Modellierungsunterstützung für die rollenbasierte Delegation in prozessgestützten Informationssystemen

Der Beitrag stellt einen integrierten Ansatz zur Modellierung und Durchsetzung von Delegationsregeln in prozessbasierten Informationssystemen vor. Basierend auf einem generischen formalen Metamodell werden delegationsbezogene Konflikte und entsprechende Lösungsstrategien diskutiert. Zudem wird eine UML-Erweiterung, eine prototypische Implementierung und eine Fallstudie anhand realer Geschäftsprozesse beschrieben.

DOI 10.1007/s11576-014-0433-3

Die Autoren

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

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

Eingegangen: 2013-03-26
Angenommen: 2014-02-03
Angenommen nach zwei Überarbeitungen durch Prof. Dr. Becker.
Online publiziert: 2014-08-23

This article is also available in English via http://www.springerlink.com and http://www.bise-journal.org: Schefer-Wenzl S, Strembeck M (2014) Modeling Support for Role-Based Delegation in Process-Aware Information Systems. Bus Inf Syst Eng. doi: 10.1007/s12599-014-0343-3.

Zusätzliche Information ist in der Online-Version dieses Beitrags (doi: 10.1007/s11576-014-0433-3) enthalten.

1 Einleitung

Ein Geschäftsprozess besteht aus einer Menge von Aufgaben, die durch eine Organisation in einer vorgegebenen Reihenfolge durchgeführt werden, um bestimmte Ziele zu erreichen (siehe z. B. zur Muehlen und Indulska 2010). Wird die Ausführung eines Geschäftsprozesses durch ein Informationssystem unterstützt, so werden Geschäftsprozesse häufig auch als Workflows bezeichnet (siehe z. B. Weske 2012). Zur Unterstützung der sicheren Ausführung eines Workflows müssen die daran beteiligten Personen die Rechte besitzen, die für die Ausführung ihrer jeweiligen Aufgaben benötigt werden (siehe z.B. Georgiadis et al. 2001; Oh und Park 2003; Strembeck und Mendling 2011; Thomas und Sandhu 1997; Wainer et al. 2003). In den vergangenen Jahren hat sich die rollenbasierte Zugriffskontrolle (engl. *role-based access control*, RBAC; siehe z. B. Ferraiolo et al. 2007; Sandhu et al. 1996) zu einem De-facto-Standard für die Zugriffskontrolle in softwarebasierten Systemen entwickelt. Hierbei werden Rollen verwendet, um verschiedene Arbeitsplatzprofile und Zuständigkeiten innerhalb einer Organisation und/oder eines Informationssystems zu modellieren. Rechte (engl. *permissions*) werden Rollen gemäß der Aufgaben (engl. *tasks*) zugeteilt, die die Mitglieder der jeweiligen Rolle erfüllen müssen. Anschließend können die Rollen an menschliche Benutzer vergeben werden (siehe z. B. Strembeck 2010). Darüber hinaus werden Rollen auch als abstraktes Konzept für die Delegation von Rechten (siehe z. B. Crampton und Khambhammettu 2008b; Wainer et al. 2007) sowie für die Zuordnung von Pflichten (engl. *duties*) verwendet, die über sogenannte Verpflichtungen (engl. *obligations*) definiert werden (siehe z. B. Schaad und Moffett 2002; Strembeck 2005; Zhao et al. 2007).

Das Konzept der Delegation dient dazu, die Verwaltung von Rechten und Pflichten zu flexibilisieren. Hierbei kann ein Subjekt (i. d. R. ein menschlicher Benutzer) seine Aufgaben, Pflichten oder Rollen an ein anderes Subjekt delegieren (siehe z. B. Schaad und Moffett 2002). In weiterer Folge handelt das Subjekt, an das eine Delegation gerichtet ist (der Delegationsempfänger, engl. *delegatee*) im Namen des delegierenden Subjekts (der Delegierende, engl. *delegator*). Die Delegation ist ein bekanntes Konzept, das in vielen Anwendungsbereichen Verwendung findet (siehe z. B. Crampton und Khambhammettu 2008c; Hasebe et al. 2010; Ravichandran und Yoon 2006).

Neben der Delegation von Rechten und Pflichten unterstützt der in diesem Beitrag vorgestellte Ansatz die Definition sogenannter *Entailment Constraints*.[1] Eine

[1] Da sich die Bezeichnung „Entailment Constraint" kaum sinnerhaltend in die deutsche Sprache übertragen lässt, verwenden wir in diesem Beitrag durchgehend die englische Bezeichnung.

aufgabenbasierte Entailment Constraint (engl. *task-based entailment constraint*) schränkt die Anzahl der Subjekte (i. d. R. Personen) ein, die eine Aufgabe$_x$ ausführen können, wenn ein bestimmtes Subjekt zuvor eine andere Aufgabe$_y$ ausgeführt hat. Die Definition von Entailment Constraints hat somit direkte Auswirkungen auf die Anzahl der Subjekte und Rollen, die bestimmte Aufgaben ausführen dürfen oder müssen (siehe z. B. Russell et al. 2005; Schefer et al. 2011; Strembeck und Mendling 2010, 2011; Tan et al. 2004; Warner und Atluri 2006; Wolter et al. 2008). Eine häufig verwendete Art von Entailment Constraints sind sogenannte *Mutual Exclusion Constraints* (deutsch etwa: „gegenseitige Ausschlusseinschränkungen"). Mutual Exclusion Constraints können verwendet werden, um Richtlinien zur Behandlung von Interessenkonflikten zu definieren. Insbesondere können hiermit verschiedene Arten der Aufgabentrennung (engl. *separation of duties*) durchgesetzt werden. Im Wesentlichen wird durch Mutual Exclusion Constraints festgelegt, dass zwei oder mehr Aufgaben durch verschiedene Personen ausgeführt werden müssen, um Interessenskonflikte zu vermeiden. Ein Interessenskonflikt entsteht beispielsweise als Ergebnis der gleichzeitigen Zuordnung von zwei sich gegenseitig ausschließenden Modellelementen (z. B. zwei sich ausschließenden Rechten oder Aufgaben) zu ein und demselben Subjekt. Eine zweite Art von Entailment Constraint sind sogenannte *Binding Constraints* (deutsch etwa: „Aufgabenbindung"). Im Gegensatz zu Mutual Exclusion Constraints definieren Binding Constraints, dass miteinander verbundene Aufgaben immer durch dasselbe Subjekt oder dieselbe Rolle auszuführen sind (siehe z. B. Strembeck und Mendling 2010, 2011; Tan et al. 2004; Wainer et al. 2003). Binding Constraints können in subjektbasierte und rollenbasierte Einschränkungen unterteilt werden (siehe z. B. Strembeck und Mendling 2010, 2011).

Die zusätzliche Modellkomplexität, die sich durch die gemeinsame Verwendung von Delegationsmechanismen und Entailment Constraints ergibt, kann in prozessbasierten Informationssystemen zu einer Reihe von Problemen führen (siehe z. B. Crampton und Khambhammettu 2008a; Schaad 2001). Insbesondere müssen sowohl zur Entwurfszeit als auch zur Laufzeit Konsistenzprüfungen durchgeführt werden, um die jederzeitige Konsistenz des zugehörigen RBAC-Modells auch bei der Delegation von Aufgaben, Rollen oder Pflichten sicherstellen zu können. Im Rahmen des vorliegenden Beitrags diskutieren wir die Einbeziehung von Delegationen bei der Prüfung und Sicherstellung der Konsistenz prozessbezogener RBAC-Modelle.

Der vorliegende Beitrag kombiniert und erweitert hierzu unsere vorherigen Veröffentlichungen Schefer und Strembeck (2011b) und Schefer-Wenzl et al. (2012) zum gleichen Themenbereich: Der Ansatz basiert auf einem formalen Metamodell, das die Kernelemente von Prozessmodellen und RBAC-Modellen miteinander integriert. Dieses Metamodell sowie verschiedene Algorithmen zur Erkennung von Delegationskonflikten wurden in Schefer-Wenzl et al. (2012) vorgestellt. Eine entsprechende Erweiterung der Unified Modeling Language (UML; OMG 2011b) unterstützt die Modellierung der zugehörigen Konzepte. Unsere UML-Erweiterung für die Spezifikation von RBAC-Delegationsmodellen mithilfe von erweiterten UML-Aktivitätendiagrammen wurde in Schefer und Strembeck (2011b) zum ersten Mal beschrieben.

Unser Ansatz verwendet Techniken der modellgetriebenen (Software-) Entwicklung (model-driven development (MDD); siehe z. B. Schmidt 2006; Selic 2003; Stahl und Völter 2006), um die Modellierung und Durchsetzung von Delegationsrichtlinien in Geschäftsprozessen zu unterstützen. Im Kontext von MDD ermöglicht ein sogenanntes *Computation-independent model* (CIM) die generische Beschreibung einer Anwendungsdomäne (oder Subdomäne). Das CIM ist unabhängig von einer bestimmten Modellierungssprache oder Technologie. Ein CIM kann anschließend verwendet werden, um ein *Platform-independent model* (PIM) der jeweiligen Domäne zu erstellen. Da das PIM unabhängig von einer bestimmten Softwareplattform und daher aus Implementierungssicht neutral ist, wird es üblicherweise mithilfe einer standardisierten Modellierungssprache beschrieben (zum Beispiel durch MOF-basierte Sprachen wie BPMN oder UML). Ein PIM beschreibt die Struktur eines Softwaresystems, die Elemente/Ergebnisse, die durch das System erzeugt werden oder den Kontroll- und Objektfluss in einem System. Schließlich beschreibt ein *Platform-specific model* (PSM) die Realisierung/Umsetzung eines Softwaresystems durch plattformspezifische Technologien und Werkzeuge.

Neben der Zusammenführung der Ergebnisse aus Schefer und Strembeck (2011b) und Schefer-Wenzl et al. (2012) stellen wir in diesem Beitrag die folgenden neuen Ergebnisse vor: Der vorliegende Beitrag beschreibt verschiedene Lösungsstrategien für Konflikte, die aus der Delegation von Rollen, Aufgaben oder Pflichten entstehen können. Weiterhin wurden alle Konzepte, die durch das formale Metamodell und die zugehörige UML-Erweiterung definiert werden, als Teil der *BusinessActivity Library and Runtime Engine* implementiert (zum Download erhältlich unter BAL 2012). Diese erweiterte Softwareplattform stellt eine Laufzeitumgebung für Geschäftsprozesse, das zugehörige RBAC-Modell sowie die entsprechenden Delegationsrichtlinien zur Verfügung. Um die praktische Anwendbarkeit des Ansatzes zu zeigen, wurde eine Fallstudie anhand realer Geschäftsprozesse durchgeführt.

Unser Beitrag weist die folgende Gliederung auf: Der 2. Abschnitt führt wichtige Begriffe ein und enthält die Definitionen des prozessbezogenen RBAC-Delegationsmodells auf der CIM-Ebene. Abschnitt 3 behandelt verschiedene Konflikte, die bei der Delegation von Rollen, Aufgaben oder Pflichten auftreten können. Weiterhin stellen wir entsprechende Konfliktlösungsstrategien vor, die die Konsistenz der Delegationsmodelle sicherstellen. Nachfolgend beschreibt Abschn. 4 Algorithmen zur automatischen Erkennung der in Abschn. 3 erläuterten Konflikte. Abschnitt 5 beschreibt eine UML-Erweiterung zur Modellierung der Delegationsrichtlinien auf der PIM-Ebene. In Abschn. 6 erläutern wir den Ablauf einer Fallstudie zur praktischen Anwendbarkeit unserer UML-Erweiterung. Im Anschluss gibt Abschn. 7 einen Überblick über unsere Softwareplattform zur Laufzeitunterstützung für prozessbezogene RBAC-Delegationsmodelle auf der PSM-Ebene. Abschließend diskutiert Abschn. 8 verwandte Arbeiten und Abschn. 9 fasst die Ergebnisse des Beitrags zusammen.

2 Prozessbezogene RBAC-Delegationsmodelle

Abbildung 1 zeigt einen vereinfachten Geschäftsprozess für die Behandlung von Kreditanträgen, der als Standard-UML-Aktivitätsdiagramm visualisiert ist und im Weiteren als laufendes Beispiel dient. Wie üblich, müssen auch bei der Ausführung dieses Prozesses menschliche

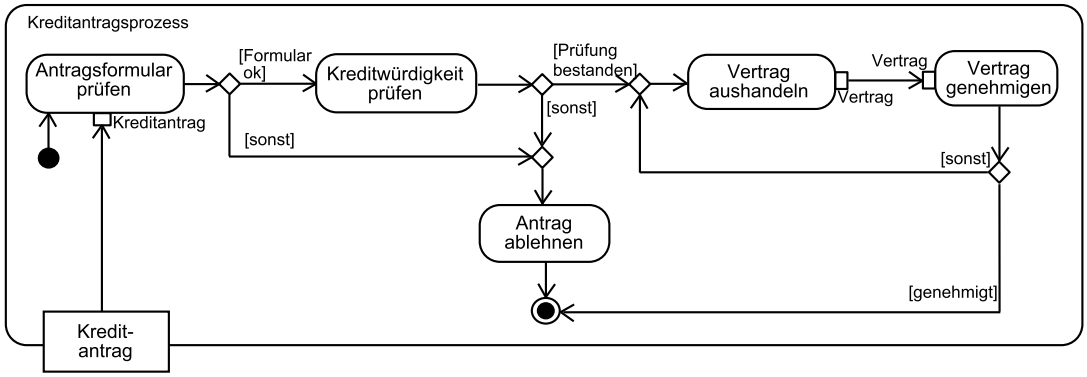


Abb. 1 Kreditantragsprozess modelliert als Standard-UML-Aktivitätendiagramm

Benutzer und Softwareprogramme bestimmte Aufgaben erfüllen. Jede Aufgabe in einem Workflow (wie etwa die Prüfung der Kreditwürdigkeit eines Kunden oder die Verhandlung eines Vertrags) ist mit bestimmten Zugriffsrechten verknüpft (z. B. Rechten zum Zugriff auf Kundendaten bzw. zur Unterzeichnung eines Kreditvertrags).

Eine Pflicht (engl. *duty*) definiert eine Aktion, die durch ein bestimmtes Subjekt auszuführen ist, um gesetzliche oder organisationsinterne Bestimmungen zu erfüllen (siehe z. B. Schefer und Strembeck 2011a; Strembeck 2005). Zum Beispiel hängt die Aufgabe „Vertrag aushandeln" aus **Abb. 1** mit einer Pflicht „Vorvertragliche Informationspflichten erfüllen" zusammen, die sich aus den zugehörigen gesetzlichen Vorgaben ergibt.[2] Ein Subjekt, das für die Durchführung dieser Aufgabe und die Erfüllung der damit zusammenhängenden Pflichten zuständig ist, benötigt somit auch die für die Ausführung erforderlichen Zugriffsrechte (siehe z. B. Strembeck 2005; Zhao et al. 2007).

Die Geschäftsprozesse und Softwaresysteme einer Organisation werden häufig mithilfe grafischer Modellierungssprachen spezifiziert. Da heute verfügbare Standardmodellierungssprachen (wie z. B. UML oder BPMN) keine Modellelemente zur Abbildung von Rollen, Rechten, Pflichten oder den zugehörigen Delegationsrichtlinien (im Sinne von RBAC) zur Verfügung stellen, müssen derartige Informationen den Modellen zumeist über rein textuelle Beschreibungen (oft in Form von „Prosatext") hinzugefügt werden (siehe z. B. Recker et al. 2006). Die fehlende Verbindung der Prozessmodelle auf der einen und der entsprechenden RBAC-Modellelemente auf der anderen Seite führt häufig zu mangelnder Transparenz und zu Inkonsistenzen zwischen Prozessbeschreibungen und den tatsächlichen ausgeführten Prozessinstanzen (siehe z. B. Wolter et al. 2009). Vor allem im Rahmen der Delegation, wo die Zuordnung des ursprünglichen Subjekts zu einer Aufgabe geändert wird, ist die Definition, Überwachung und tatsächliche Durchsetzung der delegierten Aufgaben und Pflichten schwierig. Das Problem wird noch verstärkt, falls ein Softwaresystem die mehrstufige Delegation von Aufgaben, Rechten und Pflichten erlaubt (siehe z. B. Barka und Sandhu 2000b).

2.1 Delegation im Geschäftsprozesskontext

Das in diesem Beitrag beschriebene Delegationsmodell erlaubt die Delegation von Rollen, Aufgaben und Pflichten. Im Wesentlichen kann jedes Subjekt ein ihm selbst zugewiesenes Recht (zur Durchführung einer Aufgabe in einem Geschäftsprozess) und die damit zusammenhängenden Pflichten an ein anderes Subjekt delegieren. Eine permanente Delegation gilt hierbei für alle Instanzen eines bestimmten Geschäftsprozesses (z. B. jede Instanz des Kreditantragsprozesses). Eine temporäre Delegation gilt hingegen nur für eine bestimmte Prozessinstanz (z. B. Herrn Mayers Kreditantragsprozess).

Im Kontext der rollenbasierten Zugriffskontrolle wird häufig das Konzept der sogenannten Delegationsrollen verwendet (siehe z. B. Joshi und Bertino 2006; Shang und Wang 2008; Zhang et al. 2003b). Auch im hier vorgestellten Ansatz wird durch den Delegierenden zunächst eine Delegationsrolle erstellt, der anschließend die zu delegierenden Aufgaben und Pflichten zugewiesen werden (ähnlich wie in Zhang et al. 2003b). Auf diese Weise ist jede Pflicht mit einer bestimmten Aufgabe verbunden (siehe z. B. Schefer und Strembeck 2011a). Damit eine Aufgabe, eine Pflicht oder eine Rolle delegiert werden kann, muss diese zunächst als delegierbar gekennzeichnet werden (siehe unten). Der Delegierende kann dann alle oder einen Teil seiner delegierbaren Aufgaben, Pflichten oder Rollen delegieren, indem er diese einer Delegationsrolle zuweist. Nachfolgend werden die Delegationsrollen den Delegationsempfängern zugewiesen und können für temporäre oder permanente Delegation verwendet werden.

Allgemein werden Delegationsrollen sowie die Zuweisungen von Aufgaben und Pflichten zu diesen Delegationsrollen durch das jeweils delegierende Subjekt verwaltet. Alle anderen Rollen werden als reguläre Rollen bezeichnet und üblicherweise durch den (oder die) Sicherheitsbeauftragten der jeweiligen Organisation verwaltet.

Neben den oben erwähnten Basiselementen (Rollen, Rechte/Aufgaben, Pflichten), ermöglicht der hier vorgestellte Ansatz die Definition von Entailment Constriants. Insbesondere erlaubt der Ansatz die Definition von *Static Mutual-Exclusion (SME) Constraints* und *Dynamic Mutual-Exclusion (DME) Constraints*. Im Allgmeinen werden Mutual Exclusion Constraints in Workflows verwendet, um Interessenskonflikte zu vermeiden (siehe z. B. Botha und Eloff 2001; Casati et al. 2001; Strembeck und Mendling 2010, 2011; Tan et al. 2004; Wainer et al. 2003; Warner und Atluri 2006). Eine SME-Constraint definiert,

[2] Die Pflicht „Vorvertragliche Informationspflichten erfüllen" ist in **Abb. 1** nicht dargestellt (siehe unten).

dass zwei sich statisch gegenseitig ausschließende Aufgaben nicht ein und demselben Subjekt zugewiesen werden dürfen – d. h. kein Subjekt darf in den Besitz zweier sich statisch ausschließender Rechte/Aufgaben gelangen. Im Gegensatz dazu definiert eine DME-Constraint, dass zwei sich dynamisch gegenseitig ausschließende Aufgaben nicht in ein und derselben Prozessinstanz durch dasselbe Subjekt ausgeführt werden dürfen – d. h. ein Subjekt darf zwei sich dynamisch ausschließende Rechte/Aufgaben zwar besitzen aber nicht gleichzeitig benutzen. Neben Mutual Exclusion Constraints können in unserem Ansatz auch *Subject-Binding (SB) Constraints* (deutsch etwa: „Subjektbindung“) und *Role-Binding (RB) Constraints* (deutsch etwa: „Rollenbindung“) definiert werden (siehe z. B. Strembeck und Mendling 2010, 2011). Hierbei definiert eine SB-Constraint, dass zwei aneinander gebundene Aufgaben durch ein und dieselbe Person ausgeführt werden müssen. Eine RB-Constraint definiert, dass die aneinander gebundenen Aufgaben durch Mitglieder derselben Rolle auszuführen sind, aber nicht notwendigerweise durch ein und dieselbe Person.

Um die jederzeitige korrekte Durchsetzung der Entailment Constraints sicherzustellen, müssen diese Einschränkungen auch im Kontext der Delegation berücksichtigt werden (siehe Abschn. 4). Zum Beispiel darf die Delegation von Aufgaben, Pflichten oder Rollen einen Delegationsempfänger nicht dazu berechtigen, zwei sich statisch ausschließende Aufgaben durchzuführen (d. h. zwei Aufgaben, für die ein SME-Constraint definiert ist). Falls jedoch z. B. ein Subject-Binding zwischen zwei Aufgaben definiert ist, müssen diese beiden Aufgaben immer gemeinsam delegiert werden. Andernfalls könnte die SB-Constraint nicht erfüllt werden.

2.2 Generisches Metamodell für prozessbezogene RBAC-Delegationsmodelle

Dieser Abschnitt beschreibt formale Definitionen für prozessbezogene RBAC-Delegationsmodelle auf der CIM-Ebene. Die Definitionen 1–3 fassen die Definitionen für prozessbezogene RBAC-Modelle zusammen (für weitere Details siehe Strembeck und Mendling 2011). Die formalen Definitionen können in weiterer Folge als Grundlage für die Erweiterung beliebiger Prozessmodellierungssprachen und/oder Softwaresysteme verwendet werden. In diesem Beitrag beschreiben wir exemplarisch eine entsprechende UML-Erweiterung (siehe Abschn. 5).

Definition 1 (Prozessbezogenes RBAC-Modell) Es sei $PRM = (E, Q, D)$ ein prozessbezogenes RBAC-Modell, wobei sich $E = S \cup R \cup P_T \cup P_I \cup T_T \cup T_I$ auf paarweise disjunkte Mengen des Modells, $Q = rh \cup rsa \cup tra \cup es \cup er \cup ar \cup pi \cup ti$ auf Relationen und $D = sb \cup rb \cup sme \cup dme$ auf Mutual Exclusion und Binding Constraints beziehen, sodass

- Für die Mengen des Metamodells gilt:
  - Ein Element von S wird *Subjekt* genannt. $S \neq \emptyset$.
  - Ein Element von R wird *Rolle* genannt. $R \neq \emptyset$.
  - Ein Element von P_T wird *Prozesstyp* genannt. $P_T \neq \emptyset$.
  - Ein Element von P_I wird *Prozessinstanz* genannt.
  - Ein Element von T_T wird *Aufgabentyp* genannt. $T_T \neq \emptyset$.
  - Ein Element von T_I wird *Aufgabeninstanz* genannt.
- Für die Relationen des Metamodells gilt ($\mathcal{P}$ bezeichnet die Potenzmenge):
  1. Die Relation $rh : R \mapsto \mathcal{P}(R)$ wird als *Rollenhierarchie* bezeichnet. Für $rh(r_s) = R_j$ bezeichnen wir r_s als *Senior-Rolle* und R_j als die Menge der direkten *Junior-Rollen*. Die Relation rh^* definiert die Vererbung in der Rollenhierarchie, sodass $rh^*(r_s) = R_{j^*}$ alle direkten und transitiven Junior-Rollen umfasst, aus denen die Senior-Rolle r_s erbt. Die Rollenhierarchie enthält keine Zyklen, d. h. für jedes $r \in R$: $rh^*(r) \cap \{r\} = \emptyset$.
  2. Die Relation $rsa : S \mapsto \mathcal{P}(R)$ wird als *Zuweisung von Rollen zu Subjekten* bezeichnet (engl. role-to-subject assignment). Für $rsa(s) = R_s$ bezeichnen wir s als *Subjekt* und $R_s \subseteq R$ als die Menge der *Rollen, die diesem Subjekt zugeordnet sind.*

     Diese Relation impliziert eine Relation *Rollenbesitz* (engl. role ownership) $rown : S \mapsto \mathcal{P}(R)$, sodass für jedes Subjekt s alle direkten und vererbten Rollen enthalten sind, d. h. $rown(s) = \bigcup_{r \in rsa(s)} rh^*(r) \cup rsa(s)$. Die Relation $rown^{-1} : R \mapsto \mathcal{P}(S)$ liefert alle Subjekte, die einer Rolle zugeordnet sind (direkt oder transitiv über eine Rollenhierarchie).
  3. Die Relation $es : T_I \mapsto S$ wird als *Zuordnung des ausführenden Subjekts* (engl. executing subject) bezeichnet. Für $es(t) = s$ bezeichnen wir s als das *ausführende Subjekt* und t als die *ausgeführte Aufgabeninstanz.*
  4. Die Relation $er : T_I \mapsto R$ wird als *Zuordnung der ausführenden Rolle* (engl. executing role) bezeichnet. Für $er(t) = r$ bezeichnen wir r als die *ausführende Rolle* und t als die *ausgeführte Aufgabeninstanz.*
  5. Die Relation $tra : R \mapsto \mathcal{P}(T_T)$ wird als *Zuweisung der Aufgaben zu Rollen* bezeichnet (engl. task-to-role assignment). Für $tra(r) = T_r$ bezeichnen wir r als *Rolle* und $T_r \subseteq T_T$ als die Menge der Aufgaben, die r zugewiesen sind.

     Diese Relation impliziert eine Relation *Aufgabenbesitz* (engl. task ownership) $town : R \mapsto \mathcal{P}(T_T)$, sodass für jede Rolle r die von ihren Junior-Rollen geerbten Aufgaben in der Menge enthalten sind, d. h. $town(r) = \bigcup_{r_{inh} \in rh^*(r)} tra(r_{inh}) \cup tra(r)$. Die Relation $town^{-1} : T_T \mapsto \mathcal{P}(R)$ liefert die Menge der Rollen, die einer Aufgabe zugeordnet sind (direkt oder transitiv über die Rollenhierarchie).
  6. Die Relation $ti : (T_T \times P_I) \mapsto \mathcal{P}(T_I)$ wird als *Aufgabeninstanziierung* (engl. task instantiation) bezeichnet. Für $ti(t_T, p_I) = T_i$ bezeichnen wir $T_i \subseteq T_I$ als Menge der *Aufgabeninstanzen*, $t_T \in T_T$ wird als *Aufgabentyp* und $p_I \in P_I$ wird als *Prozessinstanz* bezeichnet.
  7. Die Relation $ar : S \mapsto R$ liefert die derzeit *aktivierte Rolle* (engl. active role) eines Subjekts. Für $ar(s) = r$ bezeichnen wir s als Subjekt und r als die aktivierte Rolle von s.[3]
  8. Weiterhin ermöglicht das Modell die Definition von Entailment Constraints in Form von Subject Binding, Role Binding, Static Mutual Exclusion und Dynamic Mutual Exclusion Constraints für Aufgabentypen. Die entsprechenden Konsistenzbedingungen sind

[3] Wir nehmen an, dass jedes Subjekt jederzeit die Rollen aktivieren (oder deaktivieren) kann, die diesem Subjekt direkt oder indirekt über die Rollenhierarchie zugewiesen wurden (siehe z. B. Ferraiolo et al. 1999; Sandhu et al. 1996)

in Strembeck und Mendling (2011) spezifiziert:

Die Relation $sb : T_T \mapsto \mathcal{P}(T_T)$ wird als ***Subject-Binding*** (deutsch etwa: „Subjektbindung“) bezeichnet. Für $sb(t) = T_{sb}$ bezeichnen wir t als die *bindende Aufgabe* und $T_{sb} \subseteq T_T$ als die Menge der *gebundenen Aufgaben*.

Die Relation $rb : T_T \mapsto \mathcal{P}(T_T)$ wird als ***Role-Binding*** (deutsch etwa: „Rollenbindung“) bezeichnet. Für $rb(t) = T_{rb}$ bezeichnen wir t als die *bindende Aufgabe* und $T_{rb} \subseteq T_T$ als die Menge der *gebundenen Aufgaben*.

Die Relation $sme : T_T \mapsto \mathcal{P}(T_T)$ wird als ***Static Mutual Exclusion*** (deutsch etwa: „statischer gegenseitiger Ausschluss“) bezeichnet. Für $sme(t_1) = T_{sme}$ mit $T_{sme} \subseteq T_T$ bezeichnen wir jedes Paar t_1 und $t_x \in T_{sme}$ als *sich statisch gegenseitig ausschließende Aufgaben*.

Die Relation $dme : T_T \mapsto \mathcal{P}(T_T)$ wird als ***Dynamic Mutual Exclusion*** (deutsch etwa: „dynamischer gegenseitiger Ausschluss“) bezeichnet. Für $dme(t_1) = T_{dme}$ mit $T_{dme} \subseteq T_T$ bezeichnen wir jedes Paar t_1 und $t_x \in T_{dme}$ als *sich dynamisch gegenseitig ausschließende Aufgaben*.

Definition 2 beinhaltet Regeln für die statische Korrektheit der prozessbezogenen RBAC-Modelle, die die Konsistenz der entsprechenden Modellelemente und Elementbeziehungen zum Entwurfszeitpunkt sicherstellen.

Definition 2 Es sei $PRM = (E, Q, D)$ ein prozessbezogenes RBAC-Modell. *PRM* wird als statisch korrekt bezeichnet, wenn die folgenden Bedingungen erfüllt sind:

1. Aufgaben können sich nicht selbst ausschließen:

   $\forall t_2 \in sme(t_1)\colon t_1 \neq t_2$ und
   $\forall t_2 \in dme(t_1)\colon t_1 \neq t_2$

2. Bidirektionalität von Mutual Exclusion Constraints:

   $\forall t_2 \in sme(t_1)\colon t_1 \in sme(t_2)$ und
   $\forall t_2 \in dme(t_1)\colon t_1 \in dme(t_2)$

3. Aufgaben können nicht an sich selbst gebunden werden:

   $\forall t_2 \in sb(t_1)\colon t_1 \neq t_2$ und
   $\forall t_2 \in rb(t_1)\colon t_1 \neq t_2$

4. Bidirektionalität von Binding Constraints:

   $\forall t_2 \in sb(t_1)\colon t_1 \in sb(t_2)$ und
   $\forall t_2 \in rb(t_1)\colon t_1 \in rb(t_2)$

5. Zwei Aufgaben schließen sich entweder statisch oder dynamisch gegenseitig aus:

   $\forall t_2 \in sme(t_1)\colon t_2 \notin dme(t_1)$

6. Entweder SME-Constraint oder Binding Constraint:

   $\forall t_2 \in sme(t_1)\colon t_2 \notin sb(t_1) \wedge t_2 \notin rb(t_1)$

7. Entweder DME-Constraint oder Subject Binding Constraint:

   $\forall t_2 \in dme(t_1)\colon t_2 \notin sb(t_1)$

8. Konsistenz von Aufgabenzuweisung und SME:

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

9. Konsistenz von Rollenzuweisung und SME:

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

Definition 3 spezifiziert Regeln für die dynamische Korrektheit eines prozessbezogenen RBAC-Modells, d. h. Regeln, die nur zur Laufzeit anhand einer konkreten *Prozessinstanz* geprüft werden können.

Definition 3 Es sei $PRM = (E, Q, D)$ ein prozessbezogenes RBAC-Modell und P_I die Menge der zugehörigen Prozessinstanzen. *PRM* wird als dynamisch korrekt betrachtet, wenn die folgenden Bedingungen erfüllt sind:

1. In derselben Prozessinstanz muss sich das ausführende Subjekt der SME-Aufgaben unterscheiden: $\forall t_2 \in sme(t_1),\ pi \in P_I : \forall t_x \in ti(t_2, pi), t_y \in ti(t_1, pi) : es(t_x) \cap es(t_y) = \emptyset$
2. In derselben Prozessinstanz muss sich das ausführende Subjekt der DME Aufgaben unterscheiden: $\forall t_2 \in dme(t_1),\ pi \in P_I : \forall t_x \in ti(t_2, pi), t_y \in ti(t_1, pi) : es(t_x) \cap es(t_y) = \emptyset$
3. In derselben Prozessinstanz müssen die rollengebundenen Aufgaben dieselbe ausführende Rolle haben: $\forall t_2 \in rb(t_1),\ pi \in P_I : \forall t_x \in ti(t_2, pi),\ t_y \in ti(t_1, pi) : er(t_x) = er(t_y)$
4. In derselben Prozessinstanz müssen die subjektgebundenen Aufgaben dasselbe ausführende Subjekt haben: $\forall t_2 \in sb(t_1),\ pi \in P_I : \forall t_x \in ti(t_2, pi), t_y \in ti(t_1, pi) : es(t_x) = es(t_y)$

Abbildung 2 zeigt ein konzeptuelles Modell (dargestellt als Klassendiagramm), das die wesentlichen Elemente und Beziehungen von prozessbezogenen RBAC-Delegationsmodellen skizziert. Ein grafisches Metamodell ist prinzipiell gut geeignet, um den Zusammenhang zwischen verschiedenen Artefakten darzustellen. Es können jedoch meist nicht alle formalen Zusammenhänge und Invarianten der Modellelemente visuell dargestellt werden. Definition 4 enthält daher die formalen Definitionen der wesentlichen Elemente des Metamodells. Insbesondere kombinieren wir hierbei bekannte Konzepte mehrerer bestehender Delegationsmodelle (siehe z. B. Crampton und Khambhammettu 2008a; Hasebe et al. 2010; Schaad und Moffett 2002; Zhang et al. 2003b) und integrieren sie in das in den Definitionen 1–3 beschriebene Metamodell.

Definition 4 (Prozessbezogenes RBAC-Delegationsmodell) Es sei $PRDM = (E, Q, D, DL)$ ein prozessbezogenes Delegationsmodell, wobei E sich auf die paarweise disjunkten Mengen des Metamodells bezieht, Q auf die Relationen, D auf die Entailment Constraints und DL auf Relationen für Delegationsrichtlinien.

Die zusätzlichen Mengen des prozessbezogenen RBAC-Delegationsmodells sind:

- Ein Element von *RR* wird als *Reguläre Rolle (engl. regular role)* bezeichnet. $RR \subseteq R$.
- Ein Element von *DR* wird als *Delegationsrolle (engl. delegation role)* bezeichnet. $DR \subseteq R$
- Ein Element von *DRT* wird als *Temporäre Delegationsrolle (engl. temporary delegation role)* bezeichnet. $DRT \subseteq DR$.
- Ein Element von DT_T wird als *delegierbarer Aufgabentyp (engl. delegable task type)* bezeichnet. $DT_T \subseteq T_T$.
- Ein Element von DU_T wird als *Pflichtentyp (engl. duty type)* bezeichnet.
- Ein Element von DU_I wird als *Pflichtinstanz (engl. duty instance)* bezeichnet.
- Ein Element von DDU_T wird als *delegierbarer Pflichtentyp (engl. delegable duty type)* bezeichnet. $DDU_T \subseteq DU_T$.

Nachfolgend definieren wir die Relationen für das RBAC-Delegationsmodell: $DL = rrh \cup drh \cup creator \cup drpi \cup trra \cup trdel \cup dta \cup rrsa \cup drsa \cup dui \cup res \cup rer$ ($\mathcal{P}$ bezeichnet die Potenzmenge):

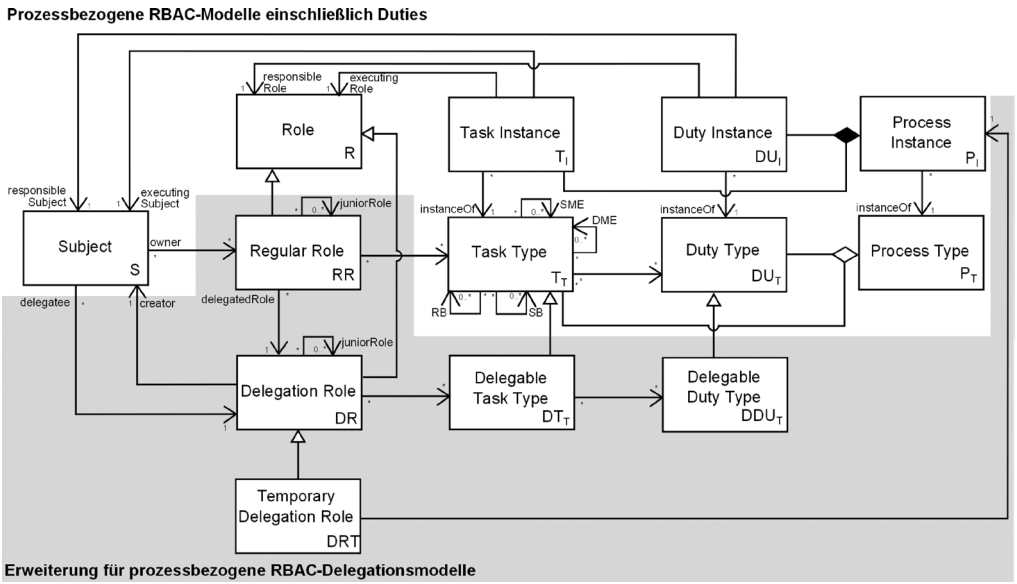


Abb. 2 Hauptelemente von prozessbezogenen RBAC-Delegationsmodellen (Konzeptüberblick)

1. Die Rollen R werden in reguläre Rollen (RR) und Delegationsrollen (DR) unterteilt. In RBAC können Rollen miteinander in Beziehung stehen und eine (oder mehrere) Rollenhierarchie(n) bilden, wobei Senior-Rollen die Berechtigungen von ihren Junior-Rollen erben. In einer Bank könnte etwa eine Senior-Rolle „Bankmanager“ alle Rechte ihrer Junior-Rolle „Bankkaufmann“ erben. Wenn „Bankkaufmann“ selbst Junior-Rollen hat, erbt der „Bankmanager“ transitiv alle Berechtigungen von den ihm transitiv untergeordneten Junior-Rollen. Um die ungewollte Vererbung von Rechten zu verhindern, besteht die reguläre Rollenhierarchie nur aus regulären Rollen. Die nachfolgende definierte Relation ersetzt die Relation aus Definition 1.1:

   Die Relation $rrh : RR \mapsto \mathcal{P}(RR)$ wird als ***reguläre Rollenhierarchie*** (engl. regular role hierarchy) bezeichnet. Für $rrh(r_s) = RR_j$ bezeichnen wir $r_s \in RR$ als die *reguläre Senior-Rolle* und $RR_j \subseteq RR$ als die Menge der direkten regulären Junior-Rollen. Die Relation rrh^* definiert die Vererbung in der Rollenhierarchie, sodass $rrh^*(r_s) = RR_{j*}$ alle direkten und transitiven Junior-Rollen umfasst, aus denen die Senior-Rolle r_s erbt. Die reguläre Rollenhierarchie enthält keine Zyklen, d. h. für jedes $r \in RR : rrh^*(r) \cap \{r\} = \emptyset$.
2. Die Delegationsrollen können in einer Delegationsrollenhierarchie (engl. delegation role-hierarchy) angeordnet werden. Hierbei ist zu beachten, dass jede Delegationsrolle reguläre Junior-Rollen oder delegierte Juinor-Rollen haben kann (siehe z. B. Zhang et al. 2003b). Die Delegationsrollen dürfen jedoch *keine* regulären Senior-Rollen haben, um ungewollte Vererbung der Berechtigungen in der regulären Rollenhierarchie zu verhindern:

   Die Relation $drh : DR \mapsto \mathcal{P}(R)$ wird als ***Delegationsrollenhierarchie*** bezeichnet. Für $drh(dr_s) = R_j$ bezeichnen wir $dr_s \in DR$ als *Senior-Delegationsrolle* und $R_j \subseteq R$ als die *Menge der direkten Junior-Delegationsrollen*. Die Relation drh^* definiert die Vererbung in der Rollenhierarchie, sodass $drh^*(dr_s) = R_{j*}$ alle direkt und transitiv untergeordneten Rollen umfasst, aus denen die Senior-Rolle dr_s erbt. Die Delegationsrollenhierarchie enthält keine Zyklen, d. h. für jedes $r \in R : drh^*(r) \cap \{r\} = \emptyset$.
3. Jedes Subjekt kann eine beliebige Anzahl von Delegationsrollen erstellen. Im weiterer Folge handelt der Ersteller (engl. creator) als der Delegierende (engl. delegator) seiner Delegationsrollen. Wenn zum Beispiel das Subjekt „Alice“ eine Delegationsrolle „Sommerpraktikant“ erstellt, kann sie Teile (oder alle) der ihr zugeteilten Aufgaben und Pflichten an die Rolle „Sommerpraktikant“ delegieren:

   Die Relation $creator(dr) : DR \mapsto S$ wird als ***Delegationsrollenersteller*** bezeichnet. Für $creator(dr) = s$, bezeichnen wir $dr \in DR$ als *Delegationsrolle* und $s \in S$ als den *Ersteller dieser Delegationsrolle*.
4. Standardmäßig ist eine Delegationsrolle permanent. Bei einer temporären Delegation muss eine temporäre Delegationsrolle erstellt werden. Wenn das Subjekt „Alice“ z. B. in den Urlaub fahren möchte und noch Kreditanträge hat, die nicht bearbeitet sind, kann Alice eine Delegationsrolle „Sommerpraktikant“ erstellen, die nur für die nicht vollständig bearbeiteten Anträge Gültigkeit hat:

   Die Relation $drpi : DRT \mapsto \mathcal{P}(P_I)$ wird als ***temporäre Delegationsrollenzuordnung*** (engl. temporary delegation role mapping) bezeichnet. Für $drpi(drt) = P_{drt}$, bezeichnen wir $drt \in DRT$ *temporäre Delegationsrolle* und $P_{drt} \subseteq P_I$ als die *Menge von Prozessinstanzen, für die drt gültig ist*.
5. Aufgabentypen werden regulären Rollen zugewiesen, um die Berechtigungen der jeweiligen Rolle zu definieren. Die folgende Relation ersetzt die Relation zur Zuweisung der Aufgaben zu Rollen *tra* aus Definition 1.5:

   Die Relation $trra : RR \mapsto \mathcal{P}(T_T)$ wird als ***Zuweisung von Aufgaben zu regulären Rollen*** (engl. task-to-regular role assignment) bezeichnet. Für $trra(r) = T_r$ bezeichnen wir $r \in RR$ *reguläre Rolle* und $T_r \subseteq T_T$ als die *Menge der Aufgaben, die r zugeordnet sind*. Die Relation $trra^{-1} : T_T \mapsto \mathcal{P}(RR)$ liefert die Menge der regulären Rollen, denen eine bestimmte Aufgabe zugeordnet ist.

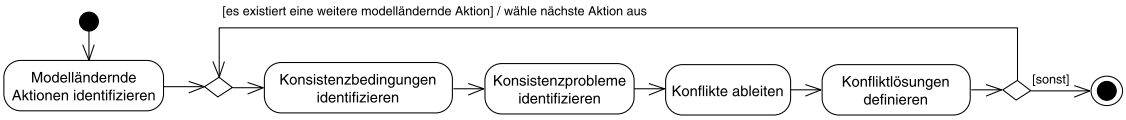


Abb. 3 Allgemeiner Prozess zur Identifikation von Konflikten und Konfliktlösungen

6. Aufgabentypen können als delegierbar definiert werden. Nur delegierbare Aufgaben können Delegationsrollen zugeordnet werden. Ein Subjekt kann eine (delegierbare) Aufgabe delegieren, indem es diese Aufgabe einer Delegationsrolle zuweist:

   Die Relation $trdel : DR \mapsto \mathcal{P}(DT_T)$ wird als ***Aufgabe-zu-Rollen-Delegation*** (engl. task-to-role delegation) bezeichnet. Für $trdel(dr) = DT_{dr}$ heißt $dr \in DR$ *Delegationsrolle* and $DT_{dr} \subseteq DT_T$ die *Menge der delegierten Aufgaben, die dr zugewiesen sind*. Die Relation $trdel^{-1} : DT_T \mapsto \mathcal{P}(DR)$ gibt die Menge der Delegationsrollen an, denen eine bestimmte delegierbare Aufgabe zugeordnet ist.
7. Eine Pflicht (engl. duty) definiert eine Aktion, die durch ein bestimmtes Subjekt durchgeführt werden muss, um gesetzliche oder organisatorische Bestimmungen einzuhalten. In Zusammenhang mit einem Geschäftsprozess wird jede Pflicht einer Aufgabe zugeordnet (siehe Schefer und Strembeck 2011a):

   Die Relation $dta : T_T \mapsto \mathcal{P}(DU_T)$ wird als ***Pflicht-zu-Aufgaben-Zuordnung*** (engl. duty-to-task assignment) bezeichnet. Für $dta(t) = DU_x$ heißt $t \in T_T$ *Aufgabentyp* und $DU_x \subseteq DU_T$ die *Menge der Pflichten, die diesem Aufgabentyp zugeordnet sind*.
8. Delegierbare Aufgaben können nur delegiert werden, wenn alle damit zusammenhängenden Pflichten ebenfalls delegierbar sind: $\forall t_x \in trdel(dr) : \forall du \in dta(t_x) : du \in DDU_T$.
9. Reguläre Rollen werden Subjekten zugeordnet. Dadurch erhalten Subjekte die Rechte zur Ausführung der entsprechenden Aufgaben und Pflichten. Die folgende Relation ersetzt die Relation zur Zuweisung von Rollen zu Subjekten *rsa* aus Definition 1.2:

   Die Relation $rrsa : S \mapsto \mathcal{P}(RR)$ wird als ***Zuweisung regulärer Rollen zu einem Subjekt*** (engl. regular-role-to-subject assignment) bezeichnet. Für $rrsa(s) = RR_s$ bezeichnen wir $s \in S$ als *Subjekt* und $RR_s \in RR$ als die *Menge der regulären Rollen, deren Besitzer s ist*. Die Relation $rrsa^{-1} : RR \mapsto \mathcal{P}(S)$ liefert alle Subjekte, die einer regulären Rolle zugeordnet sind.
10. Delegationsrollen werden Delegationsempfängern zugeordnet. Nachfolgend sind die jeweiligen Delegationsempfänger berechtigt und verpflichtet die delegierten Aufgaben und Pflichten auszuführen:

    Die Relation $drsa : S \mapsto \mathcal{P}(DR)$ wird als ***Delegationsrollen-zu-Subjekt-Zuweisung*** (engl. delegation-role-to-subject assignment) bezeichnet. Für $drsa(s) = DR_s$ bezeichnen wir $s \in S$ als *Delegationsempfänger* und $DR_s \in DR$ als die *Menge der Delegationsrollen, deren Besitzer s ist*. Die Relation $drsa^{-1} : DR \mapsto \mathcal{P}(S)$ liefert alle Delegationsempfänger, die einer Delegationsrolle zugeordnet sind.
11. Für jeden Pflichtentyp kann eine beliebige Anzahl an Instanzen erzeugt werden:

    Die Relation $dui : (DU_T \times P_I) \mapsto \mathcal{P}(DU_I)$ wird als ***Pflichteninstanziierung*** (engl. duty instantiation) bezeichnet. Für $dui(du_T, p_I) = DU_i$ heißt $DU_i \subseteq DU_I$ *Menge der Pflichtinstanzen*, $du_T \in DU_T$ *Pflichtentyp* und $p_I \in P_I$ *Prozessinstanz*.
12. Die Relation $res : DU_I \mapsto S$ wird als ***Zuordnung des zuständigen Subjekts*** (engl. responsible-subject mapping) bezeichnet. Für $res(du) = s$ heißt $s \in S$ das *zuständige Subjekt* und $du \in DU_I$ die *zugehörige Pflichtinstanz*.
13. Innerhalb derselben Prozessinstanz ist ein Subjekt, das eine Aufgabe ausführt, auch für die Erfüllung aller damit zusammenhängenden Pflichten zuständig:

    $\forall du \in dta(t_1), p_i \in P_I:$
    $\forall t_x \in ti(t_1, p_i), du_x \in dui(du, p_i):$
    $es(t_x) = res(du_x)$
14. Die Relation $rer : DU_I \mapsto R$ wird als ***Zuordnung der zuständigen Rolle*** (engl. responsible-role mapping) bezeichnet. Für $rer(du) = r$ heißt $r \in R$ die *zuständige Rolle* und $du \in DU_I$ die *zugehörige Pflichtinstanz*.

3 Identifikation und Lösung von Delegationskonflikten

Bedingt durch die immanente Komplexität von prozessbezogenen Delegationsmodellen können verschiedene Arten von Konflikten auftreten. In unserem Delegationsmodell haben wir 14 potenzielle Konflikte identifiziert, die bei der Delegation von Rollen, Aufgaben oder Pflichten geprüft werden müssen, um unzulässige Aufgabenzuweisungen zu vermeiden. **Abbildung 3** zeigt ein UML-Aktivitätsdiagramm, das den allgemeinen Prozess skizziert, den wir für die Identifikation verschiedener Konflikte und Konfliktlösungen angewandt haben. Insbesondere bestimmen wir zuerst die Aktionen, die ein (konsistentes) RBAC-Modell ändern (wie etwa das Zuweisen neuer Aufgaben zu Rollen oder die Definition neuer Constraints). Als nächstes identifizieren wir Konsistenzbedingungen, die eingehalten werden müssen, wenn die jeweilige Änderungsaktion durchgeführt wird. Auf Grundlage dieser Konsistenzbedingungen identifizieren wir potenzielle Konsistenzprobleme und leiten die jeweiligen Konflikte ab. In einem letzten Schritt definieren wir mögliche Lösungsstrategien für jeden der Konflikte. Diese Schritte werden für jede Änderungsaktion wiederholt.

Jeder der identifizierten Konflikte bezieht sich hierbei direkt auf die Konsistenzbedingungen für prozessbezogene Delegationsmodelle (siehe Abschn. 2.2). Viele dieser Konsistenzbedingungen wurden zuvor auch bereits durch andere Wissenschaftler identifiziert (siehe z. B. Botha und Eloff 2001; Crampton und Khambhammettu 2008a, 2008b; Strembeck und Mendling 2011; Zhang et al. 2003b). **Tabelle 1** zeigt die Zusammenhänge zwischen verschiedenen Delegationskonflikten und der oder den jeweiligen formalen Konsistenzbedingung(en).

Nachfolgend werden die einzelnen potenziellen Delegationskonflikte und die jeweiligen Lösungsstrategien ausführlich diskutiert. Die Konflikte werden durch Anwendung entsprechender Algorithmen erkannt (siehe Abschn. 4). In weiterer Folge können diese Konflikte

Tab. 1 Zusammenhang zwischen den formellen Konsistenzbedingungen und den Algorithmen

Delegationskonflikt	Konsistenzbedingung
Erstellerkonflikt	Definition 4.3
Konflikt mit der delegierbaren Aufgabe	Definition 4.6
Konflikt mit der delegierbaren Pflicht	Definition 4.8
Konflikt mit der Zuweisung der zu delegierenden Aufgabe (town)	Definitionen 4.6 und 4.9
Konflikt mit der Zuweisung der zu delegierenden Rolle (rown)	Definitionen 4.3 und 4.9
SME-Konflikt bei Aufgabenzuweisung	Definitionen 1.8, 2.8, und 4.5
SME-Konflikt bei Rollenzuweisung	Definitionen 1.8, 2.9, und 4.5
Konflikt mit der SB-Delegation	Definitionen 1.8, 2.3, 2.4, 3.2, und 4.5
Konflikt mit der RB-Delegation	Definitionen 1.8, 2.3, 2.4, 3.3, und 4.5
Konflikt mit der SB-Pflichtendelegation	Definitionen 1.8 4.5, 4.7, und 4.14
Konflikt mit der RB-Pflichtendelegation	Definitionen 1.8 4.5, 4.7, und 4.14
Konflikt mit der Selbstdelegation	Definitionen 4.1 und 4.2
Konflikt mit der zyklischen Delegation	Definition 4.2
Konflikt mit der temporären Delegationsrolle	Definition 4.4

durch Anwendung einer der 21 vordefinierten Lösungsstrategien aufgelöst werden. Diese Lösungen verhindern inkonsistente Delegationrelationen oder Laufzeitzuweisungen. Für die meisten Konflikte gibt es mehrere alternative Lösungen. Die Entscheidung, welche der möglichen Lösungen angewandt werden sollte, verlangt normalerweise spezielles Wissen der jeweiligen Anwendungsdomäne (z. B. Bank, Versicherung, Krankenhaus, Energieerzeuger) und ist stark abgängig vom jeweiligen organisatorischen Kontext sowie der entsprechenden RBAC-Konfiguration. Eine formale Beschreibung der Lösungsstrategien findet sich in Online-Anhang A.

3.1 Identifikation und Lösung von Delegationskonflikten

Erstellerkonflikt (engl. creator conflict): Ein Erstellerkonflikt tritt auf, wenn ein Subjekt versucht, eine Delegationsrolle zu delegieren, die er/sie nicht erstellt hat. Nur der Ersteller einer Delegationsrolle kann diese delegieren und ihr Delegationsempfänger zuweisen. Zum Beispiel versucht in **Abb. 4a** Subjekt s_1 Aufgabe t_x an die Delegationsrolle dr_y zu delegieren. t_x ist prinzipiell delegierbar, was in **Abb. 4a** durch einen Pfeil am Aufgabensymbol mit dem Buchstaben D symbolisiert wird. s_1 ist jedoch nicht der Ersteller von dr_y und kann daher nicht an diese Rolle delegieren.

Lösungen für Erstellerkonflikte: Um diesen Konflikt zu lösen, können wir die Aufgabe an eine der Delegationsrollen des Delegierenden delegieren (siehe Lösung 1 in Online-Anhang A). Alternativ kann der Konflikt gelöst werden, indem zuerst die jeweilige Delegationsrolle entfernt wird. Dann kann der Delegierende eine neue Delegationsrolle mit demselben Namen erstellen und in weiterer Folge an diese delegieren (siehe Lösung 2 in Online-Anhang A und **Abb. 4a**).

Konflikt mit einer delegierbaren Aufgabe (engl. delegable task conflict): Ein Konflikt mit einer delegierbaren Aufgabe tritt auf, wenn ein Subjekt versucht, eine Aufgabe zu delegieren, die nicht als delegierbar definiert ist. In **Abb. 4b** kann die Aufgabe t_x z. B. nicht an die Delegationsrolle dr_y delegiert werden, weil t_x nicht delegierbar ist. Dieser Konflikt kann potenziell auch auftreten, wenn der Delegierende versucht, eine Rolle zu delegieren während zumindest eine der Aufgaben, die dieser Rolle zugewiesen sind, nicht delegierbar ist.

Lösungen für Konflikt mit einer delegierbaren Aufgabe: Dieser Konflikt kann nur gelöst werden, indem die Aufgabe(n), die delegiert werden soll(en), als delegierbar definiert wird/werden (siehe **Abb. 4b** und Lösung 3).

Konflikt mit einer delegierbaren Pflicht (engl. delegable duty conflict): Ein Konflikt mit einer delegierbaren Pflicht tritt auf, wenn ein Subjekt versucht, eine Aufgabe zu delegieren, die an eine nicht delegierbare Pflicht gebunden ist. Pflichten müssen immer durch das ausführende Subjekt der jeweiligen Aufgabe erfüllt werden. Wenn also eine Aufgabe delegiert wird, muss daher auch die jeweilige Pflicht delegierbar sein. Dieser Konflikt kann weiters auftreten, wenn ein Delegierender versucht, eine Rolle zu delegieren und eine der Aufgaben, die dieser Rolle zugewiesen sind, eine nicht delegierbare Pflicht enthält. In **Abb. 4c** kann Aufgabe t_x nicht an die Delegationsrolle dr_y delegiert werden, weil die Pflicht du_x nicht als delegierbar definiert ist.

Lösungen für Konflikt mit einer delegierbaren Pflicht: Dieser Konflikt kann gelöst werden, indem die jeweiligen Pflichten als delegierbar definiert werden (siehe Lösung 4). Alternativ können die Pflichten, die den Konflikt ausgelöst haben, entfernt werden, um den Konflikt zu lösen (siehe Lösung 5). Diese Lösung ist aber in realen Anwendungsszenarien selten möglich und wird hier nur der Vollständigkeit halber erwähnt.

3.2 Identifikation und Lösung von Zuweisungskonflikten

Konflikt bei der Zuweisung der zu delegierenden Aufgabe (town) (engl. delegator task ownership conflict): Ein Konflikt bei der Zuweisung der zu delegierenden Aufgabe tritt auf, wenn ein Subjekt versucht, eine Aufgabe zu delegieren, die es nicht über die regulären Rollenmitgliedschaften besitzt. Ein Subjekt kann nur Aufgaben und Rollen delegieren, deren direkter oder transitiver Eigentümer es ist. In **Abb. 5a** versucht Subjekt s_1 Aufgabe t_x an seine Delegationsrolle dr_y zu delegieren. Das Subject s_1 ist jedoch nicht im Besitz von t_x.

Lösungen für Konflikte bei der Zuweisung der zu delegierenden Aufgabe (town): Der Konflikt kann gelöst werden, indem die Aufgabe einer der regulären Rollen im Besitz des Delegierenden zugeordnet wird (siehe Lösung 6). Alternativ kann dem Delegierenden eine reguläre Rolle zugeordnet werden, der die jeweilige Aufgabe zugeordnet ist (siehe Lösung 7).

Konflikt bei der Zuweisung der zu delegierenden Rolle (rown) (engl. delegator role ownership conflict): Ein Konflikt bei der Zuweisung der zu delegierenden Rolle tritt auf, wenn ein Subjekt versucht, eine Rolle zu delegieren, die es nicht besitzt. In **Abb. 5b** versucht Subjekt s_1 z. B. die reguläre Rolle rr_j an seine Delegationsrolle dr_y zu delegieren, indem es rr_j als Junior-Rolle von dr_y definiert. Das Subject s_1 ist jedoch nicht im Besitz von rr_j.

Lösungen für Konflikte bei der Zuweisung der zu delegierenden Rolle (rown):

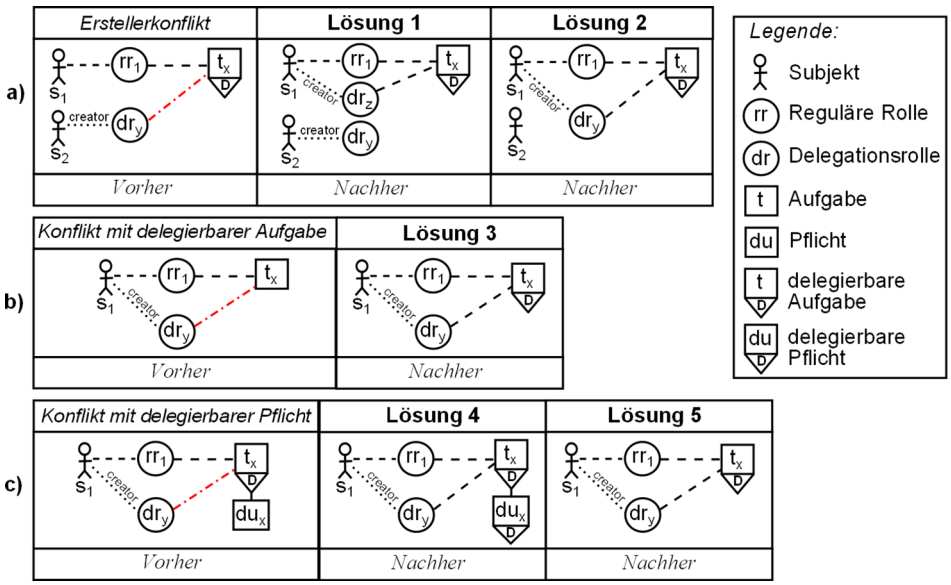


Abb. 4 Lösung von Konflikten mit Ersteller (**a**), delegierbarer Aufgabe (**b**) und delegierbarer Pflicht (**c**)

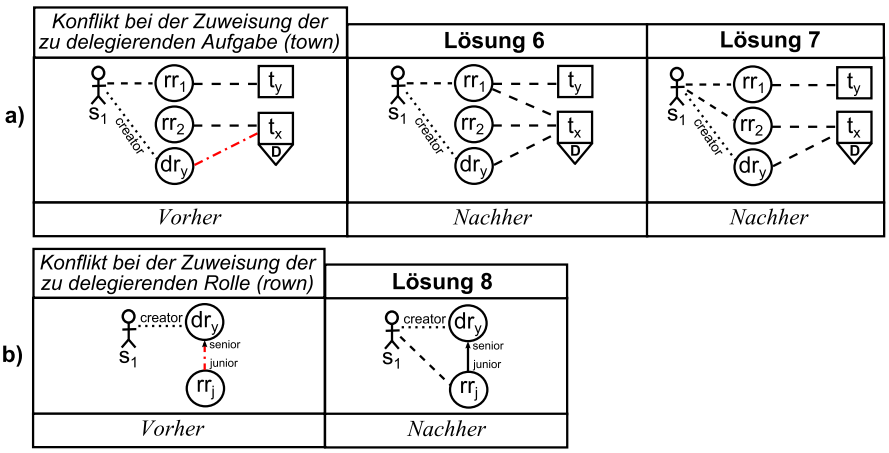


Abb. 5 Lösung von Zuweisungskonflikten: town (**a**) und rown (**b**)

Der Konflikt kann gelöst werden, indem dem Delegierenden (direkt oder transitiv) die Rolle zugewiesen wird, die er zu delegieren versucht (siehe Lösung 8).

3.3 Identifikation und Lösung von SME-Konflikten

SME-Konflikt bei der Aufgabenzuweisung (engl. task-assignment SME conflict): Ein SME-Konflikt bei der Aufgabenzuweisung kann auftreten, wenn die Delegation einer Aufgabe an eine Rolle oder einer Rolle an eine Rolle zur Zuweisung von zwei SME-Aufgaben zu ein und derselben Rolle führen würde. **Abbildung 6a** zeigt ein Beispiel, in dem eine Delegationsrolle dr_y die Aufgabe t_y besitzt, die als statisch ausschliessend zu einer anderen Aufgabe t_x definiert ist.

Lösungen für SME-Konflikte bei der Aufgabenzuweisung: Zur Auflösung des Konflikts in **Abb. 6a** kann die SME-Constraint zwischen den beiden Aufgabentypen entfernt oder in einer DME-Constraint geändert werden (siehe Lösungen 9 und 10). Alternativ kann Aufgabe t_y aus dr_y entfernt werden oder die Aufgabe t_y, die den Konflikt verursacht, gelöscht werden (siehe Lösungen 11 und 12). Beachten Sie, dass einige dieser Lösungen, wie etwa das Entfernen einer Constraint oder einer Aufgabe, in realen Anwendungsszenarien selten möglich sind und daher vor allem der Vollständigkeit halber angeführt werden.

SME-Konflikt bei der Rollenzuweisung (engl. role-assignment SME conflict): Ein SME-Konflikt bei der Rollenzuweisung tritt auf, wenn die geplante Delegation einer Aufgabe an eine Rolle oder einer Rolle an eine Rolle dazu führen würde, dass zwei sich statisch ausschliessende Aufgaben demselben Subjekt zugeordnet wären. Als Folge davon wäre der Delegationsempfänger berechtigt, zwei SME-Aufgaben auszuführen. **Abbildung 6b** zeigt ein Beispiel, in dem die Delegation von Aufgabe t_x an die Delegationsrolle dr_y zu einem Konflikt bei der Rollenzuweisung führen würde, weil der Delegationsempfänger s_1 dann das Recht hätte, die beiden SME-Aufgaben t_z und t_x auszuführen. In ähnlicher Weise ist bei der Delegation einer Rolle an eine Delegationsrolle oder bei der Zuordnung eines Delegationsempfängers zu einer Delegationsrolle auf Rollenzuweisungskonflikte zu achten.

Lösungen für SME-Konflikte bei der Rollenzuweisung: Zur Vermeidung dieses Konflikts können dieselben Lösungen wie für Konflikte bei der Aufgabenzuweisung angewandt werden (siehe Lösungen 9–12). Weiterhin kann Lösung 13 angewandt werden, indem die Zuweisung, die den Konflikt verursacht, zwischen der regulären Rolle rr_z und dem Subjekt s_1 (siehe **Abb. 6b**) entfernt wird. Zudem kann der Konflikt (theoretisch) durch Entfernen des Subjekts s_1, gelöst werden (siehe Lösung 14).

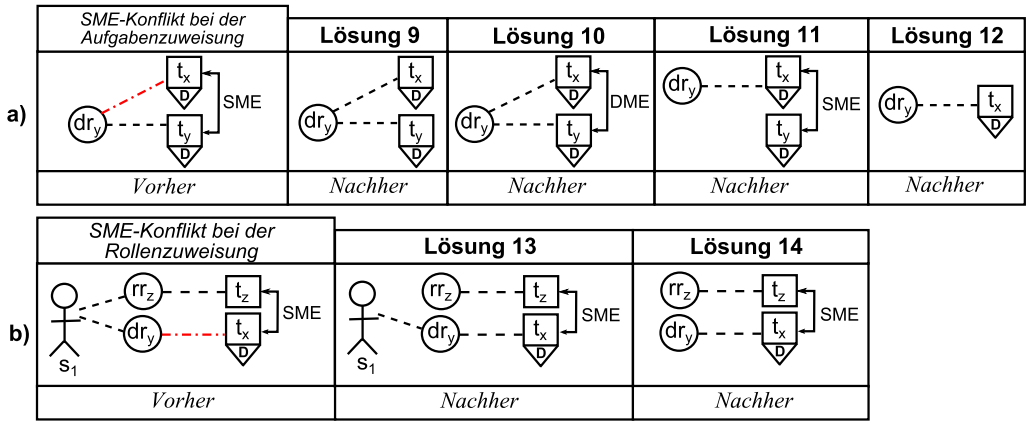


Abb. 6 Lösung der SME-Konflikte mit Aufgaben- (**a**) und Rollenzuordnung (**b**)

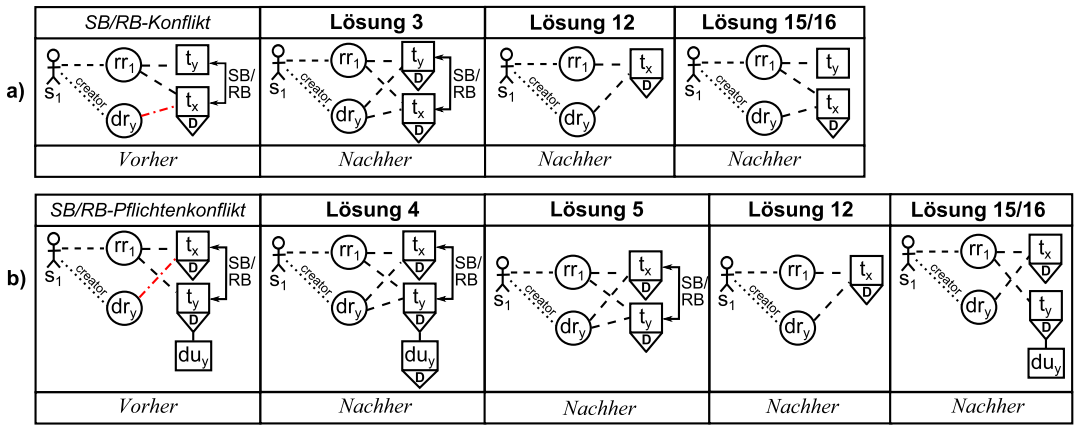


Abb. 7 Lösung von SB/RB- (**a**) und SB/RB-Pflicht- (**b**) Delegationskonflikten

3.4 Identifikation und Lösung von Binding-Konflikten

SB-Delegationskonflikt (engl. SB delegation conflict): Ein SB-Delegationskonflikt tritt auf, wenn ein Subjekt versucht, eine Aufgabe zu delegieren, die per Subject-Binding an eine oder mehrere nicht-delegierbare Aufgabe(n) gebunden ist. Derart gebundene Aufgaben müssen jedoch immer durch dasselbe Subjekt ausgeführt werden. Wenn also eine Aufgabe delegiert wird, müssen auch alle subjektgebundenen Aufgaben derselben Delegationsrolle zugewiesen werden. Andernfalls kann die SB-Constraint nicht erfüllt werden. In **Abb. 7a** wird eine SB-Constraint für t_x und t_y definiert. Daher muss das Subjekt, das t_x ausführt, auch t_y ausführen. Wird t_x an dr_y delegiert, kommt es zu einem SB-Konflikt, da t_y nicht als delegierbar definiert ist. Um dennoch die SB-Constraint zu erfüllen, müssen beide Aufgaben an dr_y delegiert werden.

Lösungen für SB-Delegationskonflikte: Dieser Konflikt kann gelöst werden, indem alle subjektgebundenen Aufgaben als delegierbar definiert werden (siehe Lösung 3). Alternativ kann die Aufgabe oder die SB-Constraint, die den Konflikt auslöst, entfernt werden (siehe Lösungen 12 und 15). In **Abb. 7a** kann t_x delegiert werden, wenn t_y als delegierbar definiert wird. Die Delegation ist auch möglich, wenn t_y oder die SB-Constraint zwischen t_x und t_y entfernt wird.

RB-Delegationskonflikt (engl. RB delegation conflict): Ein RB-Delegationskonflikt liegt vor, wenn ein Subjekt versucht, eine Aufgabe zu delegieren, die per Role-Binding an eine oder mehrere nicht delegierbare Aufgabe(n) gebunden wurde. Rollengebundene Aufgaben müssen aber immer von Mitgliedern derselben Rolle ausgeführt werden. Wenn also eine Aufgabe delegiert wird, müssen alle rollengebundenen Aufgaben derselben Delegationsrolle zugeordnet werden. Andernfalls kann die RB-Constraint nicht erfüllt werden. In **Abb. 7a** wird eine RB-Constraint für t_x and t_y definiert. Daher müssen t_x und t_y immer derselben Rolle zugeordnet werden. Bei der Delegation von t_x an dr_y kommt es zu einem RB-Konflikt, da t_y nicht als delegierbar definiert wurde.

Lösungen für RB-Delegationskonflikte: Dieser Konflikt kann gelöst werden, indem alle rollengebundenen Aufgaben als delegierbar definiert werden (siehe Lösung 3). Alternativ kann die Aufgabe oder die RB-Constraint, die den Konflikt auslöst, entfernt werden (siehe Lösungen 12 und 16). In **Abb. 7a** kann t_x delegiert werden, wenn t_y als delegierbar definiert wird. Danach können beide Aufgaben an dr_y delegiert werden. Die Delegation ist auch möglich, wenn t_y oder die RB-Constraint für t_x und t_y entfernt wird.

SB-Pflichten-Delegationskonflikt (engl. SB duty delegation conflict): Sobald ein Subjekt versucht, eine Aufgabe zu delegieren, die per Subject-Binding an andere Aufgaben gebunden ist, kommt es zu einem SB-Pflichten-Delegationskonflikt, wenn eine der subjektgebundenen Aufgaben mit einer

nicht delegierbaren Pflicht verbunden ist. In diesem Fall kann die entsprechende subjektgebundene Aufgabe nicht delegiert werden. Um die SB-Constraint zu erfüllen, müssen aber alle subjektgebundenen Aufgaben delegiert werden. In **Abb. 7b** ist eine SB-Constraint für t_x und t_y definiert. Weiterhin ist t_y mit einer Pflicht du_y verbunden. Wenn Subjekt s_1 versucht, t_x an dr_y zu delegieren, muss es auch alle subjektgebundenen Aufgaben und damit zusammenhängenden Pflichten delegieren. In diesem Beispiel ist du_y nicht delegierbar. Daher kommt es zu einem SB-Pflichten-Delegationskonflikt.

Lösungen für SB-Pflichten-Delegationskonflikte: Der Konflikt kann gelöst werden, indem die jeweilige Pflicht du_y als delegierbar definiert wird (siehe Lösung 4). Alternativ kann die Pflicht, die den Konflikt auslöst, gelöscht werden, die subjektgebundene Aufgabe t_y die mit du_y verbunden ist, kann gelöscht werden oder die SB-Constraint kann entfernt werden (siehe Lösungen 5, 12 und 15).

RB-Pflichten-Delegationskonflikt (engl. RB duty delegation conflict): Versucht ein Subjekt, eine Aufgabe zu delegieren, die per Role-Binding an andere Aufgaben gebunden ist, kommt es zu einem RB-Pflichten-Delegationskonflikt, wenn eine der rollengebundenen Aufgaben mit einer nicht-delegierbaren Pflicht verbunden ist. In diesem Fall kann die jeweilige rollengebundene Aufgabe nicht delegiert werden. Um die RB-Constraint zu erfüllen, müssen alle rollengebundenen Aufgaben an dieselbe Delegationsrolle delegiert werden. Wenn Subjekt s_1 in **Abb. 7b** versucht, t_x an dr_y zu delegieren, müssen auch alle rollengebundenen Aufgaben und damit zusammenhängende Pflichten an dr_y delegiert werden. In diesem Beispiel ist du_y nicht delegierbar. Daher kommt es zu einem RB-Pflichten-Delegationskonflikt.

Lösungen für RB-Pflichten-Delegationskonflikte: Der Konflikt kann gelöst werden, indem die jeweilige Pflicht du_y als delegierbar definiert wird (siehe Lösung 4). Alternativ kann die Pflicht, die den Konflikt auslöst, gelöscht werden, die mit du_y verbundene rollengebundenen Aufgabe t_y kann gelöscht werden oder die RB-Constraint kann entfernt werden (siehe Lösungen 5, 12 und 16).

3.5 Identifikation und Lösung von Vererbungskonflikten

Selbstdelegationskonflikt (engl. self-delegation conflict): Ein Selbstdelegationskonflikt tritt auf, wenn an Rolle an sich selbst delegiert werden soll. Generell kann eine Rolle nicht ihre eigene Junior-Rolle sein (siehe **Abb. 8a** und Strembeck und Mendling 2010, 2011).

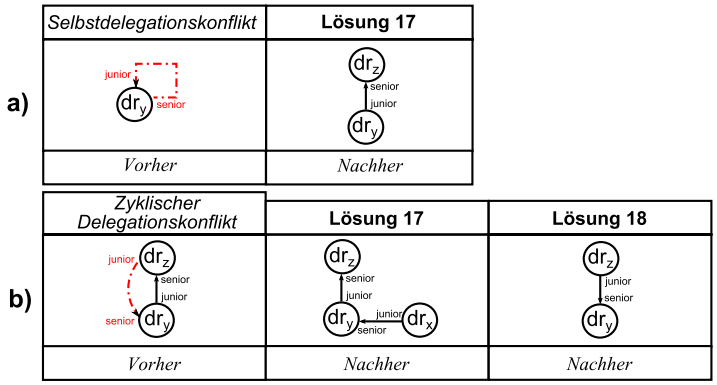


Abb. 8 Lösung von Selbstdelegations- (**a**) und zyklischen Delegationskonflikten (**b**)

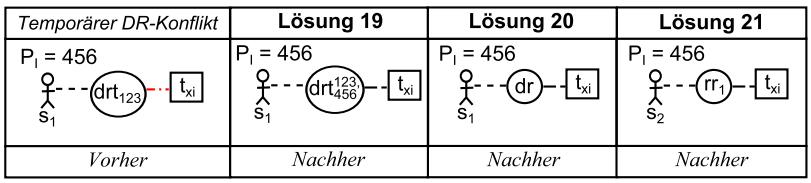


Abb. 9 Lösung von Laufzeitkonflikten

Lösung für Selbstdelegationskonflikte: Dieser Konflikt kann gelöst werden, indem eine andere Junior- oder Senior-Rolle gewählt wird, sodass eine Vererbungsrelation zwischen zwei *verschiedenen* Rollen definiert wird (siehe **Abb. 8a** und Lösung 17).

Zyklischer Delegationskonflikt (engl. cyclic delegation conflict): Ein zyklischer Delegationskonflikt tritt auf wenn durch die Delegation ein Zyklus in der Rollenhierarchie entstehen würde (siehe **Abb. 8b** und Strembeck und Mendling 2010, 2011).

Lösungen für zyklische Delegationskonflikte: Dieser Konflikt kann gelöst werden, indem eine andere Rolle delegiert wird, die noch nicht Teil derselben Rollenhierarchie ist (siehe Lösung 17). In **Abb. 8b** wird Lösung 17 angewandt, indem eine neue Vererbungsbeziehung zwischen dr_x und dr_y definiert wird, wobei jedoch die bestehende Vererbungsbeziehung zwischen dr_y und dr_z erhalten bleibt. Außerdem kann die bestehende Vererbungsbeziehung zwischen dr_y und dr_z entfernt werden, bevor die inverse Vererbungsbeziehung mit dr_z als Junior-Rolle von dr_y definiert wird (siehe Lösung 18).

3.6 Identifikation und Lösung von Laufzeitkonflikten

Neben den Laufzeitkonflikten gemäß Schefer et al. (2011) und Strembeck und Mendling (2010) kann noch ein weiterer Konflikt im Zusammenhang mit temporären Delegationsrollen auftreten.

Konflikt mit temporären Delegationsrollen (engl. temporary delegation role conflict): Ein Konflikt mit temporären Delegationsrollen tritt auf, wenn das verantwortliche Subjekt nicht berechtigt ist, eine bestimmte Aufgabeninstanz auszuführen, weil die temporäre Delegationsrolle nicht für die jeweilige Prozessinstanz gilt. Jede temporäre Delegationsrolle gilt nur für bestimmte Prozessinstanzen. Daher ist der Delegationsempfänger einer temporären Delegationsrolle berechtigt, alle delegierten Aufgaben nur innerhalb dieser Prozessinstanzen auszuführen. In **Abb. 9** wird Subjekt s_1 der temporären Delegationsrolle *drt* zugewiesen, wobei *drt* nur für die Prozessinstanz 123 gilt. Die aktuelle Prozessinstanz ist jedoch 456. Daher darf s_1 die delegierten Aufgaben in dieser Prozessinstanz nicht ausführen.

Lösung des Konflikts mit temporären Delegationsrollen: Der Konflikt kann gelöst werden, indem die betreffende Prozessinstanz zu den Prozessinstanzen hinzufügt wird, für welche die temporäre Delegationsrolle gilt (siehe Lösung 19).

Algorithm 1 prüft, ob die Delegation eines bestimmten Aufgabentyps an eine bestimmte Delegationsrolle zulässig ist

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

Algorithm 2 prüft, ob die Delegation einer bestimmten Rolle an eine Delegationsrolle zulässig ist

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

Eine andere Lösung ist die Änderung der temporären Delegationsrolle zu einer permanenten Delegationsrolle (siehe Lösung 20). Danach sind die Delegationsempfänger berechtigt, alle Instanzen der delegierten Aufgaben durchzuführen. Alternativ kann *drt* in eine permanente Delegationsrolle geändert werden. Weiterhin kann man ein anderes ausführendes Subjekt zuordnen, das die Berechtigung zur Ausführung der jeweiligen Aufgabe besitzt (siehe Lösung 21).

4 Algorithmen zur Identifikation von Delegationskonflikten

In diesem Abschnitt beschreiben wir Algorithmen zur Identifikation der in Abschn. 3 vorgestellten Delegationskonflikte. Zur Unterstützung einer systematischen Konfliktbehandlung schlagen wir die folgenden drei Schritte vor: Zunächst muss ein Delegationskonflikt erkannt werden. Nach der Identifikation des Konflikts wird entschieden, wie dieser Konflikt zu lösen ist, d. h., welche Lösungsstrategie angewandt werden soll. Nach Anwendung der Lösungsstrategie kann die gewünschte Delegation durchgeführt werden.

Die Algorithmen 1–3 prüfen die Konsistenz eines prozessbezogenen RBAC-Delegationsmodells zum Entwurfszeitpunkt, bevor eine neue Delegationsbeziehung erstellt wird. Algorithmus 4 prüft die Laufzeitkonsistenz eines prozessbezogenen RBAC-Delegationsmodells. Insbesondere prüft es, ob eine temporäre De-

Algorithm 3 prüft, ob eine bestimmte Delegationsrolle einem bestimmten Delegationsempfänger zugewiesen werden darf

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

Algorithm 4 prüft, ob eine konkrete Aufgabeninstanz, die in einer bestimmten Prozessinstanz ausgeführt wird, einem bestimmten Delegationsempfänger zugeordnet werden darf

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

legationsrolle das jeweilige Subjekt berechtigt, eine delegierte Aufgabe in einer bestimmten Prozessinstanz durchzuführen. Diese Algorithmen ergänzen die in Schefer et al. (2011) und Strembeck und Mendling (2010) vorgestellten Algorithmen, die potenzielle Konflikte prozessbezogener RBAC-Modelle erkennen, die nicht mit Delegation zusammenhängen.

5 Eine UML-Erweiterung zur Modellierung prozessbezogener Delegationsmodelle

Die Unified Modeling Language (UML) OMG (2011b) ist heute der De-facto-Standard für die Modellierung von Softwaresystemen. Ein Vorteil der UML ist die Möglichkeit, alle Artefakte eines Softwaresystems mithilfe derselben Sprache (bzw. Sprachfamilie) abbilden zu können. Da die UML von Haus aus keine nativen Sprachelemente zur Abbildung von Sicherheitseigenschaften zur Verfügung stellt, wurden in der jüngeren Vergangenheit bereits mehrere Ansätze für entsprechende UML-Erweiterungen vorgestellt (siehe z. B. Basin et al. 2006; Jürjens 2005; Rodriguez et al. 2006; Rodriguez und de Guzman 2007). Im Zusammenhang mit der Modellierung von prozessbasierten Softwaresystemen, kann Modellierungsunterstützung für die Delegation von Rollen, Aufgaben und Pflichten helfen, die Kommunikation zwischen Softwareentwicklern, Sicherheitsexperten, Experten der jeweiligen Anwendungsdomäne (häufig kurz: Domänenexperten) und anderen Interessensgruppen zu verbessern (siehe z. B. Mouratidis und Jürjens 2010).

Das UML-Metamodell basiert auf der OMG Meta Object Facility (MOF; OMG 2011a) und definiert die abstrakte Syntax aller UML-Diagrammtypen. Der UML-Standard stellt im Wesentlichen zwei Optionen zur Verfügung, um das zugehörige Metamodell an einen bestimmten Anwendungsbereich anzupassen (OMG 2011b): (a) Definition von UML-Profilen: Ein UML-Profil darf das UML-Metamodell jedoch nicht verändern und kann daher nur verwendet werden, um bestehende UML-Metaklassen in limitiertem Maß für spezielle Anwendungsdomänen zu erweitern. UML-Profile stellen somit lediglich einen eingeschränkten Erweiterungsmechanismus zur Verfügung (siehe OMG 2011b, S. 660); (b) Erweiterung des UML-Metamodells: auf diese Weise können dem UML-Metamodell neue Metaklassen mit neuer Semantik sowie Beziehungen zwischen diesen Metaklassen hinzugefügt werden. Da die von uns neu einzuführenden Modellelemente für prozessbezogene Delegation die Spezifikation neuer Metaklassen mit neuer und im originalen UML-Metamodell nicht verfügbaren Semantik erfordern, haben wir für unsere UML-Erweiterung die zweite Option (Erweiterung des UML-Metamodells) gewählt.

Dieser Abschnitt stellt die UML-Erweiterung mit dem Namen *BusinessActivityDelegations* vor, die die Modellierung von Delegationsrichtlinien für Rollen, Aufgaben und Pflichten auf Grundlage der formalen Definitionen aus Abschn. 2.2 ermöglicht. Zu diesem Zweck erweitern wir das *BusinessActivities*-Package (Strembeck und Mendling 2011), das eine UML-Erweiterung für prozessbezogene RBAC-Modelle zur Verfügung stellt. Darüber hinaus wurden alle Teile des *BusinessActivityDelegations*-Package als Erweiterung der „BusinessActivity-Library and Runtime-Engine“ implementiert (siehe Abschn. 7).

5.1 UML-Metamodell-Erweiterung

Basierend auf dem formalen Metamodell für prozessbezogene RBAC-Delegationsmodelle, das in Abschn. 2 beschrieben wurde, zeigt **Abb. 10** eine entsprechende UML-Erweiterung auf der PIM-Ebene. In dieser UML-Erweiterung ist eine *BusinessActivity* eine spezielle UML-Aktivität (siehe **Abb. 10**). Diese kann neben den neu eingeführten Modellelementen (siehe unten) auch alle für Standard-UML-Aktivitäten verfügbaren Modellelemente beinhalten. Eine *BusinessAction* entspricht einer Aufgabe (engl. task) in einem Geschäftsprozess und umfasst alle Berechtigungen zur Ausführung der jeweiligen Aufgabe (siehe Strembeck und Mendling 2011 für weitere Details zu BusinessActivities und BusinessActions). Die neue Metaklasse *Duty* ist ein spezieller UML-Classifier (siehe **Abb. 10**) und wird verwendet um zu definieren, dass ein Subjekt die Pflicht hat eine bestimmte Aktion auszuführen (siehe Schefer und Strembeck 2011a). Die Verbindung zwischen den Metaklassen Duty und BusinessAction stellt sicher, dass ein Subjekt, dem eine Pflicht zugeordnet wird, auch alle Berechtigungen zur Durchführung dieser Pflichten erhält. *Role* und *Subject* sind spezialisierte UML-Classifier (Strembeck

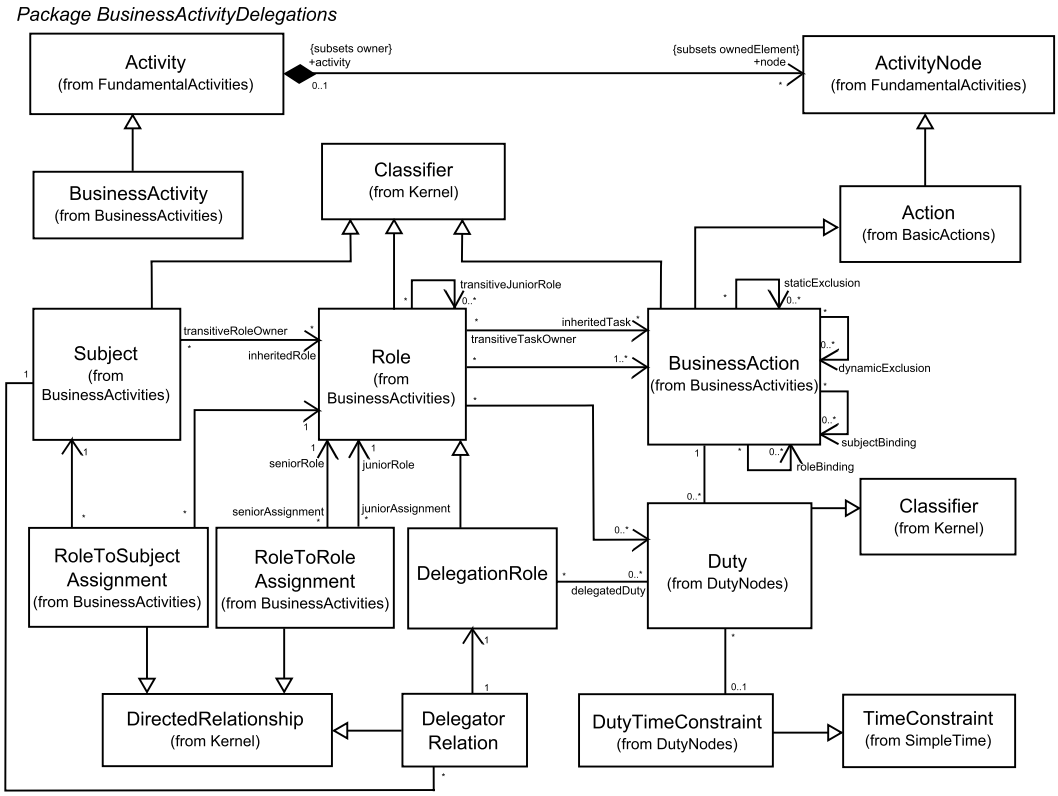


Abb. 10 UML-Metamodell-Erweiterung *BusinessActivityDelegations*

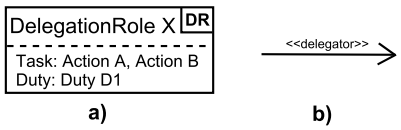


Abb. 11 Visualisierung der (**a**) Delegationsrollen und (**b**) der Delegierendenbeziehung

und Mendling 2011), die mit BusinessActions und Duties verknüpft werden (siehe **Abb. 10**). Weiterhin kann eine Pflicht mit eine *DutyTimeConstraint* verknüpft werden. Wenn ein DutyTimeConstraint abgelaufen ist, ohne dass die entsprechende Pflicht erfüllt wurde, kann eine sogenannte *Kompensationsaktion* (engl. compensation action) ausgeführt werden (siehe auch Schefer und Strembeck 2011a).

Die Metaklasse *DelegationRole* definiert einen spezialisierten Rollentyp. Jeder DelegationRole können andere Rollen, BusinessActions, und/oder Duties zugeordnet werden (siehe **Abb. 10**). Eine *DelegatorRelation* ist eine UML-DirectedRelationship und verbindet eine DelegationRole mit einem Subjekt, das als der Besitzer (der Delegierende, engl. delegator) dieser DelegationRole auftritt. Nur der jeweilige Besitzer einer DelegationRole darf Rollen, BusinessActions oder Duties an seine eigenen DelegationRoles delegieren (siehe OCL-Invariante 1 in Abschn. 5.2). **Abbildung 11** zeigt Darstellungsoptionen zur Visualisierung der Delegationsrollen und Delegierendenbeziehung. Es ist hierbei zu beachten, dass diese Beziehungen formal durch die zugehörige UML-Erweiterung definiert werden und daher unabhängig von ihrer grafischen Darstellung existieren. Auf diese Weise kann das grafische Symbol, das Instanzen einer Metaklasse repräsentiert, bei Bedarf leicht gegen ein anderes Symbol ausgetauscht werden.

Jede DelegationRole kann einem (oder mehreren) Delegationsempfänger (engl. delegatee) zugewiesen werden. Der Delegationsempfänger wird dadurch berechtigt, die zugehörigen Aufgaben und Pflichten durchzuführen (siehe OCL-Invariante 2). Ein Delegierender kann eine reguläre Rolle delegieren, indem diese reguläre Rolle als Junior-Rolle für eine seiner DelegationRoles definiert wird. Alle BusinessActions und Duties, die dieser regulären Rolle zugewiesen sind, müssen delegierbar sein (siehe auch Abschn. 2.2). Hierbei ist insbesondere zu beachten, dass reguläre Rollen zwar Junior-Rollen einer DelegationRole sein können, es aber nicht möglich ist, eine reguläre Rolle als Senior-Rolle einer DelegationRole zu definieren, um ungültige Vererbung von Berechtigungen zu vermeiden (siehe OCL-Invariante 3). Zur Delegation einer BusinessAction weist der Delegierende die BusinessAction der jeweiligen DelegationRole zu. Nur wenn eine BusinessAction delegierbar ist, kann sie an eine DelegationRole delegiert werden (siehe OCL-Invarianten 4 und 6). Um die Delegation von Pflichten in UML-Modellen umzusetzen muss die zu delegierende Pflicht ebenfalls als delegierbar definiert werden (siehe OCL-Invarianten 5, 7 und 9). Nach der Zuweisung eines Delegationsempfängers verliert der Delegierende seine Verpflichtung zur Erfüllung der delegierten Pflicht. Optional ist es aber möglich eine Pflicht zur Überprüfung zu definieren (engl. review duty; Schaad und Moffett 2002), sodass der Delegierende die korrekte Umsetzung der delegierten Pflichten durch den Delegationsempfänger überprüfen muss (siehe OCL-Invarianten 8 und 9).

Um die Möglichkeit zur zeitlichen Beschränkung einer Delegation zu berücksichtigen (siehe z. B. Barka und Sandhu 2000b), können DelegationRoles

Tab. 2 Abbildung des generischen Metamodells auf die UML-Erweiterung

Generische Definition	Abgedeckt durch
Definition 4.1: $rrh : RR \mapsto \mathcal{P}(RR)$	Metamodell-Erweiterung: Beziehung RoleToRoleAssignment (siehe **Abb. 10** und Strembeck und Mendling 2011)
Definition 4.2: $drh : DR \mapsto \mathcal{P}(R)$	Metamodell-Erweiterung: Beziehung RoleToRoleAssignment (siehe **Abb. 10**) und OCL-Invariante 3
Definition 4.3: $creator(dr) : DR \mapsto S$	Metamodell-Erweiterung: Metaklasse DelegatorRelation (siehe **Abb. 10**) und OCL-Invariante 1
Definition 4.4: $drpi : DRT \mapsto \mathcal{P}(P_I)$	OCL-Invarianten 10 und 11
Definition 4.5: $trra : RR \mapsto \mathcal{P}(T_T)$	Metamodell-Erweiterung: Beziehung zwischen Role und BusinessAction (siehe **Abb. 10** und Strembeck und Mendling 2011)
Definition 4.6: $trdel : DR \mapsto \mathcal{P}(DT_T)$	Metamodell-Erweiterung: Beziehung zwischen Role und BusinessAction (siehe **Abb. 10**) und OCL-Invarianten 4, 6 und 15
Definition 4.7: $dta : T_T \mapsto \mathcal{P}(DU_T)$	Metamodell-Erweiterung: Beziehung zwischen Duty und BusinessAction (siehe **Abb. 10**)
Definition 4.8: $\forall t_x \in trdel(dr) : \forall du \in dta(t_x) : du \in DDU_T$	OCL-Invarianten 5, 7, 8, 9 und 16
Definition 4.9: $rrsa : S \mapsto \mathcal{P}(RR)$	Metamodell-Erweiterung: Beziehung RoleToSubjectAssignment (siehe **Abb. 10** und Strembeck und Mendling 2011)
Definition 4.10: $drsa : S \mapsto \mathcal{P}(DR)$	Metamodell-Erweiterung: Beziehung RoleToSubjectAssignment (siehe **Abb. 10**) und OCL-Invariante 2
Definition 4.11: $dui : (DU_T \times P_I) \mapsto \mathcal{P}(DU_I)$	Implizit durch unsere Metamodellerweiterung und die Spezifikation von UML Aktivitätsdiagrammen definiert (siehe **Abb. 10** und OMG 2011b; Schefer und Strembeck 2011a)
Definition 4.12: $res : DU_I \mapsto S$	OCL-Invariante 12
Definition 4.13: $\forall du \in dta(t_1), p_i \in P_I : \forall t_x \in ti(t_1, p_i), du_x \in dui(du, p_i) : es(t_x) = res(du_x)$	OCL-Invariante 13
Definition 4.14: $rer : DU_I \mapsto R$	OCL-Invariante 14

entweder für temporäre oder für permanente Delegation definiert werden (siehe OCL-Invarianten 10 and 11). Weiterhin unterstützen wir ein- und mehrstufige Delegation für BusinessActions und Pflichten. Bei einer einstufigen Delegation kann eine delegierte BusinessAction oder Pflicht nicht durch den Delegationsempfänger weiter delegiert werden (siehe Barka und Sandhu 2000b). Die entsprechende Kennzeichnung erfolgt durch die Attribute *delegable* und *isDelegated* für jede BusinessAction und für jede Pflicht. Der Wert von isDelegated wird auf *true* gesetzt, sobald die jeweilige BusinessAction oder Pflicht delegiert wurde. Bei der einstufigen Delegation wird das Attribut delegable auf *false* gesetzt, wenn das isDelegated-Attribut einer BusinessAction oder einer Pflicht auf *true* gesetzt ist (siehe OCL-Invarianten 6 und 7). Ein Vorteil der einstufigen Delegation ist, dass ein Delegierender, der möglicherweise verpflichtet ist, die Durchsetzung von delegierten Aufgaben oder Pflichten zu überwachen, die Kontrolle darüber behält, wer für die Durchführung der delegierten BusinessAction zuständig ist bzw. wer die delegierte Pflicht ausführt. Bei Bedarf kann jedoch jederzeit auch die mehrstufige Delegation verwendet werden, indem stattdessen OCL-Invarianten 15 und 16 verwendet werden.

5.2 OCL-Invarianten

Ein grafisches UML-Modell kann häufig nicht alle Zusammenhänge oder Arten von Einschränkungen erfassen, die für die Beschreibung einer Anwendungsdomäne von Belang sind. Aus diesem Grund können Zusammenhänge, die sich in grafischen UML-Modellen nicht oder nur mühsam ausgedrücken lassen, mithilfe der Object Constraint Language (OCL; OMG 2014) definiert werden. Zur besseren Lesbarkeit des Beitrags werden in diesem Abschnitt exemplarisch lediglich drei OCL-Invarianten für unsere UML-Erweiterung gezeigt. Die vollständige Liste der OCL-Invarianten für die BusinessActivityDelegations Erweiterung sind in Online-Anhang B zu finden. **Tabelle 2** liefert darüber hinaus einen Überblick, wie die einzelnen Definitionen aus Abschn. 2.2 in der BusinessActivityDelegations Erweiterung abgebildet werden.

5.3 Beispiel für BusinessActivityDelegation-Modelle

In **Abb. 12** wird der Kreditantragsprozess aus **Abb. 1** durch Verwendung der neu eingeführten Modellelemente erweitert. Der Prozess in **Abb. 12a** umfasst fünf Aufgaben, von denen drei als BusinessActions definiert sind. Die BusinessActions sind mit Pflichten verbunden: Die BusinessAction *Kreditwürdigkeit prüfen* ist mit der Pflicht *Antragstellerrating prüfen* verbunden, die BusinessAction *Vertrag aushandeln* ist mit der Pflicht *Vorvertragliche Pflichten erfüllen* verbunden und die BusinessAction *Vertrag genehmigen* ist mit der Pflicht *Fertigen Vertrag prüfen* verbunden. Weiterhin wird die Kompensationsaktion *Pflicht neu zuweisen* ausgelöst, wenn die Pflicht *Antragstellerrating prüfen* nicht rechtzeitig erfüllt wird.

Abbildung 12b zeigt die Pflicht *Antragstellerrating prüfen*, die mit der BusinessAction *Kreditwürdigkeit prüfen* verbunden ist. Sie ist mit einem DutyTime-Constraint und einer Kompensationsak-

OCL-Invariante 1 Der Delegierende (engl. delegator) einer Duty, einer BusinessAction oder einer Rolle muss das zu delegierende Modellelement besitzen.

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

OCL-Invariante 6 Jede BusinessAction definiert ein Attribut „isDelegated“, das angibt, ob eine bestimmte BusinessAction bereits delegiert wurde oder nicht. Wurde sie bereits delegiert, kann sie nicht weiter delegiert werden (einstufige Delegation, siehe Barka und Sandhu 2000b).

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

OCL-Invariante 11 Ist eine DelegationRole nur für die temporäre Delegation vorgesehen (isTemporary=true), so ist der Wert des Attributs „relatedProcessInstance“ gleich der ID der zugehörigen Prozessinstanz.

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

tion verknüpft. Die DutyTimeConstraint drückt aus, dass die Pflicht *Antragsteller-rating prüfen* innerhalb von drei Zeiteinheiten (z. B. Tagen) nach Beginn der jeweiligen BusinessAction ausgeführt werden muss. Andernfalls wird die Kompensationsaktion *Pflicht neu zuweisen* ausgeführt.

Die Zuständigkeit für die Pflichten wird in **Abb. 12c** dargestellt und zeigt die Rolle des Bankangestellten, die das Recht zur Ausführung der drei BusinessActions besitzt und die zugehörigen Pflichten, die im Kreditantragsprozess definiert sind, erfüllen muss. Auf diese Weise sind Subjekte, denen diese Rolle zugewiesen wird, für die Erfüllung der enstprechenden Pflichten und die Ausführung der damit zusammenhängenden BusinessActions zuständig. In unserem Beispiel wird das Subjekt M. Meyer der Rolle Bankangestellter zugewiesen und muss daher auch die damit zusammenhängenden Pflichten erfüllen. Weiterhin tritt M. Meyer als Delegierender für die permanente Delegationsrolle Sommerpraktikant auf. Durch Zuweisung der Delegationsrolle an das Subjekt J. Smith wird dieses Subjekt zum Delegationsempfänger (siehe **Abb. 12c**)

6 Fallstudie für die Modellierung eines prozessbezogenen RBAC-Delegationsmodells

Zur Evaluierung der praktischen Anwendbarkeit unseres Ansatzes haben wir eine Fallstudie durchgeführt, in der die BusinessActivityDelegations Erweiterung auf reale Prozesse angewandt wurde. Diese Fallstudie basiert auf einer Sammlung realer Prozessmodelle eines großen österreichischen Schulzentrums. Die Sammlung besteht aus 30 Prozessen, die durch Mitglieder der Schule während einer Prozessmanagementini-

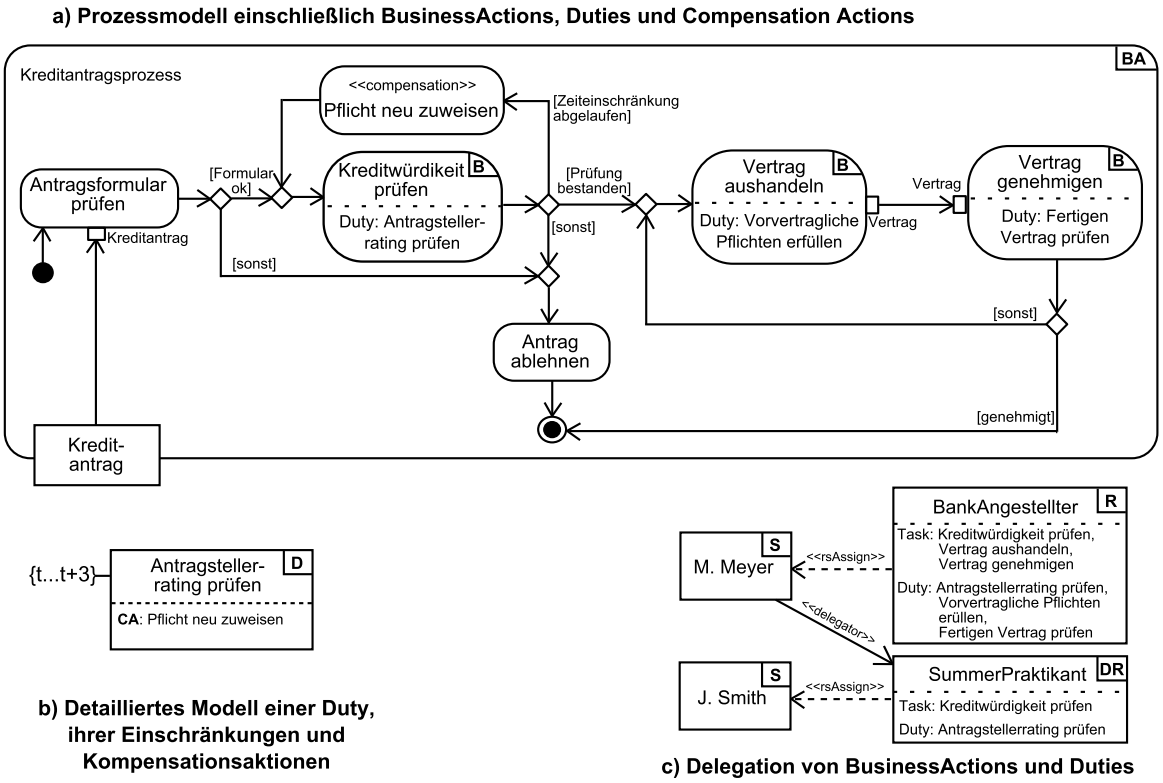


Abb. 12 Erweiterter Kreditantragsprozess

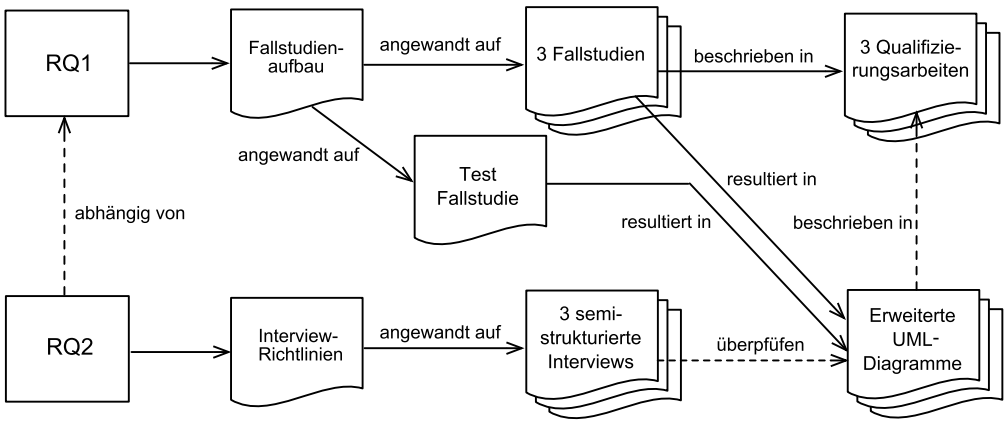


Abb. 13 Multimethoden-Forschungsdesign

tiative gesammelt wurden. Der Kontrollfluss einiger dieser Prozesse wurde grafisch visualisiert und zeigte den Ablauf der Aufgaben und die jeweils autorisierten/zuständigen Personen an. Die Visualisierung im Rahmen der Prozessmanagementinitiative erfolgte jedoch mit einer nicht-standardisierten Ad-hoc-Notation. Weiterhin lag für die meisten der Prozesse nur eine Dokumentation in Form von textuellen/tabellarischen Aufgabenbeschreibungen vor. Die Prozessbeschreibungen beinhalteten außerdem Referenzen zu rechtlichen Anforderungen (z. B. bestimmte Paragrafen des österreichischen Unterrichtsgesetzes) sowie andere interne oder externe Bestimmungen.

In der hier beschriebenen Fallstudie wurden die Prozesse, die Delegationsszenarien enthielten, mithilfe unserer UML-Erweiterung (siehe Abschn. 5) neu modelliert. Diese Fallstudie ist Teil einer größeren Multimethodenstudie, die in Schefer-Wenzl et al. (2013) detailliert beschrieben ist. Unsere Multimethodenstudie bestand aus zwei aufeinander folgenden Forschungsphasen (siehe **Abb. 13**). Die beiden zugehörigen Forschungsfragen waren: Welche Hindernisse gibt es für die Verwendung unserer UML-Erweiterung durch Modellierer mit grundlegenden UML-Modellierungskenntnissen (RQ1)? Welche Hindernisse gibt es für die Verwendung der resultierenden Prozessmodelle für Nutzer, die keine explizite softwaretechnische und/oder sicherheitstechnische Vorbildung haben (RQ2)? Bezüglich RQ1 haben wir verschiedene Fallstudien entworfen (eine davon war die Fallstudie zur Modellierung von Delegationsaspekten). Für die Beantwortung

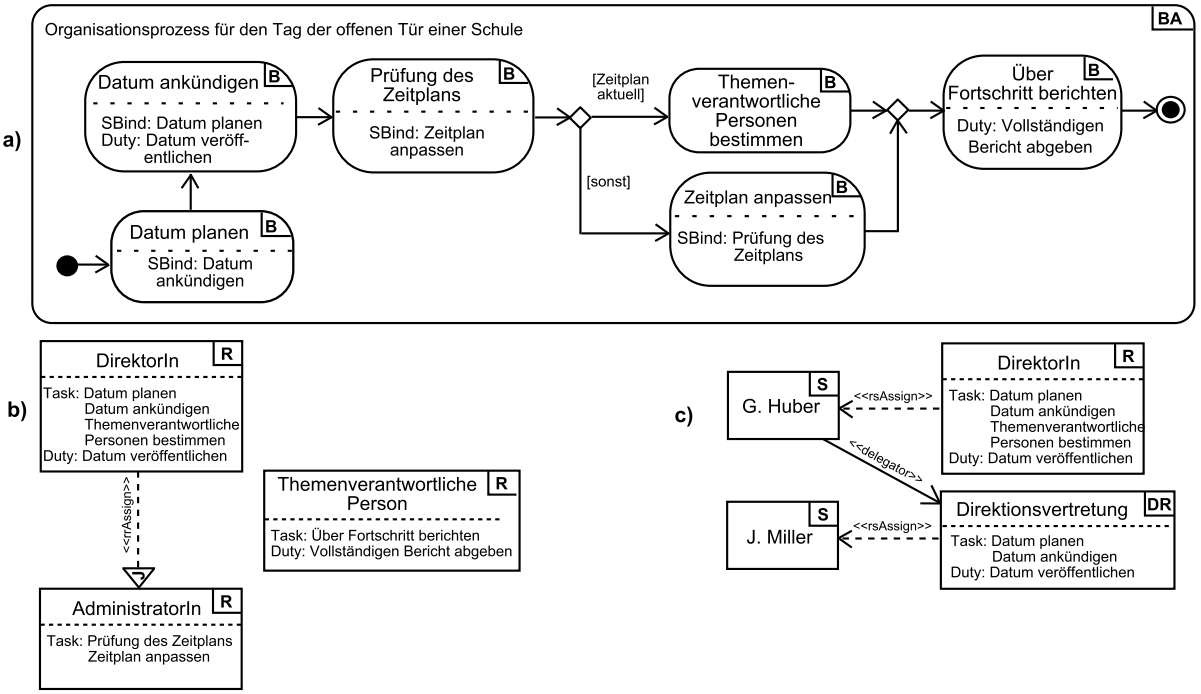


Abb. 14 Beispielprozess an einer österreichischen Schule

Tab. 3 Fragen aus den semistrukturierten Interviews

Q1	Bieten die Prozessmodelle einen Mehrwert für die Schule? Wenn ja, inwieweit können die Schule/die Mitglieder der Schule von den erweiterten Prozessmodellen profitieren?
Q2	Wie können die erweiterten Prozessmodelle potenziell in der Schule verwendet werden?
Q3	Was halten Sie von unserem Ansatz zur integrierten Darstellung von Prozessmodellen und damit zusammenhängenden Sicherheitsaspekten? Vor- und Nachteile?
Q4	Haben Sie Verständnisprobleme mit verschiedenen Teilen der Prozesse? Wenn ja, welche Teile sind leicht verständlich und welche sind schwer oder unverständlich?
Q5	Haben Sie Vorschläge dazu, wie die grafische Darstellung der Prozesse verbessert werden kann?

von RQ2 wurden semistrukturierte Interviews durchgeführt. Eine detaillierte Beschreibung der Multimethodenstudie kann in Schefer-Wenzl et al. (2013) nachgelesen werden.

Abbildung 14 zeigt einen vereinfachten Beispielprozess aus unserer Fallstudie. Die gesamte Sammlung der in dieser Fallstudie modellierten Prozesse ist in Vondal (2012) dokumentiert. **Abbildung 14a** zeigt eine BusinessActivity, die den Prozess zur Organisation eines Tags der offenen Tür modelliert. Alle durch unsere UML-Erweiterung neu eingeführten Modellelemente werden in diesem Beispielprozess verwendet. Der in **Abb. 14a** dargestellte Prozess enthält sechs BusinessActions. Zwei dieser BusinessActions sind mit Pflichten verbunden. Weiterhin definiert der Prozess zwei Subject Binding Constraints zwischen „Datum planen“ und „Datum ankündigen“ sowie zwischen „Prüfen, dass der Plan auf dem neuesten Stand ist“ und „Zeitplan anpassen“. In **Abb. 14** werden die Rollen und die entsprechend zugewiesenen Aufgaben und Pflichten dargestellt. **Abbildung 14c** zeigt, welche Aufgaben und Pflichten ein Besitzer der Rolle Direktor an die Rolle Direktionsvertretung delegieren kann.

Nach der Neu-Modellierung der Prozesse mithilfe unserer UML-Erweiterung, wurden die neuen Prozessmodelle der Fallstudie bewertet, indem wir semistrukturierte Interviews mit drei Mitgliedern der Schule durchführt haben. Darunter waren der Schuldirektor, ein Lehrer und ein Mitglied der Schulverwaltung. Dieser Ansatz wurde gewählt, da Interviews eine wichtigste und anerkannte Methode in der Fallstudienforschung sind (siehe z. B. Runeson und Höst 2009). Weiterhin wird für eine qualitative Fallstudie empfohlen, Subjekte aus verschiedenen Teilen der Organisation zu wählen, um Mitglieder verschiedener Rollen an den Gesprächen zu beteiligen (siehe Corbin und Strauss 2008).

Die Gespräche wurden nach den Richtlinien aus Hove und Anda (2005) erstellt. Der Gesprächsleitfaden bestand aus fünf offenen Fragen. Jedes Gespräch dauerte zwischen 20 und 25 Minuten. Die Antworten wurden vor Ort in Form von Notizen aufgezeichnet und danach durch den Gesprächsführer analysiert. **Tabelle 3** zeigt die Hauptfragen, die während der Interviews gestellt wurden.

Im Rahmen der Interviews wurden uns insbesondere zwei Vorteile der visuell modellierten Prozesse kommuniziert. Erstens betonte der Direktor, dass neue Mitarbeiter, die mit den Abläufen in der Schule nicht vertraut sind, nun eine umfassende und leicht verständliche diagrammbasierte Dokumentation der Kernprozesse sowie der damit zusammenhängenden Möglichkeiten

für Delegationen zur Hand haben. Dies hätte das Potenzial, Arbeitsaufgaben und die Kommunikation mit anderen Mitgliedern der Schule während der ersten Wochen ihrer Arbeit an der Schule zu erleichtern. Die Äußerungen des Schuldirektors stützen die Vermutung, dass Prozessmodelle geeignete Kommunikationsinstrumente für Personen sind, die keine explizite software-technische Vorbildung haben (siehe z. B. Dumas et al. 2012). Weiterhin erhielten wir das Feedback, dass die Prozessbeschreibungen *vor* Durchführung der Fallstudie inkonsistent und uneinheitlich waren. Zudem merkten die Gesprächspartner an, dass mit Zugriffskontrollinformationen angereicherte Prozessmodelle das allgemeine Bewusstsein unter den Mitarbeitern der Schule verbessern, wie genau die verschiedenen Sicherheitsanforderungen mit den Kernprozessen verbunden sind. Alle drei Mitarbeiter der Schule gaben an, dass die Prozessmodelle leicht zu verstehen sind (z. B. Aufgaben- und Rollenbezeichnungen, Abfolge der Aufgaben, Beziehungen zwischen Pflichten und Aufgaben).

Unsere Fallstudie wurde speziell dazu entworfen, um unseren Modellierungsansatz zu bewerten. Daher hat die Fallstudie potenziell Einschränkungen hinsichtlich der Allgemeingültigkeit unserer Erkenntnisse für andere UML-Erweiterungen. Eine wichtige Einschränkung ergibt sich aus der Beschränkung der Studie auf eine einzelne Organisation. Die Beobachtungen sind daher möglicherweise spezifisch für die Domäne der weiterführenden Schulen in Österreich. Innerhalb dieser Domäne deckt die Studie jedoch eine große Menge verschiedener Aspekte ab, die vom Prozessmanagement der Schule bis zu Notfallevakuierungsverfahren reichen. Dennoch müssen künftige Studien zeigen, ob die Ergebnisse dieser Studie auch für andere Anwendungsbereiche/Domänen und andere Organisationstypen gelten.

Des weiteren gibt es potenzielle Einschränken der Allgemeingültigkeit der aus den drei Interviews abgeleiteten Ergebnisse. Zunächst können die Interviewergebnisse nicht verallgemeinert werden, weil die Gesprächspartner alle Mitglieder der selben Institution waren. Zudem bestand das potenzielle Risiko einer Voreingenommenheit des Gesprächsführers, da dieser gleichzeitig ein Autor der bewerteten UML-Erweiterungen ist. Diese Doppelrolle kann die Gespräche potenziell beeinträchtigt haben. Um dieses Risiko zu minimieren, versuchte der Gesprächsleiter jedoch, zu beobachten, statt das Gespräch zu lenken, und ermutigte die Gesprächspartner dazu, frei zu sprechen.

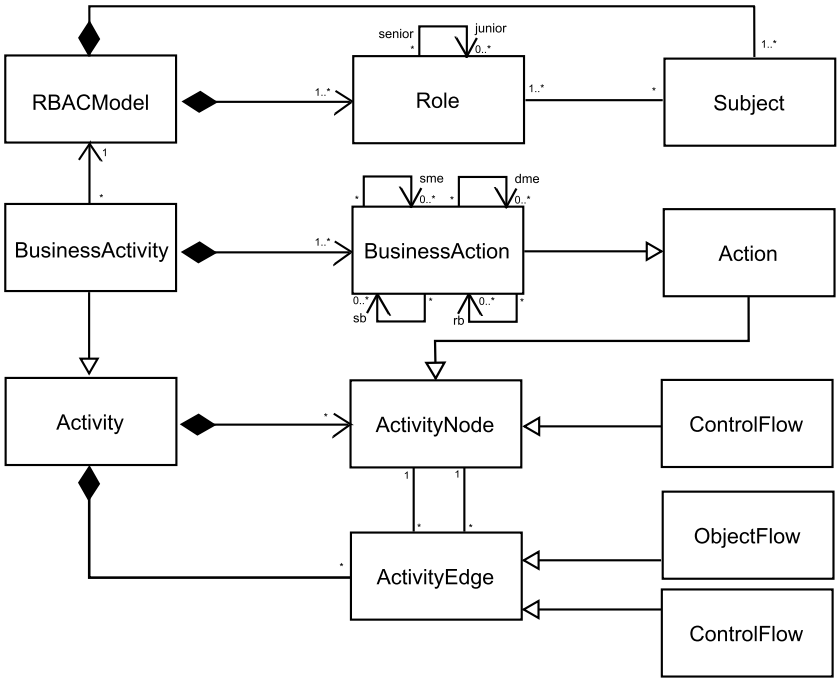


Abb. 15 Klassenmodell der BusinessActivity-Library and Runtime-Engine (Ausschnitt; Strembeck und Mendling 2011)

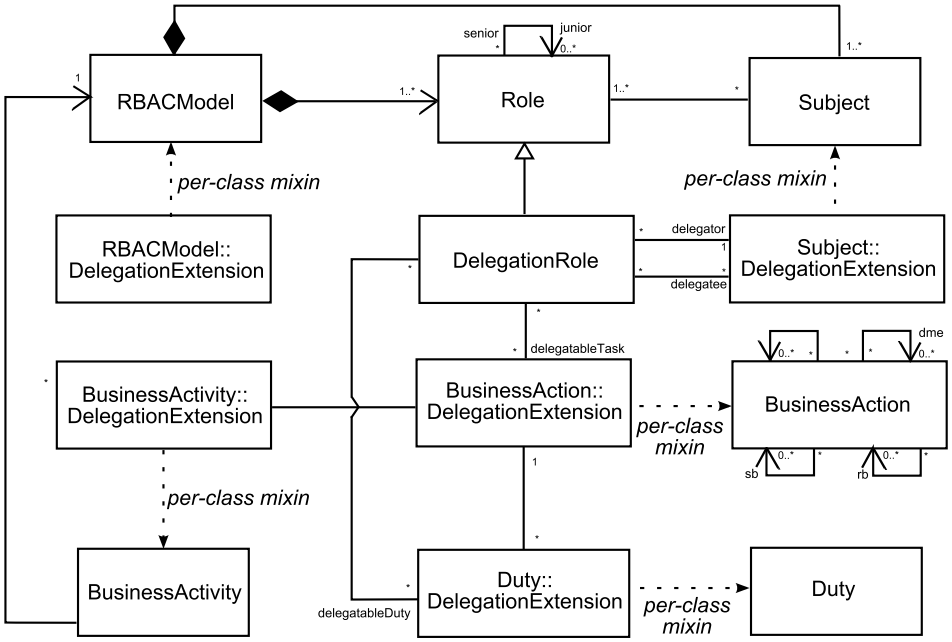
7 Plattformunterstützung

Um die Definition und Durchsetzung prozessbezogener RBAC-Delegationsmodelle auf der PSM-Ebene zu ermöglichen, wurde eine entsprechende Erweiterung für die *BusinessActivity Library and Runtime-Engine* implementiert. Dieser Abschnitt gibt einen Überblick über diese Erweiterung (zum Download verfügbar unter BAL 2012).

Die BusinessActivity Library and Runtime-Engine stellt eine Laufzeitumgebung für prozessbezogene RBAC-Modelle zur Verfügung. Sie unterstützt alle Elemente von prozessbezogenen RBAC-Modellen und stellt Funktionen zur Verwaltung der jeweiligen Laufzeitinstanzen zur Verfügung (siehe auch Strembeck und Mendling 2011). Weiterhin setzt unsere Laufzeitumgebung automatisch alle Constraints durch, die über OCL-Invarianten definiert wurden (siehe Abschn. 5.2). **Abbildung 15** zeigt einen Ausschnitt der wichtigsten Klassenbeziehungen der BusinessActivity Library and Runtime-Engine.

Die Delegationserweiterung wurde als Extension Package implementiert und erweitert die BusinessActivity Library and Runtime-Engine um Unterstützung für prozessbezogene RBAC-Delegationsmodelle. **Abbildung 16** zeigt die wesentlichen Klassenbeziehungen dieser Erweiterung. Die BusinessActivity Library and Runtime-Engine sowie das Extension Package wurden in der Programmiersprache Extended Object Tcl (XOTcl, siehe z. B. Neumann und Sobernig 2009, 2011; Neumann und Zdun 2000) implementiert. XOTcl ist eine objektorientierte Erweiterung der Scriptsprache Tcl (Ousterhout 1990). Im wesentlichen ist XOTcl eine C-Bibliothek, die dynamisch in eine Tcl-kompatible Umgebung geladen und in C-Programme eingebettet werden kann. Unter anderem stellt XOTcl einen Mixin-Mechanismus zur Verfügung (siehe Zdun et al. 2007). XOTcl-Mixin-Klassen unterstützen das dynamischen Abfangen von Methodenaufrufen. Sie ermöglichen eine flexible Erweiterung von Klassen ohne die Vererbungshierarchie verändern zu müssen. XOTcl unterstützt sowohl sogenannte Per-Object-Mixins als auch Per-Class-Mixins. Per-Object-Mixins können einzelne Objekte individuell um zusätzliche Funktiona-

Abb. 16 Klassenmodell des Delegationspaketes



litäten erweitern. Im Gegensatz dazu erweitern Per-Class-Mixins bestimmte Klassen und damit jede Instanz der erweiterten Klasse um zusätzliche Funktionalitäten (siehe Zdun et al. 2007). Beide XOTcl-Mixinkonstrukte gelangen in unserem Extension Package zur Anwendung, um bestimmte Funktionalitäten für eine Klasse oder ein Objekt dynamisch zu aktivieren oder zu deaktivieren (siehe **Abb. 16**).

8 Verwandte Arbeiten

Wir unterscheiden in diesem Beitrag zwischen drei Arten verwandter Arbeiten. Zunächst gibt es Ansätze, die vor allem darauf abzielen, Delegationsaspekte in rollenbasierte Zugriffskontrollmodelle zu integrieren. Zweitens behandeln einige wenige Ansätze die Delegation von Pflichten/Verpflichtungen. Drittens gibt es eine Reihe verschiedener Delegationsansätze für Workflow-/Geschäftsprozessumgebungen. Viele der verwandten Ansätze sind komplementär zu unserer Arbeit und können mit unseren prozessbezogenen RBAC-Delegationsmodellen kombiniert werden.

Tabelle 4 gibt einen Überblick über die verwandten Arbeiten. Bezüglich der Konzepte und Artefakte, die in den Abschn. 2, 3, 4 und 5 vorgestellt wurden, verwenden wir ein $\surd$, wenn ein verwandter Ansatz ähnliche und/oder vergleichbare Unterstützung für ein bestimmtes Konzept bietet und ein Δ, wenn ein verwandter Ansatz zumindest teilweise Unterstützung für einen bestimmten Aspekt bietet.

In den letzten Jahren wurden verschiedene Ansätze im Bereich der rollen- und rechtebasierten Delegation publiziert. In Barka und Sandhu (2000b) wird das RBDM-Framework für die Charakterisierung rollenbasierter Delegationsmodelle vorgestellt, das beispielsweise zwischen permanenter und temporärer, teilweiser und vollständiger sowie einstufiger und mehrstufiger Delegation unterscheidet. All diese Konzepte wurden auch in unser Delegationsmodell integriert. Ein formales Modell und einige Erweiterungen für RBDM werden in Barka und Sandhu (2000a) vorgestellt. RDM2000 (Zhang et al. 2003a) ist eine Erweiterung von RBDM zur Unterstützung mehrstufiger Delegation. Weiterhin wird eine rollenbasierte deklarative Sprache vorgeschlagen, um die zugehörigen Delegationsrichtlinien zu definieren. Ähnlich wie unser Ansatz, stellt RBDM2000 Toolunterstützung zur Verfügung und berücksichtigt Separation-of-Duty-Richtlinien.

In Zhang et al. (2003b) wird ein Delegationsmodell auf Rechtebasis (PBDM) vorgestellt, das die Delegation von Rollen und Rechten ermöglicht. Auch in PBDM werden Delegationsrollen verwendet, um Rechte an Benutzer zu delegieren. Die meisten der in Zhang et al. (2003b) vorgestellten Konzepte wurden auch in unserem Delegationsmodell berücksichtigt. Unterstützung für prozessbezogene Entailment Constraints ist in Zhang et al. (2003b) jedoch auf statische Separation-of-Duty-Richtlinien beschränkt, während unser Ansatz auch die Definition von Binding Constraints ermöglicht. Ein zu Zhang et al. (2003b) ähnlicher Ansatz wird in Hasebe et al. (2010) präsentiert. Insbesondere wird in Hasebe et al. (2010) das auf RBAC96 (Sandhu et al. 1996) aufbauende *Capability-Based Delegation Model* (CRBAC) vorgestellt. CRBAC ermöglicht die Delegation von Rollen und Rechten durch den Transfer von Capabilities. Im Vergleich zu unserer Arbeit unterstützt jedoch keiner dieser Ansätze die Delegation von Pflichten. Zusätzlich zum in Hasebe et al. (2010) vorgestellten Delegationsmodell ermöglicht PBDM (Zhang et al. 2003b) die Erkennung verschiedener Delegationskonflikte. Im Gegensatz zu unserem Beitrag werden jedoch keine entsprechenden Lösungsoptionen für diese Konflikte diskutiert.

Sohr et al. (2012) stellen einen UML-basierten Ansatz für die modellgetriebene Spezifikation rollenbasierter Delegationsrichtlinien vor. Im Gegensatz zu unserem Ansatz stellt Sohr et al. (2012) jedoch keine Verbindung zu Geschäftsprozessmodellen oder prozessbasierten Softwaresystemen her. Weiterhin beruht der Ansatz aus Sohr et al. (2012) nicht auf einem generischen Metamodell und behandelt weder Binding Constraints noch die Delegation von Pflichten. Sohr et al. verwenden inbesondere Standard UML-Klassen- und Objekt-

Tab. 4 Vergleich verwandter Arbeiten

	Delegation von Rollen	Delegation von Aufgaben	Delegation von Pflichten	Entailment constraints	Konflikt-identifikation	Konflikt-lösung	Formales Metamodell (CIM Ebene)	Modellierungsunterstützung (PIM Ebene)	Toolunterstützung (PSM Ebene)
Rollenbasierte Delegationsmodelle									
Barka and Sandhu (2000a, 2000b)	√						Δ		
Zhang et al. (2003a)	√			Δ			Δ		Δ
Shang und Wang (2008), Zhang et al. (2003b)	√			Δ	Δ		Δ		
Hasebe et al. (2010)	√						Δ		
Sohr et al. (2012)	√			Δ				Δ	
Delegationsmodelle für Verpflichtungen									
Cole et al. (2001)			√						
Schaad und Moffett (2002)			√				Δ		
Ghorbel-Talbi et al. (2010, 2011)	√		√				Δ		
Delegation in Geschäftsprozessen									
Gaaloul und Charoy (2009); Gaaloul et al. (2011, 2010)	Δ	√		√			√		Δ
Atluri und Warner (2005)		√		Δ	Δ		Δ		
Wainer et al. (2007)	√	√		Δ			Δ		Δ
Crampton und Khambhammettu (2008c)	√	√		Δ			Δ		
Crampton und Khambhammettu (2008a)	√	√		Δ	Δ		Δ		
Prozessbasierte RBAC-Delegationsmodelle (unser Ansatz)	√	√	√	√	√	√	√	√	√

diagramme für die grafische Visualisierung von Delegationsrichtlinien. Toolunterstützung sowie die Behandlung von Konflikten werden in Sohr et al. (2012) nicht thematisiert.

Obwohl auch Pflichten oder Verpflichtungen delegiert werden können, wurde dieser Delegationstyp in der Literatur bislang nur wenig beachtet. Eine der ersten Veröffentlichungen in dieser Richtung Cole et al. (2001) beleuchtet verschiedene Arten der Delegation von Verpflichtungen. Auch in Schaad und Moffett (2002) wird explizit die Delegation von Pflichten behandelt. In Ghorbel-Talbi et al. (2010, 2011) wurde ein spezielles Delegationsmodell für Verpflichtungen eingeführt. Im Vergleich zu unserer Arbeit behandelt jedoch keiner dieser Ansätze die Delegation von Pflichten in einem Geschäftsprozesskontext. Weiterhin werden weder prozessbezogene Entailment Constraints, noch entsprechende Modellierungs/Toolunterstützung oder die Erkennung und Lösung von Konflikten thematisiert.

Zusätzlich zu den bereits erwähnten gibt es auch einige Beiträge, die das Konzept der Delegation im Kontext von Geschäftsprozessen behandeln. In Atluri und Warner (2005) wird zum Beispiel das Konzept der bedingten Delegation vorgestellt. Der Beitrag betrachtet in diesem Zusammenhang auch verschiedene Arten von Constraints, wie etwa Separation-of-Duty-Richtlinien. Weiterhin werden drei Arten von Konflikten sowie ein Algorithmus zur Zuordnung von Aufgaben zu Subjekten (vergleichbar mit Algorithmus 4 aus Abschn. 4) vorgestellt. In Wainer et al. (2007) wird ein formales Modell für die rollenbasierte und aufgabenbasierte Delegation in Workflows beschrieben. Im Vergleich zu unserem Beitrag trifft Wainer et al. (2007) jedoch keine Unterscheidung zwischen Subject-Binding und Role-Binding Constraints. Weiterhin wird die Erkennung und Auflösung von delegationsbezogenen Konflikten in Wainer et al. (2007) nicht behandelt. Ähnliche Ansätze finden sich auch in Crampton und Khambhammettu, Crampton und Khambhammettu (2008a, 2008c), wobei keiner dieser Beiträge entsprechende Modellierungsunterstützung zur Verfügung stellt und nur in eingeschränktem Maße auf die Erkennung von Konflikten eingegangen wird.

Es gibt nur wenige Beiträge, welche Entailment Constraints und die sich daraus potenziell ergebenden Konflikte im Kontext von Delegation behandeln. Gaaloul und Charoy (2009) und Gaaloul et al. (2011, 2010) stellen einen formalen Ansatz für die Integration der Delegation von Aufgaben in das RBAC-Modell vor, der ebenfalls Separation-of-Duty- und Binding-of-Duty-Richtlinien berücksichtigt. Im Vergleich zu Gaaloul und Charoy (2009) und Gaaloul et al. (2011, 2010) behandelt unser Ansatz auch die Delegation von Pflichten und stellt zudem eine entsprechende UML-Erweiterung zur Verfügung, um die grafische Visualisierung der prozessbezo-

Zusammenfassung / Abstract

Sigrid Schefer-Wenzl, Mark Strembeck

Modellierungsunterstützung für die rollenbasierte Delegation in prozessgestützten Informationssystemen

Der Beitrag stellt einen integrierten Ansatz für die Modellierung und Durchsetzung von Delegationsrichtlinien in prozessbasierten Informationssystemen vor. In diesem Kontext wird eine entsprechende Erweiterung für rollenbasierte Zugriffskontrollmodelle (RBAC) beschrieben. Diese Erweiterung ist insofern generisch, als sie prinzipiell verwendet werden kann, um beliebige prozessbasierte Informationssysteme oder Prozessmodellierungssprachen mit Konstrukten für RBAC-Delegationsmodelle zu erweitern. Des weiteren befasst sich der Beitrag mit der Identifikation delegationsbezogener Konflikte sowie den zugehörigen Lösungsstrategien. Insbesondere ist der Ansatz darauf ausgelegt, die Konsistenz der RBAC-Modelle sowohl zur Entwurfszeit als auch zur Laufzeit sicherzustellen. Basierend auf einem formalen Metamodell wird zudem eine UML-Erweiterung für die Delegation von Rollen, Aufgaben und Pflichten vorgestellt. Diese UML-Erweiterung kann einerseits gemeinsam mit bereits bestehenden UML-Erweiterungen verwendet werden und demonstriert andererseits das prinzipielle Vorgehen zur Integration der neuen Modellkonstrukte in eine standardisierte Modellierungssprache. Zur Evaluierung der praktischen Anwendbarkeit des Ansatzes wurde eine Fallstudie an einem Realweltbeispiel durchgeführt. Weiterhin wurden alle vorgestellten Modellkonstrukte als Erweiterung der „BusinessActivity Library and Runtime Engine“ implementiert.

Schlüsselwörter: Delegation, Geschäftsprozesse, Pflichten, RBAC, Sicherheit, Zugriffskontrolle

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

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

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

genen Delegationskonzepte zu ermöglichen. In Shang und Wang (2008) wird eine Erweiterung von PBDM vorgestellt, die die Einhaltung von Constraints bei der Delegation berücksichtigt. Im Gegensatz zu unserer Arbeit konzentriert sich Shang und Wang (2008) aber auf statische Separation-of-Duty-Richtlinien und behandelt nur sehr knapp damit zusammenhängende Konflikte. Weiterhin werden lediglich rollenbasierte Einschränkungen besprochen, während unser Beitrag aufgabenbasierte Einschränkungen betrachtet. In Crampton und Khambhammettu (2008a) bespricht Crampton das „Satisfiability“-Problem von Workflows im Zusammenhang mit Delegation. Crampton stellt hierzu auch einen Algorithmus zur Verfügung, der bestimmt, ob eine Delegation zulässig ist oder nicht. Der Algorithmus unterscheidet jedoch nicht zwischen verschiedenen Konflikttypen und stellt keine entsprechenden Konfliktlösungen zur Verfügung. Weiterhin konzentrieren sich Crampton und Khambhammettu (2008a) insbesondere auf die aufgaben- und rollenbasierte Delegation, während unser Ansatz zusätzlich die Delegation von Pflichten ermöglicht. In Schaad (2001) diskutiert Schaad Delegationskonflikte, die im Rahmen des RBAC96-Modells auftreten können. Hierbei konzentiert er sich vor allem auf Konflikte, die im Zusammenhang mit Separation-of-Duty-Richtlinien stehen. Weiterhin ist der Ansatz darauf ausgerichtet, die Konflikte nach Durchführung der Delegation zu erkennen, während unsere Algorithmen Konflikte bereits bei (bzw. vor) der Ausführung der Delegation erkennen. Daher werden in unserem Ansatz Konflikte erkannt und aufgelöst, bevor sie zu einer inkonsistenten RBAC-Konfiguration führen können.

Nach unserem Wissen stellt dieser Beitrag den ersten Ansatz zur systematischen Delegation von Aufgaben, Pflichten und Rollen im Geschäftsprozesseskontext dar. Im Gegensatz zu anderen Ansätzen berücksichtigen wir durchgängig auch Mutual Exclusion Constraints und Binding Constraints und stellen Strategien für die Lösung der einzelnen Konflikttypen zur Verfügung (siehe **Tab. 4**).

9 Zusammenfassung

Dieser Beitrag stellte einen Ansatz zur integrierten Modellierung von Delegationskonzepten vor. Unser Ansatz basiert auf einem generischen Metamodell auf

der CIM-Ebene. Ausgehend von diesem Metamodell enthält unser Beitrag generische Algorithmen und Lösungsstrategien für Konflikte, die sich bei der Delegation von Aufgaben, Pflichten und Rollen ergeben können. Besondere Aufmerksamkeit wurde hierbei der Berücksichtigung von Mutual Exclusion Constraints und Binding Constraints gewidmet. Unser Ansatz ist so ausgelegt, dass Konflikte erkannt und gelöst werden, bevor sie zu einer inkonsistenten RBAC-Konfiguration führen. Auf diese Weise stellen wir die jederzeitige Konsistenz der prozessbezogenen RBAC-Delegationsmodelle sicher.

Auf der PIM-Ebene enthält unser Ansatz eine UML-Erweiterung für die Modellierung von Prozessen und zugehörigen Delegationsrichtlinien. Weiterhin wurden alle Elemente des generischen Metamodells als Extension Package für die BusinessActivity Library and Runtime-Engine implementiert (zum freien Download verfügbar von BAL 2012). Zur Evaluierung unserer UML-Erweiterung wurden eine Fallstudie sowie zugehörige Interviews durchgeführt. Zur Fortführung unserer Arbeit planen wir die Durchführung weiterer (Industrie-)Fallstudien. Weiterhin möchten wir untersuchen, wie andere sicherheitsbezogene Konzepte in die Delegation integriert werden können. Beispielsweise möchten wir die Secure-Object-Flows-(SOF-)Erweiterung, die in Hoisl et al. (2014) vorgestellt wird, mit dem hier vorgestellten Ansatz integrieren. Davon abgesehen wollen wir unser generisches Metamodell auf der CIM-Ebene verwenden, um andere Prozess-Modellierungsprachen (wie etwa BPMN) mit einer Delegationserweiterung auszustatten und potenzielle Unterschiede zwischen verschiedenen Modellierungssprachen bezüglich dieser Sicherheitserweiterungen analysieren.

Literatur

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Gaaloul K, Charoy F (2009) Task delegation based access control models for workflow systems. In: Proceedings of the 9th IFIP conference on e-business, e-services, and e-society (I3E)

Gaaloul K, Zahoor E, Charoy F Godart C (2010) Dynamic authorisation policies for event-based task delegation. In: Proceedings of the 22nd International conference on advanced information systems engineering (CAiSE)

Gaaloul K, Proper E, Charoy F (2011) An extended RBAC model for task delegation in workflow systems. In: Proceedings of the workshops on business informatics research

Georgiadis CK, Mavridis I, Pangalos G, Thomas RK (2001) Flexible team-based access control using contexts. In: Proceedings of the 6th ACM symposium on access control models and technologies (SACMAT), S 21–27

Ghorbel-Talbi MB, Cuppens F, Cuppens-Boulahia N (2010) Negotiating and delegating obligations. In: Proceedings of the International conference on management of emergent digital ecosystems (MEDES)

Ghorbel-Talbi MB, Cuppens F, Cuppens-Boulahia N, Metayer DL, Piolle G (2011) Delegation of obligations and responsibility. In: Proceedings of the International information security and privacy conference (SEC)

Hasebe K, Mabuchi M, Matsushita A (2010) Capability-based delegation model in RBAC. In: Proceedings of the 15th ACM symposium on access control models and technologies (SACMAT), S 109–118

Hoisl B, Sobernig S, Strembeck M (2014) Modeling and enforcing secure object flows in process-driven SOAs: an integrated model-driven approach. Software and Systems Modeling 2:513–548

Hove SE, Anda B (2005) Experiences from conducting semi-structured interviews in empirical software engineering research. In: Proceedings of the 11th IEEE International software metrics symposium (METRICS)

Joshi JBD, Bertino E (2006) Fine-grained role-based delegation in presence of the hybrid role hierarchy. In: Proceedings of the 11th ACM symposium on access control models and technologies (SACMAT), S 81–90

Jürjens J (2005) Sound methods and effective tools for model-based security engineering with UML. In: Proceedings of the 27th International conference on software engineering (ICSE)

Mouratidis H, Jürjens J (2010) From goal-driven security requirements engineering to secure design. International Journal of Intelligent Systems 25(8):813–840

Neumann G, Sobernig S (2009) XOTcl 2.0 – a ten-year retrospective and outlook. In: Proceedings of the sixteenth annual Tcl/Tk conference

Neumann G, Sobernig S (2011) An overview of the next scripting toolkit. In: Proceedings of the the 18th annual Tcl/Tk conference

Neumann G, Zdun U (2000) XOTcl, an object-oriented scripting language. In: Proceedings of Tcl2k: the 7th USENIX Tcl/Tk conference

Oh S, Park S (2003) Task-role-based access control model. Information Systems 28(6):533–562

OMG (2011a) Meta object facility (MOF) core specification. Version 2.4.1, formal/2011-08-07. The Object Management Group. http://www.omg.org/spec/MOF. Abruf am 2012-02-27

OMG (2011b) Unified modeling language (OMG UML): superstructure. Version 2.4.1, formal/2011-08-06. The Object Management Group. http://www.omg.org/spec/UML

OMG (2014) Object constraint language specification. Version 2.4, formal/2014-02-03. The Object Management Group. http://www.omg.org/spec/OCL. Abruf am 2014-04-25

Ousterhout J (1990) Tcl: an embeddable command language. In: Proceedings of the winter USENIX conference

Ravichandran A, Yoon J (2006) Trust management with delegation in grouped peer-to-peer communities. In: Proceedings of the 11th ACM symposium on access control models and technologies (SACMAT), S 71–80

Recker J, Indulska M, Rosemann M, Green P (2006) How good is BPMN really? Insights from theory and practice. In: 14th European conference on information systems

Rodriguez A, de Guzman IGR (2007) Obtaining use case and security use cases from secure business process through the MDA approach. In: Proceedings of the interna-

tional workshop on security in information systems (WOSIS)
Rodriguez A, Fernandez-Medina E, Piattini M (2006) Towards a UML 2.0 extension for the modeling of security requirements in business processes. In: Proceedings of the international conference on trust and privacy in digital business (TrustBus)
Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14(2):131–164
Russell N, Hofstede AHMT, Edmond D (2005) Workflow resource patterns: identification, representation and tool support. In: Proceedings of the 17th conference on advanced information systems engineering (CAiSE'05). Lecture notes in computer science, Bd 3520. Springer, Heidelberg, S 216–232
Sandhu R, Coyne E, Feinstein H, Youman C (1996) Role-based access control models. IEEE Computer 29(2):38–47
Schaad A (2001) Detecting conflicts in a role-based delegation model. In: Proceedings of the 17th annual computer security applications conference (ACSAC), S 117–126
Schaad A, Moffett JD (2002) Delegation of obligations. In: Proceedings of the 3rd International workshop on policies for distributed systems and networks (POLICY)
Schefer S, Strembeck M (2011a) Modeling process-related duties with extended UML activity and interaction diagrams. Electronic Communications of the EASST 37
Schefer S, Strembeck M (2011b) Modeling support for delegating roles, tasks, and duties in a process-related RBAC context. In: International workshop on information systems security engineering (WISSE). Lecture notes in business information processing. Springer, Heidelberg
Schefer S, Strembeck M, Mendling J, Baumgrass A (2011) Detecting and resolving conflicts of mutual-exclusion and binding constraints in a business process context. In: Proceedings of the 19th International conference on cooperative information systems (CoopIS). Lecture notes in computer science, Bd 7044. Springer, Heidelberg
Schefer-Wenzl S, Strembeck M, Baumgrass A (2012) An approach for consistent delegation in process-aware information systems. In: Proceedings of the 15th international conference on business information systems (BIS). Lecture notes in business information processing, Bd 117. Springer, Heidelberg
Schefer-Wenzl S, Sobernig S, Strembeck M (2013) Evaluating a UML-based modeling framework for process-related security properties: a qualitative multi-method study. In: Proceedings of the 21st European conference on information systems (ECIS), Utrecht
Schmidt DC (2006) Model-driven engineering – guest editor's introduction. IEEE Computer 39(2):25–31
Selic B (2003) The pragmatics of model-driven development. IEEE Software 20(5):19–25
Shang Q, Wang X (2008) Constraints for permission-based delegations. In: Proceedings of the 8th IEEE International conference on computer and information technology workshops (CITWORKSHOPS), S 216–223
Sohr K, Kuhlmann M, Gogolla M, Hu H, Ahn GJ (2012) Comprehensive two-level analysis of role-based delegation and revocation policies with UML and OCL. Information and Software Technology 54(12):1396–1417
Stahl T, Völter M (2006) Model-driven software development. Wiley, New York
Strembeck M (2005) Embedding policy rules for software-based systems in a requirements context. In: Proceedings of the 6th IEEE International workshop on policies for distributed systems and networks (POLICY)
Strembeck M (2010) Scenario-driven role engineering. IEEE Security & Privacy 8(1):28–35
Strembeck M, Mendling J (2010) Generic algorithms for consistency checking of mutual-exclusion and binding constraints in a business process context. In: Proceedings of the 18th International conference on cooperative information systems (CoopIS). Lecture notes in computer science, Bd 6426. Springer, Heidelberg
Strembeck M, Mendling J (2011) Modeling process-related RBAC models with extended uml activity models. Information and Software Technology 53(5):456–483
Tan K, Crampton J, Gunter CA (2004) The consistency of task-based authorization constraints in workflow systems. In: Proceedings of the 17th IEEE workshop on computer security foundations
Thomas RK, Sandhu RS (1997) Task-based authorization controls (TBAC): a family of models for active and enterprise-oriented autorization management. In: Proceedings of the IFIP TC11 WG11.3 11th International conference on database security XI: status and prospects, S 166–181
Vondal F (2012) Modellierung von Delegation in prozessbezogenen RBAC-Modellen – Eine Fallstudie. Bachelor thesis, WU Vienna
Wainer J, Barthelmess P, Kumar A (2003) W-RBAC – a workflow security model incorporating controlled overriding of constraints. International Journal of Cooperative Information Systems 12(4):455
Wainer J, Kumar A, Barthelmess P (2007) DW-RBAC: a formal security model of delegation and revocation in workflow systems. Information Systems 32(3):365–384
Warner J, Atluri V (2006) Inter-instance authorization constraints for secure workflow management. In: Proceedings of the 11th ACM symposium on access control models and technologies (SACMAT), S 190–199
Weske M (2012) Business process management: concepts, languages, architectures, 2. Aufl. Springer, Heidelberg
Wolter C, Schaad A, Meinel C (2008) A transformation approach for security enhanced business processes. In: Proceedings of the IASTED International conference on software engineering
Wolter C, Menzel M, Schaad A, Miseldine P, Meinel C (2009) Model-driven business process security requirement specification. Journal of Systems Architecture 55(4):211–223
Zdun U, Strembeck M, Neumann G (2007) Object-based and class-based composition of transitive mixins. Information and Software Technology 49(8):871–891
Zhang L, Ahn GJ, Chu BT (2003a) A rule-based framework for role-based delegation and revocation. ACM Transations on Information System Security 6(3):404–441
Zhang X, Oh S, Sandhu R (2003b) PBDM: a flexible delegation model in RBAC. In: Proceedings of the 8th ACM symposium on access control models and technologies (SACMAT), S 149–157
Zhao G, Chadwick D, Otenko S (2007) Obligations for role based access control. In: Proceedings of the 21st International conference on advanced information networking and applications workshops (AINAW), S 424–431
zur Muehlen M, Indulska M (2010) Modeling languages for business processes and business rules: a representational analysis. Information Systems 35(4):379–390