Kaskadierender Widerruf von Delegationen in prozessbasierten Informationssystemen

David Hopfmüller, Sigrid Schefer-Wenzl und Mark Strembeck
Institut für Wirtschaftsinformatik und Neue Medien
Wirtschaftsuniversität Wien
{vorname.nachname}@wu.ac.at

Abstract: Das *BusinessActivities Framework* stellt einen integrierten, modellgetriebenen Ansatz zur Modellierung von Sicherheitseigenschaften für prozessbasierte Informationssysteme zur Verfügung. Ein wichtiger Teil des BusinessActivity Frameworks ermöglicht die Modellierung von rollenbasierten Zugriffskontrollmodellen. In diesem Kontext ist auch die Möglichkeit zur Delegation von Rollen und Rechten vorgesehen. Der vorliegende Beitrag erweitert den Ansatz um einen Algorithmus für den kaskadierenden Widerruf von Delegationen. Der Ansatz umfasst eine prototypische Implementierung, die als Erweiterung der BusinessActivity LRE (von engl.: Library and Runtime Engine) fungiert.

1 Einleitung

Ein Geschäftsprozess besteht aus einer Menge an Aufgaben, die in einer Organisation durchgeführt werden müssen, um bestimmte Ziele zu erreichen. Zur Unterstützung der sicheren Ausführung von Geschäftsprozessen müssen alle Subjekte, die in einem bestimmten Prozess mitarbeiten, die benötigten Rechte besitzen (siehe z.B. [GMPT01, OP03, SM11]). In den vergangenen Jahren hat sich die *rollenbasierte Zugriffskontrolle* (engl.: role-based access control, RBAC, [FKC07, SCFY96]) sowohl in der Industrie als auch im wissenschaftlichen Bereich zu einem De-facto-Standard entwickelt. In RBAC bilden Rollen verschiedene Arbeitsbereiche und Verantwortlichkeiten innerhalb einer Organisation bzw. in einem Informationssystem ab. Einer Rolle werden einerseits Zugriffsberechtigungen gemäß ihres Aufgabenbereichs zugewiesen. In weiterer Folge werden diese Rollen menschlichen Benutzern entsprechend ihrer Arbeitsprofile zugewiesen (siehe [Str10]).

Delegation ist ein Mechanismus zur Flexibilisierung des organisationalen Rechtemanagements, der auch als wichtiges Konzept für prozessbasierte Informationssysteme identifiziert wurde (siehe z.B. [CK08]). Subjekte haben hierbei die Möglichkeit, zugewiesene Aufgaben, Pflichten oder komplette Rollen an andere Subjekte weiterzuleiten (siehe z.B. [SM02, SWSB12]). In weiterer Folge agiert der Empfänger der Delegation (engl.: delegatee) anstelle (bzw. im Auftrag) des delegierenden Subjekts (engl.: delegator). Eine *mehrstufige Delegation* liegt dann vor, wenn ein Subjekt Aufgaben, Pflichten oder Rollen, die es selbst durch Delegation erhalten hat, an ein drittes Subjekt weiter delegiert.

Der *Widerruf* (engl.: revocation) einer Delegation ist jener Vorgang, bei dem ein Subjekt anderen Subjekten zuvor delegierte Berechtigungen wiederum entzieht (siehe z.B. [HJPW01]). Grob kann zwischen einfachem und kaskadierendem Widerruf unterschieden werden. Beim *einfachen Widerruf* wird ein bestimmtes, delegiertes Recht bzw. eine bestimmte, delegierte Rolle ohne Berücksichtigung etwaiger Weiterdelegationen wieder zurückgenommen. *Kaskadierender Widerruf* beinhaltet die Aufhebung der delegierten Rechte, Pflichten oder Rollen auch aller Weiterdelegationen. Sobald dem Subjekt, an das zuerst delegiert wurde, die delegierten Berechtigungen entzogen werden, verlieren auch alle Subjekte, an die weiter delegiert wurde, diese Berechtigungen.

In diesem Beitrag präsentieren wir einen Algorithmus zum systematischen Widerruf von Delegationen in prozessbasierten Informationssystemen. Der Fokus liegt hierbei insbesondere auf der Diskussion spezieller Herausforderungen im Zuge des kaskadierenden Widerrufs. Die Evaluierung unseres Algorithmus erfolgte im Rahmen einer prototypischen Implementierung als Erweiterung einer bestehenden Softwareplattform, dem *BusinessActivities Framework*. Dieses Framework ermöglicht die integrierte Modellierung von Sicherheitseigenschaften für prozessbasierte Informationssysteme.

Abschnitt 2 liefert einen Überblick zum Thema Delegation und Widerruf in prozessbasierten Informationssystemen und geht insbesondere auf problematische Aspekte beim kaskadierenden Widerruf ein. Anschließend stellen wir den entwickelten Algorithmus und seine Integration in das BusinessActivities Framework vor. Abschnitt 3 schließt mit Informationen zur praktischen Implementierung und den parallel dazu entwickelten Tests an. In Abschnitt 4 wird ein Überblick über verwandte Arbeiten in diesem Bereich vermittelt. Abschnitt 5 fasst schließlich zusammen und schlägt einige Ideen und Anregungen zur Weiterentwicklung des Ansatzes vor[1].

2 Delegation und Widerruf in prozessbasierten Informationssystemen

2.1 Grundlagen der Delegation

Jede Aufgabe in einem IT-unterstützten Geschäftsprozess (z.B. eine Vertragsverhandlung) ist üblicherweise mit bestimmten Zugriffsrechten verknüpft (z.B. zur Unterzeichnung eines Vertrags). Daher muss jedes Subjekt, das Aufgaben in einem Prozess auszuführen hat, zur Durchführung dieser Aufgaben berechtigt sein (siehe z.B. [GMPT01, OP03, SM11]). In einem organisationalen Kontext kann eine Aufgabe auch mit bestimmten Pflichten verbunden sein. Eine Pflicht (engl.: duty) definiert eine Aktion, die aufgrund gesetzlicher oder unternehmensinterner Regeln durchgeführt werden muss (siehe z.B. [SM02, SS11a]). In dem Modell, das dieser Arbeit zugrunde liegt, sind sowohl Rollen, als auch Aufgaben und Pflichten delegierbar (siehe [SWSB12]). Eine Pflicht, die einem Subjekt (oder einer Rolle) zugewiesen wurde, wird auch als Verpflichtung (engl.: obligation) bezeichnet.

Delegation beschreibt im Rechte-Management einen Mechanismus, der es Subjekten er-

[1] Um den LNI-Vorgaben zu genügen, musste der vorliegende Artikel gekürzt werden. Die vollständige Fassung ist online auf der Homepage zum BusinessActivities Framework verfügbar: `http://wi.wu.ac.at/home/mark/BusinessActivities/library.html`

möglicht, Teile von oder alle zugewiesenen Aufgaben und Pflichten an andere Subjekte zu transferieren (siehe z.B. [ZOS03]). In Zusammenhang mit RBAC existiert eine Reihe von Modellen, die das Konzept so genannter Delegationsrollen einsetzen (siehe z.B. [JB06, SW08, ZOS03]). Bei dem in diesem Beitrag verwendeten Delegationsmodell wird eine *Delegationsrolle* von einem Subjekt erzeugt und enthält eine Menge delegierbarer Aufgaben und Pflichten (siehe [SWSB12]). Delegationsrollen werden in weiterer Folge an andere Subjekte delegiert, die dann anstelle des delegierenden Subjekts (z.B. in Stellvertretung) handeln können. Im Unterschied zu Delegationsrollen werden alle anderen Rollen daher nun als *reguläre Rollen* bezeichnet.

Wir unterscheiden prinzipiell drei Arten von Delegationsvorgängen:

- Delegation einzelner Aufgaben oder Pflichten an Delegationsrollen,
- Delegation vollständiger Rollen an Delegationsrollen,
- Delegation einer Delegationsrolle an ein Subjekt (genannt *delegatee*).

Um die Menge der delegierbaren Elemente sprachlich besser fassen zu können, soll an dieser Stelle als Oberbegriff aller delegierbaren Typen die Bezeichnung *Delegable* eingeführt werden. Ein *Delegable* kann somit vom Typ Aufgabe bzw. Recht, Pflicht oder Rolle sein, und kann zu Delegationszwecken einer Delegationsrolle hinzugefügt werden.

Zu jedem der oben genannten Delegationsvorgänge muss es in weiterer Folge auch einen entsprechenden Umkehrvorgang zum Widerruf bzw. zur Zurücknahme der Delegation geben. Die folgenden Rahmenbedingungen sind für einen wohlgeordneten Delegationsprozess und den anschließenden Widerruf von Delegationen von zentraler Bedeutung (siehe z.B. [BS00, SWSB12]):

1. Subjekte können eine beliebige Anzahl an Delegationsrollen erstellen. Das erstellende Subjekt (engl.: *creator*) wird automatisch zum *delegator* der Delegationsrolle.
2. Aufgaben (bzw. Rechte) können einer Delegationsrolle zugeordnet werden. Hierbei gelten folgende Regeln:
   (a) Die zu delegierende Aufgabe muss dem Ersteller der Delegationsrolle aktuell zugeordnet sein (durch eine reguläre Rolle oder, im Fall einer mehrstufigen Delegation, durch eine andere Delegationsrolle).
   (b) Die Aufgabe muss als delegierbar (engl.: *delegable*) definiert sein.
   (c) Soweit zugehörige Pflichten vorhanden sind, müssen diese ebenfalls als *delegable* definiert sein.
3. Delegationsrollen können an ein oder mehrere Subjekte *(delegatees)* delegiert werden. Diese erhalten dadurch das Recht (oder die Pflicht), die enthaltenen Aufgaben durchzuführen.
4. Subjekte können nur jene Aufgaben, Rollen und Delegationsrollen (weiter) delegieren, die ihnen selbst zugeordnet sind. Ist nur die einstufige Delegation (engl.:

single-step delegation) vorgesehen, beschränkt sich die Menge der delegierbaren Elemente eines Subjekts auf jene, die ihm über reguläre Rollen zugeordnet sind. Ist die mehrstufige Delegation (engl.: *multi-step delegation*) von Rollen und Rechten erlaubt, können auch per Delegationsrollen empfangene Elemente weiter delegiert werden.

5. Delegationsrollen können in einer *Rollenhierarchie* angeordnet werden (siehe z.B. [SWSB12, ZOS03]). Eine Rollenhierachie ist im Wesentlichen ein gerichteter, azyklischer Graph, der Vererbungsbeziehungen zwischen Rollen definiert. Um unerlaubte Rechteweitergabe zu vermeiden, gilt, dass reguläre Rollen nur als Junior-, nicht aber als Seniorrolle von Delegationsrollen zugelassen sind (siehe hierzu z.B. auch [ZOS03]).

6. Delegationsrollen können als *permanent* oder *temporär* definiert werden. Die Ausübung temporärer Delegationsrollen ist auf eine bestimmte Prozessinstanz beschränkt. Im Gegensatz dazu können permanente Delegationsrollen in jeder Instanz des zugehörigen Prozesstyps verwendet werden.

In diesem Beitrag werden weiterhin vier Arten von Einschränkungen (engl.: *constraints*) betrachtet, die die Kombination von Aufgaben einschränken (siehe z.B. [SM11] und Abbildung 1):

1. *Subject-binding* (SB): Auf diese Art verbundene Aufgaben müssen vom selben Subjekt ausgeführt werden, beispielsweise eine medizinische Diagnose und das Verfassen des zugehörigen Befundes durch den behandelden Arzt in einem Krankenhaus.

2. *Role-binding* (RB): Derart verbundene Aufgaben müssen von Mitgliedern derselben Rolle ausgeführt werden, aber nicht notwendigerweise vom selben Individuum – beispielsweise Zeichnungsberechtigungen in einer Firma.

3. *Static mutual exclusion* (SME): Aufgaben, die einander gegenseitig statisch ausschließen, dürfen nie derselben Rolle oder demselben Subjekt *zugeordnet* sein. Um Betrug und Missbrauch vorzubeugen, darf zum Beispiel kein Subjekt gleichzeitig Einkäufer und Controller der selben Unternehmung sein.

4. *Dynamic mutual exclusion* (DME): Aufgaben, die einander dynamisch ausschließen, dürfen zwar derselben Rolle (bzw. dem selben Subjekt) zugeordnet werden, diese Aufgaben müssen jedoch in jeder Prozessinstanz von verschiedenen Subjekten ausgeführt werden ("Mehraugenprinzip").

Die oben erwähnten Einschränkungen spielen bei der Delegation eine wichtige Rolle und erfordern einigen Aufwand, um ihre Einhaltung zu garantieren. *Subject-binding* und *role-binding* sind zudem auch für den Widerruf einer Delegation relevant, um einen konsistenten Zustand zu erhalten.

Sobald Aufgaben und Rollen delegiert werden können, entsteht die damit verbundene Anforderung, diese Delegationen auch widerrufen zu können. Dies trifft insbesondere dann

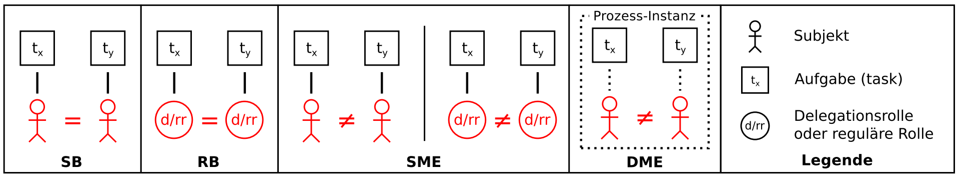

Abbildung 1: Mögliche Einschränkungen von Delegationen

zu, wenn die Delegation temporär erfolgen soll, wie es beispielsweise bei der Urlaubsvertretung in einer Firma der Fall sein kann. Es gilt der Grundsatz, dass grundsätzlich alles widerrufen werden kann, was delegiert werden kann. In Frage kommen somit:

- Widerruf einzelner *Aufgaben* (engl.: tasks) von einer Delegationsrolle,
- Widerruf von *regulären Rollen*, welche an eine Delegationsrolle delegiert wurden,
- Widerruf von *Delegationsrollen*, die an eine andere Delegationsrolle delegiert wurden,
- Entfernung einzelner *Empfänger* (engl.: delegatees) aus einer Delegationsrolle.

Für jeden dieser Widerrufsvorgänge unterscheiden wir grundsätzlich zwei Fälle:

Einfacher Widerruf: Beim einfachen Widerruf wird eine einzelne Delegation widerrufen. Bei dieser Delegation kann es sich um eine Aufgabe, Delegationsrolle oder reguläre Rolle handeln, die aus einer Delegationsrolle entfernt werden soll. Alternativ kann auch ein Empfänger einer solchen Delegationsrolle entfernt werden.

Kaskadierender Widerruf: Im Fall der mehrstufigen Delegation kann ein Empfänger die delegierten Elemente seinerseits weiterdelegieren. Beim kaskadierenden Widerruf sollen auch alle Weiterdelegationen rückgängig gemacht werden.

2.2 Problemstellungen des kaskadierenden Widerrufs

(1) Mehrfache Zuweisung

Das Grundproblem des kaskadierenden Widerrufs von Delegationen besteht darin, dass ein Empfänger eines delegierten Elements dasselbe Element mehrfach erhalten haben kann, nämlich durch eine (andere) Delegationsrolle, oder durch Zuweisung über regulärer Rollen.

Delegiert nun der Empfänger *(delegatee)* das Element seinerseits weiter, ist prinzipiell unklar, welche Instanz des Delegables weiter delegiert wurde. Zwar wäre eine Implementierung denkbar, die die einzelnen Instanzen unterscheidbar hält, diese Unterscheidung ist jedoch aus Sicht der Geschäftslogik nicht relevant und wird daher hier nicht weiter betrachtet. Allerdings bedeutet dies auch, dass die weitere Delegation nicht einfach zurückgezogen werden darf, ohne zu prüfen, ob eine zweite Quelle (engl.: *second source*) des fraglichen Elements existiert, die als Ursprung der Weitergabe in Frage kommt: Das kann entweder eine andere Delegationsrolle oder eine reguläre Rolle sein.

(2) Bestimmung der Herkunft mehrfacher Zuweisungen

Werden für ein Delegable ausschließlich Delegationsrollen als zweite Quelle gefunden, besteht die Möglichkeit, dass deren Herkunft letztendlich ausschließlich auf dieselbe (nämlich die ursprünglich zu widerrufende) Delegationsrolle zurückführt. Ist das der Fall, handelt es sich *nicht* um eine (qualifizierte) zweite Quelle.

(3) Zyklen bei mehrfacher Zuweisung

Im Zuge der Auflösung der Herkunft von Delegationsrollen, die als zweite Quelle in Frage kommen, stellt neben deren Herkunft auch die mögliche Zyklenbildung ein Problem dar. Zwar ist die direkte Bildung eines Zyklus zwischen zwei Delegationsrollen nicht zulässig (vgl. Abschnitt 2.1), allerdings könnte ein Subjekt S1 ein Element an Subjekt S2 delegieren, welches dasselbe Delegable (mittels einer eigenen Delegationsrolle) wieder zurück an S1 delegiert. Das Erkennen solcher zyklischer Graphen ist Voraussetzung für eine korrekte Implementierung des zugehörigen Modells.

(4) Weiter-Delegation von Teilmengen

Hat die ursprüngliche Delegationsrolle reguläre Rollen oder Delegationsrollen enthalten, können diese Container ihrerseits prinzipiell beliebig viele und beliebig tief verschachtelte Elemente enthalten. Der Empfänger einer solchen Delegationsrolle hat (bei aktivierter Mehrfachdelegation) nun die Möglichkeit, beliebige Teilmengen der ursprünglichen Delegationsrolle weiter zu delegieren.

Beim Auflösen derartiger Delegationsrollen sind daher nicht nur die direkten Juniorrollen (Kind-Elemente) einer Delegationsrolle zu beachten und zu behandeln, sondern gleichermaßen alle in Rollen enthaltenen Delegables.

2.3 Algorithmus für den kaskadierenden Widerruf von Delegationen

Der aus den oben genannten Anforderungen resultierende Algorithmus wurde in zwei größere Funktionen sowie einige Hilfsfunktionen aufgeteilt. Erstere sind dabei von besonderem Interesse, da sie der Verfolgung der potentiell komplexen Delegationsbeziehungen dienen. Die Prozedur *cascadingRevocation* sowie der Algorithmus *findSecondSource* sind nachfolgend in Pseudocode wiedergegeben und zudem als Erweiterung der BusinessActivity LRE (von engl.: Library and Runtime Engine) implementiert (siehe Abschnitt 3)[2]. Der Pseudocode der dort verwendeten Nebenfunktionen *taskRoleDelegationRevoke* und *roleRoleDelegationRevoke* ist aus Platzgründen nur in der erweiterten Version dieses Artikels enthalten.

[2]Der Quellcode der BusinessActivity LRE ist zum freien Download verfügbar: `http://wi.wu.ac.at/home/mark/BusinessActivities/library.html`

Wir benötigen die Menge aller Subjekte S, die Menge aller Rollen R (die alle Delegationsrollen DR miteinschließt), sowie die Menge der Aufgabentypen T_T samt seiner Teilmenge der delegierbaren Aufgabentypen DT_T. Auf Seiten der Funktionen kommt $drh^*(dr) = R_{j^*}$ zum Einsatz, welche die Menge aller direkten und transitiven Junior-Rollen einer Delegationsrolle liefert. Selbiges ist durch $rrh^*(r) = R_{j^*}$ für die reguläre Rollen-Hierarchie definiert. Das Mapping $town(dr) = T_{T_{dr}}$ liefert alle (auch über Junior-Rollen) delegierten Aufgaben einer Delegationsrolle, $rown_{rrh}(s) = R_s$ hingegen alle einem Subjekt über die reguläre Rollenhierarchie direkt und transitiv zugeordneten Rollen. Mit $rsdel(s) = DR_s$ können alle Delegationsrollen ermittelt werden, die momentan einem Subjekt zugeordnet sind. Umgekehrt kann mittels $rsdel^{-1}(dr) = S_{dr}$ auf alle Delegatees einer Delegationsrolle zugegriffen werden. $creator(dr) = s_{dr}$ wird verwendet, um den Ersteller einer Delegationsrolle zu bestimmen. Die Umkehrung $creator^{-1}(s) = DR_s$ ist in [SWSB12] nicht ausdrücklich definiert, hierfür verwenden wir folgende Definition: *Das Mapping $creator^{-1} : S \rightarrow \mathcal{P}(DR)$ wird **created delegation roles** genannt. Für ein gegebenes Subjekt $s \in S$ retourniert $creator^{-1}(s) = DR_s$ alle von diesem erstellte Delegationsrollen $DR_s \in DR$.*

Prozedur 1. *Finden und Widerrufen aller weiter delegierten Instanzen eines Delegables.*

```
Name: cascadingRevocation
Input: dr ∈ DR, revoke ∈ S ∪ R ∪ DT_T, handledSubjects ∈ S
create_empty_set delegatees
create_empty_set revokes
create_empty_set secondSources
if revoke ∈ S then
    set delegatees = revoke
    add drh*(dr) ∪ town(dr) to revokes
else
    add rsdel^-1(dr) to delegatees
    add revoke to revokes
    if revoke ∈ R then
        add rrh*(dr) ∪ town(dr) to revokes
    endif
    if revoke ∈ DR then
        add drh*(dr) ∪ town(dr) to revokes
    endif
endif
for each delegatee ∈ delegatees
    add delegatee to handledSubjects
    for each createdDg ∈ drh*(creator^-1(delegatee)) \
    ∪ town(creator^-1(delegatee))
        if createdDg ∈ revokes ∧ createdDg ∉ secondSources then
            if createdDg ∈ findSecondSource(createdDg, delegatee) then
                add createdDg to secondSources
            else
                if createdDg ∈ DT_T then
```

for each $createdDr \in creator^{-1}(delegatee) \setminus$
$\mid createdDg \in town(createdDr)$
*taskRoleDelegationRevoke *
(createdDg, createdDr, "cascation", handledSubjects)
else if $createdDg \in R$ **then**
for each $createdDr \in creator^{-1}(delegatee) \setminus$
$\mid createdDg \in drh^{*}(createdDr)$
*roleRoleDelegationRevoke *
(createdDg, createdDr, "cascation", handledSubjects)
endif; endif; endif; endfor; endfor

Algorithmus. *Bestimmen, ob es weitere Quellen als Grundlage für eine Delegation gibt.*

Name: *findSecondSource*
Input: $dg \in R \cup DT_T$, $subject \in S$, $creator \in S$
create_empty_set *checkedSubjects*
if $dg \in DR \wedge creator(dg) == subject$ **then**
return true
elseif $dg \in rown_{rrh}(subject) \cup town(rown_{rrh}(subject))$
return true
else
add *subject* to *checkedSubjects*
for each $dr \in rsdel(subject)$
if $(dg \in R \wedge dg \in drh^{*}(dr))$
$\vee (dg \in DR \wedge dg \in drh^{*}(dr))$
$\vee (dg \in DT_T \wedge dg \in town(dr))$ **then**
if $creator(dr) \in checkedSubjects$ **then**
return false
else
return *findSecondSource(dg, creator(dr), checkedSubjects)*
endif; endif; endfor
return false
endif

2.3.1 Die Hauptfunktion `cascadingRevocation`

Ist kaskadierender Widerruf gewünscht, wird die zentrale Methode `cascadingRevocation` von einer der Methoden zum einfachen Widerruf eingebunden. Diese Methode enthält grundsätzlich all jene Funktionalität, die zur Behandlung eines Delegationsschritts erforderlich ist. Sie erwartet zumindest folgende zwei Parameter:

(1) Das zu widerrufende Delegable: Bezüglich der Definition der in Frage kommenden Typen von Delegables sei auf Abschnitt 2.1 verwiesen; (2) die Delegationsrolle, aus der das Delegable widerrufen werden soll, beziehungsweise aus der ein Delegationsempfänger entfernt werden soll.

In einem ersten Schritt werden drei Listen angelegt: Die Liste `revokes` enthält die zu berücksichtigenden Delegables (Rollen können ihrerseits weitere Elemente enthalten), die Liste `delegatees` enthält alle Delegationsempfänger, die auf Weiterdelegation überprüft werden müssen. Anschließend wird überprüft, ob einer dieser Delegationsempfänger eines der Delegables weiterdelegiert hat. Die Listen `handledSubjects` und `second Sources` dienen der Zwischenspeicherung bereits gefundener Subjekte bzw. Sekundärquellen.

Ein kritischer Punkt ist jedoch, dass ein Delegable nur widerrufen werden darf, wenn dafür keine valide Sekundärquelle existiert. Diese Suche übernimmt die Methode `findSecond Source` (siehe Algorithmus *findSecondSource*), die für das Tupel (Delegable, Subjekt) sowohl die regulär zugewiesenen, als auch die per Delegation empfangenen Objekte eines Subjekts findet. Existiert eine solche Sekundärquelle, wird das Tupel nicht weiter behandelt. Wird allerdings keine solche Sekundärquelle ausfindig gemacht, muss die Delegation rückgängig gemacht werden. Dies geschieht, indem die entsprechende Methode (`role RoleDelegationRevoke` beziehungsweise `taskRoleDelegationRevoke`) mit den neuen Parametern des Delegables und der Delegationsrolle aufgerufen wird. Um unnötige Redundanzen zu vermeiden, werden entlang der Ausführung gewonnene Erkenntnisse über Schleifeniterationen hinweg zwischengespeichert. Gerade im Fall wechselseitig rekursiver Methoden kann die Laufzeit dadurch deutlich reduziert werden. Zudem wird die Liste der bereits untersuchten Subjekte gespeichert, um eine mehrfache Untersuchung zu verhindern.

2.3.2 Suche nach Sekundärquellen: `findSecondSource`

Die Methode `findSecondSource` sucht, wie bereits im vorangegangenen Abschnitt erwähnt, für ein konkretes Subjekt die Sekundärquellen eines bestimmten Delegables. Folgende Sekundärquellen kommen in Frage:

- Die mittels regulärer Rollen zugewiesenen Elemente. Diese zu finden ist vergleichsweise einfach und in die Methode `getRegularPermissions` ausgelagert.
- Ist das Delegable eine Delegationsrolle und das zu überprüfende Subjekt deren Delegator, handelt es sich um eine gültige Sekundärquelle.
- Die Ermittlung der ebenfalls als Sekundärquelle in Betracht kommenden empfangenen Delegables ist hingegen interessanter: Zuerst werden die empfangenen Delegationsrollen des Subjekts gesammelt und mittels der Funktionen `hasJuniorRole` beziehungsweise `ownsTask` daraufhin überprüft, ob das gesuchte Delegable enthalten ist. Ist dies bei keiner der Delegationsrollen der Fall, war die Suche erfolglos. Ist das Ergebnis jedoch positiv, gilt es auszuschließen, dass die in der Delegationsrolle enthaltenen Delegables auf eine andere Delegationsrolle zurückgehen, die bereits widerrufen wurde oder noch zu widerrufen ist – Delegationen also letztendlich "aufgrund ihrer selbst" legitimiert werden.

Die Rückgabe der Methode ist ein einfacher Wahrheitswert (engl.: *boolean*), der angibt, ob eine valide Sekundärquelle für das angegebene Delegable existiert oder nicht.

3 Integration in BusinessActivities

Das BusinessActivities Framework ist im Rahmen der *BusinessActivity LRE* (von engl.: Library and Runtime Engine) implementiert (für weitere Details siehe auch [SM11]). Der Aufbau ist modular gehalten, Erweiterungen wie Pflichten (engl.: *duties*) oder Delegationen können dadurch unkompliziert bei Bedarf eingebunden werden (siehe z.B.: [SS11a, SS11b, SWS13, SWSB12]). Auch die Implementierung zum Widerruf von Delegationen wurde in einer *Revocation* genannten Erweiterung gekapselt und umfasst neben den (bereits bestehenden) Methoden zum Widerruf einfacher Delegationen auch jene für den kaskadierenden Widerruf.

Eine der im Zuge der Implementierung zu lösenden Herausforderungen war die Suche nach geeigneten Parametern für die Methode `cascadingRevocation`. Dieses Problem trat erst beim kaskadierenden Widerruf von Delegationen auf, da in diesem Fall mehrere Datentypen für einen Parameter in Frage kommen. Statt der Kurznamen werden daher die vollqualifizierten Objektnamen verwendet. Über den Zugriff auf das entsprechende Objekt kann auf weitere seiner Eigenschaften zugegriffen werden, unter anderem auf den entsprechenden Objekttyp.

Zur Verifikation der Erweiterung wurde eine Reihe an Tests definiert, die jeweils eine zu erwartende Ausgangssituation modellieren. Nach Ausführung einer oder mehrerer mit der Delegation bzw. dem Widerruf in Verbindung stehender Operationen wird abschließend der erwartete Endzustand überprüft.

4 Verwandte Arbeiten

Die Überlegungen zum Widerruf von Delegationen bauen in erster Linie auf Arbeiten zur Delegation selbst auf. Es existiert mittlerweile eine Vielzahl unterschiedlicher Delegationsmodelle. Wir beschränken uns hier auf die Auswahl einiger Modelle, die unserem Delegations- und Widerrufsmodell am ähnlichsten sind.

Barka und Sandhu entwerfen in [BS00] das früheste vollständig definierte rollenbasierte Delegationsmodell RBDM (von engl.: Role-based Delegation Model) mit Widerrufsmöglichkeiten. In RBDM kann ein Benutzer komplette Rollen an andere Benutzer delegieren. Ein darauf aufbauender Ansatz ist das *PBDM* (von engl.: Permission-based Delegation Model), das von Zhang et al. in [ZOS03] beschrieben wird. In [HMM10] wurde das Capability-based Delegation Model (CRBAC) präsentiert, das viele Funktionalitäten des PBDM-Modells übernimmt. Der spezielle Fokus des CRBAC-Ansatzes liegt auf domänenübergreifender Delegation von Rollen und Rechten im Rahmen von Capability-Transfers.

Ein Beitrag in Form eines regelbasierten Frameworks für Delegationen und deren Widerruf ist in [ZAC03] zu finden, wo Zhang et al. insbesondere auf den Widerruf von Delegationen eingehen.

Der vorliegende Beitrag unterscheidet sich somit insofern grundlegend von oben genannten Arbeiten, als er umfassend konkrete Problemstellungen erörtert und löst, die sich beim kaskadierenden Widerruf in prozessbasierten Informationssystemen auf Basis von RBAC ergeben können. Die Arbeit trägt mit dem kaskadierenden Widerruf einen wesentlichen Teil zur Komplettierung des bereits vorgestellten Konzepts der Delegation im Rahmen des BusinessActivities-Frameworks bei, da Delegationen ohne ihren Gegenpart wenig Praxisrelevanz haben.

5 Zusammenfassung und Ausblick

In diesem Beitrag diskutierten wir die Problemstellungen, die sich beim kaskadierenden Widerruf von Delegationen in prozessbasierten Informationssystemen ergeben. Der kaskadierende Widerruf ermöglicht die Aufhebung von Delegationen über mehrere Stufen hinweg und erfordert die Durchführung einer Reihe von Schritten, um die Konsistenz des gesamten Zugriffskontrollmodells nicht zu gefährden. Wir stellten zunächst einen generischen Algorithmus zum systematischen Widderruf von Delegationen vor, der den kaskadierenden Widerruf in beliebigen prozessbasierten Informationssystemen adressiert. Weiterhin präsentierten wir eine konkrete prototypische Implementierung unseres Algorithmus, die im Rahmen der Erweiterung einer bestehenden Softwareplattform erfolgte.

Aufbauend auf unserem Ansatz ergeben sich einige konkrete Ansätze für weiterführende Arbeiten: Momentan ist nur der Widerruf einzelner Delegables vorgesehen. Gerade wenn eine Delegationsrolle als Sammlung von Elementen verwendet wird, die in Zusammenhang mit einem bestimmten Anwendungsfall stehen, ist es naheliegend, diese auch als Gesamtes widerrufen zu können. Zudem wäre interessant, gezielt einzelnen Subjekten Delegables wieder entziehen zu können, die diese per Mehrfach-Delegation erhalten haben, wenn sich beispielsweise Zuständigkeiten innerhalb eines Unternehmens ändern. Diese Option ist momentan nur im Rahmen des einfachen Widerrufs, also jeweils durch den unmittelbar Delegierenden, vorgesehen.

Literatur

[BS00] Ezedin Barka und Ravi Sandhu. Framework for Role-Based Delegation Models. In *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC)*, December 2000.

[CK08] Jason Crampton und Hemanth Khambhammettu. On Delegation and Workflow Execution Models. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC)*, March 2008.

[FKC07] David F. Ferraiolo, D. Richard Kuhn und Ramaswamy Chandramouli. *Role-Based Access Control*. Artech House, second. Auflage, 2007.

[GMPT01] Christos K. Georgiadis, Ioannis Mavridis, George Pangalos und Roshan K. Thomas. Flexible Team-Based Access Control Using Contexts. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT)*, May 2001.

[HJPW01] A. Hagstrom, S. Jajodia, F.P. Presicce und D. Wijesekera. Revocations - a Classification. In *Proc. of the 14th IEEE Computer Security Foundations Workshop*, 2001.

[HMM10] Koji Hasebe, Mitsuhiro Mabuchi und Akira Matsushita. Capability-Based Delegation Model in RBAC. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies (SACMAT)*, June 2010.

[JB06] James B. D. Joshi und Elisa Bertino. 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)*, June 2006.

[OP03] Sejong Oh und Seog Park. Task-role-based access control model. *Information Systems*, 28(6), 2003.

[SCFY96] Ravi Sandhu, Edward Coyne, Hal Feinstein und Charles Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2), 1996.

[SM02] Andreas Schaad und Jonathan D. Moffett. Delegation of Obligations. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY)*, June 2002.

[SM11] Mark Strembeck und Jan Mendling. Modeling Process-related RBAC Models with Extended UML Activity Models. *Information and Software Technology*, 53(5), 2011.

[SS11a] Sigrid Schefer und Mark Strembeck. Modeling Process-Related Duties with Extended UML Activity and Interaction Diagrams. In *Electronic Communications of the EASST, Vol. 37*, March 2011.

[SS11b] Sigrid Schefer und Mark Strembeck. Modeling Support for Delegating Roles, Tasks, and Duties in a Process-Related RBAC Context. In *Proc. of the International Workshop on Information Systems Security Engineering (WISSE)*, 2011.

[Str10] Mark Strembeck. Scenario-Driven Role Engineering. *IEEE Security & Privacy*, 8(1), 2010.

[SW08] Qinghua Shang und Xingang Wang. Constraints for Permission-Based Delegations. In *Proceedings of the 8th IEEE International Conference on Computer and Information Technology Workshops (CITWORKSHOPS)*, 2008.

[SWS13] Sigrid Schefer-Wenzl und Mark Strembeck. Modeling Context-Aware RBAC Models for Mobile Business Processes. *International Journal of Wireless and Mobile Computing*, 6(5):448–462, 2013.

[SWSB12] Sigrid Schefer-Wenzl, Mark Strembeck und Anne Baumgrass. An Approach for Consistent Delegation in Process-Aware Information Systems. In *Proc. of the 15th International Conference on Business Information Systems (BIS)*, Jgg. 117, 5 2012.

[ZAC03] Longhua Zhang, Gail-Joon Ahn und Bei-Tseng Chu. A rule-based framework for role-based delegation and revocation. *ACM Trans. Inf. Syst. Secur.*, 6(3):404–441, August 2003.

[ZOS03] Xinwen Zhang, Sejong Oh und Ravi Sandhu. PBDM: a flexible delegation model in RBAC. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, SACMAT '03, Seiten 149–157, New York, NY, USA, 2003. ACM.