# Dezentrale Rollenbasierte Autorisierung und Administrierung

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
der Universität Dortmund
am Fachbereich Informatik

eingereicht von

**Mario Lischka**

Dortmund
2006

Tag der mündlichen Prüfung: 8. Mai 2006
Dekan: Prof. Dr. Bernhard Steffen
Gutachter: Prof. Dr. Horst F. Wedde
Prof. Dr. Joachim Biskup

# Zusammenfassung

Gegenstand dieser Arbeit ist die rollenbasierte Zugriffskontrolle und deren Administrierung, bei denen ein dezentraler Ansatz zu Grunde gelegt wird. Dies wird durch die Einführung so genannter Autorisierungssphären erreicht, die es ermöglichen, hybride Zugriffsstrategien für bestimmte Teile einer Organisation dezentral zu spezifizieren und dabei die lokalen Erfordernisse zu berücksichtigen. Dabei besteht auch die Möglichkeit, die Aspekte einer verteilten Umgebung bei der Modellierung der Zugriffsprivilegien miteinzubeziehen.
Im Gegensatz zu anderen rollenbasierten Zugriffssystemen wird eine Trennung zwischen der eigentlichen Rollenhierarchie und dem Aufbau einer Organisation in Bereiche, Abteilungen und Arbeitsgruppen vorgenommen. Diese werden als Organisationseinheiten modelliert und gemäß den Gegebenheiten der Organisation hierarchisch strukturiert. Dies ermöglicht eine Vereinfachung der Rollenhierarchie, da in dieser Arbeit die Zugriffsprivilegien an Rollen in einer Organisationseinheit vergeben werden.

Eine völlig autonome Spezifizierung von Zugriffsprivilegien ist aus organisatorischer Sicht nicht sinnvoll. Daher sind die Autorisierungssphären gemäß den Gegebenheiten der Organisation hierarisch geordnet. Hierdurch ist es möglich, ein Vererbungsprinzip einzuführen, dem gemäß in einer übergeordneten Autorisierungssphäre festgelegt werden kann, welche der eigenen Zugriffsstrategien auch in bestimmten untergeordneten Autorisierungssphären gelten sollen. Sofern die Organisationsstruktur kein reines Einliniensystem ist, kann es aufgrund der hybriden Zugriffsprivilegien in Autorisierungssphären, die zwei direkte Vorgänger haben, zu Vererbungskonflikten kommen. Diese zeigen sich dadurch, dass zwei gegensätzliche Zugriffsprivilegien an diese Autorisierungssphäre vererbt werden. Sobald diese Vererbungskonflikte erkannt werden, können sie mittels verschiedener vorgestellter Lösungsstrategien aufgehoben werden.

In modernen Organisationen ist ein Zugriff auf Daten einer anderen Abteilung unerlässlich. Dabei müssen sowohl die Zugriffsstrategien beachtet werden, die den Arbeitsablauf für die Rolle festlegen, als auch die Zugriffsstrategien, die zur Sicherung der Daten spezifiziert werden. Im vorliegenden dezentralen Ansatz bedeutet dies, dass zwei Autorisierungssphären bei einer entsprechenden Zugriffsanfrage involviert sind. Wenn diese nicht zum selben Ergebnis kommen, liegt ein Koordinierungskonflikt vor. Dieser muss von einer übergeordneten koordinierenden Autorisierungssphäre durch entsprechende Regeln aufgelöst werden.

Zusätzlich zur Spezifizierung der Zugriffsstrategien sowie deren Vererbung und Koordinierung wird auch die eigentliche Umsetzung des Vererbungsprinzips und der Koordinierung von Zugriffen mittels fest vorgegebener Regeln in einer Datalog–Sprache modelliert. Durch die durchgehende Verwendung einer Datalog–Sprache bei der Auswertung einer Zugriffsanfrage kann nachgewiesen werden, dass die in dieser Arbeit fest vorgegebenen Regeln, das Vererbungsprinzip und die Koordinierung der Zugriffsanfragen korrekt umsetzen.

Neben den Zugriffsregeln müssen auch Rollen, Organisationseinheiten, Autorisierungssphären etc. und die entsprechenden Hierarchien dezentral administriert werden. Dabei ist nicht ein einzelner Administrator zuständig, sondern die Mitglieder eines Autorisierungsteams sind gemeinschaftlich verantwortlich.Diese werden hierzu in einen administrativen Prozess eingebunden, der in zwei Schritte unterteilt ist. Zunächst wird von einem einzelnen Benutzer ein Administrierungsvorschlag initiiert, der dann auf unterschiedliche Weise vom Autorisierungsteam bestätigt werden muss. Durch zusätzliche Administrierungsregeln können die Rechte zur Initiierung und auch die Beteiligung an der Bestätigung modelliert werden.

Die eigentlichen Abläufe bei der Bestätigung lassen sich formal nicht mittels Datalog spezifizieren. Daher wird eine neue Klasse von Petri–Netzen, sogenannte Administrationsnetze (Admin-Netze), eingeführt. Diese Administrationsnetze erlauben eine detaillierte Spezifizierung der Prozesse, wobei sowohl die beteiligten Administratoren als auch die Vorschläge, inklusive verschiedener Versionen und Teilvorschläge, erfasst werden. Die Interaktion mit den Administratoren wird durch spezielle Transitionen modelliert. Komplexere Abläufe lassen sich mittels Subnetze strukturieren. Hierdurch wird auch die formale Analyse der Administrationsnetze erleichtert.

In Erinnerung an

Hilde und Philipp Wiederschein

## Danksagung

Diese Arbeit entstand am Lehrstuhl Informatik III der Universität Dortmund unter der Betreuung von Prof. Dr. Horst F. Wedde. Ihm gilt mein Dank für die Gelegenheit, diese Promotion zu verwirklichen, für die Unterstützung und die vielen konstruktiven Gespräche.

Ich danke Prof. Dr. Joachim Biskup für sein Engagement als zweiter Gutachter dieser Arbeit. Das von ihm initiierte *Kolleg Sicherheit* bot mir mehrfach die Gelegenheit, die Ansätze dieser Arbeit in einem frühen Stadium zu diskutieren. Seine Anmerkungen und Kommentare waren mir eine wertvolle Hilfe.
Prof. Dr. Heiko Krumm danke ich für die Mitwirkung am Promotionsverfahren und seine Anregungen und Kommentare im *Kolleg Sicherheit*. Ebenso gilt mein Dank Dr. Peter Kemper für die Mitwirkung an diesem Promotionsverfahren.

Während der letzten Phase war Dr. Arnim Wedig ein wertvoller Gesprächspartner, dessen Kritik und Anregungen ich sehr zu schätzen weiß. Bei allen Mitarbeitern und Ehemaligen des Lehrstuhls Informatik III möchte ich mich für die gute und kollegiale Zusammenarbeit bedanken.

Mein tiefster Dank gilt Sonja Hoffmeister, deren Unterstützung und Zuspruch ich mir während der gesamten Zeit, in der diese Arbeit entstanden ist, sicher sein konnte, für die aufmunternden Worte ebenso wie für die Anmerkungen und Kommentare aus dem Blickwinkel einer anderen wissenschaftlichen Disziplin.
Letztlich geht mein innigster Dank an meine Eltern Renate und Aloys, die mir erst ermöglicht haben, diesen Weg einzuschlagen. Sie und meine Schwester Sandra gaben mir den Rückhalt und Ansporn, ohne den eine solche Arbeit nicht entstehen kann.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Kapitel 1

# Motivation für eine dezentrale rollenbasierte Zugriffskontrolle

Die rollenbasierte Zugriffskontrolle bzw. Role Based Access Control (RBAC) wurde von [FK92] eingeführt, um im Gegensatz zur Discretionary Access Control (DAC) und Mandatory Access Control (MAC), eine Trennung von Zugriffsprivilegien und Benutzern zu erreichen. Durch dieses Grundkonzept will man den Bedürfnissen von nicht militärischen[1] Organisationen Rechnung tragen, da es für diese nicht wichtig ist, welche Sicherheitsfreigabe eine Person hat, sondern welche Position diese innerhalb der Organisation ausfüllt. Daher werden alle Zugriffsprivilegien an eine Rolle geknüpft, die einer Stellenbeschreibung in der Organisation entspricht. Der Benutzer erhält die Zugriffsprivilegien über die ihm zugewiesen Rollen. Falls sich seine Position in der Organisation ändert, müssen nicht individuell vergebene Zugriffsprivilegien geändert werden. Stattdessen wird dem Benutzer eine andere Rolle zugewiesen, an die dann die benötigten Zugriffsprivilegien geknüpft sind.

Die rollenbasierte Zugriffskontrolle hat sich in den letzten Jahren als drittes grundlegendes Konzept der Autorisierungskontrolle neben DAC und MAC etabliert [SV01]. Dazu hat auch die Standardisierung der rollenbasierten Zugriffskontrolle beigetragen, wie sie zunächst in [FBK99] vorgeschlagen wurde und in ihrer überarbeiteten Form in [FSG+01] spezifiziert wurde. In [OSM00] wurde gezeigt, wie Mandatory und Discretionary Access Control durch eine rollenbasierte Zugriffskontrolle ersetzt werden können.

Trotz der Erleichterung bei der Zuweisung der Zugriffsprivilegien[2] an die Benutzer, die durch den rollenbasierten Ansatz erreicht wird, ist die Modellierung der Rollen eine große Herausforderung. In einigen Arbeiten wie [JSSB97, JSSS01] sowie [OHL03] wird hierzu eine zentrale Administrierungsstelle vorgesehen. Diese muss dann aber sehr genau Detailkenntnisse über die Stellenspezifikationen und die Arbeitsabläufe in den verschiedenen Abteilungen haben, um die entsprechenden Zugriffsrechte korrekt modellieren zu können. Falls diese zentrale Administrierungsstelle stets bei Änderungswünschen aus den Abteilungen involviert werden muss, kann dadurch in größeren und dynamischen Organisationen unter Umständen schnell ein Engpass entstehen. Aus diesem Grund wird in dieser Arbeit ein vollständig **dezentrales Management** für eine rollenbasierte Zugriffskontrolle entwickelt, bei dem die verschiedenen Abteilungen, Projekte und anderen Einheiten in einer Organisation selbst für die Spezifizierung der Zugriffsrechte verantwortlich sind. Dabei konzentriert sich diese Arbeit auf die Entwicklung einer **Autorisierungssprache**, die diese dezentrale Definition der Zugriffsstrategien ermöglicht und umsetzt. Eine dezentrale Auswertung wird unterstützt und gewisse Annahmen über die Bearbeitung einer Zugriffsanfrage getroffen.

[1] Mit dem DAC Konzept, insbesondere dem Bell–LaPadula Modell, können militärische Kommandostrukturen nachgebildet werden, bei denen es auf die persönliche Sicherheitsfreigabe des Benutzers und die Sicherheitseinstufung des Objektes ankommt.

[2] Eine genaue Begriffsdefinition für "*Zugriffsprivilegien*", "*Objekt*" sowie "*Rolle*" erfolgt im Abschnitt 3.1.

Für die Zuweisung von bestimmten Zugriffsprivilegien sind in diesem Ansatz nicht *einzelne* Administratoren für bestimmte Bereiche verantwortlich, wie es u.a. in [SBC+97, SM99] vorgeschlagen wird oder gar ein zentraler Administrator, wie in [OHL03] oder u.a. [JSSB97] angenommen wird. Dies ist für komplexe Systeme nicht adäquat, da diese Administratoren sowohl die organisatorische Verantwortung, als auch das technische Verständnis haben müssen. Aus diesem Grund sind in dieser Arbeit sogenannte **Autorisierungsteams** gemeinschaftlich für die Administrierung verantwortlich, um sowohl die fachliche Kompetenz als auch die organisatorische Verantwortung mit einzubringen. Dabei sollen insbesondere die Zugriffsprivilegien[3] dezentral definiert werden. Hierzu werden sogenannte **Autorisierungssphären** eingeführt, in denen modular die Zugriffsregeln der einzelnen Bereiche und Abteilungen zusammengefasst werden. Die Grundzüge dieses Ansatzes wurden erstmalig in [WL01] vorgestellt und in [WL03b, WL04a] um die Aspekte der Administrierung erweitert.

Um eine große Flexibilität bei der Gestaltung der Zugriffsregeln zu haben, ist das in dieser Arbeit vorgestellte Autorisierungssystem nicht zwingend als "*offenes System*"[4] oder "geschlossenes System"[5] konzipiert, sondern als **hybrides System**, bei dem sowohl explizite Verbote, als auch explizite Erlaubnisse in den Zugriffsregeln formuliert werden können. Dies ist auch aus organisatorischer Sicht sinnvoll, da sich in kritischen Bereichen, etwa bei den personenbezogenen Daten der Personalabteilung, eher die Vorgehensweise eines "*geschlossenen Systems*" anbietet, während bei anderen Daten, wie allgemeinen Produktinformationen, ein "*offenes System*" adäquater ist.

Die Zugriffsprivilegien werden mittels einer geschichteten Datalog–Sprache [CGT89] spezifiziert[7]. Die dezentrale Spezifizierung führt dazu, dass die Regelsätze auf den einzelnen Ebenen relativ klein sind, welches sowohl eine schnellere Auswertung begünstigt, als auch bei der Administrierung eine leichtere Übersicht erlaubt. Zur Festlegung, welche Subjekte bzw. Rollen auf welche Objekte zugreifen dürfen, werden sogenannte **Feature Prädikate** definiert. Dabei werden insbesondere die Aspekte, die beim Zugriff auf ein Dateisystem von Bedeutung sind, berücksichtigt. Außerdem sollten bei den Zugriffsanfragen in einer dezentralen und verteilten Umgebung auch die besonderen Gegebenheiten, wie Standort des Benutzers und der Daten sowie die verwendeten Kommunikationskanäle für die Autorisierung berücksichtigt werden. Das genaue Vorgehen und die zur Modellierung der Netzwerkumgebung definieren **Kontext Feature Prädikate** und **Kontext Trust Prädikate** wurden in [WL04b] zum ersten Mal vorgestellt und in dieser Arbeit wieder aufgegriffen.

Eine dezentrale Spezifizierung der Zugriffsregeln für die jeweiligen Autorisierungssphären bedeutet jedoch nicht, dass die einzelnen Autorisierungsteams völlig autonom über die Zugriffsstrategien entscheiden können und die hierarchische Struktur der Organisation unberücksichtigt bleibt. Dies würde nach Ansicht der Organisationstheorie auch zur völligen Auflösung der Organisation führen (vgl. u.a. [PDF99, S.223]). Statt dessen soll das dezentrale Management in dem hier vorgestellten Ansatz insbesondere im Bezug auf die Zugriffsstrategien eng an das allgemeine betriebswirtschaftliche Organisationsprinzip der Delegierung[8] angelegt werden. Dabei wird die Entscheidungskompetenz an eine untergeordnete Ebene, d.h. an das Autorisierungsteam, delegiert. Eine Restverantwortung verbleibt (jedoch) beim übergeordneten Autorisierungsteam und der verwalteten Autorisierungssphäre.

Dies ist besonders bei den Zugriffen zwischen verschiedenen Abteilungen wichtig, die im Abschnitt 1.3 detaillierter vorgestellt werden. Im folgenden Abschnitt wird zunächst die besondere Zuordnung von Rollen in Organisationseinheiten erläutert. Anschließend wird im Abschnitt 1.2 die

[3] Die in dieser Arbeit vorgestellte Administrierung umfasst sämtliche Aspekte des Autorisierungssystems.

[4] Bei einem "*offenen System*" werden bei den Zugriffsregeln nur die verbotenen Zugriffe festgelegt. Alle Zugriff, die nicht explizit aufgeführt sind, werden also gewährt[6].

[5] Bei einem "*geschlossen System*" werden bei den Zugriffsregeln nur die erlaubten Zugriffe festgelegt. Alle Zugriffe, die nicht explizit erlaubt werden, sind verboten und werden verweigert[6].

[6] "*offene*" und *"geschlossene Systeme*" sind somit zwei gegensätzliche Ansätze.

[7] Da zur Modellierung der Zugriffsprivilegien auch Negationen von Prädikaten benötigt werden, muss eine geschichtete (stratified) Datalog–Sprache verwendet werden.

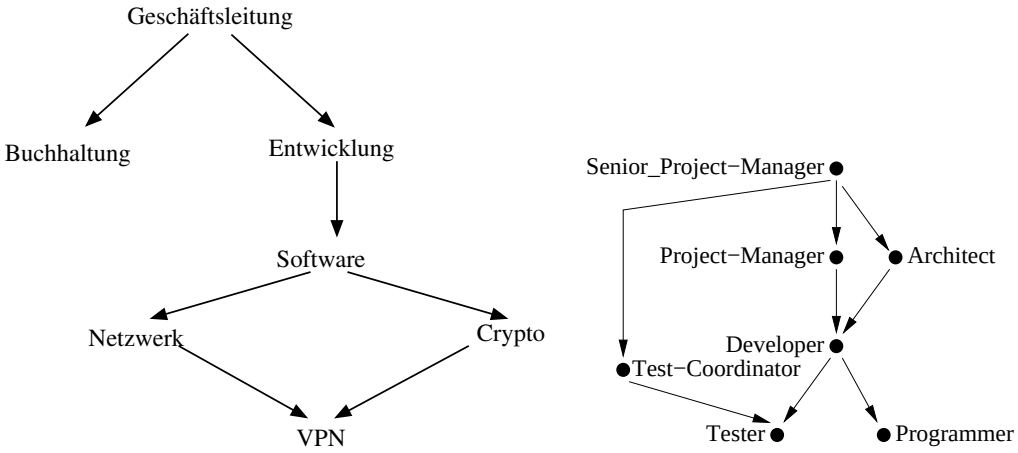
[8] Eine Übersicht hierzu bieten [PDF99, Rob01].

Vererbung von Zugriffsregeln entlang der Organisationshierarchie vorgestellt. Die Möglichkeiten der Administrierung des vorliegenden Ansatzes werden im Abschnitt 1.4 aufgezeigt. Abschließend wird im Abschnitt 1.5 der weitere Aufbau der Arbeit präsentiert.

## 1.1 Rollen und Organisationseinheiten

Bei der Zuweisung von Rollen wird im Standard RBAC [FSG+01] wie auch in anderen Ansätzen stets die Rollenhierarchie und die Organisationsstruktur verknüpft. Entsprechend gäbe es im Standard RBAC einen Project-Manager-Softwareentwicklung, der ein Projekt in der Abteilung Softwareentwicklung leitet, und für ein Projekt der Marketingabteilung einen Project-Manager-Marketing, die nichts miteinander zu tun haben. Dies würde in modernen Organisationen, die verstärkt Projekte als Organisationskonzept einsetzen, dazu führen, dass bei der Erzeugung eines Projektes Crypto die Rollenhierarchie entsprechend um Positionen Project-Manager-Crypto und Programmer-Crypto erweitert werden müsste. Dabei ist davon auszugehen, dass Project-Manager-Softwareentwicklung und Project-Manager-Crypto ebenso wie Programmer-Softwareentwicklung und Programmer-Crypto jeweils ähnlich Aufgaben und damit auch Zugriffsprivilegien haben, die nun neu zugewiesen werden müssen.

Stattdessen wird in dieser Arbeit ein Ansatz verfolgt, bei dem die Rollenhierarchie auf möglichst allgemeine Stellenbezeichnungen wie Project-Manager oder Programmer beschränkt wird, wie sie in Abbildung 1.1(b) zu sehen sind. Gleichzeitig ist der Bereich, die Abteilung oder das Projekt, in dem eine entsprechende Rolle eingenommen wird, von Bedeutung. Dieses Vorgehen ermöglicht später auch eine Vererbung der Zugriffsregeln für einzelne Rollen an untergeordnete Ebenen der Organisation. Dieser Ansatz wird in [WL01] zum ersten Mal vorgestellt.




(a) Ausschnitt: Hierarchie der Organisationseinheiten

(b) Ausschnitt: Rollenhierarchie

Abbildung 1.1: Ausschnitt für den Aufbau einer beispielhaften Organisation

Zur Erläuterung der Vorgehensweise einer Organisation wird in dieser Arbeit beispielhaft ein Softwareunternehmen verwendet. Ein Ausschnitt aus dessen Organisationsstruktur ist in Abbildung 1.1(a) zu sehen[9]. Unterhalb der Geschäftsleitung ist ein Bereich die Entwicklung, von der im Ausschnitt nur die Abteilung für Software aufgeführt ist. Innerhalb dieser Abteilung gibt es gem. Annahme mehrere Projekte, die sich mit der Programmierung in den Gebieten Netzwerk, Crypto

[9]Diese Art der Darstellung dient der Visualisierung der Halbordnung $<_{\mathcal{U}}$, mit der die interne Organisationsstruktur abgebildet wird. Diese $<_{\mathcal{U}}$ Halbordnung wird im Abschnitt 3.2 definiert. Bei einer betriebswirtschaftlichen Sicht wird im Allgemeinen auf die Pfeilspitzen verzichtet. Wenn sie im betriebswirtschaftlichen Kontext verwendet werden, deuten die Pfeilspitzen je nach Richtung Weisungsbefugnis bzw. Beschwerdestelle an (vgl. [SZ99, Kie99]).

und VPN (Virtual Privat Netzwerk) beschäftigen. Diese verschiedenen Elemente einer Organisation werden in dieser Arbeit als **Organisationseinheiten** bezeichnet.

Die Zugriffsrechte werden an Rollen geknüpft, und diese Rollen können einzelnen Benutzern zugewiesen werden. Wie eingangs erwähnt, ist es gerade bei dieser Projektorganisation nicht sinnvoll, einem Benutzer, der im Projekt Crypto beim Softwareentwurf mitwirken soll, die Rolle Developer für die gesamte Organisation zuzuweisen. Damit hätte er sowohl alle Rechte im Projekt Netzwerk, aber auch entsprechenden Zugriff auf die Daten in den Projekten Crypto und VPN.

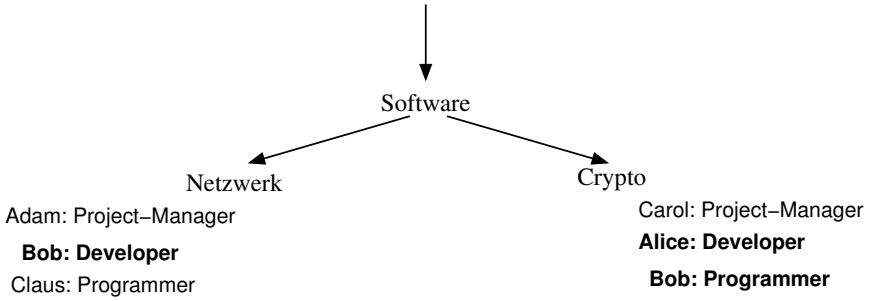



Abbildung 1.2: Beispiel für die Zuordnung von Rollen in Organisationseinheiten

Stattdessen wird einem Benutzer eine **Rolle in einer Organisationseinheit** zugewiesen. Die Zugriffsrechte an den Objekten werden dementsprechend auch an bestimmte Rollen in bestimmten Organisationseinheiten vergeben. In dem Beispiel in Abbildung 1.2 hat die Person Bob die Rolle Developer in der Organisationseinheit Netzwerk und außerdem die Rolle Programmer in der Organisationseinheit Crypto, um dort einen personellen Engpass auszugleichen. Aufgrund der Verknüpfung von Rollen in einer Organisationseinheit kann Bob allerdings keine Vorteile, d.h. mehr Zugriffsrechte, erlangen, indem er versucht, seine Rolle Developer in Crypto auszunutzen.

## 1.2 Vererbung von Zugriffsregeln

Ein wichtiger Aspekt der dezentralen rollenbasierten Zugriffskontrolle ist die Definition der Zugriffsprivilegien auf den verschiedenen Ebenen bzw. Autorisierungssphären der Organisation. Dabei ist die Grundidee, dass auf den oberen Autorisierungssphären nur allgemeine Zugriffsregeln festgelegt werden, sowie grundsätzliche Richtlinien zum Beispiel zum Datenschutz, die in der gesamten Organisation beachtet werden müssen. Dieser Ansatz wurde in [WL01] zum ersten Mal vorgestellt und wird in dieser Arbeit präzisiert.

Im angegebenen Beispiel wird durch die Autorisierungssphäre Software entsprechend die grundsätzlichen Zugriffsstrategien für die einzelnen Rollen festgelegt. Diese Zugriffsstrategien werden an die untergeordneten Autorisierungssphären vererbt und müssen dort beachtet werden. Wobei auch entsprechende Einschränkungen für die Vererbung möglich sind. Die untergeordneten Autorisierungssphären können diese geerbten Zugriffsstrategien ergänzen, um so selbst ihre Arbeitsabläufe im Detail festlegen zu können. Wie dies genau erfolgt, wird durch ein **Vererbungsprinzip** spezifiziert, das in dieser Arbeit entwickelt wird. Durch spezielle Vererbungsregeln kann von jedem Autorisierungsteam festgelegt werden, welche Zugriffsstrategien in bestimmten untergeordneten Autorisierungssphären beachtet werden müssen.
Dabei wird angenommen, dass an den Knoten, an denen die Auswertung einer Zugriffsanfrage für eine bestimmte Autorisierungssphäre vorgenommen wird, durch Replizierung[10] die Vererbungsregeln der übergeordneten Autorisierungssphären bekannt sind.

Problematisch wird die Vererbung bei einem hybriden System von Zugriffsstrategien immer dann, wenn es wie im Fall von VPN mehrere direkt übergeordnete Autorisierungssphären gibt. In

[10] In [Tat01] wurden verschiedene Protokolle zur Replizierung von Dateien im Dragon Slayer System untersucht, die zunächst auch für die Replizierung der Zugriffsstrategien verwendet werden können.

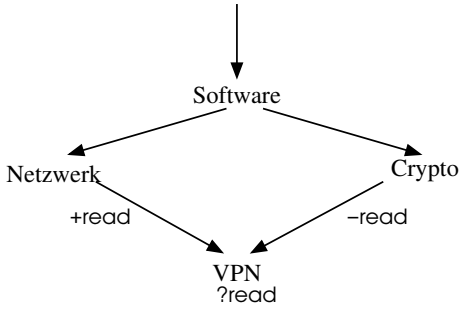



Abbildung 1.3: Beispiel für einen Vererbungskonflikt

der Organisationstheorie spricht man von Mehrlinien–Organisation. Aufgrund des hybriden Ansatzes kann es so zu Konflikten zwischen den vererbten Zugriffsstrategien von Netzwerk auf der einen Seite und Crypto auf der anderen Seite kommen. In diesem Beispiel wird zunächst angenommen, dass Organisationseinheiten und Autorisierungssphären identisch[11] sind. Als Beispiel vererbt, wie in Abbildung 1.3 skizziert, die Autorisierungssphäre Netzwerk eine Zugriffsstrategie, die den Lesezugriff für ein bestimmtes Subjekt in einer Rolle auf ein bestimmtes Objekt erlaubt. Dagegen vererbt die Autorisierungssphäre Crypto eine Zugriffsregel, die für das gleiche Subjekt in der entsprechenden Rolle einen Lesezugriff auf das Objekt verbietet. Diese gegensätzlichen Aussagen bei einer Zugriffsanfrage werden als **Vererbungskonflikt** bezeichnet, der adäquat gelöst werden muss.

Dies erfolgt durch entsprechende Administrierungsprozesse, die im Abschnitt 1.4 noch detaillierter vorgestellt werden. Die formale Definition des Vererbungsprinzips und Vererbungskonfliktes erfolgt im Kapitel 6.

## 1.3 Zugriffe auf Daten anderer Organisationseinheiten

Die Datenobjekte werden in dem vorliegenden Ansatz genau der Organisationseinheit zugeordnet, die diese für ihre Arbeitsabläufe zwingend als Ressourcen benötigt. Darüber hinaus müssen auch Subjekte, die Rollen in anderen Organisationseinheiten aktiviert haben, auf diese Objekte zugreifen können. Gemäß [Seg03] ist bei modernen Methoden der Qualitätssicherung diese Art von Zugriffen noch stärker nötig, um entsprechende Kontrollen und Überprüfungen zu ermöglichen. Dabei stellt ein Subjekt, dessen aktive Rolle in der einen Organisationseinheit liegt, eine Zugriffsanfrage für ein Objekt, das einer anderen Organisationseinheit zugeordnet ist. Solange die Organisationseinheiten zur selben Autorisierungssphäre gehören, werden genau wie bei Zugriffen innerhalb einer Organisationseinheit, die geerbten Zugriffsstrategien der übergeordneten Autorisierungssphären und die Zugriffsstrategien der Autorisierungssphäre ausgewertet. Problematisch wird es bei einem hybriden System, wenn die beiden Organisationseinheiten zu verschiedenen Autorisierungssphären[12] gehören.

Am Beispiel von Stundenzetteln, die zur Abrechnung von Softwareprojekten unter anderem von Crypto genutzt werden, soll zunächst näher betrachtet werden, welche Autorisierungssphären bei dieser Art von Zugriffsanfragen involviert sind. Wie in Abbildung 1.4 skizziert, sind die Objekte vom Typ Stundenzettel bei der Buchhaltung angesiedelt. Zur inhaltlichen Überprüfung müssen aber auch die Manager auf diese Objekte zugreifen können, wenn sie die entsprechende Verantwortung im Projekt haben.
Bei einer Zugriffsanfrage müssen dann die Zugriffsregeln von verschiedenen Autorisierungssphären berücksichtigt werden. Die Zugriffsregeln der Buchhaltung können festlegen, dass ein Subjekt in

[11] Im Abschnitt 3.3 wird in der Vereinbarung 3.1 festgelegt, wie die Beziehung zwischen Organisationseinheiten und Autorisierungssphären genau ist. Diese Vereinbarung wird durch die Annahmen des Beispiels nicht verletzt.

[12] Wie schon im Abschnitt 1.2 erwähnt wird (weiterhin) angenommen, dass jede Organisationseinheit einer Autorisierungssphäre gleichen Namens zugeordnet ist.

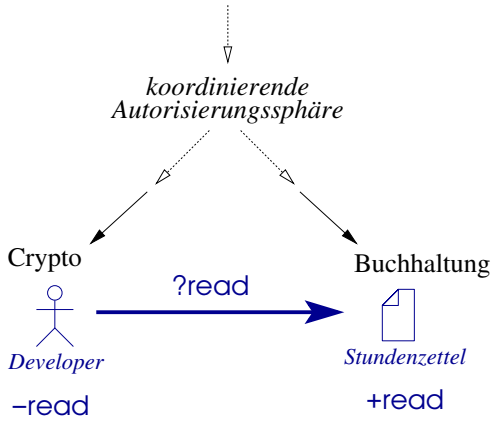



Abbildung 1.4: Beispiel für Zugriffe auf Objekte anderer Organisationseinheiten

einer übergeordneten Rolle auf die Stundenzettel eines Subjektes in einer untergeordneten Rolle zugreifen darf. Dies ist jedoch relativ ungenau, da aufgrund dieser Regel auch ein Developer aus dem Projekt Crypto auf den Stundenzettel eines Programmer des Projektes zugreifen darf. Je nach Größe des Projektes soll aber die Aufgabe der Kontrolle der Stundenzettel beim Project-Manager oder Senior Project-Manager liegen. Einem Developer soll der Zugriff nicht gestattet sein. Dies würde jedoch bedeuten, dass das Autorisierungsteam der Buchhaltung Detailwissen über die Arbeitsabläufe im Software Projekt Crypto haben muss. Im vorliegenden dezentralen Ansatz soll dies vermieden werden. Daher werden entsprechend auch die Zugriffsregeln des Softwareprojektes Crypto ausgewertet, in denen die Details zu diesen Arbeitsabläufen modelliert sind und das Subjekt, das die Zugriffsanfrage durchführt, eine Rolle aktiviert hat.

Somit werden im vorliegenden dezentralen Ansatz bei einer Zugriffsanfrage eines Subjekts, das eine Rolle in einer Organisationseinheit aktiviert hat, auf ein Objekt einer anderen Organisationseinheit stets die Zugriffsstrategien der beiden zuständigen Autorisierungssphären[13] ausgewertet.

Aufgrund der hybriden Zugriffsregeln kann es bei der Auswertung der beiden Zugriffsstrategien zu gegensätzlichen Ergebnissen kommen. Beispielhaft wird nach den Zugriffsregeln von Buchhaltung der Lesezugriff erlaubt, während das Softwareprojekt Crypto den Lesezugriff explizit verbietet, wenn ein Subjekt in der Rolle Developer auf ein Objekt vom Typ Stundenzettel zugreifen will. Dies wird als **Koordinierungskonflikt** aufgefasst, der entsprechend gelöst werden muss[14]. Im Gegensatz zu einem Vererbungskonflikt[15] tritt ein Koordinierungskonflikt auch dann auf, wenn keine intensive Kooperation nötig ist und die internen Arbeitsabläufe der verschiedenen Organisationseinheiten nicht zwingend ausgetauscht werden müssen.

In der Organisationstheorie [PDF99, Rob01] ist die vorgesetzte Stelle für den Informationsfluss zwischen den untergebenen Stellen verantwortlich. Übertragen auf den vorliegenden Ansatz wird dieser Informationsfluss dann durch eine übergeordnete Autorisierungssphäre koordiniert. Diese Autorisierungssphäre ist nicht zwingend die oberste Autorisierungssphäre, sondern wird so gewählt, dass sie gemäß der Organisationshierarchie möglichst nahe an den direkt beteiligten Autorisierungssphären liegt[16]. Im obigen Beispiel wäre dies bei der Organisationsstruktur aus Abbil-

[13] Da Autorisierungssphären auch für mehrere Organisationseinheiten verantwortlich sein können, ist es nicht zwingend das immer zwei Autorisierungssphären involviert sind. In diesem Fall wird die Anfrage, wie zu Beginn des Abschnittes erwähnt, in der einen Autorisierungssphäre ausgewertet.

[14] Dabei wird sich später zeigen, dass es nicht von Bedeutung ist, ob es lokale oder geerbte Zugriffsstrategien sind, die zum Ergebnis einer Autorisierungssphäre führen.

[15] Ein Vererbungskonflikt kann nur dann auftreten, wenn zwei Organisationseinheiten, die zu verschiedenen Autorisierungssphären gehören, eine gemeinsame untergeordnete Organisationseinheit und eine untergeordnete Autorisierungssphäre angelegen, um zu kooperieren und dazu entsprechende Arbeitsabläufe zu entwickeln.

[16] In Einliniensystemen ist dies stets die unterste Autorisierungssphäre verantwortlich, die beiden Autorisierungs-

dung 1.1(a) die Autorisierungssphäre Geschäftsleitung. Im Fall eines Zugriffes von Netzwerk nach Crypto wäre so die Autorisierungssphäre Software, die für die Koordinierung **verantwortliche Autorisierungssphäre**.

Wenn eine Einzelfallentscheidung bei jedem Koordinierungskonflikt durch die übergeordnete Autorisierungssphäre durchgeführt werden müsste, wäre diese schnell mit diesen Anfragen überflutet. Eine Anwendung der Lösungsstrategien, die für Vererbungskonflikte entwickelt werden, auf dieses Problem ist theoretisch auch möglich. Dabei müssen dann Informationen über die internen Arbeitsabläufe und die daraus resultierenden Zugriffsstrategien ausgetauscht werden. Allerdings soll diese Vorgehensweise durch den dezentralen Ansatz gerade vermieden werden. Außerdem würden die (lokalen) Regelsätze der einzelnen Autorisierungssphären größer und damit auch unübersichtlicher werden.
Daher werden generelle Regeln für die Koordinierung durch das Autorisierungsteam der übergeordneten Autorisierungssphäre festgelegt, die als **verantwortliche** und **koordinierende** Autorisierungssphäre formal im Kapitel 7 definiert wird. Im vorgestelltem Ansatz greift diese koordinierende Autorisierungssphäre auf die Entscheidungen der beiden untergeordneten Autorisierungssphären zurück und muss kein Detailwissen über die lokalen Abläufe haben. Die allgemeinen Bedingungen, unter denen ein Koordinierungskonflikt von der verantwortlichen Autorisierungssphäre zu einer Erlaubnis oder einem Verbot aufgelöst wird, können mittels **Entscheidungsregeln** festgelegt werden. Bei diesen Entscheidungsregeln können die Eigenschaften der Objekte und Einschränkungen auf Rollen oder Organisationeneinheiten analog zu den lokalen Zugriffsstrategien mittels Feature Prädikate vorgenommen werden. Die Gegebenheiten der verteilten Umgebung können mittels Kontext Feature Prädikate und Kontext Trust Prädikate ebenfalls bei der Koordinierung berücksichtigt werden. Dadurch kann der Koordinationsbedarf (möglichst) im Vorfeld adäquat gelöst und Konflikte somit vermieden werden. Falls sich die vorhandenen Koordinierungsregeln als nicht (mehr) adäquat erweisen, können sie —wie alle anderen Zugriffsregeln— mittels eines administrativen Prozesses durch andere Regeln ersetzt werden.

## 1.4 Administrierung und Entscheidungsprozesse

Neben der Administrierung der Rollenzuordnung sowie der im Abschnitt 1.1 angesprochenen Hierarchien der Rollen, Organisationseinheiten und Autorisierungssphären[17] sind vor allem die Zugriffsprivilegien von Interesse. Diese können zum einen als lokale Zugriffsstrategien vergeben werden, zum anderen besteht die Möglichkeit, Zugriffsstrategien an untergeordnete Autorisierungssphären zu vererben. Darüber hinaus müssen zur Auflösung von Koordinierungskonflikten in der jeweiligen koordinierenden Autorisierungssphäre entsprechende Strategien spezifiziert werden.

Diese Aufgaben an einzelne Subjekte in dedizierten Administratorrollen zu delegieren wie es in [SBC+97, SM99] vorgeschlagen wird, ist nicht sinnvoll. Zum einen sind die Administrierungsaufgaben mit den "normalen" Rollen verbunden, so wird der Projekt–Manager bei der Zuteilung eines neuen Mitgliedes zum Projekt involviert sein. Zum anderen wird in der Organisationstheorie [PDF99, S. 224f] eine ganze Reihe von Partizipierungsmöglichkeiten wie Anhörung, Meinungsbildung, Abstimmung, Veto-Recht etc. aufgeführt, deren Anwendung einen unterschiedlichen Partizipierungsgrad bei der Delegierung ermöglicht. Die Delegierung der Administrierungsaufgaben an ein einzelnes Subjekt ist somit nur eine von vielen Möglichkeiten.

Daher wird in der vorliegenden Arbeit ein zweistufiges Verfahren zur Administrierung entwickelt, das bereits in [WL03b, WL04a] vorgestellt wurde. Bei diesem Ansatz wird eine Änderung zunächst von bestimmten Subjekten in entsprechenden Rollen initiiert. Diese Vorschläge bedürfen der Zustimmung von weiteren Subjekten in festgelegten Rollen. Somit ist keine einzelne Person für die Administrierung allein verantwortlich.
Während die einzelnen Rechte zur Administrierung und die beteiligten Subjekte mittels Prädikaten

sphären übergeordnet ist. Die verantwortlichen Autorisierungssphären in komplexeren Mehrliniensystemen werden im Abschnitt 7.2 bestimmt.

[17] Weitere zu administrierende Elemente werden noch in den Kapiteln 3 und 4 vorgestellt.

und Regeln dezentral spezifiziert werden können, lassen sich die Prozesse, die bei der Zustimmung eines Vorschlages notwendig sind, auf diese Weise nur informell erfassen.

Obwohl diese Administrierungsprozesse sich auch als eine Art Workflow auffassen lassen, sind bisherige formale Arbeiten in diesem Bereich nicht sehr hilfreich, da inhaltliche Aspekte wie in [Aal98, ABtHK00] nur informell modelliert werden. Daher wurde in [WL03b, WL04a] eine neue Klasse von Petri–Netzen — genauer Prädikat–Transitionsnetze — zur Modellierung der Prozesse bei der Administrierung eingeführt, sogenannte **Administrationsnetze (Admin–Netze)**. Diese ermöglichen eine detaillierte und formale Spezifizierung der Vorgehensweise. Diese Spezifizierung lässt sich formal auf ihre Korrektheit überprüfen. Durch die Erweiterung der Prädikat–Transitionsnetze können Details wie verschiedene beteiligte Subjekte, unterschiedliche Vorschläge und Reaktionen darauf explizit modelliert werden.

## 1.5 Aufbau der Arbeit

Im folgenden Kapitel 2 wird zunächst ein genereller Überblick über andere relevante Veröffentlichungen zu den Themengebieten dieser Arbeit gegeben. Die weitere Arbeit ist dann in drei Teile gegliedert, wobei zunächst die *Rollenbasierte Zugriffskontrolle in verteilten Dateisystemen* in den Kapiteln 3 und 4 dargelegt wird. Im zweiten Teil, der die Kapitel 5 bis 8 umfasst, wird die *dezentrale Spezifizierung von Zugriffsstrategien* vorgestellt. Der dritte Teil beschäftigt sich dann in den Kapiteln 9 bis 11 mit der *Administrierung der modularen rollenbasierten Zugriffskontrolle*.

Im Kapitel 3 werden zunächst die grundlegenden Konzepte der **Rollen** und **Organisationseinheiten** sowie die **Autorisierungssphären** präsentiert. Anschließend wird der grundlegende Teil einer rollenbasierte Zugriffssprache entwickelt, mit dem alle Aspekte der betroffenen Rollen und Objekte sowie deren Eigenschaften zur Zugriffskontrolle herangezogen werden können. Diese Eigenschaften können später bei der Auflösung der Vererbungskonflikte als Unterscheidungsmerkmale genutzt werden. Um die Gegebenheiten von verteilten Systemen adäquat bei der Autorisierung berücksichtigen zu können, beschäftigt sich das Kapitel 4 mit einer Erweiterung, die es ermöglich, Aspekte wie den Ort oder die Verbindung zwischen den beteiligten Komponenten, in die Zugriffsstrategien mit einfließen zu lassen.

Im Kapitel 5 wird gezeigt, wie die Zugriffsregeln dezentral definiert werden können. Anschließend wird im Kapitel 6 die **Vererbung von Zugriffsstrategien** an untergeordnete Autorisierungssphären vorgestellt. Da ein Problem bei der Vererbung von hybriden Zugriffsregeln die auftretenden **Vererbungskonflikte** sind, werden allgemeine Lösungsstrategien präsentiert. Der Zugriff zwischen verschiedenen Autorisierungssphären wird im Kapitel 7 eingeführt und es wird dargelegt, wie mögliche **Koordinierungskonflikte** durch die verantwortliche Autorisierungssphäre gelöst werden können. Kapitel 8 beschäftigt sich mit der dezentralen Auswertung einer Zugriffsanfrage sowie der Umwandlung der Zugriffsregeln in ein geschichtetes Datalog–Programm.

Die Verantwortlichkeiten für die kooperative Definition von Zugriffsregeln können mit Hilfe von Administrationsregeln modelliert werden, die im Kapitel 9 vorgestellt werden. Da die einzelnen administrativen Prozesse mit Hilfe von Regeln nicht explizit modelliert werden können, werden im Kapitel 10 **Administrationsnetze** definiert, die dies im Detail ermöglichen. Die erweiterten Anwendungsmöglichkeiten von Administrierungsnetzen im Bereich der Modellierung von Verhandlungsabläufen werden im Kapitel 11 präsentiert.

Im Kapitel 12 wird diese Arbeit mit einem Fazit abgeschlossen, sowie ein Ausblick auf die weiteren Möglichkeiten vorgenommen, die sich durch diese Arbeit ergeben.

Die oben vorgestellte Reihenfolge der Kapitel 3 bis 8 ermöglicht eine Buttom-Up Definition der Datalog Prädikate, sodass stets nur auf bekannte Prädikate zurückgegriffen wird. Die dezentrale Auswertung beginnt dagegen mit dem Prädikat `query`, das im Abschnitt 8.1 definiert wird und erfolgt dann im Wesentlichen in umgekehrter Reihenfolge entlang der Schichtung der Prädikate, die in Abbildung 8.1 auf Seite 120 dargestellt ist.

Im Verlauf des ersten und zweiten Teils der Arbeit werden **Beispielregeln** angegeben, um die Möglichkeiten der Autorisierungsteams für die lokalen Zugriffsstrategien, Vererbungsregeln und Entscheidungsregeln aufzuzeigen. Die **Regeln**, die für das Erben von Zugriffsregeln und die Koordinierung angegeben, werden sind Vorlagen, die automatisch in die jeweiligen Autorisierungssphären übernommen werden[18].

In Folgenden werden einige Notationen benutzt, die an dieser Stelle bereits eingeführt werden sollen, um das weitere Verständnis zu erleichtern. In logischen Formeln werden Variablennamen grundsätzlich klein geschrieben und sind stets kursiv gedruckt (z.B. $a$, $x$, $as$, $x_2$). Konstantennamen beginnen immer mit einem Großbuchstaben oder einer Zahl und werden außerhalb der Formeln auch in Sans Serif geschrieben (z.B. Manager, Developer oder Entwicklung).
Des Weiteren wird im Text an einigen Stellen auf *'das System'* verwiesen. Dies soll dann stets als Kurzform für das in dieser Arbeit konzipierte Autorisierungs- und Administrierungssystem verstanden werden.

[18]Dies geschieht zumeist beim Anlegen einer Autorisierungssphäre oder bei einer Veränderung der Beziehung zwischen den Autorisierungssphären, wobei weitere Details bei den jeweiligen Prädikaten erläutert werden.

# Kapitel 2

# Bisherige Arbeiten

Um eine dezentrale rollenbasierte Autorisierung und die dazugehörige Administrierung zu entwickeln, wird auf verschiedene Arbeiten anderer Autoren zurückgegriffen. Dieses Kapitel gibt einen Überblick über die verschiedenen Themengebiete und ist wie folgt aufgebaut.

Zunächst werden im Abschnitt 2.1 die grundlegenden Arbeiten zur rollenbasierten Zugriffskontrolle vorgestellt sowie deren Standardisierung. Die Arbeiten zur Administrierung von RBAC werden im Abschnitt 2.2 besprochen. Da in der vorliegenden Arbeit zur Spezifizierung der Zugriffsstrategien eine Datalog–Sprache definiert wird, werden die wichtigsten Arbeiten zu Datalog im Abschnitt 2.3 angesprochen. Die Zustimmung zur Administrierung wird mit einer erweiterten Klasse von Petri–Netzen modelliert. Im Abschnitt 2.4 werden die entsprechenden grundlegenden Arbeiten diskutiert. Aus den verschiedenen Strukturierungsformen einer Organisation, die in Abschnitt 2.5 vorgestellt werden, lassen sich entsprechende Konsequenzen für die Struktur der Autorisierungssphären ableiten.

## 2.1 Rollenbasierte Zugriffskontrolle

In einer der ersten Arbeiten zur rollenbasierten Zugriffskontrolle (RBAC) von [FK92] werden sehr stark die Unterschiede zum *Discretionary Access Control (DAC)* und *Mandatory Access Control (MAC)* herausgestellt[1], sowie die Notwendigkeit eines neuen Ansatzes motiviert. Beim rollenbasierten Ansatz können jedem Subjekt bestimmte Rollen zugeordnet werden und ein Subjekt kann verschiedene Rollen aktivieren. Dabei muss mindestens eine Rolle aktiv sein, um sogenannte *Transaktionen* auszuführen.
In späteren Arbeiten entsprechen Transaktionen den Zugriffsrechten. Diese Transaktionen können für Subjekte mit bestimmten aktiven Rollen an Objekte geknüpft werden. Eine genaue Definition eines Subjektes wird nicht gegeben. Allerdings wird eine Rollenhierarchie eingeführt, wenn auch nicht formal definiert. Entlang dieser Rollenhierarchie kann die Möglichkeit "Transaktionen durchzuführen" weitergeben werden. Als ein besonders wichtiger Aspekt, der durch die rollenbasierte Zugriffskontrolle unterstützt werden soll, wird die von [CW87] vorgestellte "*Separation of Duty*" genannt.

Neben einer ausführlicheren Motivation von RBAC, die die anlaufenden Maßnahmen zu Standardisierung unterstützen sollte, werden in [FCK95] die grundlegenden Elemente von RBAC definiert. Dazu gehören Benutzer (User), denen Rollen zugeordnet werden können. Diesen Rollen werden bestimmte Operationen zugeordnet, die wiederum mit Objekten verknüpft sind. Ein Subjekt repräsentiert in [FCK95] einen aktiven Benutzer. Dieses kann nur Rollen aktivieren, die ihm bzw. dem Benutzer, der durch das Subjekt repräsentiert wird, zugeordnet wurden. Dabei ist es möglich, verschiedene Einschränkungen bezüglich der Aktivierung von Rollen zu spezifizieren. Neben der "*statischen Separation of Duty*", bei der bestimmte Kombinationen von Rollenzuordnungen nicht

[1] Eine Übersicht zu allen drei Ansätzen DAC, MAC und RBAC findet sich in dem Tutorium von [SV01].

zugelassen werden, wird auch eine "*operationale Separation of Duty*" eingeführt, bei der bestimmte Operationen zu einer Geschäftsfunktion (business function) zusammengefasst werden und kein einzelner Benutzer alle Operationen ausführen darf.

In [NO96] wird die Beziehung zwischen Rollen anhand eines Rollengraphen ausführlich diskutiert. Bei diesem Ansatz ist eine Rolle genau dann einer anderen Rolle untergeordnet, wenn sie eine Teilmenge der Zugriffsprivilegien hat. Aufbauend darauf wird in [NO99] detaillierter auf mögliche Konflikte zwischen den Rollen und zwischen den Privilegien eingegangen.

In [BVS00, BVS02] wird ein algebraischer Ansatz zur Komposition von verschiedenen Zugriffsstrategien vorgestellt. Dabei können diese Zugriffsstrategien von beliebigen Stellen auch durch heterogene "Systeme" spezifiziert werden. Dieser Ansatz greift bei der Komposition auf entsprechende Interfaces der heterogenen Systeme zurück. Dabei wird immer von einem Subjekt $s$ ausgegangen, dass mittels einer Methode $a$ auf ein Objekt $o$ zugreifen will, wobei kein expliziter rollenbasierter Ansatz zu Grunde gelegt wird[2]. Allerdings werden keine hybriden Zugriffsstrategien zugelassen.

In [WJ01, WJ02, WJ03] wird ebenfalls eine Algebra für Zugriffsstrategien vorgestellt. Die verschiedenen Zugriffsstrategien umfassen dann unterschiedliche Zugriffsprivilegien, wobei diese nicht dezentral definiert sind, sondern gemäß dem dort vorgestellten Beispiel aus dem Bankenbereich, unterschiedliche Situationen widerspiegeln. Kernpunkt des Ansatzes sind sogenannte "*states*" die eine Abbildung von logischen Aussagen ("*propositions*") auf die Tripel von Subjekt, Objekt und Zugriffsmethode $(s, o, \pm)$ darstellen, für die die logischen Aussagen gültig sind. Von einem anfänglichem *state*, der in den Beispielen keine Zugriffsrechte enthält, können durch die Kompositionsoperatoren einem Benutzer weitere Zugriffsrechte hinzugefügt werden.
Diese Algebra unterstützt für alle Zugriffsstrategien eine hybride Definition der Zugriffsprivilegien. Allerdings werden die Operationen nur durchgeführt, wenn die Menge der möglichen Zugriffe dabei konsistent bleibt. Dies bedeutet, dass bei widersprüchlichen Zugriffsrechten – im Sprachgebrauch der vorliegenden Arbeit Konflikte genannt – die Operation nicht (vollständig) durchgeführt wird. Dadurch werden Konflikte auf eine fest vorgegebene Weise aufgelöst.

Eine Erweiterung zum RBAC–Ansatz, der nicht nur zeitliche Einschränkungen bezüglich der Rollenaktivierung und Deaktivierung erlaubt, sondern auch komplexere Bedingungen zwischen diesen zulässt, wird in [BBF00, BBF01] präsentiert. Eine ähnliche Vorgehensweise wird bereits in [Tho97] vorgestellt, wobei dieser Ansatz stark von dem Anwendungsbeispiel eines Behandlungsteams in einem Krankenhaus geleitet ist.
Diese Aspekte wurden jedoch bei der Standardisierung von RBAC nicht berücksichtigt. Weitere Möglichkeiten der rollenbasierten Zugriffskontrolle sind vor allem im *ACM Workshop on Role-Based Access Control (RBAC)* und im daraus hervorgegangenen *ACM Symposium on Access Control Models and Technologies (SACMAT)* zu finden.

### 2.1.1 Standard RBAC

Ein erster Schritt in Richtung Standardisierung der rollenbasierten Zugriffskontrolle ist der Artikel von [SCFY96], der eine "family of reference models" vorstellt. Dabei werden RBAC Systeme mit bestimmten Eigenschaften in verschiedene Kategorien zusammengefasst. Die Eigenschaften, nach denen unterschieden wird, sind zum einen das Vorhandensein einer Rollenhierarchie, zum anderen die Möglichkeit *Constraints* bezüglich der Zuordnung von Rollen zu definieren. Aus den Kombinationen dieser Eigenschaften werden dann vier verschiedene Kategorien gebildet. Diese werden in späteren Veröffentlichungen als "RBAC96" Modelle bezeichnet. Jede dieser Kategorien wird in [San96] die entsprechende Administrierung gegenübergestellt, sodass eine entsprechende Hierarchie von Administrierungskategorien gebildet wird.

[2]Inwieweit die einzelnen Zugriffsstrategien auf einen rollenbasierten Ansatz beruhen, bleibt offen.

In [San96] kann ein User immer die zugeordnete Rolle und alle untergeordneten Rollen aktivieren. Diese Möglichkeit wird in [San98] über die zusätzliche "*Activation*" Hierarchie eingeschränkt, diese ergänzt die Vererbung der Privilegien entlang der Rollenhierarchie ("inheritence-hierarchy").

Eine NIST Standardisierung der rollenbasierten Zugriffskontrolle wird zum ersten Mal von [FBK99] vorgeschlagen. Dabei werden die statische Zuordnung der Rollen auf der einen Seiten und die dynamische Aktivierung der Rollen auf der anderen Seite als Unterscheidungsmerkmale festgelegt. Die verschiedenen Möglichkeiten werden anhand einer Referenzimplementierung für einen Web-Service aufgezeigt.

Demgegenüber werden von [SFK00] vier Stufen von RBAC Modellen vorgeschlagen. In der untersten Stufe "Flat RBAC" können Benutzer Rechte durch Rollen erhalten, wobei der Benutzer mehrere Rollen gleichzeitig nutzen kann. Die nächste Stufe, die "Hierarchical RBAC", muss darüber hinaus eine Rollenhierarchie als Halbordnung über alle Rollen unterstützen. Dabei kann diese Rollenhierarchie entweder eingeschränkt oder uneingeschränkt sein. Die darauf aufbauende Stufe "Constrained RBAC" muss entweder statische oder dynamische "Separation of Duty" unterstützen, die genaue Implementierung wird offen gelassen . Die vierte Stufe "Symmetric RBAC" beinhaltet die Anforderungen von allen vorangegangen Stufen insbesondere also eine Rollenhierarchie und Constraints. Zusätzlich muss eine Möglichkeit bereitgestellt werden, die Zuordnung von Zugriffsrechten und Rollen zu analysieren. Wobei auch Rechte, die über die Rollenhierarchie weitergegeben werden, entsprechend in einem Review mit einzubeziehen sind.

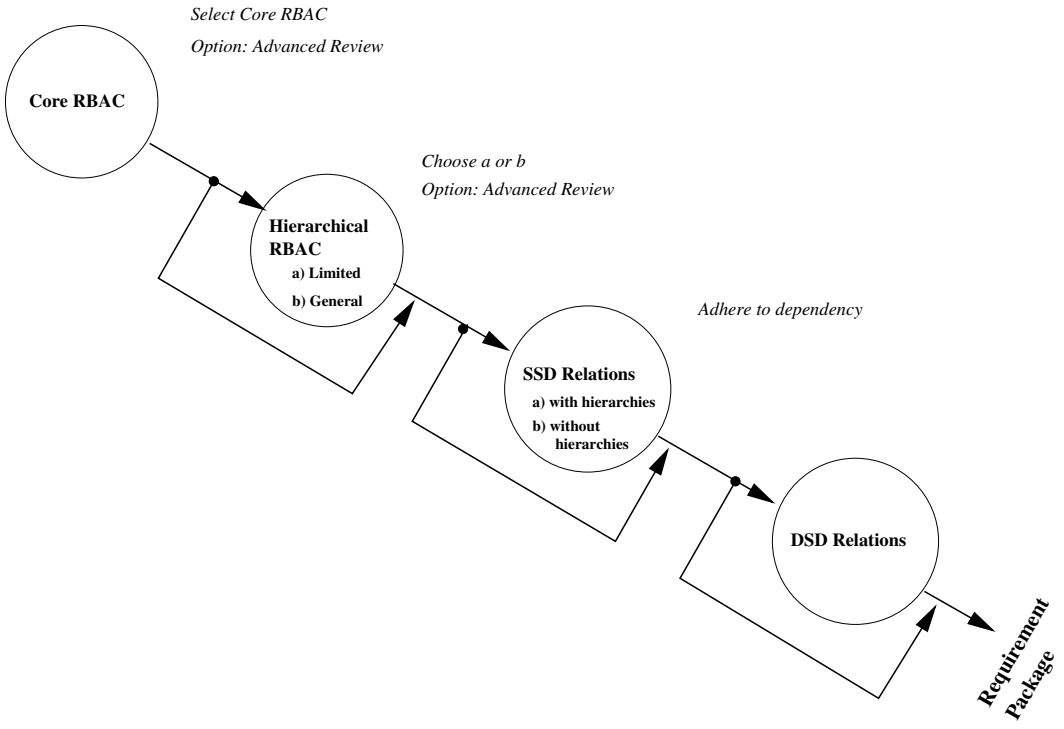



Abbildung 2.1: Hierarchie der RBAC Modelle gemäß Vorschlag zum NIST Standard von [FSG+01]

Diese Einstufung wird in [FSG+01] verändert und ist in Abbildung 2.1 dargestellt. So ist die Möglichkeit eines Reviews bereits optional in der ersten Stufe enthalten. Die darauf aufbauenden Stufen können die Möglichkeiten der vorangegangenen Stufe enthalten, sind aber nicht, wie in [SFK00], dazu verpflichtet. Die zweite Stufe bleibt "Hierarchical RBAC" entweder mit einer eingeschränkten oder allgemeinen Rollenhierarchie. Die dritte Stufe beinhaltet nun lediglich die "Static

Separation of Duty", während die vierte Stufe nun die "Dynamic Separation of Duty" enthält. Die Abbildung 2.1 stellt die Schritte zur Zusammenstellung der Anforderungen und Funktionalitäten eines beliebigen RBAC Systems gemäß NIST–Standard dar.

Neben dem aus [FSG+01] hervorgegangenen offiziellen Standard [ANS04], gibt das Buch von [FKC03] eine Übersicht über den gesamten Standardisierungsansatz.

### 2.1.2 Spezifizierung von Zugriffsstrategien

In den bisher vorgestellten Arbeiten wird an Beispielen erläutert, welche Rolle bestimmte Operationen auf ein konkretes Objekt ausführen darf, allerdings wird nicht genau erläutert wie dies in einer konkreten Anwendung realisiert werden soll.
Für RBAC96 [SCFY96] wird in [AS00] eine Sprache vorgestellt, die eine Spezifizierung der Constraints für die Zugriffskontrolle ("*Role Constraint Language (RCL2000)*") erlaubt. Diese Constraints betrachten allerdings keine Objekttypen oder andere Eigenschaften von Objekten. Es besteht auch keine Möglichkeit, auf andere Constraints Bezug zu nehmen. Zur weiteren Auswertung muss diese Sprache in eine eingeschränkte Prädikatenlogik erster Ordnung umgewandelt werden.

Diese Umwandlung entfällt bei der "Authorization Specification Language", die in [JSS97, JSSB97, JSSS01] diskutiert wird, da es sich um eine Datalog–Sprache[3] handelt. Mit dieser Sprache kann ein "System Security Officer" Regeln für den Zugriff auf Objekte spezifizieren. Hierzu werden zum einen Prädikate definiert, die bestimmte Eigenschaften von Subjekten und Objekten abfragen können[4]. In diesem Ansatz sind Subjekte sowohl Benutzer, Gruppen von Benutzern als auch Rollen, wobei die beiden letzteren jeweils eine eigene Hierarchie aufweisen. Benutzern können bestimmte Rollen zugewiesen werden und Benutzer können Mitglieder in einer Gruppe sein. Zum anderen werden Prädikate definiert, mit denen konkrete Zugriffe formuliert werden können, abhängig davon, ob die Prädikate im Rumpf gültig sind. Mit Hilfe dieser Prädikate können beliebige Zugriffspolitiken (offen, geschlossen oder hybrid) formuliert werden. Konflikte[5], die aufgrund der hybriden Zugriffsregeln entstehen, können entlang der beiden Hierarchien aufgelöst werden.

## 2.2 Administrierung von RBAC

Bei der Administrierung von RBAC geht es darum, die verschiedenen Zuordnungen insbesondere von Benutzer und Rolle sowie Rolle und Zugriffsrechte genauer zu beschreiben. Zeitgleich mit den Bemühungen um eine Standardisierung von RBAC wurden auch Ansätze zu einem einheitlichen Vorgehen bei der Administrierung in [SBC+97, SBM99] vorgeschlagen, die von den Autoren als *"ARBAC97"* bezeichnet werden. Dieses Konzept wird entlang der verschiedenen Beziehungskanten in verschiedene Teile aufgegliedert.
Die Zuweisung von Benutzern zu bestimmten Rollen wird als "User–Role Assignment (URA97)" detaillierter in [SB97] vorgestellt. Entsprechende Implementierungen für Unix [SA98a] und Windows NT [SA98b] wurden realisiert.
Die Zuweisung von Zugriffsprivilegien an bestimmte Rollen wird als "Permission-Role Assignment (PRA97)" in [SP98] detaillierter besprochen. Außerdem wird für die Administrierung der Beziehungen zwischen den Rollen ein "Role-Role-Assignment (RRA97)"entwickelt.
Bei allen diesen Ansätzen wird einer Administratorrolle das Recht zugewiesen, bei bestimmten Rollen, die innerhalb eines sogenannten "Role-Range" liegen, neue Beziehungen hinzuzufügen oder zu entfernen. Dabei kann das Recht der Rollenzuweisung an Benutzer zusätzlich über Rollen eingeschränkt werden, die der Benutzer zuvor haben muss. Das Recht auf Zuweisung von Zugriffsprivilegien kann auf solche Privilegien eingeschränkt werden, die bereits einer anderen Rolle

[3] Weitere Arbeiten zu Datalog werden im Abschnitt 2.3 vorgestellt.
[4] Ausgehend von dieser Vorgehensweise wird im Kapitel 3 eine umfangreiche Liste an Feature Prädikaten vorgestellt.
[5] Im Kontext dieser Arbeit wird dies als Inkonsistenz bezeichnet (vgl. Abschnitte 5.2, 6.2.2 und 7.3.1).

zugeordnet sind.

In [SM99] wird *"ARBAC99"* als eine Erweiterung von ARBAC97 präsentiert, die vor allem die Änderungen durch die Entwicklung des Standard RBAC berücksichtigt (vgl. Abschnitt 2.1.1). Bei all diesen Ansätzen werden die Administrierungsrechte einer einzelnen Rolle zugeordnet. Für komplexe dezentrale Autorisierung ist die gemeinschaftliche Administrierung durch Autorisierungsteams sinnvoller.

Ein graphisches Framework zur Administrierung von RBAC wird in [KMPP04] präsentiert. Mit diesem können auch bestimmte Constraints graphisch dargestellt werden.

Die Möglichkeiten von RBAC in Unternehmen werden in [KSM03] betrachtet. Die Problematik, Rollen in einer bestehenden Unternehmensstruktur zu identifizieren, wird in [KSS03] diskutiert. Dabei werden Data–Mining Ansätze vorgestellt, um bestimmte Zugriffsprivilegien in Rollen zusammenzufassen.

## 2.3 Datalog

Datalog wurde ursprünglich als Verbindung zwischen Datenbank–Abfrage–Sprachen und logischer Programmierung entwickelt. Eine Übersicht hierzu bietet [CGT89] bzw. [CGT90].

In [JSS97, JSSB97] wurde zum ersten Mal eine Datalog–Sprache verwendet, um Regeln für die Autorisierung zu formulieren. Der Vorteil dieser Vorgehensweise liegt darin, dass relativ einfach Bedingungen formuliert werden können, die entweder von bekannten Fakten, wie Rollenzuordnung, Objekttyp etc., oder anderen Bedingungen, wie anderen Regeln, abhängen. Daher wird dieser Ansatz auch in [WL01] aufgegriffen und in [WL03b, WL04a] auf die Administration, sowie in [WL04b] für verteilte Umgebungen übertragen.

Ein Datalog–Programm besteht aus Regeln, der Form $L_0 \leftarrow L_1 \& \ldots \& L_n$. Dabei ist $L_i$ ein Literal der Form $p_i(t_1, \ldots, t_{k_i})$ wobei $p_i$ ein Prädikat Symbol ist und $t_j, 1 \leq j \leq k_i$, Terme darstellen. $L_0$ wird auch als Kopf und $L_1 \ldots L_n$ als Körper oder Rumpf bezeichnet. Ein Term ist entweder eine Konstante oder eine Variable. Fakten sind die Aussagen über die reale Welt und werden auch als *Extensional Database (EDB)* bezeichnet. Die Prädikate, die diese Fakten abfragen, werden als EDB-Prädikate oder kurz "EPreds" bezeichnet. Das Datalog–Programm stellt eine *Intensional Database (IDB)* dar, bei der mit Hilfe von weiteren Prädikaten, sogenannten $IPreds$, zusätzliche Fakten abgeleitet werden können. In einem Datalog–Programm kann gemäß [CGT90, S. 81f] die Menge der Prädikatsnamen $Pred$ in zwei disjunkte Teilmengen $IPred$ und $EPred$ zerlegt werden, folglich gilt $Pred = EPred \cup IPred$ und $EPred \cap IPred = \emptyset$.

**Definition 2.1 (Datalog–Programm)**
Ein Datalog–Programm $P$ ist eine endliche Menge von Hornklauseln, sodass für alle Klauseln $C \in P$ gilt, dass entweder $C \in EDB$ gilt, oder C ist eine Regel, für die gilt:

1. Das Prädikat im Kopf von $C$ ist Element von $IPred$, und

2. alle Variablen im Kopf von $C$ kommen auch im Körper von $C$ vor.

Die Verwendung von Negationen in allgemeinen Datalog–Programmen ist kritisch. Für die Formulierung von bestimmten Bedingungen der Zugriffskontrolle sind Negationen jedoch notwendig. Allgemein wurden verschiedene Erweiterungen für Negationen vorgestellt [vG88, She88, ABW88]. In dieser Arbeit wird ein geschichtetes Datalog–Programm (stratified) verwendet [GL88, vG89, vGRS91]. Bei einem geschichtetem Datalog können alle Prädikate so angeordnet werden, dass stets nur Prädikate auf gleicher oder untergeordneter Schicht verwendet werden und Negationen nur für Prädikate in untergeordneten Schichten auftauchen. Diese Einschränkung muss bei der Spezifizierung der Prädikate berücksichtigt werden und wird im Abschnitt 8.2 noch detailliert betrachtet.

## 2.4 Petri–Netze

Zur Modellierung der Administrationsabläufe wird eine formal abgeschlossene Klasse von Prädikat/Transitionsnetzen (PrT) definiert, die ihrerseits von [GL79, GL81, Gen87] vorgestellt werden[6]. Weitere Arbeiten ([MV86]; [LR94]; [LMP02]) beschäftigen sich mit den verschiedenen Analysemöglichkeiten für PrT–Netze.

Diese PrT–Netze werden um mehrere Techniken erweitert. Zum einen um Inhibitor Kanten, die in [Val83] erwähnt werden. Diese erlauben das Fehlen eines bestimmten Tokens bei der Aktivierung mit einzubeziehen. Zum anderen werden noch Zeitkonstraints zugelassen [GMMP91], die es erlauben den Zeitraum, in dem Transitionen schalten können, einzuschränken.

Der Vorteil von Petri–Netzen liegt in der Möglichkeit einer formalen Analyse insbesondere durch Invarianten. Eine ausführliche Diskussion über verschieden Analyseverfahren findet sich in [Bau96] und [Des98]. In [Sta90] findet sich eine Übersicht über die verschiedenen Klassen von Petri–Netzen und die Invarianten Analyse. Ebenso bietet [GV03] eine aktuelle Übersicht über Petri–Netze und berücksichtigt zusätzlich noch Themen, wie Entfaltung und Invarianten Analyse von Coloured Petri–Netzen. Weitere Analysemöglichkeit sind die Untersuchung von Erreichbarkeitsgraphen und Stubborn Sets. In [CH93] wird eine Erweiterung von Coloured Petri–Netze um Inhibitor Kanten gegeben, die bei der Analyse in komplementäre Stellen mit einer maximalen Anzahl von Token umgewandelt werden.

Aufgrund der Erweiterungen in den Admin–Netzen ist keines der bisher bekannten Tools in der Lage diese direkt zu untersuchen. Daher wurde speziell für Admin–Netze ein Analysetool entwickelt [Den05], das ein kombiniertes Verfahren von T–Invarianten und Erreichbarkeitsanalyse verwendet. Die Berechnung der Invarianten erfolgt nach der *Fourier–Motzkin–Methode* wie sie in [MS82] vorgestellt wurde. Diese ist auch als Algorithmus von Farkas [Far02] in der Literatur bekannt (vgl. u.a. [MV86] [Sta90]). Die Details zu diesem Tool und seinen Möglichkeiten werden im Abschnitt 11.6 gesondert vorgestellt.

Die Steuerung von Workflows mittels Petri–Netzen wird in [Aal98, ABtHK00] vorgestellt und als *"WorkFlow net"* bezeichnet. Allerdings beruhen diese Netze auf einfachen Stellen–Transitions Netzen[8]. Damit kann zwar die Reihenfolge der Abläufe auch bei Verzweigungen und anderen Strukturen dargestellt werden, allerdings müssen alle Inhalte zu diesen Workflows durch die Beschriftungen an den Transitionen ausgedrückt werden. Ebenso können Gründe und Bedingungen für Entscheidungen über ein alternatives Vorgehen nicht modelliert werden.

## 2.5 Grundlegende Strukturen von Organisationen

In der Organisationstheorie wird der Aufbau einer Organisation in zwei grundlegende Kategorien geteilt (vgl. [Kie99, PDF99, SZ99, Rob01]). Die dadurch entstehenden Strukturen zwischen Organisationseinheiten bilden die Weisungs- und Entscheidungsrechte innerhalb einer Organisation ab. Für den vorliegenden Ansatz eines dezentralen Autorisierungssystems bedeutet dies, dass dies auch Autorisierungssphären sind, die entsprechende Zugriffsstrategien an untergeordnete Autorisierungssphären weitergeben können.

### 2.5.1 Einliniensystem

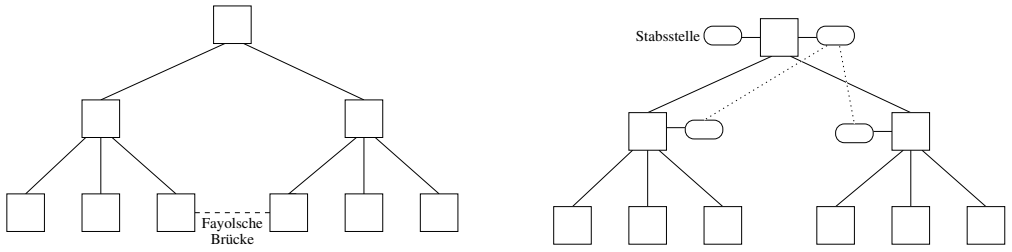
Beim Einliniensystem, das in [Fay16, Fay29] vorgestellt wurde, gibt es eine strikte Hierarchie und damit eine eindeutige Struktur der Verantwortlichkeiten und Anweisungen. Das Problem ist jedoch, dass die übergeordneten Stellen wegen einer fehlenden Spezialisierung auf die Leitungsaufgaben überfordert werden, da es zu Überlastungen und damit zu Informationsverlust kommen

[6] Als eine weitere Klasse von sogenannten "High–Level Petri–Netzen" wurde von [Jen87] *Coloured Petri–Netze* eingeführt[7] und in [Jen96a, Jen96b, Jen96c] detaillierter präsentiert.

[7] Die gleichnamigen Netze von [SW78] werden in diesen Veröffentlichungen nicht erwähnt.

[8] wie sie von Petri vorgestellt wurden [Pet62, Pet66]

kann. Außerdem ist der Anweisungsweg auch zugleich Informationsweg / Dienstweg. Um diese Informationswege abzukürzen, werden sogenannte Fayolsche Brücken eingebaut, über die untergeordnete Stellen direkt kommunizieren können.




(a) Allgemeines Einliniensystem (b) Stabliniensystem mit Stabshierarchie

Abbildung 2.2: Einliniensysteme

#### Stabliniensystem

Um die Probleme der Überforderung zu reduzieren, wurde das sogenannte Stabliniensystem entwickelt, das auf Konzepte im militärischen Bereich zurückgeht. Hier beraten spezialisierte Stabsstellen bei den Entscheidungen und der entsprechenden Kontrolle der Umsetzung bzw. Durchführung. Diese Stabstellen stehen abseits der eigentlichen Hierarchie und haben keine Weisungsberechtigung gegenüber Organisationseinheiten, die der Organisationseinheit, für die sie tätig sind, untergeordnet sind.

Bei einer Variante dieses Systems kann auch die Möglichkeit eingeräumt werden anderen Stabsstellen Weisungen zu geben. Dies wird als Stabshierarchie bezeichnet. Weitere Varianten und Kritiken an diesem System finden sich in [SZ99, S. 275-278].

### 2.5.2 Mehrliniensystem

Bei der in [Tay11] vorgestellten Mehrlinienstruktur sind die übergeordneten Organisationseinheiten auf bestimmte Funktionen der Leitung[9], wie Durchführung, Qualitätskontrolle etc. spezialisiert und können den untergeordneten Organisationseinheiten entsprechende Anweisungen geben. Zwar erleichtert dies die Spezialisierung, jedoch sind die Informationen stark verstreut und es kann zu Konflikten zwischen den verschiedenen Anweisungen kommen, die nur schwer zu lösen sind (vgl. [PDF99, S 229f.]). Statt diesem allgemeinen Mehrliniensystem sind in der Praxis, die beiden folgenden zu finden.

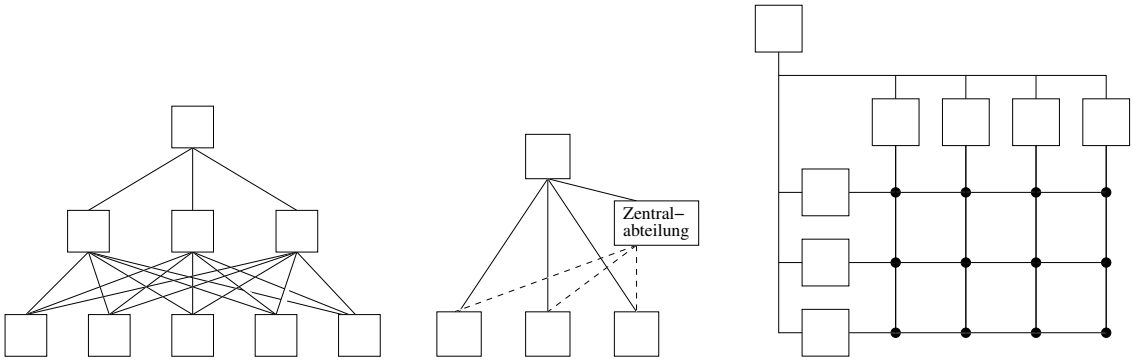
#### Zentralabteilung

Eine Zentralabteilung, die direkt bei der Unternehmensleitung angesiedelt ist, ist häufig aus Stabsstellen für Finanzen, Planung, Recht oder Ähnlichem hervorgegangen. Diese Abteilung darf für einen engen Zuständigkeitsbereich Anweisungen an untergeordnete Organisationseinheiten Weisungen geben.

#### Matrixorganisation

Bei einer Matrixstruktur ist die Organisation in die beiden Dimensionen Verrichtungsprinzip (z.B. Beschaffung, Produktion, Absatz) und Objektprinzip (z.B. die Produkte A, B, C) aufgegliedert. Gegenüber den untergeordneten Organisationseinheiten der Matrix, die in Abbildung 2.3(c) durch Punkte dargestellt sind, ist jeweils eine Organisationseinheit aus jeder Dimension weisungsberechtigt. Diese können gegenüber Weisungen des anderen ein Veto einlegen und müssen sich dann im

[9] Dieses System wird auch als Funktionsmeistersystem bezeichnet.




(a) Allgemeines Mehrliniensystem (b) Zentralabteilung (c) Matrixorganisation

Abbildung 2.3: Mehrliniensysteme

Konsens einigen.

Für die weitere Konzipierung des Autorisierungssystems ist es wichtig festzustellen, dass sich in einer Organisationseinheit maximal zwei Weisungslinien treffen können. Dies findet sich insbesondere in den entsprechenden **Hierarchien der Autorisierungssphären** wieder.

# Teil I

# Rollenbasierte Zugriffskontrolle in verteilten Dateisystemen

# Kapitel 3

# Grundlegende Elemente einer modularen rollenbasierten Zugriffssprache

Bisherige Autorisierungssysteme nutzen die Stellenbeschreibungen in einer Organisation, um daraus Rollen zu bilden und sie aufgrund der Aufgabenanalyse mit den notwendigen Zugriffsprivilegien für bestimmte Objekte zu versehen. Der Ansatz der vorliegenden Arbeit für ein Autorisierungssystem berücksichtigt zudem die Strukturierung der Organisation, d.h. z.B. die Aufgabensynthese in einzelne Abteilungen, Arbeitsgruppen und Projekten, um so eine dezentrale Administrierung zu erreichen. Die Zugriffsprivilegien können "vor Ort" in den Abteilungen und Projekten festgelegt werden, damit die Arbeitsabläufe für jede Organisationseinheit optimal angepasst werden. Diese Organisationseinheiten, die eine Abteilung, eine Arbeitsgruppe oder Ähnliches in einer beliebigen Organisation sein können, sind zumeist hierarchisch strukturiert[1]. Dabei macht es keinen Unterschied, ob es sich bei der Organisation um ein Unternehmen, eine Regierungsstelle oder eine öffentliche Einrichtung handelt. Da es unter Umständen zu aufwendig ist, die Zugriffsprivilegien in jeder Arbeitsgruppe einzeln zu regeln, oder wenn in Projekten die Arbeitsabläufe gleich sein sollen, werden sogenannte **Autorisierungssphären** eingeführt, die festlegen, für welche Teilmenge der Organisationseinheiten die gleichen Zugriffsprivilegien gelten sollen. Diese Autorisierungssphären sind genauso wie die Organisationseinheiten hierarchisch strukturiert und bilden damit die Gegebenheiten der Organisation im Autorisierungssystem ab. Dies ermöglicht es, neben einer dezentralen Modellierung der Zugriffsprivilegien auch bestimmte Zugriffsprivilegien durch übergeordnete Autorisierungssphären durchzusetzen. Bevor die Modellierung der Zugriffsprivilegien im Teil 2 genauer vorgestellt wird, werden in diesem Kapitel noch die weiteren Elemente des Autorisierungssystems — neben Organisationseinheiten und Autorisierungssphären — näher erläutert.

Wie eingangs erwähnt, werden die Zugriffsprivilegien an Rollen geknüpft. Da die Zugriffsprivilegien dezentral modelliert werden, können diese nicht nur an eine bestimmte Rolle vergeben werden, sondern an Rollen in einer Organisationseinheit. Konsequenterweise werden einem Benutzer auch Rollen in Organisationseinheiten zugewiesen[2]. Damit kann ein Benutzer, der verschiedene Rollen in unterschiedlichen Organisationseinheiten hat, wie Bob im Szenario aus Abschnitt 1.1 (vgl. Abbildung 1.2), diese nicht ausnutzen, um mehr Zugriffe zu erlangen, indem er die Zugriffsprivilegien einer anderen Rolle einnimmt, die ihm in einer anderen Organisationseinheit zugewiesen wurde.
Um festzulegen, welche Zugriffsprivilegien für die aktuelle Aufgabe genutzt werden sollen, kann der Benutzer jeweils eine ihm zugewiesene Rolle aktivieren. Diese aktive Rolle wird entsprechend bei der dezentralen Auswertung berücksichtigt.
Es ist offensichtlich, dass nicht jede Rolle in allen Organisationseinheiten besetzt ist. So wird

[1] oder es kann zumindest eine Halbordnung zwischen den Organisationseinheiten aufgestellt werden.

[2] In anderen Ansätzen wie [JSSS01, FSG+01] (vgl. Abschnitt 2.1.2 bzw. 2.1.1) werden die Rollen organisationsweit zugewiesen.

von den Rollen in Abbildung 1.1(b) die Rolle Programmierer nur in den Projekten unterhalb der Entwicklungsabteilung Software vergeben sein, während Aufsichtratsvorsitzender nur in der Geschäftsleitung zu finden ist (vgl. Abbildung 1.1(a)). Um jedoch eine Durchsetzung von bestimmten Zugriffsprivilegien zu ermöglichen, sind **alle** Rollen innerhalb einer Organisation bekannt und die Strukturierung dieser Rollen gemäß einer Rollenhierarchie ist überall identisch[3]. Dies wird als Rollenschema in allen Autorisierungssphären berücksichtigt, um die Zugriffsprivilegien auf die Objekte zu modellieren. In [SZ99, S. 212ff] wird die Standardisierung der Rollen als eine Möglichkeit dargestellt, den Koordinationsaufwand bezüglich der Aufgabenanalyse und damit das Verständnis der Rolle zu minimieren, da allen Beteiligten, das Rollenverhalten sowie die Qualifikation und Kenntnisse bekannt sind.

Sämtliche Daten, die in Form von Dateien im Filesystem gespeichert werden, aber auch alle Verzeichnisse des Filesystems, werden im weiteren Verlauf der Arbeit als **Objekte** bezeichnet. Die Informationen, die diese Objekte enthalten, werden zum einen benötigt, um die Aufgaben einer Organisationseinheit zu erfüllen, zum anderen können sie auch wie im Beispielszenario das Produkt der Organisationseinheit sein. Dementsprechend sind die Objekte einzelnen Organisationseinheiten eindeutig zugeordnet. Somit liegt die Verantwortung für die Zugriffsprivilegien auf diese Objekte bei der entsprechenden Autorisierungssphäre der Organisationseinheit.
Da es — vor allem in einem Filesystem — zu aufwändig wäre, für jedes einzelne Objekt die Zugriffsprivilegien separat zu vergeben, werden bestimmte Eigenschaften der Objekte berücksichtigt. Dies kann neben dem Speicherungsort in einem bestimmten Verzeichnis[4], auch in erster Linie die Zugehörigkeit des Objektes zu einer bestimmten Organisationseinheit sein. Innerhalb einer Organisationseinheit ist es jedoch sinnvoll, eine weitere Differenzierung vorzunehmen. Dazu wird eine Typisierung von Objekten eingeführt, die über die von den Filetypen und MIME-Types (vgl. [SGG00, S. 350ff] bzw. [BF93, FB96]) hinausgehen kann und die Möglichkeit einer hierarchischen Strukturierung bietet. Ähnlich wie bei Rollen, werden nicht alle Typen von Objekten in jeder Organisationseinheit verwendet. Jedoch sind innerhalb der Organisation alle Typen bekannt und die Typenhierarchie ist in allen Organisationseinheiten identisch. Hinzu kommen sogenannte freie Attribute [WS00], die zusätzlich Rückschlüsse auf die Inhalte der Objekte zulassen.

Die Zugriffsprivilegien werden mit Hilfe einer Datalog–Sprache modelliert, mit der die Zugriffsrechte für eine bestimmte Gruppe von Rollen in entsprechenden Organisationseinheiten auf wohldefinierte Objekte durch sogenannte Feature Prädikate spezifiziert werden. Dieser Ansatz wurde von [JSSB97] vorgestellt[5] und in [WL01] aufgegriffen.

Im folgenden Abschnitt 3.1 werden zunächst die grundlegenden Elemente *Subjekt* und *Objekt* definiert. Weitere Elemente des Autorisierungssystems werden anschließend vorgestellt und die entsprechenden Feature Prädikate definiert, mit denen diese in den Zugriffsregeln referenziert werden können. Im Abschnitt 3.2 wird die Struktur der Organisation und die Gliederung in Organisationseinheiten näher betrachtet. Darauf aufbauend werden in Abschnitt 3.3 Autorisierungssphären eingeführt, die es erlauben, die Verantwortlichkeiten für die Zugriffsstrategien zu dezentralisieren. Schließlich wird im Abschnitt 3.4 das erweiterte Konzept für Rollen in Organisationseinheiten diskutiert. Im Abschnitt 3.5 werden die Eigenschaften von Objekten, auf denen in den Zugriffsregeln verwiesen werden kann, genauer betrachtet. Abschließend werden im Abschnitt 3.6 die Zugriffsmethoden des Autorisierungssystems erläutert und weitere Feature Prädikate präsentiert, die es ermöglichen, vorangegangene Zugriffe zu berücksichtigen, um so Konzepte wie “Chinese–Wall” umzusetzen. Ergänzend dazu werden im Kapitel 4 Elemente eingeführt, die in einer verteilten Umgebung zusätzlich bei der Autorisierung berücksichtigt werden können.

[3] Dies kann mit Hilfe von Synchronisierungsalgorithmen (vgl. [CDK02]) erreicht werden.
[4] Auf die Speicherung in einer verteilten Umgebung und die daraus folgenden Konsequenzen für die Zugriffsprivilegien wird im Kapitel 4 ausführlich eingegangen.
[5] siehe auch Abschnitt 2.1.2

## 3.1 Subjekte und Objekte

Die Zugriffsprivilegien werden durch eine Datalog–Sprache modelliert, die in dieser Arbeit spezifiziert wird. Ein Zugriff erfolgt durch ein Subjekt, das mit einer bestimmten Methode durch eine Applikation auf ein Objekt zugreift. Im RBAC Standard ist ein Subjekt eine aktive Einheit, die auf Veranlassung eines Benutzers handelt [FKC03, S. 58]. Die Methode, mit der ein Zugriff auf das Objekt erfolgen soll, kann zum Beispiel "Lesen", "Schreiben" oder "Ausführen" sein. Das Recht auf ein bestimmtes Objekt mittels einer bestimmten Methode zugreifen zu dürfen wird als **Zugriffsprivileg** bezeichnet. Diese Zugriffsprivilegien werden mittels Zugriffsregeln modelliert, die bei einer Anfrage ausgewertet werden[6]. Eine **Zugriffsstrategie** wird durch Zugriffsregeln modelliert. Um zu beschreiben, für welche Subjekte ein Zugriff auf Objekte gewährt werden soll, werden in diesem Kapitel sogenannte **Feature Prädikate** definiert, die somit den Gültigkeitsbereich der Zugriffsregeln festlegen.

In den grundlegenden Arbeiten [JSS97, JSSB97] werden Subjekte als Benutzer oder Gruppen von Benutzern definiert[7]. Dies ermöglicht eine einfache Ableitung von Rechten, die anderen Subjekten gewährt werden. Im RBAC Standard ist dagegen ein Subjekt eine aktive Einheit, die auf Veranlassung eines Benutzers handelt und deren Zugang zu Objekten kontrolliert werden soll [FKC03, S. 58]. Dabei können jedem Benutzer mehrere Subjekte zugeordnet werden und jedes Subjekt kann eine unterschiedliche Menge von aktiven Rollen haben.

Mit den Prädikaten im Kapitel 4 können noch weitere Aspekte zu den ausführenden Programmen modelliert werden, sodass die aktiven Programme, die von einem Benutzer kontrolliert werden, wesentlich genauer modelliert werden können. Aus diesem Grund sind **Subjekte** im vorliegenden Autorisierungssystem Benutzer, die im System bekannt sind. Jedem Subjekt, von dem eine Zugriffsanfrage ausgeht, die autorisiert werden soll, kann der Benutzer zugeordnet werden, der direkt an einem Host des Systems eingeloggt ist und eine Rolle aktiviert hat.

Die Menge $\mathcal{S}$ enthalte alle im System bekannten Benutzer. Alle Variablen für Element der Menge $\mathcal{S}$, die in den Zugriffsregeln verwendet werden, sind in der Menge $\mathcal{V}_\mathcal{S}$ enthalten. Für die Definition der Prädikate werden **Subjektterme** $\mathcal{ST}$ über die Menge der Variablen über $\mathcal{S}$ und die Menge der Subjekte definiert ($\mathcal{ST} = \mathcal{V}_\mathcal{S} \cup \mathcal{S}$).

Die Anfragen der Subjekte beziehen sich immer auf Objekte, deren Zugriffe vom Autorisierungssystem verwaltet wird. Um diese Zugriffsanfragen entscheiden zu können, muss die Menge dieser Objekte bekannt sein. Diese Menge aller bekannten Objekte wird als $\mathcal{O}$ definiert. Die Menge $\mathcal{V}_\mathcal{O}$ enthält alle verwendeten Variablen über diese Objektmenge $\mathcal{O}$. Für die Definition der Prädikate werden Terme über die Menge der Objekte $\mathcal{O}$ und die Menge der Variablen über die Objekte benötigt, die als **Objektterme** $\mathcal{OT} = \mathcal{V}_\mathcal{O} \cup \mathcal{O}$ bezeichnet werden.

Um die Feature Prädikate besser anhand von Beispielen erläutern zu können, sei an dieser Stelle kurz ein beispielhaftes Access Prädikat `read` mit der Form $\mathbf{read}(s, o) \leftarrow \mathcal{F}$ eingeführt. Dies ist genau dann gültig, wenn es für die gegebenen Variablenbelegung $s$ und $o$ eine gültige Belegung von Feature Prädikaten $\mathcal{F}$ im Rumpf gibt. In diesem Fall soll dem Subjekt $s$ ein lesender Zugriff auf das Objekt $o$ gestattet sein. Die tatsächlichen Definitionen werden insbesondere im Kapitel 5 gegeben.

## 3.2 Organisationseinheiten

Die verschiedenen Einheiten einer Organisation wie Bereiche, Abteilungen, Arbeitsgruppen oder Projekte werden im Autorisierungssystem durch sogenannte Organisationseinheiten oder auch *Units* dargestellt. In [WL01] wurden diese Organisationseinheiten noch als *Group* bezeichnet. In [OG00, SBM99] sind *Groups* jedoch Mengen von Benutzern und diesen Mengen können gemäß

[6] Eine detaillierte Beschreibung erfolgt im Kapitel 5.
[7] In [JSS97] sind Subjekte zusätzlich auch Rollen.

[OG00] Rollen zugewiesen werden. Um Verwechslungen zu vermeiden, wird in dieser Arbeit nun die Bezeichnung *Unit* oder *Organisationseinheit* verwendet. Die Menge $\mathcal{U}$ enthält alle Organisationseinheiten bzw. Units[8], die im Autorisierungssystem bekannt sind.

In einer Organisation sind die verschiedenen Abteilungen stets hierarchisch strukturiert. Diese Struktur wird mittels einer Halbordnung $<_{\mathcal{U}}$ zwischen Organisationseinheiten im Autorisierungssystem abgebildet. Dabei kann eine Organisationseinheit beliebig viele untergeordnete Organisationseinheiten haben. Wie im Abschnitt 2.5 erläutert, kann die Zahl der übergeordneten Organisationseinheiten auf zwei beschränkt werden[9].

In Abbildung 3.1 sind für die Computerfirma aus dem Szenario im Abschnitt 1.1 die möglichen Organisationseinheiten aufgeführt. Dabei bedeutet eine Kante von $u_h$ nach $u_l$, dass $(u_l, u_h) \in <_{\mathcal{U}}$ bzw. um den transitiven Fall auch mit einzubeziehen $u_l <_{\mathcal{U}} u_h$ gilt. Diese Beispielhierarchie wird im Folgenden zur Erläuterung genutzt.

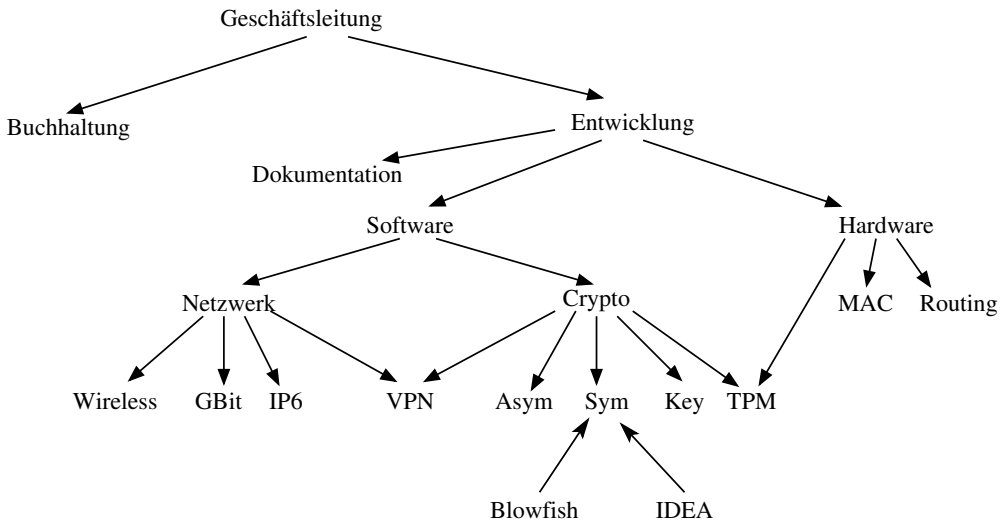



Abbildung 3.1: Organisationseinheiten

Für die Modellierung der Zugriffsregeln wird eine Menge von Variablen über Organisationseinheiten benötigt, die mit $\mathcal{V}_{\mathcal{U}}$ bezeichnet wird. Zur Definition der verschiedenen Prädikate werden Terme über Organisationseinheiten ($\mathcal{UT}$) als Vereinigung der Variablen und Konstanten über Organisationseinheiten definiert ($\mathcal{UT} = \mathcal{V}_{\mathcal{U}} \cup \mathcal{U}$).

### 3.2.1 Feature Prädikate zu Organisationseinheiten

Ähnlich der Gruppenzugehörigkeit in UNIX Systemen gehört in diesem Ansatz jedes Objekt zu genau einer Organisationseinheit. Dies geschieht nicht nur, um einfach eine Kompatibilität zu erreichen, sondern gemäß Organisationstheorie muss jede organisatorische Einheit über Ressourcen verfügen, die zur Erledigung der Aufgabe benötigt werden [PDF99, S. 219].

Um diese Zugehörigkeit zu einer Organisationseinheit bei den Zugriffsregeln berücksichtigen zu können, wird das folgende Feature Prädikat definiert.

**Definition 3.1 (Zugehörigkeit von Objekten zu einer Organisationseinheit)**

**unitof**$(o, u)$

ist ein binäres Prädikat, das dann gilt, wenn das Objekt $o \in \mathcal{OT}$ zur Organisationseinheit $u \in \mathcal{UT}$ gehört. Für jedes Element $o \in \mathcal{O}$ gibt es ein solches $u \in \mathcal{U}$.

[8] Die Begriffe Unit und Organisationseinheit werden synonym verwendet, allerdings leiten sich die Bezeichner bei der formalen Darstellung von Unit ab, um keine Verwechslung mit Objekten zu haben.

[9] Diese Einschränkung wird speziell in Abschnitt 3.3 auch für Autorisierungssphären erfolgen.

Bei der Auswertung dieses Feature Prädikates liegt entweder eine Belegung für beide Parameter vor, sodass dann anhand der entsprechenden Informationen, die dem Autorisierungssystem zur Verfügung stehen müssen[10], entschieden werden kann, ob das Feature Prädikat gültig ist oder nicht. Falls nur die Belegung für das Objekt bekannt ist, erhält der Interpreter durch dieses Prädikat eine eindeutige Belegung für den Parameter $u$. Falls die Belegung des Parameters $o$ (noch) nicht bekannt ist, dafür aber die Organisationseinheit $u$, werden alle möglichen Belegungen für den Parameter $o$ zurückgeliefert. Dies kann die Performance der Regelauswertung beeinflussen und muss bei der Spezifizierung der Zugriffsregeln berücksichtigt werden.

Grundsätzlich gehört jede Organisationseinheit zu genau einer Autorisierungssphäre, die dann für diese Unit die Zugriffspolitiken spezifiziert. Das entsprechende Feature Prädikat wird im Abschnitt 3.3.1 definiert.

In einer Zugriffsregel können mehrere Variablen über Organisationseinheiten verwendet werden[11], die bei der Auswertung der Zugriffsregel durch den Interpreter der Datalog–Sprache durchaus auch mit den gleichen Werten belegt werden können. Um dieses Abfragen zu können und bei Bedarf auszuschließen, wird das folgende Feature Prädikat definiert.

**Definition 3.2 (Gleichheit Organisationseinheiten)**
Das binäre Prädikat

$$\textbf{equal\_unit}(u_1, u_2)$$

ist genau dann gültig, wenn die beiden Terme über Organisationseinheiten $u_1, u_2 \in \mathcal{UT}$ die gleiche Belegung in $\mathcal{U}$ haben.

Um zu verhindern, dass eine Zugriffsregel mit einer Belegung gültig ist, bei der die Variablen $u_1$ und $u_2$ den gleichen Wert haben, muss $\neg\mathsf{equal\_unit}(u_1, u_2)$ im Rumpf enthalten sein. Dieses Feature Prädikat wird an Stelle der spezielle Prädikat Symbole "$<, =, \neq$" aus [CGT90, S. 210f] verwendet, um eine einheitliche Notation zu gewährleisten. Die Auswertung ist im Prototypen des Interpreters [Hal05] realisiert. Die Vergleichsoperatoren für Terme über Organisationseinheiten sind ebenfalls als Feature Prädikate realisiert.

### 3.2.2 Feature Prädikate zur Hierarchie der Organisationseinheiten

Die Relation $<_{\mathcal{U}}$ zwischen den Organisationseinheiten kann mittels der folgenden drei Feature Prädikate in den Zugriffsregeln berücksichtigt werden.

**Definition 3.3 (direkte Unitbeziehung)**

$$\textbf{unitin}(u_1, u_2)$$

ist ein binäres Prädikat, das dann gültig ist, falls für beiden Organisationseinheiten $u_1, u_2 \in \mathcal{UT}$ gilt $(u_1, u_2) \in <_{\mathcal{U}}$.

Wenn die Beispielorganisation in Abbildung 3.1 zu Grunde gelegt wird, wäre das Feature Prädikat $\mathsf{unitin}(Sym, Crypto)$ gültig, $\mathsf{unitin}(IDEA, Crypto)$ hingegen nicht, da keine direkte Relation in $<_{\mathcal{U}}$ existiert.

Daher wird neben diesem Prädikat für die direkten Beziehungen auch das folgende Prädikat definiert, mit dem die Transitivität $(u_1 <_{\mathcal{U}} u_2)$ innerhalb der Hierarchie der Organisationseinheiten einfach getestet werden kann.

**Definition 3.4 (indirekte Unitbeziehung)**
Die indirekte Unitbeziehung kann über die folgenden Regeln für $u_1, u_2 \in \mathcal{UT}$ abgefragt werden, wobei $u' \in \mathcal{V}_{\mathcal{U}}$:

$$\textbf{subunit}(u_1, u_2) \leftarrow \mathsf{unitin}(u_1, u_2)$$
$$\textbf{subunit}(u_1, u_2) \leftarrow \mathsf{subunit}(u_1, u')\&\mathsf{unitin}(u', u_2)$$

[10] Dabei ist die genaue Implementierung dieses Zugriffes in dieser Arbeit nicht von Bedeutung.

[11] Da bisher nur ein einziges Prädikat definiert wurde, lässt sich noch kein Beispiel hierfür geben. Daher wird auf die Beispielregeln 5.1 auf Seite 69 verwiesen.

Durch die freie Variable $u'$ und der rekursiven Definition dieses Feature Prädikates kann ein Interpreter für eine Datalog–Sprache alle transitiven Beziehungen in der Hierarchie der Organisationseinheiten finden. Bei einer Anfrage nach $\mathsf{subunit}(IDEA, Crypto)$ würde zunächst die erste Zeile überprüft werden. Da $\mathsf{unitin}(IDEA, Crypto)$ wie oben gesehen nicht erfolgreich ist, würde die zweite Zeile genommen und geprüft, ob es in $\mathsf{subunit}(IDEA, u')\&\mathsf{unitin}(u', Crypto)$ eine gültig Belegung für $u'$ gibt. Hierzu werden zunächst vom Interpreter der Datalog–Sprache[12] alle möglichen Belegungen für $\mathsf{unitin}(u', Crypto)$ gesucht, dies ist gemäß der Beispielhierarchie aus Abbildung 3.1 VPN, Asym, Sym, Key und TPM. Ein Abgleich mit $\mathsf{unitin}(IDEA, u')$ ergibt Sym als einzige mögliche Belegung. Somit ist $\mathsf{subunit}(IDEA, Sym)\&\mathsf{unitin}(Sym, Crypto)$ überprüft. Die Auflösung von $\mathsf{subunit}(IDEA, Sym)$ nach $\mathsf{unitin}(IDEA, Sym)$ gemäß der ersten Zeile von Definition 3.4, ergibt schon eine gültige Belegung. Somit ist das Prädikat $\mathsf{subunit}(IDEA, Crypto)$ gültig.

Mit dem Prädikat subunit() kann sowohl auf direkte, als auch auf indirekte Beziehung getestet werden, da sich dies bei der Entwicklung der Beispielregeln als praktischer erwiesen hat. Wenn jedoch durch das Prädikat subunit() ausschließlich die indirekten Beziehungen getestet werden könnten[13], müssen für den Fall, dass sowohl indirekte als auch direkte Beziehungen zulässig sind, zwei Zugriffsregeln eingefügt werden, die einmal auf direkte und einmal auf indirekte Beziehung testen. Da davon auszugehen ist, dass es in den meisten Fällen egal ist, ob es sich um eine direkte oder indirekte Beziehung handelt, würde dies zu einer Fülle von ähnlichen Zugriffsregeln führen, die sich jeweils nur durch das Prädikat subunit() und unitin() unterscheiden würden. Da ähnliche Feature Prädikate auch für die Hierarchien über Autorisierungssphären (Def. 3.9), Rollen (Def. 3.15) und Objekttypen (Def. 3.21) definiert werden, würde bei einer strikten Trennung zwischen direkter und indirekter Beziehung, auch bei den entsprechenden Hierarchien, die Zahl der ähnlichen Regeln weiter steigen.

Auf der anderen Seite kann für eine Zugriffsregel, bei der zwei Organisationseinheiten $u_1$ und $u_2$ nur in indirekter Beziehung stehen sollen, die direkte Beziehung einfach negiert werden. Dann würde der entsprechende Ausschnitt aus der Regel lauten:

$$\mathsf{subunit}(u_1, u_2)\&\neg\mathsf{unitin}(u_1, u_2)$$

Zusätzlich hat es sich bei der Entwicklung der Beispielregeln als sinnvoll herausgestellt, auch eine möglichst identische Belegung der Parameter bei den Vergleichen zwischen den Elementen der Hierarchien zuzulassen. Bei der Definition des entsprechenden Feature Prädikates wird auf alle bisher definierten Feature Prädikate zurückgegriffen.

**Definition 3.5 (erweiterte Unitbeziehung)**
Die indirekte Unitbeziehung kann über die folgenden Regeln für $u_1, u_2 \in \mathcal{UT}$ abgefragt werden, wobei $u' \in \mathcal{V}_{\mathcal{U}}$:

$$\begin{aligned}
&\textbf{subuniteq}(u_1, u_2) \leftarrow \mathsf{equal_unit}(u_1, u_2)\\
&\textbf{subuniteq}(u_1, u_2) \leftarrow \mathsf{unitin}(u_1, u_2)\\
&\textbf{subuniteq}(u_1, u_2) \leftarrow \mathsf{subunit}(u_1, u')\&\mathsf{unitin}(u', u_2)
\end{aligned}$$

Die zweite und dritte Zeile könnten auch durch $\mathsf{subuniteq}(u_1, u_2) \leftarrow \mathsf{subunit}(u_1, u_2)$ ersetzt werden. Dies hätte jedoch in den Fällen, in denen das Prädikat $\mathsf{unitin}(u_1, u_2)$ gültig ist, eine schlechtere Performance des Interpreters durch den unnötigen Aufruf von $\mathsf{subunit}(u_1, u_2)$ zur Folge.

[12] Die Details zum Prototypen für die Autorisierung finden sich in [Hal05].

[13] Dies könnte dadurch erreicht werden, indem statt der Definition 3.4 das Prädikat subunit() wie folgt definiert würde:

$$\begin{aligned}
&\mathsf{subunit}(u_1, u_2) \leftarrow \mathsf{unitin}(u_1, u')\&\mathsf{unitin}(u', u_2)\\
&\mathsf{subunit}(u_1, u_2) \leftarrow \mathsf{subunit}(u_1, u')\&\mathsf{unitin}(u', u_2)
\end{aligned}$$

## 3.3 Autorisierungssphären

Wie in der Motivation zu dieser Arbeit dargestellt, soll die Spezifizierung der Zugriffsprivilegien dezentral erfolgen. Diese dezentrale Spezifizierung in jeder einzelnen Organisationseinheit zuzulassen würde aber auch voraussetzen, dass in jeder Organisationseinheit eine Gruppe von Benutzern vorhanden ist, die kompetent ist, die entsprechenden Zugriffsstrategien zu formulieren[14]. Davon ist nicht in jeder beliebigen Organisation auszugehen. Zudem erscheint eine Koordination bei der Spezifizierung der Zugriffsstrategien über mehrere Organisationseinheiten hinweg durchaus sinnvoll. Aus diesem Grunde werden Autorisierungssphären eingeführt, in denen für eine bestimmte Teilmenge der Organisationseinheiten die Zugriffsstrategien spezifiziert werden. Die **Zugriffsstrategie** einer Autorisierungssphäre umfasst alle in ihr spezifizierten Zugriffsregeln. Die Menge $\mathcal{AS}$ enthält alle im Autorisierungssystem definierten Autorisierungssphären.

Für die Delegierung von Privilegien und die Vererbung von Zugriffsstrategien ist es notwendig, dass die Autorisierungssphären in direkter Beziehung stehen. Dieses wird mittels einer Relation $<_{\mathcal{AS}}$ definiert. In Abbildung 3.2 ist eine solche Hierarchie von Autorisierungssphären für die Beispielorganisation dargestellt.

Einer Autorisierungssphären können beliebig viele Autorisierungssphären direkt untergeordnet sein. Jedoch kann eine Autorisierungssphäre **maximal zwei direkt übergeordnete** Autorisierungssphären. Damit lassen sich beim Stabliniensystem, die Stabstelle und die vorgesetzte Organisationseinheit mit eigenen Autorisierungssphären versehen und entsprechend bei der Matrixorganisation die beiden Achsen mit unterschiedlichen Autorisierungssphären. Wie in Abschnitt 2.5 erläutert, stellt diese Einschränkung für beliebige Organisationsstrukturen kein Problem dar.

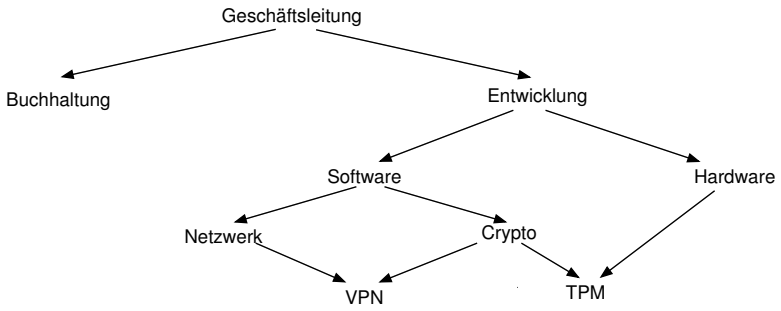



Abbildung 3.2: Autorisierungssphären und deren Hierarchie

Da die Zugriffsstrategien nicht direkt von einer Organisationseinheit definiert werden, sondern von der zuständigen Autorisierungssphäre, wird eine entsprechende Zuordnung benötigt. Jede Organisationseinheit $u$ wird mittels einer Abbildungsfunktion genau einer Autorisierungssphäre $as$ zugeordnet:

$$sphereof : \mathcal{U} \rightarrow \mathcal{AS}$$

Wobei wie oben beschrieben, mehrere Organisationseinheiten zu einer Autorisierungssphäre gehören können. Für die Beispielorganisation sind in Abbildung 3.3 die Autorisierungssphären und die zugeordneten Organisationseinheiten zu sehen. Dabei ist es zulässig, aber nicht zwingend erforderlich, dass Bezeichner aus der Menge der Autorisierungssphären $\mathcal{AS}$ auch in der Menge der Organisationseinheiten $\mathcal{U}$ auftauchen.

Die Hierarchie der Autorisierungssphären steht im engen Zusammenhang mit der zuvor im Abschnitt 3.2 beschriebenen Hierarchie der Organisationseinheiten. Sowohl die Organisationseinheiten als auch die Autorisierungssphären sind durch die Relation $<_{\mathcal{U}}$ bzw. $<_{\mathcal{AS}}$ teilgeordnet. Diese beiden Relationen dürfen nicht unabhängig voneinander sein, da sie beide die Struktur der Organisation widerspiegeln. Daher muss Folgendes für beliebige Organisationseinheiten und deren Autorisierungssphären gelten:

[14]Diese Gruppe von Benutzern wird im weiteren Verlauf als Autorisierungsteam bezeichnet. Dieses wird in Definition 9.1 im Kapitel 9 noch formal beschrieben.

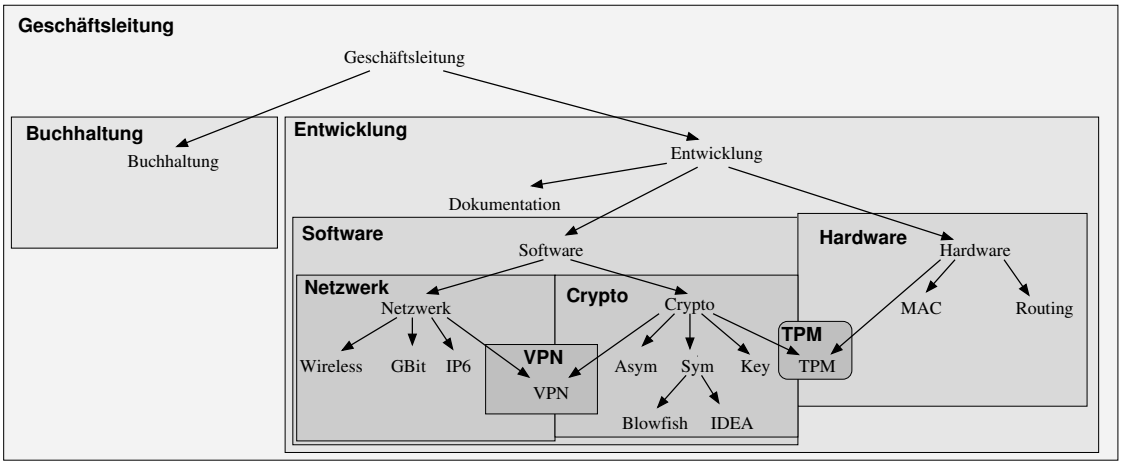
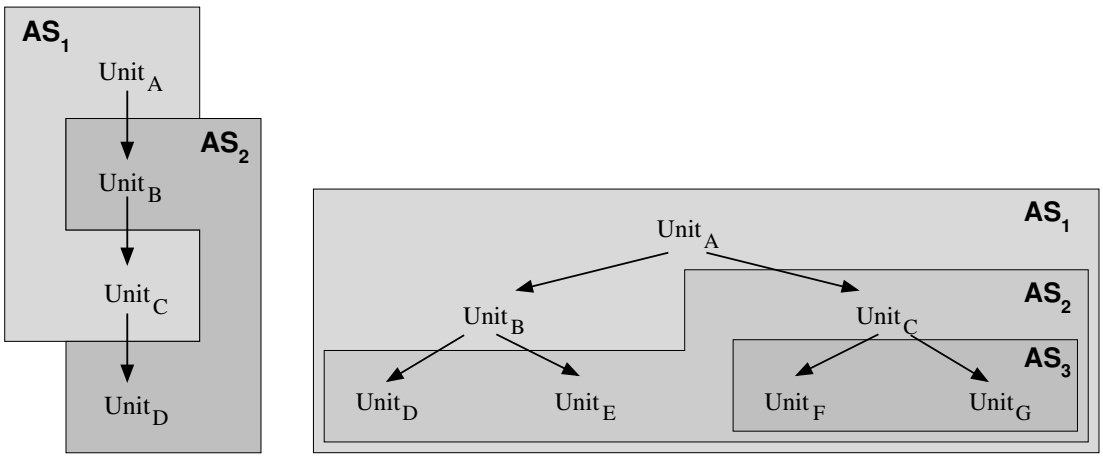



Abbildung 3.3: Autorisierungssphären und deren Organisationseinheiten innerhalb einer Organisation

**Vereinbarung 3.1**

$$\forall u_1, u_2 \in \mathcal{U} : u_1 <_{\mathcal{U}} u_2 \Rightarrow sphereof(u_1) <_{\mathcal{AS}} sphereof(u_2) \vee sphereof(u_1) = sphereof(u_2)$$

Dabei ist zu beachten, dass die Rückrichtung nicht immer gilt, da zwei Units unter Umständen nicht zueinander in Beziehung stehen können, jedoch die zugehörigen Autorisierungssphären. So gilt, wenn die Hierarchien aus Abbildung 3.1 und 3.2 sowie die Zuordnung in Abbildung 3.3 zu Grunde gelegt werden, $sphereof(IP6) <_{\mathcal{AS}} sphereof(Documentation)$ aber nicht die Beziehung $IP6 <_{\mathcal{U}} Documentation$.




(a) unzulässige Anordnung (b) zulässige Anordnung

Abbildung 3.4: Konstellationen zwischen Autorisierungssphären und Organisationseinheiten

Eine untergeordnete Autorisierungssphäre kann die Zugriffsstrategien der übergeordneten Autorisierungssphären erben[15]. Durch die Forderung in Vereinbarung 3.1 wird auch verhindert, dass sich Kreise bei der Auswertung der Zugriffsregeln der einzelnen Autorisierungssphären bilden.

[15]mehr dazu in der Definition 6.1 im Kapitel 6 .

Eine Konstellation, wie in Abbildung 3.4(a), wäre demnach nicht zulässig, da die beiden Autorisierungssphären $AS_1$ und $AS_2$ durch eine Relation $<_{\mathcal{AS}}$ nicht so in Beziehung gesetzt werden können, dass für alle Organisationseinheiten $\mathsf{Unit_A}, \mathsf{Unit_B}, \mathsf{Unit_C}$ und $\mathsf{Unit_D}$ die Vereinbarung 3.1 erfüllt ist.

Die Konstellation in Abbildung 3.4(b) wäre zulässig, da nicht gefordert ist, dass alle Organisationseinheiten, die zu einer Autorisierungssphäre gehören, einen zusammenhängenden Teilgraphen bilden müssen. Bei der Zuordnung der Organisationseinheiten zu Autorisierungssphären muss dies auch überprüft werden, da jede Autorisierungssphäre maximal zwei übergeordnete Autorisierungssphären haben kann.

Für den Verweis auf Autorisierungssphären bei der Spezifizierung der Regeln wird eine Menge von Variablen über Autorisierungssphären benötigt, die als $\mathcal{V}_{\mathcal{AS}}$ bezeichnet werden. Zur Definition der verschiedenen Prädikate werden Terme über Autorisierungssphären $\mathcal{AST}$ definiert als Vereinigung der Variablen und Konstanten über Autorisierungssphären ($\mathcal{AST} = \mathcal{V}_{\mathcal{AS}} \cup \mathcal{AS}$).

### 3.3.1 Feature Prädikate zu Autorisierungssphären

Jede Organisationseinheit gehört immer zu genau einer Autorisierungssphäre, sodass hierdurch einfach festgestellt werden kann, wer direkt für die Definition der Zugriffsregeln verantwortlich ist. Diese können von übergeordneten Autorisierungssphären entweder beim Zugriff zwischen verschiedenen Autorisierungssphären außer Kraft gesetzt werden, wie im Kapitel 7 dargestellt wird, jedoch muss auch hierbei die verantwortliche Autorisierungssphäre festgestellt werden. Das folgende Feature Prädikat wird dazu genutzt, diese Beziehung zwischen Organisationseinheit und Autorisierungssphäre zu prüfen.

**Definition 3.6 (Autorisierungssphäre)**

$$\textsf{sphereof}(u, as)$$

ist ein binäres Prädikat, das dann gültig ist, wenn die Unit $u \in \mathcal{UT}$ zur Autorisierungssphäre $as \in \mathcal{AST}$ gehört. Für jedes Element $u \in \mathcal{U}$ gibt es ein solches $as \in \mathcal{AS}$.

In Bezug auf die Beispielorganisation in Abbildung 3.3 auf Seite 28 würde dies Folgendes bedeuten: $\textsf{sphereof}(Blowfish, Crypto)$ wäre ein gültiges Feature Prädikat. Dagegen wäre das Prädikat $\textsf{sphereof}(IDEA, Software)$ nicht gültig, da Crypto die zugehörige Autorisierungssphäre ist und Software eine übergeordnete Sphäre von Crypto. Falls bei der Auswertung einer Zugriffsregel die Organisationseinheit nicht genau festgelegt ist, und stattdessen eine Variable $u \in \mathcal{V}_{\mathcal{U}}$ bei der Definition verwendet wird, wäre das Prädikat $\textsf{sphereof}(u, Netzwerk)$ dann gültig, wenn $u$ einen der Werte aus der Menge {Network, Wireless, GBit, IP6} annimmt. Da VPN in einer eigenen Autorisierungssphäre ist, wäre das Prädikat bei dieser Belegung von $u$ nicht gültig.

Die eindeutige Zugehörigkeit eines Objektes zu einer Organisationseinheit lässt sich auch durch das folgende Regelfragment, genau einer Autorisierungssphäre $as$ zuordnen:

$$\textsf{unitof}(o, u) \& \textsf{sphereof}(u, as)$$

Für eine beliebige aber feste Belegung von $o$ wird der freien Variablen $u$ ein eindeutiger Wert zugewiesen, bei der das erste Prädikat gültig ist. Mit dieser Belegung für $u$ wird dann eine Belegung für die Variable $as$ gesucht, mit der das zweite Prädikat gültig wird. Somit lässt sich auch eine direkte Verantwortlichkeit für die Zugriffsregeln eines Objektes festlegen.

Um bei der Verwendung von Variablen über Autorisierungssphären festlegen zu können, ob zwei Variablen $as_1, as_2 \in \mathcal{V}_{\mathcal{AS}}$ die gleiche Belegung haben dürfen, wird das folgende Feature Prädikat definiert.

**Definition 3.7 (Gleichheit Autorisierungssphären)**
Das binäre Prädikat

$$\textbf{equal\_sphere}(as_1, as_2)$$

ist genau dann gültig, wenn die beiden Autorisierungssphärenterme $as_1, as_2 \in \mathcal{AST}$ die gleiche Belegung in $\mathcal{AS}$ haben.

Analog zur Gleichheit von Organisationseinheiten in Definition 3.2 wird dieses Prädikat auch bei der Definition eines Feature Prädikates zum Vergleich von Autorisierungssphären gemäß der Relation $<_{\mathcal{AS}}$ verwendet.

### 3.3.2 Feature Prädikate zur Hierarchie der Autorisierungssphären

Um die Relation $<_{\mathcal{AS}}$, die zwischen den Autorisierungssphären definiert ist, in den Zugriffsregeln nutzen zu können[16], werden die folgenden Prädikate, die diese Relation für zwei gegebene Autorisierungssphären prüfen, benötigt.

**Definition 3.8 (direkte Beziehung von Autorisierungssphären)**

$$\textbf{spherein}(as_1, as_2)$$

ist ein binäres Prädikat, das dann gültig ist, wenn für die beiden Autorisierungssphären $as_1, as_2 \in \mathcal{AST}$ gilt: $(as_1, as_2) \in <_{\mathcal{AS}}$.

Gemäß der Hierarchie der Autorisierungssphären in Abbildung 3.2, würden die Prädikate spherein$(Hardware, Entwicklung)$ und spherein$(VPN, Crypto)$ gültig sein. Dagegen wäre die Belegung spherein$(Netzwerk, Entwicklung)$ nicht gültig, da hier keine direkte Kante besteht, sondern nur eine indirekte Kante über die Autorisierungssphäre Software. Um neben der direkten Beziehung auch die Transitivität innerhalb der Hierarchie der Autorisierungssphären einfach testen zu können, wird das folgende Prädikat definiert. Wie schon in der Definition 3.4 wird dieses Prädikat rekursiv definiert und benutzt eine freie Variable.

**Definition 3.9 (indirekte Beziehung von Autorisierungssphären)**
Die indirekte Beziehung von Autorisierungssphären kann über die folgenden Regeln für $as_1, as_2 \in \mathcal{AST}$ abgefragt werden wobei $as' \in \mathcal{V}_{\mathcal{AS}}$:

$$\textbf{subsphere}(as_1, as_2) \leftarrow \text{spherein}(as_1, as_2)$$
$$\textbf{subsphere}(as_1, as_2) \leftarrow \text{subsphere}(as_1, as') \& \text{spherein}(as', as_2)$$

Wie zuvor erwähnt, ist bei einer Auswertung von subsphere$(Netzwerk, Entwicklung)$ die erste Zeile der Definition 3.9 nicht gültig. Die Variable $as'$ wird nach Auswertung des Prädikates spherein$(as', Entwicklung)$ vom Interpreter mit der Belegung Software und Hardware getestet. Während mit der Belegung Hardware keine gültiges Feature Prädikat spherein$(Netzwerk, as')$ vorliegt, ist das Prädikat spherein$(Netzwerk, Software)$ gültig und damit auch das Feature Prädikat subsphere$(Netzwerk, Entwicklung)$.

Die Vereinbarung 3.1 lässt sich wie folgt mittels Feature Prädikaten ausdrücken. Dabei muss für jede Variablenbelegung $u_l, u_h \in \mathcal{V}_{\mathcal{U}}$ gelten:

$$\text{subunit}(u_l, u_h) \Rightarrow \text{sphereof}(u_l, as_l) \& \text{sphereof}(u_h, as_h) \& \text{subsphere}(as_l, as_h)$$

Um beim Vergleich von Autorisierungssphären auch die gleiche Belegung zweier Variablen $as_1, as_2 \in \mathcal{V}_{\mathcal{AS}}$ zulassen zu können, wird das folgende Prädikat als Erweiterung der Definition 3.9 definiert.

[16] Im Kapitel 6 wird diese Hierarchien in den Regeln eingesetzt. Im Abschnitt 8.2 wird diese Hierarchie dazu genutzt eine geschichtete Datalog–Sprache zu erzeugen.

**Definition 3.10 (erweiterte Beziehung von Autorisierungssphären)**
Die erweiterte Beziehung von Autorisierungssphären kann über die folgenden Regeln für $as_1, as_2 \in \mathcal{AST}$ abgefragt werden wobei $as' \in \mathcal{V}_{\mathcal{AS}}$:

$$\textbf{subsphereeq}(as_1, as_2) \leftarrow \mathsf{equal\_sphere}(as_1, as_2)$$
$$\textbf{subsphereeq}(as_1, as_2) \leftarrow \mathsf{spherein}(as_1, as_2)$$
$$\textbf{subsphereeq}(as_1, as_2) \leftarrow \mathsf{subsphere}(as_1, as')\&\mathsf{spherein}(as', as_2)$$

## 3.4 Rollen

Bei der rollenbasierten Zugriffskontrolle werden die Zugriffsprivilegien an Rollen geknüpft. Dabei ist gemäß [FK92, FSG+01] eine Rolle eine *Stellenbeschreibung* ("job function") oder ein *Stellentitel* in einer Organisation.Gemäß [KSS03] leiten sich die Rollen von den existierenden Stellenbeschreibungen in einer Organisation ab. Die Menge $\mathcal{R}$ enthält alle im Autorisierungssystem definierten Rollen.

Während der sogenannte "Flat RBAC" noch keine Beziehungen zwischen den Rollen kennt[17], wird beim sogenannten "Hierarchical RBAC" eine Vererbung von Zugriffsprivilegien entlang einer Rollenhierarchie eingeführt. Diese automatische Vererbung entlang der Rollenhierarchie wird in diesem Autorisierungssystem durch eine explizite Weitergabe der Privilegien durch sogenannte Ableitungsregeln ersetzt[18], wie sie auch schon in den grundlegenden Arbeiten (u.a. [JSS97]) vorgeschlagen wurden.

Die Relation $<_{\mathcal{R}}$ zwischen den Rollen wird auch als Rollenhierarchie bezeichnet. Beim Standard RBAC wird zwischen allgemeinen und eingeschränkten Rollenhierarchien unterschieden. Während Erstere eine beliebige Struktur haben dürfen, sind bei Letzterer nur Baumstrukturen oder invertierte Baumstrukturen zulässig, d.h. eine Rolle darf mehrere Nachfolger haben, aber nur genau einen Vorgänger oder umgekehrt (vgl. [FSG+01, S. 235] bzw. [FKC03, S.85]). Es gibt keine eindeutigen Aussagen dazu, dass jede Rolle mit einer anderen in Beziehung stehen muss[19]. Die Beispiele und Erläuterungen legen dies jedoch nahe. In diesem Modell muss sowohl bei einer allgemeinen als auch bei einer eingeschränkten Rollenhierarchie jede Rolle mit einer anderen in Beziehung stehen. Daher soll die folgende Vereinbarung stets gelten.

**Vereinbarung 3.2**

$$\forall r \in \mathcal{R} \exists r' \in \mathcal{R} : r <_{\mathcal{R}} r' \vee r' <_{\mathcal{R}} r$$

Dagegen wird bei einer allgemeinen Rollenhierarchie ein einziges minimales oder maximales Element nicht gefordert, wie dies beim Standard–RBAC für eine allgemeine Rollenhierarchie der Fall ist. Bei einer eingeschränkten Rollenhierarchie gibt es auf jeden Fall ein minimales oder maximales Element. Aus organisatorischer Sicht gibt es fast immer einen "obersten Verantwortlichen", sodass dies das maximale Element wäre.

In Abbildung 3.5 ist ein Beispiel für eine entsprechende Rollenhierarchie in einer Computerfirma zu sehen, die im Folgenden zur Erläuterung eingesetzt werden soll.

Für die Spezifikation der Regeln wird die Menge der Variablen $\mathcal{V}_{\mathcal{R}}$ über diese Menge der Rollen $\mathcal{R}$ benötigt. Rollenterme $\mathcal{RT}$, die bei der Definition der verschiedenen Prädikate benötigt werden, seien die Vereinigung von Rollenvariablen und Rollenkonstanten, $\mathcal{RT} = \mathcal{V}_{\mathcal{R}} \cup \mathcal{R}$.

### 3.4.1 Feature Prädikate zu Rollen

Als Besonderheit erfolgt bei diesem Ansatz die Zuordnung einer Rolle zu einem Benutzer nicht wie im RBAC Standard für die gesamte Organisation, sondern nur für eine bestimmte Organisationseinheit. Dies kann mit dem folgenden Feature Prädikat abgefragt werden.

[17] Details zum RBAC Standards finden sich im Abschnitt 2.1.1.
[18] Mehr bei der Definition der entsprechenden Prädikate im Kapitel 5.
[19] Dies ist besonders beim Anlegen von neuen Rollen problematisch, wie im Abschnitt 9.3.1 erläutert wird.

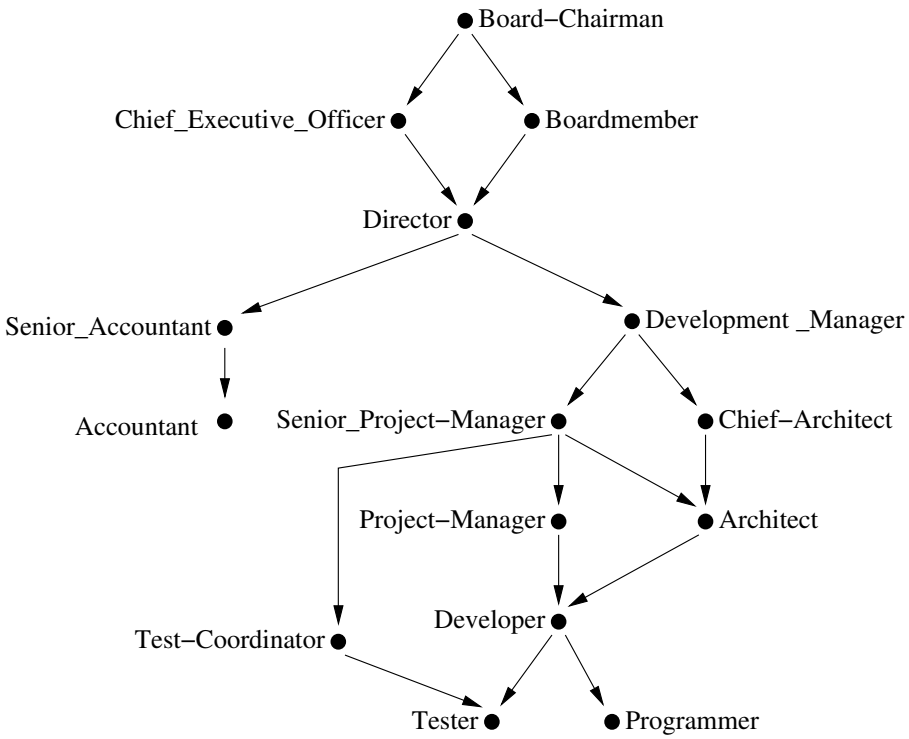



Abbildung 3.5: Rollen innerhalb der Organisation

**Definition 3.11 (Rollenzuweisung)**
Das dreistellig Prädikat

$$\textbf{role(}s, u, r\textbf{)}$$

ist gültig, wenn dem Subjekt $s \in \mathcal{ST}$ die Rolle $r \in \mathcal{RT}$ in der Unit $u \in \mathcal{UT}$ zugewiesen wurde.

Wenn, wie im Abschnitt 1.1 erläutert, ein Benutzer Bob die Rolle Tester in der Unit Crypto innehat und in der Unit Netzwerk die Rolle Programmer, dann kann mit diesem Prädikat und dem Prädikat unitof(o,u) aus Definition 3.1 die folgenden Regeln definiert werden:

**Beispielregel 3.1**

$$\mathsf{read}(s, o) \leftarrow \mathsf{role}(s, u, Tester) \& \mathsf{unitof}(o, u)$$

Aufgrund dieser Regel dürfte Bob ein Objekt, das zur Unit Crypto gehört lesen, eines das zur Gruppe Netzwerk gehört dürfte er aufgrund dieser Regel jedoch nicht lesen[20]. Dies liegt daran, dass er in der Unit Netzwerk nicht die Rolle Tester hat. Dass er in einer anderen Unit diese Rolle inne hat, ist aufgrund der obigen Regeln irrelevant. Durch die Nutzung von freien Variablen kann die Regel so umgeformt werden, dass es ausreichend ist, in einer beliebigen Unit die Rolle Tester zu haben, um auf Objekte der Unit Netzwerk zuzugreifen.

$$\mathsf{read}(s, o) \leftarrow \mathsf{role}(s, u, Tester) \& \mathsf{unitof}(o, Netzwerk)$$

Bei diesen beiden vorangegangen Regeln wird nicht berücksichtigt, ob Bob die Rolle Programmer oder Tester aktiviert hat, er bekommt Zugriff auf die Objekte in Netzwerk. Zu Beginn einer

[20] Es kann natürlich sein, dass noch eine weitere Regel greift, die Leserechte auch für die Rolle Programmer zulässt.

Auswertung ist es jedoch nötig festzustellen, in welcher Rolle ein Subjekt gerade aktiv ist und dementsprechend die Zugriffsanfrage gestellt hat. Da immer nur eine Rolle aktiviert ist, kann dies aus dem Kontext der Zugriffsanfrage mit dem folgenden Feature überprüft werden.

**Definition 3.12 (Rollenaktivierung)**
Das dreistellig Prädikat

$$\textbf{activ\_role}(s, u, r)$$

ist gültig, wenn dem Subjekt $s \in \mathcal{ST}$ die Rolle $r \in \mathcal{RT}$ in der Unit $u \in \mathcal{UT}$ zugewiesen wurde und diese Rolle $r$ bei der Anfrage aktiviert ist.

Die Reduzierung auf eine einzige aktive Rolle gegenüber beliebig vielen beim Standard–RBAC scheint zu keinen tatsächlichen Einschränkungen zu führen, da jederzeit die aktive Rolle gewechselt werden kann. In [FKC03] wird auch davon ausgegangen, dass durch mehrere aktive Rollen die Zahl der möglichen Zugriffe größer wird. Dies ist bei hybriden Zugriffsstrategien jedoch nicht immer der Fall, da an manche Rollen auch ein Verbot eines Zugriffs geknüpft sein kann, während an einer anderen Rolle eine Erlaubnis für dasselbe Objekt geknüpft ist. Da diese Arten von Konflikten in der vorliegenden Arbeit nicht behandelt werden sollen, wird nur eine einzige aktive Rolle zugelassen.

Durch die Feature Prädikate $\mathsf{activ\_role}(s, r, u)\&\mathsf{sphereof}(u, as)$ kann zu Beginn einer Auswertung[21] die aktive Rolle und darüber auch die zugehörige Autorisierungssphäre bestimmt werden, sodass diese für die weitere Auswertung entsprechend berücksichtigt werden kann.

Wenn Bob die Rolle Tester in der Organisationseinheit Crypto aktiviert hat, kann er aufgrund der folgenden Regel keine Objekte $o$ lesen, die zur Organisationseinheit Netzwerk gehören.

**Beispielregel 3.2**

$$\mathsf{read}(s, o) \leftarrow \mathsf{activ\_role}(s, u, Programmer)\&\mathsf{unitof}(o, u)\&\mathsf{equal\_unit}(u, Netzwerk)$$

Analog zu den Feature Prädikaten für Organisationseinheiten und Autorisierungssphären wird auch für Rollen ein Feature Prädikat definiert, mit dem zwei Variablen über Rollen auf Gleichheit geprüft werden können.

**Definition 3.13 (Rollen Gleichheit)**
Das binäre Prädikat

$$\textbf{equal\_role}(r_1, r_2)$$

ist genau dann gültig, wenn die beiden Rollenterme $r_1, r_2 \in \mathcal{RT}$ die gleiche Belegung in $\mathcal{R}$ haben.

Neben der Verwendung in der Rollenhierarchie kann dieses Prädikat auch eingesetzt werden, um allgemeine Rollenzuweisungen und gerade aktive Rolle zu vergleichen.

$$\mathsf{activ\_role}(s, u_1, r_1)\&\mathsf{role}(s, u_2, r_2)\&\neg\mathsf{equal\_role}(r_1, r_2)$$

Die obigen Feature Prädikate sind dann (alle) gültig, wenn das Subjekt $s$ neben der aktiven Rolle $r_1$ in der Organisationseinheit $u_1$ noch eine weitere Rolle $r_2$ in einer Organisationseinheit $u_2$ aufweist. Dabei müssen die Rollen $r_1$ und $r_2$ verschieden sein, können jedoch in der gleichen Organisationseinheit liegen, da keine Bedingung an die Variablen $u_1$ und $u_2$ geknüpft sind.

### 3.4.2 Feature Prädikate zur Rollenhierarchie

Analog zu den Hierarchien über Organisationseinheiten und Autorisierungssphären werden im Folgenden die Feature Prädikate für den Vergleich von Rollen gemäß einer Relation $<_{\mathcal{R}}$ definiert. Diese Feature Prädikate werden vor allem dazu genutzt, die Weitergabe von Privilegien entlang der Rollenhierarchie zu modellieren.

Die direkte Beziehung zwischen zwei Rollen gemäß der Relation $<_{\mathcal{R}}$ kann mit dem folgenden Feature Prädikat abgefragt werden.

[21] Nähere Ausführungen dazu erfolgen im Abschnitt 8.1.

**Definition 3.14 (direkte Rollenbeziehung)**

$$\textbf{rolein}(r_1, r_2)$$

ist ein binäres Prädikat, das dann gültig ist, wenn für die beiden Rollen $r_1, r_2 \in \mathcal{RT}$ gilt: $(r_1, r_2) \in <_{\mathcal{R}}$.

Analog zu den indirekten Beziehungen für Autorisierungssphären und Organisationseinheiten (vgl. Definitionen 3.4 und 3.9) werden transitive Beziehungen ($r_1 <_{\mathcal{R}} r_2$) zwischen Rollen rekursiv definiert.

**Definition 3.15 (indirekte Rollenbeziehung)**
Die indirekte Rollenbeziehung kann über die folgenden Regeln für $r_1, r_2 \in \mathcal{RT}$ abgefragt werden, wobei $r' \in \mathcal{V}_{\mathcal{R}}$:

$$\textbf{subrole}(r_1, r_2) \leftarrow \mathsf{rolein}(r_1, r_2)$$
$$\textbf{subrole}(r_1, r_2) \leftarrow \mathsf{subrole}(r_1, r')\&\mathsf{rolein}(r', r_2)$$

Auf diese Feature Prädikate aufbauend ist das folgende Prädikat auch dann gültig, wenn die Belegung der beiden Parameter gleich ist.

**Definition 3.16 (erweiterte Rollenbeziehung)**
Die indirekte erweiterte Rollenbeziehung kann über die folgenden Regeln für $r_1, r_2 \in \mathcal{RT}$ abgefragt werden, wobei $r' \in \mathcal{V}_{\mathcal{R}}$:

$$\textbf{subroleeq}(r_1, r_2) \leftarrow \mathsf{equal\_role}(r_1, r_2)$$
$$\textbf{subroleeq}(r_1, r_2) \leftarrow \mathsf{rolein}(r_1, r_2)$$
$$\textbf{subroleeq}(r_1, r_2) \leftarrow \mathsf{subrole}(r_1, r')\&\mathsf{rolein}(r', r_2)$$

Mit Hilfe dieses und des entsprechenden Feature Prädikates für Organisationseinheiten lässt sich eine Regel spezifizieren, durch die der Lesezugriff für eine größere Menge von Subjekten in bestimmten Rollen gewährt wird.

**Beispielregel 3.3**

$$\mathsf{read}(s, o) \leftarrow \mathsf{activ\_role}(s, u, r)\&\mathsf{subroleeq}(r, Developer)\& \\ \mathsf{unitof}(o, u)\&\mathsf{subuniteq}(u, Netzwerk)$$

In diesem Beispiel muss das Subjekt die Rolle Developer oder eine entsprechende untergeordnete Rolle gemäß der Rollenhierarchie in Abbildung 3.5 in der Organisationseinheit Netzwerk oder einer gemäß Abbildung 3.1 untergeordneten Organisationseinheit aktiviert haben, um Lesezugriff auf Objekte der entsprechenden Organisationseinheit zu erhalten. Diese Objekte können mit den im folgenden Abschnitt definierten Feature Prädikaten genauer beschrieben werden.

## 3.5 Eigenschaften von Objekten

Bisher kann bei Zugriffsstrategien nur die Zugehörigkeit eines Objektes zu einer Organisationseinheit als Merkmal genutzt werden (vgl. Def. 3.1). Wie zu Beginn des Kapitels erwähnt, sind Objekte Dateien, die in einem Dateisystem verwaltet werden. Darüber hinaus können Objekte wie in einem UNIX–System auch Verzeichnisse oder spezielle Blockdevices sein. Im DRAGON SLAYER System wurde zusätzlich zu Verzeichnissen auch die Möglichkeit vorgestellt, Dateien noch einmal in sogenannte *Parts* zu unterteilen [WL99].

Da das vorliegende Autorisierungssystem besonders Dateisysteme berücksichtigt, kann die Verzeichnisstruktur ebenfalls bei der Zugriffskontrolle herangezogen werden. Bereits bei UNIX Filesystemen kann ein Benutzer nur auf Dateien zugreifen, wenn er entsprechende Rechte auf allen Verzeichnissen des Pfades hat. So wird die inhaltliche Strukturierung, die durch die Verzeichnisse

eines Dateisystem ermöglicht wird, auch bei der Zugriffskontrolle genutzt.
Wie in [JSSB97] beschrieben, können im Bereich Relationale Datenbanken die Objekte Tabellen und Relationen sein, bzw. bei objekt–orientierten Datenbanken die Klassen und die Instanzen. Auch in diesem Anwendungsgebiet lassen sich die Objekte entsprechend hierarchisch strukturieren.

Für das vorliegende Autorisierungssystem kann dies durch eine einzige Objekthierarchie über die Relation $<_{\mathcal{O}}$ modelliert werden. Dabei können Verzeichnisse weitere Verzeichnisse, spezielle Blockdevices oder Dateien enthalten, Dateien können in Parts unterteilt werden. Die entsprechenden Feature Prädikate, um diese Relation bei den Zugriffsregeln zu berücksichtigen, werden im Abschnitt 3.5.1 definiert.

Neben den verschiedenen Arten von Objekten können Dateien mit einem eindeutigen Typ versehen werden, der eine weitere inhaltliche Unterscheidung für die Zugriffsregeln ermöglicht. Diese Typisierung für das vorliegende Autorisierungssystem stellt eine Erweiterung der MIME-Types [BF93, FB96] dar, da neben den allgemeinen Typen wie Image oder Text beliebige andere Typen definiert werden und in Beziehung zueinander gesetzt werden können. Die Menge $\mathcal{T}$ enthält alle im Autorisierungssystem bekannten Typen von Objekten bzw. Objekttypen, die mittels einer Relation $<_{\mathcal{T}}$ geordnet werden können. In [JSSS01] wurde nur eine gemeinsame Hierarchie für Objekte und Typen eingeführt, sodass im Graphen alle Objekte unterhalb des Typs standen. Dies hat zur Folge, dass Verzeichnisstrukturen nicht berücksichtigt werden können. Analog zum Rollenschema aus Abschnitt 3.4 ist die Menge der Objekttypen $\mathcal{T}$ und die Relation $<_{\mathcal{T}}$ in allen Organisationseinheiten und Autorisierungssphären identisch. Ein Beispiel für die möglichen Typen von Objekten und deren hierarchische Strukturierung, mit denen im Folgenden gearbeitet wird, findet sich in Abbildung 3.6. Die entsprechenden Feature Prädikate werden im Abschnitt 3.5.2 definiert.

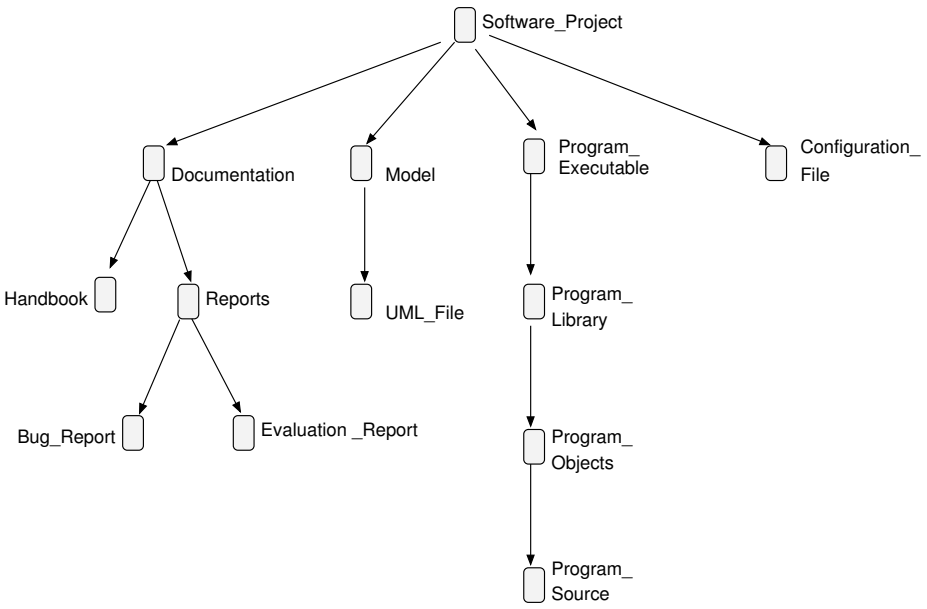



Abbildung 3.6: Objekttypen und deren hierarchische Strukturierung innerhalb der Beispielorganisation

Im Dragon Slayer System sind sogenannte *freie Attribute* eingeführt, um zusätzliche Metainformationen zu den Objekten zu speichern [WS00, Olb01]. Diese Informationen können ebenfalls bei der Zugriffskontrolle berücksichtigt werden. So können diese freien Attribute dazu genutzt werden, ein beliebiges System von "Clearance" Informationen an die Objekte zu binden. Die entsprechenden Feature Prädikate werden im Abschnitt 3.5.3 definiert.

### 3.5.1 Feature Prädikate zur Hierarchie der Objekte

Die Hierarchie der Objekte stellt die Verzeichnisstruktur des Dateisystems dar. Diese Hierarchie kann zum einen durch direkte Zugriffe auf das Dateisystem bei jeder Anfrage überprüft werden. Zum anderen ist die Abbildung der Verzeichnisstruktur in eine eigene Datenstruktur im Autorisierungssystem[22] möglich. Da sämtlich Anfragen über das Autorisierungssystem laufen, werden auch alle möglichen Änderungen in der Verzeichnisstruktur bekannt[23].

Unabhängig von der Art der Implementierung können die entsprechenden Prädikate analog zu denen für die anderen Relationen definiert werden[24].

**Definition 3.17 (direkte Zugehörigkeit von Objekten)**

$$\textbf{objectin}(o_1, o_2)$$

das binäre Prädikat ist gültig, wenn für die Objekte $o_1, o_2 \in \mathcal{OT}$ gilt: $(o_1, o_2) \in <_{\mathcal{O}}$.

Die indirekte Zugehörigkeit von Objekten kann dazu genutzt werden, Zugriffsregeln für Dateien und Objekte, die unterhalb eines bestimmten Verzeichnisses liegen, einfach zu definieren[25].

**Definition 3.18 (indirekte Zugehörigkeit von Objekten)**
Die indirekte Objektbeziehung kann über die folgenden Regeln für $o_1, o_2 \in \mathcal{OT}$ abgefragt werden, wobei $o' \in \mathcal{V}_{\mathcal{O}}$:

$$\textbf{subobject}(o_1, o_2) \leftarrow objectin(o_1, o_2)$$
$$\textbf{subobject}(o_1, o_2) \leftarrow subobject(o_1, o') \& objectin(o', o_2)$$

Bei der Entwicklung der Beispielregeln für diese Arbeit ist ein Feature Prädikat, das die Gleichheit zweier Variablen über Objekte, testet nicht nötig, da dies durch Ableitungsregeln (Def. 5.2) abgedeckt wird. Bei Bedarf wäre es kein Problem, einen Feature Prädikat **equal_object**$(o_1, o_2)$ analog zu den Definitionen 3.2, 3.7 und 3.13 zu definieren.

### 3.5.2 Feature Prädikate zu Typen von Objekten

Neben diesen bereits in den grundlegenden Arbeiten[26] zu Datenbanken und Dateisystemen vorgestellten Feature Prädikaten, werden auch Typen von Objekten vorgestellt. Diese Typen von Objekten werden, wie im vorangegangen Abschnitt 3.5 erläutert, mit der Hierarchie der Objekte vermischt. Dies erscheint für Dateisysteme nicht angebracht, da die Verzeichnisstruktur auf die Objekthierarchie $<_{\mathcal{O}}$ abgebildet wird. Somit wird eine weitere Relation $<_{\mathcal{T}}$ über die Typen der Objekte definiert, die als Hierarchie der Objekttypen bezeichnet wird. Diese kann ebenfalls mit den folgenden Feature Prädikate abgefragt werden.

In den Zugriffsregeln werden Variablen über die Objekttypen verwendet. Diese Variablen für Elemente der Menge $\mathcal{T}$ sind in der Menge $\mathcal{V}_{\mathcal{T}}$ enthalten. Für die Definition der Prädikate werden **Terme über Objekttypen** $\mathcal{TT}$ über die Menge der Variablen über $\mathcal{T}$ und die Menge der Objekttypen definiert ($\mathcal{TT} = \mathcal{V}_{\mathcal{T}} \cup \mathcal{T}$).

Der Typ eines Objektes kann mit dem folgenden Feature Prädikat abgefragt werden, das so auch schon in den grundlegenden Arbeiten (vgl. Abschnitt 2.1.2) definiert wurde.

**Definition 3.19 (Objekttyp)**

$$\textbf{typeof}(o, t)$$

ist ein binäres Prädikat, das dann gültig ist, wenn Objekt $o \in \mathcal{OT}$ den Objekttyp $t \in \mathcal{TT}$ hat.

[22] Diese Vorgehensweise wurde für den Prototyp des Interpreters [Hal05] gewählt.
[23] Ob diese dann auch tatsächlich durchgeführt werden, kann entsprechend überprüft werden.
[24] In den grundlegenden Arbeiten u.a. [JSSB97] (vgl. Abschnitt 2.1.2) und in [WL01] wurde dieses Prädikat mit dirin bezeichnet. Die Umbenennung erfolgt, um ein einheitliches Namensschema für alle Hierarchien zu erhalten.
[25] In vorangegangenen Arbeiten (u.a. [JSSB97, WL01]) wurde dieses Prädikat als in bezeichnet.
[26] u.a. [JSSB97] vgl. Abschnitt 2.1.2

Somit können die Objekttypen, die in Abbildung 3.6 für die Beispielorganisation zu sehen sind, in den Zugriffsregeln genutzt werden, um nicht nur die entsprechende Organisationseinheit des Objektes zu berücksichtigen.

**Beispielregeln 3.4**

$$\mathsf{read}(s,o) \leftarrow \mathsf{role}(s,u,Developer)\&\mathsf{unitof}(o,u)\&\mathsf{typeof}(o,UML\_File)$$
$$\mathsf{read}(s,o) \leftarrow \mathsf{role}(s,u,Developer)\&\mathsf{unitof}(o,u)\&\mathsf{typeof}(o,Program\_Source)$$
$$\mathsf{read}(s,o) \leftarrow \mathsf{role}(s,u,Developer)\&\mathsf{unitof}(o,u)\&\mathsf{typeof}(o,Program\_Library)$$

Auf Grund dieser Beispielregel, hat ein Benutzer in der Rolle Developer Lesezugriff auf alle Objekte in seiner Organisationseinheit, die vom Typ UML_File sind, außerdem auf alle Objekte vom Typ Program_Source und durch die dritte Regel auf alle vom Typ Program_Library.

Da es jedoch sehr umständlich ist, für jeden Objekttyp eine eigene Regel aufzustellen, ist die Relation der Objekttypen $<_{\mathcal{T}}$ sehr hilfreich, um Regeln für bestimmte Mengen von Objekttypen aufzustellen, wie z.B. alle Arten von Programmen. Dies war in den vorangegangenen Arbeiten (u.a. [JSS97]) nur bedingt möglich.

Die Beziehung zwischen den Objekttypen kann über die folgenden beiden Featureprädikate abgefragt werden.

**Definition 3.20 (direkte Objekttypenbeziehung)**

$$\textbf{typein}(t_1,t_2)$$

ist ein binäres Prädikat, bei dem $(t_1,t_2) \in <_{\mathcal{T}}$ für die beiden Objekttypen $t_1,t_2 \in \mathcal{TT}$ gilt.

Die indirekte Beziehung zwischen Objekttypen kann analog zu den anderen indirekten Beziehungen für Autorisierungssphären, Organisationseinheiten oder Rollen (vgl. Definitionen 3.4 3.9, 3.15) definiert werden.

**Definition 3.21 (indirekte Beziehung zwischen Objekttypen)**
Die indirekte Beziehung zwischen Objekttypen kann über die folgenden Regeln für $t_1,t_2 \in \mathcal{TT}$ abgefragt werden, wobei $t' \in \mathcal{V}_{\mathcal{T}}$:

$$\textbf{subtype}(t_1,t_2) \leftarrow \mathsf{typein}(t_1,t_2)$$
$$\textbf{subtype}(t_1,t_2) \leftarrow \mathsf{subtype}(t_1,t')\&\mathsf{typein}(t',t_2)$$

Analog zu den vorangegangenen Mengen wie Rollen und Organisationseinheiten ist es auch hilfreich überprüfen zu können, ob zwei Terme $t_1,t_2 \in \mathcal{TT}$ bei der Auswertung die gleiche Belegung haben.

**Definition 3.22 (Objekttyp Gleichheit)**
Das binäre Prädikat

$$\textbf{equal\_type}(t_1,t_2)$$

ist genau dann gültig, wenn die beiden Terme über Objekttypen $t_1,t_2 \in \mathcal{TT}$ die gleiche Belegung in $\mathcal{T}$ haben.

An Stelle einer ganzen Reihe von Regeln wie es in den Beispielregeln 3.4 nötig war kann, jetzt einem Subjekt in der Rolle Developer sehr einfach Leserecht über alle Objekte, die zum Bereich Programmquellen unterhalb des Objekttyps Program_Executable gehören, gegeben werden.

**Beispielregeln 3.5**

$$\begin{aligned}\mathsf{read}(s,o) \leftarrow\ & \mathsf{role}(s,u,Developer)\&\mathsf{unitof}(o,u)\& \\ & \mathsf{typeof}(o,t)\&\mathsf{subtype}(t,Program\_Executables)\end{aligned}$$
$$\begin{aligned}\mathsf{read}(s,o) \leftarrow\ & \mathsf{role}(s,u,Developer)\&\mathsf{unitof}(o,u)\& \\ & \mathsf{typeof}(o,t)\&\mathsf{subtype}(t,Documentation)\end{aligned}$$
$$\mathsf{read}(s,o) \leftarrow \mathsf{role}(s,u,Developer)\&\mathsf{unitof}(o,u)\&\mathsf{typeof}(o,Documentation)$$

Die zweite und dritte Regel im obigen Regelsatz geben einem Developer Lesezugriff auf alle Objekte, die etwas mit der Dokumentation zu tun haben. Dies sind alle Objekttypen unterhalb von Documentation einschließlich. Letzteres erfordert noch eine zusätzliche Regel, da bisher kein Feature Prädikat **subtypeeq**$(t_1, t_2)$ analog zu subroleeq$(r_1, r_2)$ (vgl. Def. 3.16) definiert wurde.

An Stelle dieses Feature Prädikates ist es speziell für Objekttypen sinnvoll, direkt spezifizieren zu können, ob ein Objekt aus einer bestimmten Menge von Objekten stammt.

**Definition 3.23 (erweiterte Beziehung von Objekttypen)**
Die Typzugehörigkeit eines Objektes kann über die folgenden Regeln für $o \in \mathcal{OT}$ und $t' \in \mathcal{V}_T$ abgefragt werden:

$$\textbf{subtypeof}(o, t) \leftarrow \text{typeof}(o, t)$$
$$\textbf{subtypeof}(o, t) \leftarrow \text{typeof}(o, t')\&\text{typein}(t', t)$$
$$\textbf{subtypeof}(o, t) \leftarrow \text{typeof}(o, t')\&\text{subtype}(t', t)$$

Das Feature Prädikat ist genau dann gültig, wenn das Objekt $o$ vom Objekttype $t$ oder einem Objekttyp $t'$ der dem Objekttype $t$ gemäß der Relation $<_{\mathcal{T}}$ untergeordnet ist ($t' <_{\mathcal{T}} t$). Somit kann die Zahl der Feature Prädikate in einer Zugriffsregel zugunsten der Übersichtlichkeit reduziert werden. Falls die Menge der Objekte nur solche enthalten soll, deren Objekttyp echt unterhalb eines Objekttyps $t$ sein soll, kann entweder die erste Regel aus Regelsatz 3.6 genommen werden oder die erste Regel des folgenden Regelsatzes. Die zweite Regel ersetzt die beiden anderen Regeln aus Regelsatz 3.6.

**Beispielregeln 3.6**

$$\begin{aligned}\text{read}(s, o) \leftarrow &\text{role}(s, u, Developer)\&\text{unitof}(o, u)\& \\ &\text{subtypeof}(o, Program\_Executables)\&\neg\text{typeof}(o, Program\_Executables)\\ \text{read}(s, o) \leftarrow &\text{role}(s, u, Developer)\&\text{unitof}(o, u)\&\text{subtypeof}(o, Documentation)\end{aligned}$$

Eigentlich könnte der Typ eines Objektes auch als ein Attribut aufgefasst werden, dieses hat jedoch eine gewisse Sonderstellung, da hiermit auch eine Hierarchie verbunden ist, was bei den folgenden Attributen von Objekten nicht der Fall ist.

### 3.5.3 Feature Prädikate zu Attributen von Objekten

Wie zu Beginn des Kapitels erwähnt, sind die Autorisierungssphären für die Zugriffsstrategien der Objekte in ihren zugehörigen Organisationseinheiten verantwortlich. Darüber hinaus besteht in allen Dateisystemen für UNIX, die den Zugriff mittels MAC realisieren, die Möglichkeit, dem Besitzer eines Objektes besondere Rechte einzuräumen. Dieses soll mit Hilfe des folgenden Feature Prädikates auch in dem vorgestelltem Autorisierungssystem der Fall sein.

**Definition 3.24 (Besitz von Objekten)**

$$\textbf{owner}(s, o)$$

ist ein binäres Prädikat, das dann gültig ist, wenn der Benutzer $s \in \mathcal{ST}$ als Besitzer für das Objekt $o \in \mathcal{OT}$ eingetragen ist.

Somit kann durch die folgende Regel jedem Besitzer eines Objektes Lesezugriff gewährt werden, unabhängig davon, ob er in der aktiven Rolle Zugriff auf diesen Objekttyp hat.

**Beispielregeln 3.7**

$$\text{read}(s, o) \leftarrow \text{owner}(s, o)\&\text{unitof}(o, u)\text{activ\_role}(s, u, r)$$

Da $r$ eine freie Variable ist, kann die aktive Rolle des Subjektes beliebig sein, solange sie in der gleichen Organisationseinheit ist, zu der auch das Objekt gehört. Falls die Besitzer der Objekte generell Lesezugriff haben sollen, müsste die Regel wie folgt laut: $\mathsf{read}(s, o) \leftarrow \mathsf{owner}(s, o)$

Eine digitale Signatur über den Inhalt eines Dokumentes kann dazu verwendet werden, die Authentizität dieses Inhaltes zu bestätigen und damit weitere Zugriffe von anderen Subjekten zu erlauben. An dieser Stelle wird davon ausgegangen, dass eine Public-Key Infrastruktur existiert, die einen digitalen Schlüssel einem Subjekt $s$ zuordnen kann.

**Definition 3.25 (Objekt Signaturen)**

**signature($o, s$)**

ist ein binäres Prädikat, dass dann gültig ist, wenn zum Objekt $o$ eine digitale Signatur gespeichert wurde, die vom privaten Schlüssel erzeugt wurde, der dem Subjekt $s$ zugeordnet wird.

Die digitale Signatur kann dazu dienen, einen bestimmten Status eines Objektes zu dokumentieren und abhängig davon anderen Subjekten den Zugriff zu erlauben[27].

**Beispielregel 3.8**

$$\begin{aligned}\mathsf{read}(s, o) \leftarrow & \mathsf{typeof}(o, UML\_File)\& \\ & \mathsf{role}(s, u, Programmer)\&\mathsf{unitof}(o, u)\& \\ & \mathsf{signature}(o, s_1)\&\mathsf{role}(s_1, u, Developer)\end{aligned}$$

Auf Grund dieser Beispielregel kann ein Benutzer mit der aktiven Rolle Programmer ein Objekt vom Typ UML_File lesen, wenn dieses eine Signatur eines Benutzers mit der Rolle Developer enthält und alle zur gleichen Organisationseinheit $u$ gehören. Dabei kann das Subjekt, dessen Signatur am Objekt ist, im Moment der Auswertung der Regel auch eine andere Rolle aktiviert haben. Geprüft wird durch diese Regel lediglich, ob es diese Rolle zugewiesen hat. Um diese Vorgehensweise zu vervollständigen, muss mit weiteren Regeln festgelegt werden, dass nur Subjekte, die diese Rolle auch aktiviert haben, eine Signatur anbringen dürfen.

Im Dragon Slayer System werden sogenannte *freie Attribute* eingeführt, um zusätzliche Metainformationen zu den Objekten zu speichern [WS00, Olb01]. Um diese Attribute abfragen zu können, wird eine Menge von Vergleichsoperatoren $\mathcal{OP} = \{=, \neq\}$ benötigt, mit denen getestet wird, ob ein Attribut mit einem bestimmten Wert übereinstimmt oder nicht[28].

**Definition 3.26 (Objektattribute)**

**attribute($o, al, op, av$)**

das Vierfach–Prädikat ist gültig, wenn das Objekt $o \in \mathcal{OT}$ ein Attribut mit dem Label $al$ enthält und entsprechend dem Operator $op \in \mathcal{OP}$ der des Wert dieses Attributes mit dem String $av$ übereinstimmt bzw. nicht übereinstimmt.

In vielen Beispielen, vor allem solchen, die zeigen wie Discretionary Access Control (DAC) durch einen Rollen Basierten Ansatz realisiert werden kann [San98, OSM00], wird auch mit der Freigabestufe oder Clearance der Objekte gearbeitet. Allerding ist in [San98] nur von einem Clearance Label $\lambda(o)$ für jedes Objekt die Rede, ohne das auf weitere Details bezüglich der Realisierung eingegangen wird. Diese Labels könnten leicht mit den oben angegebenen Attributen realisiert werden. Jedoch müsste gegebenenfalls noch eine weitere Hierarchie über diese Clearance Label eingeführt werden sowie entsprechende Feature Prädikate, um diese abzufangen. Da die Zugriffsrechte

[27] Natürlich kann man auch den Zugriff bei einer vorhanden Signatur verbieten, allerdings wurde bisher mit dem Beispielprädikat read(s,o) erst die Möglichkeit geschaffen Lesezugriffe zu erlauben.

[28] Vergleichsoperatoren wie $<$ werden nicht definiert, da sie voraussetzen, dass auf den Attributwerten eine (Teil-) Ordnung definiert ist, was aber nicht der Fall ist.

in diesem Modell auch von weiteren Eigenschaften wie Typ oder Verzeichnishierarchie abhängig gemacht werden können, wird auf diese Hierarchie zunächst verzichtet.

## 3.6 Zugriffsmethoden

Bisher wurde in allen Beispielen nur der Lesezugriff geregelt. Darüber hinaus sind weitere Zugriffsmethoden nötig. Zusätzlich zum Leserecht (read) können neben dem üblichen Schreibrecht (write) und Ausführungsrecht (execute) auch weitere Rechte zum Anhängen von Informationen an eine Datei (append), zum Signieren einer Datei (sign) oder zum Ändern der Attribute (modify) betrachtet werden.

Dabei macht es keinen Sinn, eine Relation über diese Zugriffsmethoden zu definieren, da zwei unterschiedliche Methoden nicht grundsätzlich voneinander abhängig sind. So hat der Entzug des Leserechtes (-read) nichts mit dem Gewähren des Schreibrechtes (+write) zu tun. Ein konkretes Beispiel hierfür wäre in einer UNIX-Umgebung die Maildatei eines Benutzers (z.B. `/var/spool/mail/bob`), in welche alle Benutzer schreiben dürfen, aber nur der Besitzer lesen darf. Daher ist eine automatische Ableitung von Zugriffsrechten über eine Hierarchie der Zugriffsmethoden nicht zwingend möglich.

Da die Zugriffsmethoden des Autorisierungssystems möglichst flexibel sein sollen, wird von keiner konkreten Menge an Zugriffsrechten ausgegangen, sondern die obige Aufzählung als ein mögliches Beispiel verwendet. Die Menge $\mathcal{AC}$ enthält alle im System definierten Zugriffsmethoden. Nicht nur für die Definition der Access Prädikate im Kapitel 5, sondern sowohl für die Trust Level Prädikate im Kapitel 4 als auch für die folgenden Feature Prädikate werden Terme über diese Zugriffsmethoden benötigt. In den konkreten lokalen Zugriffsregeln wird meistens die Zugriffsmethode mit angegeben, trotzdem wird eine Menge von Variablen $\mathcal{V}_{\mathcal{AC}}$ über die Menge der Zugriffsmethoden benötigt.

Die Terme über die Zugriffsmethoden $\mathcal{ACT}$ sind als Vereinigung dieser Variablen und der Menge Zugriffsmethoden definiert ($\mathcal{ACT} = \mathcal{V}_{\mathcal{AC}} \cup \mathcal{AC}$).

Ein Sicherheitsprinzip, das von [FCK95] in die rollenbasierte Zugriffskontrolle eingeführt wurde, ist die "Operational Separation of Duty", bei der festgelegt und durchgesetzt wird, dass bestimmte Aufgaben von verschiedenen Benutzern durchgeführt werden. Ein gängiges Beispiel ist die Kontoauszahlung in einer Bank, bei der eine zweite Person gegenzeichnen muss. Somit wird sichergestellt, dass nicht derselbe Mitarbeiter eine Auszahlung initiiert, gegenzeichnet und später auch die Auszahlung durchführt. Durch diese Maßnahme wird der Betrug durch einen einzelnen Mitarbeiter verhindert.

Um diese Maßnahmen modellieren zu können, reichen die bisherigen Feature Prädikate nicht aus[29]. Daher wird im vorliegenden Autorisierungssystem die Historie der Zugriffe durch ein Subjekt dokumentiert. Da jede Zugriffsanfrage durch das Autorisierungssystem bestätigt werden muss, stellt es kein Problem dar, jede Auswertung der Regel, die erfolgreich abgeschlossen wird, zu speichern. Problematisch ist nur die Dauer dieser Speicherung, da die Historie vor allem bei einer großen Organisation schnell einen Umfang annimmt, der schwierig zu handhaben ist.

Diese gespeicherte Historie kann mit den folgenden Feature Prädikaten bei den Zugriffsregeln berücksichtigt werden.

**Definition 3.27 (vorangegangene Zugriffe)**

$$\textbf{done}(s, o, a)$$

das Dreifach–Prädikat ist gültig, wenn das Subjekt $s \in \mathcal{ST}$ mittels der Methode $a \in \mathcal{AC}$ auf das Objekt $o \in \mathcal{OT}$ zugegriffen hat.

[29] Über die Signatur von Objekten ließe sich das obige Bankbeispiel unter Umständen realisieren. Allerdings wären die späteren Beispiele nicht möglich.

Somit kann beispielsweise mit den folgenden Zugriffsregeln festgelegt werden, dass nur Subjekte mit der aktiven Rolle Programmer die Objekte vom Objekttyp Program_Source lesen dürfen, wenn sie zuvor die Objekte vom Objekttyp Bug_Report gelesen haben. Mit der zweiten Regel wird festgelegt, dass ein Subjekt in der aktiven Rolle Programmer nur dann lesend auf ein Objekt vom Objekttyp UML_File zugreifen darf, bis ein Subjekt in der Rolle Developer in ein Objekt vom Objekttyp Documentation geschrieben hat[30].

**Beispielregeln 3.9**

$$\begin{aligned}\mathsf{read}(s,o) \leftarrow & \mathsf{activ\_role}(s,u,Programmer)\&\mathsf{unitof}(o,u)\&\mathsf{typeof}(o,Program\_Source)\& \\ & \mathsf{done}(s,o',read)\&\mathsf{unitof}(o',u)\&\mathsf{typeof}(o',\mathsf{Bug\_Report}) \\ \mathsf{read}(s,o) \leftarrow & \mathsf{activ\_role}(s,u,Programmer)\&\mathsf{unitof}(o,u)\&\mathsf{typeof}(o,UML\_File)\& \\ & \neg\mathsf{done}(s',o',write)\&\mathsf{role}(s',u,Developer)\&\mathsf{unitof}(o',u)\&\mathsf{typeof}(o',Documentation)\end{aligned}$$

Die zweite Regel aus dem obigen Beispiel zeigt, dass die Historie der Zugriffe nicht gelöscht werden kann, sobald eine Session beendet wurde. Die Problematik der Speicherung der Historie wird im Abschnitt 9.3.2 noch einmal aufgegriffen.

Um die Suche in der Historie einschränken zu können, erscheint es sinnvoll, einen Zeitfaktor in das Feature Prädikat done einzubauen.

**Definition 3.28 (vorangegangene Zugriffe bis zu einem Zeitpunkt)**

$$\mathbf{done\_since}(s,o,a,d)$$

das vierfach Prädikat ist gültig, wenn das Subjekt $s \in \mathcal{ST}$ mittels der Methode $a \in \mathcal{AC}$ auf das Objekt $o \in \mathcal{OT}$ innerhalb der letzten $d \in I\!N$ Zeiteinheiten zugegriffen hat.

Falls es möglich ist, bei allen Zugriffsregeln die vorangegangenen Zugriffe mit einzubeziehen, auf einen bestimmten Zeitraum einzuschränken, kann *nach* dem Verstreichen des maximalen Zeitpunktes $d$ der Zugriff aus der Historie gelöscht werden.

**Beispielregel 3.10**

$$\begin{aligned}\mathsf{read}(s,o) \leftarrow & \mathsf{activ\_role}(s,u,Programmer)\&\mathsf{unitof}(o,u)\&\mathsf{typeof}(o,Program\_Source)\& \\ & \mathsf{done}(s,o',read,60)\&\mathsf{unitof}(o',u)\&\mathsf{typeof}(o',\mathsf{Bug\_Report})\end{aligned}$$

Für die Beispielregeln 3.10 müssen im Gegensatz zu der ersten Beispielregel aus 3.9 nach 60 Zeiteinheiten (Minuten) die vergangenen Lesezugriffe nicht mehr gespeichert werden, sodass dadurch die Zugriffslisten beschränkt werden können.

Da bisher nur read als Beispielprädikat für einen Lesezugriff definiert wurde, lassen sich keine komplexeren Bedingungen als die bereits gegebenen Beispielregeln zur Erläuterung der einzelnen Feature Prädikate spezifizieren. Deshalb kann an dieser Stelle nur auf die Beispielregeln in Kapitel 5 verwiesen werden. Im folgenden Kapitel 4 wird gezeigt, wie die Gegebenheiten einer verteilten Umgebung bei der Zugriffskontrolle berücksichtigt werden können. Anschließend wird im Kapitel 5 gezeigt, wie ein Autorisierungsteam mit den in diesem Kapitel vorgestellten Feature Prädikaten komplexe Bedingungen spezifizieren kann, unter denen bestimmte Zugriffsprivilegien gelten sollen. Bei der Definition dieser Feature Prädikate flossen auch die Erfahrungen aus der Entwicklungen eines verteilten Dateisystems [WL99, WS00] mit ein.

Bei der Auflösung sowohl von Vererbungskonflikten als auch von Koordinierungskonflikten können diese Feature Prädikate dazu verwendet werden, genauere Abgrenzungen zwischen den unterschiedlichen Vorgehensweisen zu spezifizieren.

[30] Dies soll in diesem Beispiel das Ende der Bearbeitungszeit der Programme markieren.

Darüber hinaus werden vor allem die Feature Prädikate zu den Autorisierungssphären dazu verwendet, die Vererbung von Zugriffsregeln und die Koordinierung von Zugriffen zwischen Organisationseinheiten aus unterschiedlichen Autorisierungssphären direkt in Datalog zu modellieren.

Für die einfachere Definition von weiteren Prädikaten enthält die Menge $\mathcal{F}$ alle Feature Prädikate, die in diesem Kapitel definiert wurden, unabhängig davon, ob es sich dabei gemäß Datalog um Extensional oder Intensional Prädikate handelt. Abschließend findet sich in Tabelle 3.1 eine Übersicht der in diesem Kapitel definierten Terme sowie der dazugehörigen Mengen und Mengen von Variablen sowie, falls vorhanden, der entsprechenden Relation.

| | Terme | | Variablen | | Mengen | Relation |
|---|---|---|---|---|---|---|
| Organisationseinheiten | $\mathcal{UT}$ | = | $\mathcal{V}_{\mathcal{U}}$ | $\cup$ | $\mathcal{U}$ | $<_{\mathcal{U}}$ |
| Autorisierungssphären | $\mathcal{AST}$ | = | $\mathcal{V}_{\mathcal{AS}}$ | $\cup$ | $\mathcal{AS}$ | $<_{\mathcal{AS}}$ |
| Rollen | $\mathcal{RT}$ | = | $\mathcal{V}_{\mathcal{R}}$ | $\cup$ | $\mathcal{R}$ | $<_{\mathcal{R}}$ |
| Subjekte | $\mathcal{ST}$ | = | $\mathcal{V}_{\mathcal{S}}$ | $\cup$ | $\mathcal{S}$ | |
| Objekte | $\mathcal{OT}$ | = | $\mathcal{V}_{\mathcal{O}}$ | $\cup$ | $\mathcal{O}$ | $<_{\mathcal{O}}$ |
| Objekttypen | $\mathcal{TT}$ | = | $\mathcal{V}_{\mathcal{T}}$ | $\cup$ | $\mathcal{T}$ | $<_{\mathcal{T}}$ |
| Zugriffsmethoden | $\mathcal{ACT}$ | = | $\mathcal{V}_{\mathcal{AC}}$ | $\cup$ | $\mathcal{AC}$ | |

Tabelle 3.1: Terme und dazugehörige Variablenmengen und Mengen

# Kapitel 4

# Berücksichtigung der lokalen Aspekte

In rollenbasierten Zugriffssystemen wie dem RBAC Standard (vgl. Abschnitt 2.1.1) wird als Subjekt $s$ immer der Prozess bezeichnet, der unter einem bestimmten Benutzerkonto läuft. Weitere Informationen, unter anderem wo dieser Prozess läuft oder auf welche anderen Prozesse er für den Zugriff zurückgreifen soll, werden nicht berücksichtigt. Für Datenbankanwendungen, bei der alle Benutzer auf einen zentralen Server oder auch Servercluster zugreifen, ist dies hinreichend. In einer verteilten Umgebung muss bei der Autorisierung auch die Sicherheit und Vertrauenswürdigkeit der beteiligten Komponenten berücksichtigt werden. Während sich bei einem zentralen Ansatz die Benutzer auf dem gleichen Server[1] anmelden (müssen), auf dem auch die Daten gespeichert sind und für die Zugriffsanfrage bei Bedarf aufgearbeitet werden, können dies in einer verteilten Umgebung verschiedene Server bzw. Hosts sein, die völlig unabhängig voneinander sind. In großen Organisationen können diese Hosts über mehreren Standorte verteilt sein, die dann zum Teil über öffentliche Leitungen miteinander kommunizieren. Außerdem müssen beim Zugriff aus anderen Staaten auf bestimmte (z.B. personenbezogene) Daten verschiedene rechtliche Grundlagen, wie das Datenschutzgesetz [BDS03], geachtet werden.

Motiviert wurde die erste Arbeit [WL04b] auch durch die Szenarien, die im Rahmen des EU–Rahmenprogrammes entwickelt wurden [DBS+01]. Anhand dieser Beschreibungen wird deutlich, wie wichtig eine Berücksichtigung der Umgebung ist, in der eine Zugriffsanfrage gestellt wird.

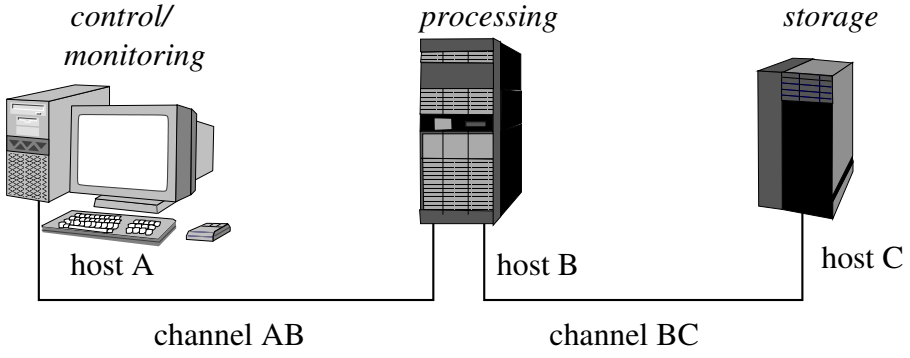



Abbildung 4.1: lokale Aspekte

In Abbildung 4.1 ist solch eine verteilte Umgebung skizziert: Zum einen der Host $A$, an dessen Konsole der Benutzer eingeloggt ist und von dem alle Zugriffsanfragen letztlich ausgehen sowie die

[1] Ein Servercluster bestehend aus Datenbankservern und Applikationsservern, die an einem Ort aufgestellt sind und zentral administriert werden, fällt für die weitere Betrachtung in die gleiche Kategorie wie ein einzelner Server.

Ergebnisse geliefert werden, auf der anderen Seite der Host $C$, auf dem die Objekte gespeichert sind und zu denen ein Zugriff mit einer bestimmten Methode autorisiert werden soll. Dazwischen kann unter Umständen noch ein weiterer Host $B$ liegen, auf dem die Applikationen laufen, die die Daten in den Objekten verarbeiten sollen. Diese Applikationen können auch direkt auf den Hosts $A$ oder $C$ laufen. Im Gegensatz dazu kann die Applikation auch auf mehrere Objekte zugreifen, die alle auf verschiedenen Hosts liegen.

Da sowohl im RBAC Standard Modell, wie auch im vorliegenden Modell, immer eine Zugriffsanfrage nach der anderen autorisiert wird, muss immer nur ein Host betrachtet werden, auf dem das Objekt für die aktuelle Zugriffsanfrage liegt. Für die Entwicklung der Datalog Sprache, mit der die Autorisierungsregeln spezifiziert werden, wird daher von drei Hosts ausgegangen, wie sie in Abbildung 4.1 zu sehen sind.

Zwischen den Hosts $A$ und $B$ besteht ein Kommunikationskanal, über den die Applikation auf Host $B$ gesteuert wird und die Ergebnisse an den Benutzer, der auf Host $A$ eingeloggt ist, geliefert werden. Der Kommunikationskanal zwischen den Hosts $B$ und $C$ dient zum Austausch der Daten, die in den Objekten gespeichert sind, sowie für die dazugehörigen Meta–Informationen, wie Attribute oder Signaturen (vgl. Abschnitt 3.5).

Für die Autorisierung in einer verteilten Umgebung sollte die Sicherheit und Vertraulichkeit aller Komponenten berücksichtigt werden, um die Vertraulichkeit und Authentizität der Daten nicht zu gefährden. Objekte, die für die Organisation besonders kritisch sind, sollten nur auf Hosts gespeichert werden, die besonders gesichert sind, sei es physikalisch, aber auch durch sichere Betriebssysteme oder verschlüsselte Dateisysteme. Allerdings ist solch ein Aufwand sinnlos, wenn ein autorisierter Benutzer auf die Inhalte dieser Objekte zugreifen kann, ohne dass die verwendete Kommunikationsverbindung entsprechend kryptographisch abgesichert ist. Ansonsten können unautorisierte Benutzer die Daten mithören oder gar verfälschen. Wenn die Organisation über mehrere Standorte verteilt ist, können die Daten bei einer unzureichend gesicherten Kommunikationsverbindung auch für Außenstehende zugänglich werden.

Ähnliches kann auch für den Host gelten, auf dem die Objekte bearbeitet werden, da hier meistens den Administratoren vertraut werden muss, dass die Applikationen korrekt installiert sind und die Informationen nicht abgehört werden[2]. Zudem muss sichergestellt werden, dass die Applikationen auch korrekt arbeiten und keinen fehlerhaften oder böswilligen Code enthalten.

Unter den gleichen Gesichtspunkten muss der Host betrachtet werden, an dessen Konsole der Benutzer eingeloggt ist. Da hier sowohl die Inhalte der Objekte, oder Teile davon, als auch die Ergebnisse der Bearbeitung angezeigt werden[3], muss den Administratoren des Hosts entsprechend vertraut werden. Für besonders kritische Objekte könnte es ratsam sein, dass die Benutzer sich nicht an jedem beliebigen Ort befinden dürfen, wenn sie darauf zugreifen wollen. Außerdem müssen bei einem Zugriff über Staatsgrenzen hinweg zum Teil Gesetze wie die zum Datenschutz beachtete werden.

Um diese Sicherheitsaspekte der verschiedenen Komponenten bei der Autorisierung berücksichtigen zu können, werden im Folgenden sogenannte **Kontext Feature Prädikate** ($\mathcal{CF}$) definiert. Bestimmte Konstellationen, wie der Aufbau und die Verschlüsselung einer Kommunikationsverbindung, aber auch der physikalische Ort eines Hosts und Informationen über seine Administratoren können zu sogenannten Vertrauensstufen zusammengefasst werden. Diese Vertrauensstufen sollten dezentral in den Autorisierungssphären vergeben werden. Zur Modellierung dieser Vertrauensstufen werden sogenannte **Kontext Trust Prädikate** ($\mathcal{CTL}$) definiert.
Zur Erläuterung dieser Prädikate wird das Beispielprädikat `read` so erweitert, dass auch diese beiden Mengen von Prädikaten im Rumpf verwendet werden dürfen: $\mathbf{read}(s, o) \leftarrow \mathcal{F}\&\mathcal{CF}\&\mathcal{CTL}$

Im nächsten Abschnitt 4.1 werden zunächst die Vertrauensstufen näher erläutert. In den daran anschließenden Abschnitten wird eine Menge von Prädikaten definiert, die es erlaubt die Eigen-

[2]Zum Beispiel indem direkt der Inhalt des Hauptspeichers oder der Auslagerungsdateien ausgelesen wird.
[3]Zum Beispiel durch eine Umleitung der Ausgabe Applikationen wie es bei X11 möglich ist.

schaften von Knoten (Abschnitt 4.2), der Kommunikationskanäle zwischen Knoten (Abschnitt 4.3) sowie die Eigenschaften der verarbeitenden Applikationen (Abschnitt 4.4) bei der Zugriffskontrolle zu berücksichtigen.

## 4.1 Vertrauensstufen

Um die einzelnen Komponenten clustern zu können, bietet sich die Einführung von Vertrauensstufen an, deren Zuteilung sich nach dem Standort, Eigenschaften der Komponente oder Ähnlichem richtet. Dabei wird diese Zuteilung auch dezentral von den Autorisierungssphären verwaltet. Die Bandbreite reicht von Vertrauensstufen, die kennzeichnen, ob einzelne Hosts zur Organisationseinheit gehören ("Internal" vs. "External"), ob die Konfiguration vertrauenswürdig ist ("Trusted" vs. "Insecure"), bis zu Kombinationen aus diesen Einstufungen ("Internal_Trusted"). Darüber hinaus kann bei Kommunikationskanälen der Grad der kryptographischen Absicherung wiedergegeben werden ("Secured", "Crypto_Secured", "Highly_Secured").

Dieses Vorgehen unterscheidet sich von der in [Her03] verwendeten und in [Jøs99] vorgestellten Methode, bei der die Vertrauensstufen aufgrund dreier Parameter ("belief", "disbelief" und "uncertainty") berechnet werden. In [BBK94] werden zwei Arten von Trust definiert, die zum einen das direkte Vertrauen ("direct trust") in die Einhaltung bestimmter Vertrauensklassen durch diese Entität wiedergibt und das Empfehlungsvertrauen ("recommendation trust") in die Einschätzungen Dritter bezüglich einer Vertrauensklasse durch diese Entität wiederspiegelt. Von den in [YKB93] vorgestellten Vertrauensklassen wären nur "KS (keeping information secret)" und "ID (provide identification)" direkt anwendbar.
Für die vorliegende Arbeit erscheinen diese Vertrauensklassen zu eng, daher wird die Möglichkeit einer flexiblen Festlegung innerhalb einer Organisation vorgezogen. Anstelle der Berechnung der Vertrauensstufen ist für das Anwendungsgebiet eine eindeutige Einstufung aufgrund bestimmter Fakten sinnvoller.

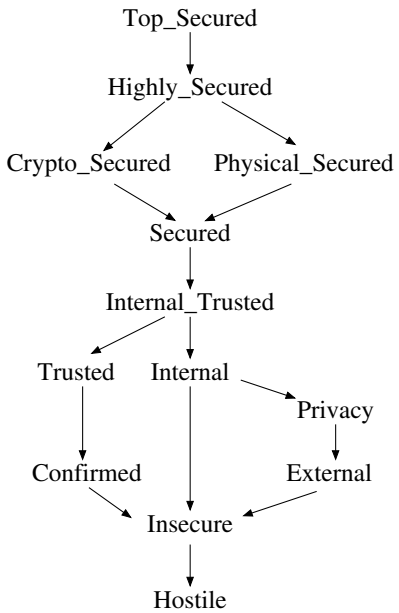



Abbildung 4.2: Beispiel für eine Vertrauensstufenhierarchie

Die Menge $\mathcal{TL}$ enthält alle Vertrauensstufen, die in einer Organisation bekannt sind. Um einfacher modellieren zu können, sodass beispielsweise Knoten, die zu einer bestimmten Menge von Vertrauensstufen gehören für die Speicherung von Objekten zugelassen sind, wird eine Halbord-

nung $<_{\mathcal{TL}}$ über die Vertrauensstufen definiert. Diese wird auch als Hierarchie der Vertrauensstufen bezeichnet. In Abbildung 4.2 ist ein Beispiel für eine solche Hierarchie der Vertrauensstufen gegeben, die auch im Folgenden verwendet wird.

Damit Autorisierungssphären auf die Einstufungen anderer Autorisierungssphären zurückgreifen können, muss die Menge der Vertrauensstufen $\mathcal{TL}$ und die entsprechende Halbordnung $<_{\mathcal{TL}}$ innerhalb der gesamten Organisation bekannt und identisch sein. Außerdem muss mit jeder Vertrauensstufe in allen Autorisierungssphären der gleiche Sicherheitsstandard assoziiert werden.

Analog zu den Rollen und Objekttypen (vgl. Abschnitt 3.4 bzw. 3.5) werden einzelne Elemente in manchen Autorisierungssphären nicht genutzt. Eine organisationsweite einheitliche Hierarchie der Vertrauensstufen verhindert fehlerhafte Einstufungen, die auftreten würden, wenn bestimmte Elemente in einer Autorisierungssphäre nicht definiert und somit Pfade unterbrochen sind. So würde es in der Beispielhierarchie in Abbildung 4.2 beim Fehlen der Stufe Privacy keine Verbindung mehr zwischen External und Internal geben. Veränderte Hierarchien würden zu erheblich größeren Komplikationen führen, zum Beispiel, wenn Privacy in einer Autorisierungssphäre statt an der angegebenen Position oberhalb von Internal eingeordnet würde.

### 4.1.1 Feature Prädikate zu Vertrauensstufen

Die Hierarchie der Vertrauensstufen ist in allen Autorisierungssphären identisch, um so die oben genannten Probleme zu vermeiden, sodass die folgenden Kontext Feature Prädikate bei der Auswertung in allen Autorisierungssphären das gleiche Ergebnis liefern.

Für die Modellierung der Autorisierungsregeln wird eine Menge von Variablen über die Vertrauensstufen benötigt, die mit $\mathcal{V}_{\mathcal{TL}}$ bezeichnet wird. Zur Definition der verschiedenen Prädikate werden Terme über Vertrauensstufen ($\mathcal{TLT}$) als Vereinigung der Variablen und Konstanten über Vertrauensstufen definiert ($\mathcal{TLT} = \mathcal{V}_{\mathcal{TL}} \cup \mathcal{TL}$).

Die Feature Prädikate für die Beziehungen zwischen Vertrauensstufen werden analog zu den entsprechenden Hierarchien für Rollen und Objekttypen (vgl. Abschnitt 3.4.2 bzw. 3.5.2) definiert.

**Definition 4.1 (direkte Beziehung von Vertrauensstufen)**

$$\textbf{levelin}(tl_1, tl_2)$$

ist ein binäres Prädikat, das bei $(tl_1, tl_2) \in <_{\mathcal{TL}}$ für die beiden Vertrauensstufen $tl_1, tl_2 \in \mathcal{TLT}$ gilt.

Die indirekte Beziehung, ob eine Vertrauensstufe in der Hierarchie unterhalb einer anderen angeordnet ist, wird wiederum rekursiv definiert.

**Definition 4.2 (indirekte Beziehung von Vertrauensstufen)**
Die indirekte Beziehung von Vertrauensstufen kann über die folgenden Regeln für $tl_1, tl_2 \in \mathcal{TLT}$ abgefragt werden,wobei $tl' \in \mathcal{V}_{\mathcal{TL}}$:

$$\textbf{sublevel}(tl_1, tl_2) \leftarrow \text{levelin}(tl_1, tl_2)$$
$$\textbf{sublevel}(tl_1, tl_2) \leftarrow \text{sublevel}(tl_1, tl')\&\text{levelin}(tl', tl_2)$$

Die Menge $\mathcal{CF}_{\mathcal{TL}}$ enthält diese beiden Kontext Feature Prädikate, die bei Kontext Trust Prädikaten eingesetzt werden können.

## 4.2 Vertrauenswürdigkeit der Hosts

Um die lokalen Aspekte bei der Autorisierung zu berücksichtigen, ist es zunächst wichtig festzustellen, an welchem Host ein Benutzer angemeldet ist, da hier alle Daten zusammenlaufen und auch kritische Informationen zugänglich sind. Daher muss bei der Autorisierung jeder Rechner eines Systems identifiziert werden können. Dies kann entweder über eindeutige Identifizierungsmerkmale der Hardware geschehen oder über Zertifikate, die die Identität des Rechners bestätigen. Sei $\mathcal{H}$ die Menge aller Rechner oder auch Hosts, die in einer Organisation bekannt sind.

Unter virtuellen Aspekten werden Eigenschaften wie die Systemsicherheit der zu Grunde liegenden Software gefasst, während physikalische Aspekte den tatsächlichen Zugang zu der Maschine und die verwendete Hardware umfassen.

Ein wichtiger Aspekt sind die physikalischen Orte, an denen die Knoten sich befinden. Zum einen müssen hier rechtliche Aspekte berücksichtigt werden. Falls Daten, die in einem Land gespeichert sind, von einem Rechner in einem anderen Land abgerufen werden, stellt dies einen Export dar, sodass z.B. bei personenbezogenen Daten das Datenschutzgesetz [BDS03] zu berücksichtigen wäre. Zum anderen spielen Sicherheitsaspekte eine Rolle, da Daten, die als streng geheim gekennzeichnet sind, nicht zugänglich sein sollten, wenn der Benutzer zum Beispiel an einem öffentlichen Terminal in einer Eingangslobby eingeloggt ist.

Dazu wird zunächst eine Menge $\mathcal{L}$ definiert, die alle bekannten bzw. relevanten Lokationen der Organisation enthält. Um einfacher bestimmte Lokationen wie "Labor X" oder Länder zusammenfassen zu können, wird über die Elemente dieser Menge $\mathcal{L}$ eine Halbordnung $<_{\mathcal{L}}$ definiert, die als Lokationshierarchie bezeichnet wird. In Abbildung 4.3 ist ein Beispiel für eine solche Lokationshierarchie angegeben, die auch im weiteren Verlauf als Beispiel dient.

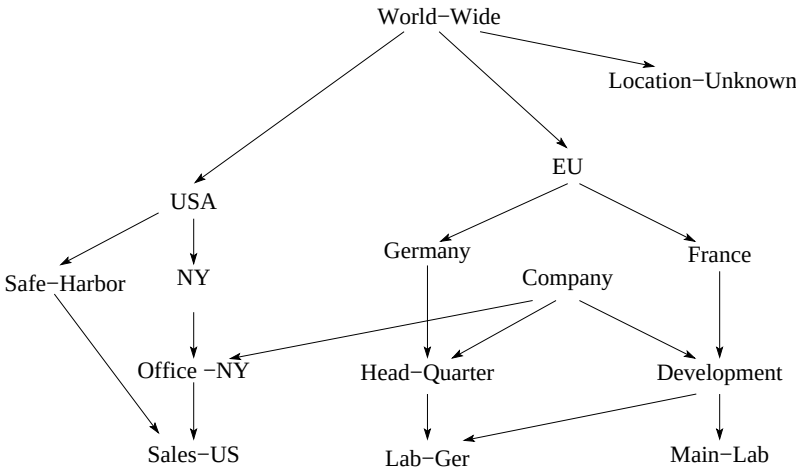



Abbildung 4.3: Beispiel für eine Lokationshierarchie

Als weiteres wichtiges Merkmal für die Sicherheit eines Hosts ist das zu Grunde liegende Betriebssystem zu nennen, da sich die verschiedenen Systeme, zum Beispiel in der Anfälligkeit für Viren und Würmerangriffe, unterscheiden oder bestimmte Reviews durchlaufen haben, um Sicherheitslücken zu schließen. Die verwendeten Betriebssysteme können sich in den verschiedenen Organisationen unterscheiden und neue Versionen oder Varianten können hinzukommen. Daher enthält die Menge $\mathcal{OS}$ alle innerhalb einer Organisation verwendeten Betriebssysteme. Neue Versionen oder auch wichtige Sicherheitsupdates können als eigene Elemente in diese Menge aufgenommen werden. Um aber nicht für jedes einzelne dieser Elemente eine eigene Einstufung in eine Vertrauensstufe definieren zu müssen, empfiehlt es sich, eine Hierarchie über die Betriebssysteme mittels einer Halbordnung $<_{\mathcal{OS}}$ zu definieren. Um diese noch weiter strukturieren zu können, kann die Menge $\mathcal{OS}$ auch abstrakte Elemente wie "Unix", "Windows" oder "MacOS"[4] enthalten. Zudem ist es sinnvoll, in die Hierarchie ein kleinstes Element "Known OS" einzufügen, um so einfach auf Betriebssysteme testen zu können, die (noch) nicht in der Menge $\mathcal{OS}$ enthalten sind, zum Beispiel wenn es sich um ein neues Sicherheitsupdate handelt. Außerdem gibt es bei vielen Betriebssystemen "Beta" bzw. "Unstable" Versionen, die zwar in der Hierarchie aufgrund der Versionsnummer höher eingeordnet werden, allerdings nicht notwendigerweise sicherer sind.
In Abbildung 4.4 ist ein Beispiel für eine Hierarchie der Betriebssysteme zu sehen. Auf dieses Beispiel beziehen sich die weiteren Ausführungen.

[4]Für die neuste Version "MacOS X" ist die Einordnung nicht ganz eindeutig, da sie auf einem Mach-Kernel beruht. Eine mögliche Hierarchie, dem dies Rechnung trägt ist in Abbildung 4.4 zu sehen.

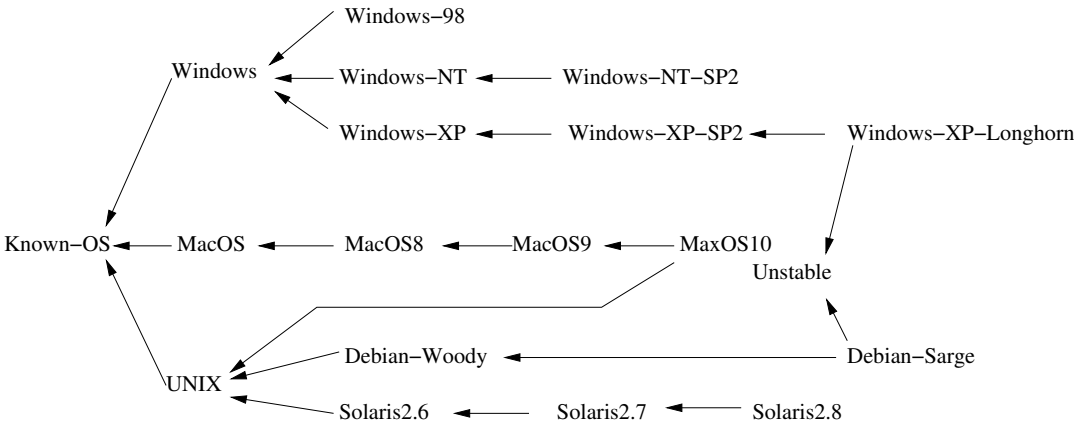



Abbildung 4.4: Beispiel[5]für eine Hierarchie von Betriebssystemen

In den letzten Jahren wurde durch die Entwicklung der *Trusted Platform Module (TPM)* [TCG03a] durch die Trusted Computing Group auch die Hardwarearchitektur einer Maschine für die Einstufung in eine Vertrauensstufe wichtig. Alle in einer Organisation bekannten Architekturen sind in der Menge $\mathcal{AR}$ enthalten. Da bei den Hardwarearchitekturen anders als bei den Betriebssystemen noch keine Hierarchie erkennbar ist, die für die Einordnung in Vertrauensstufen von Bedeutung ist, wird auf eine entsprechende Definition verzichtet.

### 4.2.1 Bestimmung der beteiligten Hosts

Die Zugriffsregeln werden in den seltensten Fällen auf konkrete Hosts verweisen, sondern über Variablen bestimmte Eigenschaften der beteiligten Hosts abgefragen. Die Menge $\mathcal{V}_\mathcal{H}$ enthält alle verwendeten Variablen, die sich auf Elemente der Menge $\mathcal{H}$ beziehen. Bei der Definition der Prädikate werden Terme über diese Variablenmenge und die Menge $\mathcal{H}$ auch als Hostterme bezeichnet ($\mathcal{HT} = \mathcal{V}_\mathcal{H} \cup \mathcal{H}$).

Wie zu Beginn des Kapitels bereits erläutert, ist ein Aspekt, der in die Autorisierung mit einbezogen werden muss, der Host, an dem der Benutzer, der als Subjekt $s$ bezeichnet wird, tatsächlich angemeldet ist. Dabei ist davon auszugehen, dass die Informationen, die durch Zugriffe dieses Subjektes gewährt werden, auch bis zu diesem Host weitergeleitet werden und sei es nur, dass sie auf dem dazugehörigen Bildschirm angezeigt werden. Bei einer eingehenden Zugriffsanfrage kann dieser Host mit Hilfe des folgenden Kontext Feature Prädikates bestimmt werden.

**Definition 4.3 (Login)**

$$\textsf{login}(s, h)$$

ist ein binäres Prädikat, das dann gültig ist, wenn das Subjekt $s \in \mathcal{ST}$ auf dem Host $h \in \mathcal{HT}$ eingeloggt ist.

Ein weiteres wichtiges Kriterium ist der Host, auf dessen Medium das Objekt gespeichert wird, da hiervon virtuelle oder physikalische Sicherheitsaspekte, sowie rechtliche Aspekte abhängen können. Für das vorliegende Autorisierungssystem wird zunächst von Medien wie Festplatten oder auch StorageNetworks ausgegangen, die permanent am Rechner verfügbar sind. Falls auch der Zugriff[6] auf ein Wechselmedium betrachtet werden soll, könnte dies gesondert abgefragt werden[7].

[5]Die Hierarchie wurde aus [WL04b] übernommen. Die Zuordnung von Debian-Sarge zu Unstable stimmt mittlerweile nicht mehr.

[6]Wobei in den meisten Fällen nur die Speicherung, also ein Schreibzugriff kritisch sein dürfte.

[7]In den meisten UNIX Systemen werden die Medien in bestimmte Verzeichnisse eingebunden, diese können mittels den Objekteigenschaften aus Abschnitt 3.5 abgefragt werden.

**Definition 4.4 (Speicherung)**

$$\textsf{storage}(o, h)$$

ist ein binäres Prädikat, das genau dann gültig ist, wenn das Objekt $o \in \mathcal{OT}$ auf einem Medium des Hosts $h \in \mathcal{HT}$ gespeichert ist.

Somit kann durch die folgende Regel[8] festgelegt werden, dass ein Subjekt nur dann Lesezugriff auf ein Objekt hat, wenn dieses auf dem gleichen Host gespeichert ist, auf dem das Subjekt eingeloggt ist.

**Beispielregel 4.1**

$$\textsf{read}(s, o) \leftarrow \textsf{login}(s, h) \& \textsf{storage}(o, h)$$

Bei der Auswertung dieser Regel ist zu Beginn die Belegung der Variablen $s$ und $o$ bekannt. Unabhängig davon, welches der beiden Kontext Feature Prädikate ausgewertet wird, erhält die Variable $h$ dadurch eine eindeutige Belegung. Nur wenn mit dieser Belegung auch das zweite Prädikat gültig ist, ist die gesamte Regel gültig.

### 4.2.2 Physikalische Eigenschaften von Hosts

Um die rechtlichen Aspekte und die physikalische Zugangssicherheit zu den Hosts berücksichtigen zu können, wird auf Informationen über den Standort der Hosts zurückgegriffen. Ein Beispiel für die möglichen Lokationen einer Organisation ist in Abbildung 4.3 zu sehen. Die Menge $\mathcal{L}$ enthält alle in der Organisation definierten Lokationen. Die Menge $\mathcal{V}_\mathcal{L}$ enthält alle Variablen über die Menge der Lokationen, die in den Zugriffsregeln verwendet werden. Terme über Lokationen $\mathcal{LT}$ sind als Vereinigung über die Menge der Lokationen $\mathcal{L}$ und die Menge der Variablen von Lokationen $\mathcal{V}_\mathcal{L}$ definiert ($\mathcal{LT} = \mathcal{L} \cup \mathcal{V}_\mathcal{L}$).

Mit dem folgenden Feature Prädikat kann die Lokation eines Hosts bestimmt werden, um diese bei einer Entscheidung über eine Zugriffsanfrage zu berücksichtigen.

**Definition 4.5 (Lokation eines Hosts)**

$$\textsf{locationof}(h, l)$$

ist ein binäres Prädikat, das genau dann gültig ist, wenn der Host $h \in \mathcal{HT}$ an der Lokation $l \in \mathcal{LT}$ steht.

Wie schon zu Beginn des Abschnittes 4.2 erläutert, ist es sinnvoll, eine Hierarchie über die Lokationen zu definieren, die dann mit den folgenden beiden Prädikaten abgefragt werden kann.

**Definition 4.6 (direkte Beziehungen zwischen Lokationen)**

$$\textsf{locationin}(l_1, l_2)$$

ist ein binäres Prädikat, das genau dann gültig ist, wenn für die beiden Lokationen $l_1, l_2 \in \mathcal{LT}$ $(l_1, l_2) \in <_\mathcal{L}$ gilt.

Ähnlich den Hierarchien von Rollen etc. (vgl. Definition 3.15), wird auch für Lokationen eine indirekte Beziehung definiert, sodass zum Beispiel alle Lokationen eines Landes einfach referenziert werden können.

[8] Zur Verdeutlichung wird wiederum das Prädikat aus Abschnitt 3.1 genommen, dessen Rumpf nun auch alle in diesem Kapitel definierten Prädikate enthalten kann.

**Definition 4.7 (indirekte Beziehung von Lokationen)**
Die indirekte Beziehung von Lokationen kann über die folgenden Regeln für $l_1, l_2 \in \mathcal{LT}$ abgefragt werden, wobei $l' \in \mathcal{V}_\mathcal{L}$:

$$\textbf{sublocation}(l_1, l_2) \leftarrow \text{locationin}(l_1, l_2)$$
$$\textbf{sublocation}(l_1, l_2) \leftarrow \text{sublocation}(l_1, l')\&\text{locationin}(l', l_2)$$

Durch die folgenden Beispielregeln, die sich auf die Hierarchie der Lokationen in Abbildung 4.3 beziehen, wird ein Zugang zu Objekten nur gewährt, wenn sich das Subjekt an einem Rechner in der EU eingeloggt hat. Um das Beispiel so kurz wie möglich zu halten, wird auf Feature Prädikate aus Kapitel 3, die die aktive Rolle des Subjekts $s$ oder den Objekttyp des Objektes $o$ einschränken, verzichtet.

**Beispielregeln 4.2**

$$\text{read}(s, o) \leftarrow \text{login}(s, h)\&\text{locationof}(h, l)\&\text{sublocation}(l, EU)$$
$$\text{read}(s, o) \leftarrow \text{login}(s, h)\&\text{locationof}(h, EU)$$

Bei einer effizienten Auswertung des Rumpfes der ersten Regel werden nacheinander die Variablen $h$ und $l$ eindeutig belegt, sodass abschließend für die Belegung der Variable $l$ überprüft werden kann, ob das Kontext Feature Prädikat $\text{sublocation}(l, EU)$ erfüllt werden kann. Da der Host als Lokation direkt EU haben könnte, muss dieser Fall mit der zweiten Regel im Regelsatz 4.2 überprüft werden. Um dies zu vermeiden, wird das folgende Kontext Feature Prädikat definiert.

**Definition 4.8 (erweiterte Beziehung von Lokationen)**
Die indirekte Beziehung von Lokationen kann über die folgenden Regeln für $l_1, l_2 \in \mathcal{LT}$ abgefragt werden, wobei $l' \in \mathcal{V}_\mathcal{L}$:

$$\textbf{sublocationof}(h, l) \leftarrow \text{locationof}(h, l)$$
$$\textbf{sublocation}(h, l) \leftarrow \text{locationof}(h, l')\text{locationin}(l', l)$$
$$\textbf{sublocation}(h, l) \leftarrow \text{locationof}(h, l')\&\text{sublocation}(l', l)$$

Die Definition ist analog zur Definition 3.23 von subtypeof(). Somit kann ein Lesezugriff für alle Subjekte, die an einem Host der Organisation ("Company") eingeloggt sind, auf alle Objekte, die auf Hosts der Organisation gespeichert sind, mit der folgenden Regel autorisiert werden.

**Beispielregel 4.3**

$$\text{read}(s, o) \leftarrow \text{login}(s, h)\&\text{sublocationof}(h, Company)\&$$
$$\text{storage}(o, h')\&\text{sublocationof}(h', Company)$$

Dabei ist zu beachten, dass nicht spezifiziert wurde, ob die Hosts $h$ und $h'$ identisch sind oder nicht.

Ein weiterer physikalischer Aspekt für die Einstufung eines Knotens wird in Zukunft sicherlich die zu Grunde liegende Hardwarearchitektur wie zum Beispiel "TPM" sein [TCG03a, TCG03c]. Im Moment erscheint die Einführung einer Hierarchie über die Hardwarearchitekturen noch nicht sinnvoll[9]. Daher wird nur ein einziges Prädikat bezüglich der Hardwarearchitektur definiert, sodass die Definition von entsprechenden Termen[10] ebenfalls unnötig wird.

**Definition 4.9 (Hardwarearchitektur)**

$$\textbf{host-arch}(h, ar)$$

ist ein binäres Prädikat, das genau dann gültig ist, wenn der Host $h \in \mathcal{HT}$ die Hardwarearchitektur $ar \in \mathcal{AR}$ hat.

Mit diesen Kontext Feature Prädikaten lassen sich die physikalischen Eigenschaften eines Hosts abfragen und bei der Einstufung der Vertrauensstufen entsprechend berücksichtigen.

[9] Die Definition würde im übrigen analog zu der über Lokationen erfolgen.
[10] Die entsprechenden Hardwarearchitekturterme würden dann mit $\mathcal{ART}$ bezeichnet werden.

### 4.2.3 Virtuelle Eigenschaften von Hosts

Neben der Lokation können weitere Eigenschaften des Rechners wichtig sein, um die Vertrauensstufe festzulegen. Eine davon ist, wie im Abschnitt 4.2 erläutert, das zu Grunde liegende Betriebssystem. Für die Definition der Prädikate, die diese Aspekte abfragen, werden Terme über Betriebssysteme $\mathcal{OST}$ benötigt, die neben den konkreten Elementen aus der Menge der bekannten Betriebssysteme $\mathcal{OS}$ auch die Menge $\mathcal{V}_{\mathcal{OS}}$ der Variablen über Elemente dieser Menge enthält.

Das konkrete Betriebssystem eines Hosts kann mit dem folgenden Prädikat abgefragt werden.

**Definition 4.10 (Betriebssystem)**

$$\textbf{host-os(}h, os\textbf{)}$$

ist ein binäres Prädikat, das genau dann gültig ist, wenn auf dem Host $h \in \mathcal{HT}$ das Betriebssystem $os \in \mathcal{OST}$ läuft.

In Abbildung 4.4 ist ein Beispiel für eine Hierarchie über die Betriebssysteme dargestellt. Diese kann über die beiden folgenden Prädikate abgefragt werden.

**Definition 4.11 (direkte Beziehungen zwischen Betriebssystemen)**

$$\textbf{osin(}os_1, os_2\textbf{)}$$

ist ein binäres Prädikat, das genau dann gültig ist, wenn für die beiden Betriebssysteme $os_1, os_2 \in \mathcal{OST}$ gilt: $(os_1, os_2) \in <_{\mathcal{OS}}$.

In den meisten Fällen wird nicht die direkte, sondern die indirekte Beziehung von Betriebssystemen von Interesse sein, da dann auch verschiedene Versionen berücksichtigt werden können. Das dazugehörige Prädikat wird wieder rekursiv definiert.

**Definition 4.12 (indirekte Beziehung von Betriebssystemen)**
Die indirekte Beziehung von Betriebssystemen kann über die folgenden Regeln für $os_1, os_2 \in \mathcal{OST}$ abgefragt werden, wobei $os' \in \mathcal{V}_{\mathcal{OS}}$:

$$\textbf{subos(}os_1, os_2\textbf{)} \leftarrow \mathsf{osin}(os_1, os_2)$$
$$\textbf{subos(}os_1, os_2\textbf{)} \leftarrow \mathsf{subos}(os_1, os')\&\mathsf{osin}(os', os_2)$$

Mit der folgenden Regel darf ein Subjekt nur lesend auf ein Objekt zugreifen, wenn es auf einem Host gespeichert ist, dessen Betriebssystem bekannt ist, aber nicht als "Unstable" eingestuft ist.

**Beispielregel 4.4**

$$\mathsf{read}(s, o) \leftarrow \mathsf{storage}(o, h)\&\mathsf{host\_os}(h, os)\& \\ \mathsf{subos}(os, Known - OS)\&\neg\mathsf{subos}(os, Unstable)$$

Außerdem kann für die Zuordnung einer Vertrauensstufe auch noch wichtig sein, wer für die Administrierung des Hosts verantwortlich ist, also das Root– oder Administrator–Passwort kennt, bzw. ein Administratorkonto hat. Hier soll nicht der konkrete Benutzer abfragt werden, sondern es reicht aus, wenn diese Aufgaben dem Team einer bestimmten Autorisierungssphäre zugeordnet werden können.

**Definition 4.13 (Zuordnung)**

$$\textbf{assigned(}h, as\textbf{)}$$

ist ein binäres Prädikat, das genau dann gilt, wenn die Administrierung von Host $h \in \mathcal{HT}$ vom Team der Autorisierungssphäre $as \in \mathcal{AST}$ durchgeführt wird.

Die folgende Regel autorisiert einen Lesezugriff für ein Subjekt auf ein Objekt dann, wenn die aktive Rolle des Subjektes in der gleichen Organisationseinheit ist, zu der auch das Objekt gehört. Darüber hinaus muss das Subjekt auf einem Host eingeloggt sein, der von der Autorisierungssphäre dieser Organisationseinheit verwaltet wird. Das Objekt muss gemäß der Regel ebenfalls auf einem Host gespeichert sein, der dieser Autorisierungssphäre zugeordnet wird.

**Beispielregel 4.5**

$$\begin{aligned}\mathsf{read}(s,o) \leftarrow & \mathsf{activ\_role}(s,u,r)\&\mathsf{unitof}(o,u)\&\mathsf{sphereof}(u,as)\& \\ & \mathsf{login}(s,h)\&\mathsf{assigned}(h,as)\&\mathsf{storage}(o,h')\&\mathsf{assigned}(h',as)\end{aligned}$$

Die beiden Hosts $h$ und $h'$, auf denen das Subjekt eingeloggt bzw. das Objekt gespeichert ist, können auch identisch sein. Die aktive Rolle des Subjektes kann bei der obigen Regel beliebig sein, da $r$ eine freie Variable ist.

Für die weiteren Definitionen werden die in den Abschnitten 4.2.1 bis 4.2.3 vorgestellten Prädikate als Kontext Feature Prädikate für Hosts zusammengefasst. Die Menge $\mathcal{CF}_{\mathcal{H}}$ enthält alle Prädikate aus den Definitionen 4.3 bis 4.13.

### 4.2.4 Einstufung von Hosts

Die Vertrauensstufe eines Hosts kann von allen im vorangegangenen Abschnitt beschriebenen Kriterien abhängen. Aus den gleichen Gründen, die schon im Abschnitt 3.3 für die Festlegung der Zugriffsregeln angeführt sind, wird keine zentrale Instanz, die die Kriterien für eine bestimmte Vertraulichkeitsstufe festgelegt, eingeführt sondern eine dezentrale Struktur unterstützt. Diese soll sowohl eine Übernahme der Einstufung durch das Autorisierungsteam, als auch eine Verfeinerung ermöglichen. Hierzu wird auf die Autorisierungssphären zurückgegriffen.

Mit dem folgenden Kontext Trust Prädikat kann die Festlegung einer bestimmten Vertrauensstufe durch eine Autorisierungssphäre spezifiziert werden.

**Definition 4.14 (Host Vertrauenstufe)**
Die Regel für Vertrauenstufen von Hosts hat die folgende Form:

$$\textbf{host\_level}(as,h,t) \leftarrow L_1\&..\&L_n$$

wobei $as \in AST$ die Autorisierungssphäre ist, die festlegt, dass dem Host $h \in \mathcal{HT}$ die Vertrauensstufe $t \in \mathcal{TLT}$ zugesprochen wird, wenn alle Prädikate $L_i \in \mathcal{F} \cup \mathcal{CF}_{\mathcal{H}}$ (mit $n \in I\!N$ und $1 \leq i \leq n$) gültig sind.

Im Folgenden sind zwei Beispiele für eine Host Trust Regel gegeben, mit denen in der Autorisierungssphäre Software die Vertrauenstufen Internal-Trusted bzw. Secured vergeben werden. Mit der dritten Regel wird ein Lesezugriff autorisiert, wenn das Subjekt auf einem Host eingeloggt ist, der von der Autorisierungssphäre Software auf der Vertrauensstufe Internal-Trusted eingestuft wurde, und das Objekt auf einem Host liegt, der die Vertrauensstufe Secured hat.

**Beispielregeln 4.6**

$$\begin{aligned}
\mathsf{host_level}(Software, h, Internal-Trusted) \leftarrow & \mathsf{sublocationof}(h, Development)\&\mathsf{host_os}(h, os)\& \\
& \mathsf{subos}(Known-OS, os)\& \\
& \neg\mathsf{subos}(os, Windows-98)\& \\
& \mathsf{assigned}(h, as)\& \\
& \mathsf{subsphereeq}(as, Software) \\
\mathsf{host_level}(Software, h, Secured) \leftarrow & \mathsf{sublocationof}(h, Development)\&\mathsf{host_os}(h, os)\& \\
& \mathsf{subos}(UNIX, os)\&\mathsf{assigned}(h, Software) \\
\mathsf{read}(s, o) \leftarrow & \mathsf{login}(s, h)\&\mathsf{storage}(o, h')\& \\
& \mathsf{host_level}(Software, h, Internal-Trusted)\& \\
& \mathsf{host_level}(Software, h', Secured)
\end{aligned}$$

Falls das Subjekt auf einem Host eingeloggt ist, der auch die Bedingungen für Secured erfüllt, ist dies nicht von Bedeutung. Sofern nicht die Belegung von $h$ mit $h'$ identisch ist, wird das Prädikat $\mathsf{host\_level}(Software, h, Internal-Trusted)$ ausgewertet.

Darüber hinaus kann sich die Einstufung eines Host in eine Vertrauensstufe in einer bestimmten Autorisierungssphäre auch von einer anderen Einstufung ableiten, welches durch folgendes Prädikat spezifiziert werden kann. Diese Einstufungen können entweder von derselben Autorisierungssphäre vorgenommen werden oder aber von einer anderen übergeordneten Autorisierungssphäre übernommen werden. Die Einschränkung auf übergeordnete Autorisierungssphäre erfolgt deshalb, um bei einer verteilten Auswertung die Replizierung der Regeln strukturiert vornehmen zu können. Diese Einschränkung gilt auch für alle weiteren Regeln zur Einstufung von Vertrauensstufen.

**Definition 4.15 (abgeleitete Host Vertrauenstufe)**
Die Regel für abgeleitete Vertrauenstufen von Hosts hat die folgende Form:

$$\textbf{derhost\_level}(as, h, t) \leftarrow L_1\& \ldots \&L_n$$

wobei $as \in \mathcal{AST}$ die Autorisierungssphäre ist, die festlegt, dass dem Host $h \in \mathcal{HT}$ die Vertrauensstufe $t \in \mathcal{TLT}$ zugesprochen wird, wenn alle Prädikate $L_i \in \mathcal{F} \cup \mathcal{CF}_{\mathcal{TL}} \cup \mathcal{CF}_{\mathcal{H}} \cup \{\text{host\_level}\}$ und $L_n \in \{\text{host\_level}\}$ (mit $n \in \mathbb{N}$ und $1 \leq i < n$) gültig sind. Alle Regeln vom Typ host müssen aus der gleichen Autorisierungssphäre $as$ oder aus einer übergeordneten Autorisierungssphäre $as'$ sein ($as <_{\mathcal{AS}} as'$).

Mit der ersten der beiden folgenden Regeln übernimmt die Autorisierungssphäre Netzwerk die Einstufung Secured von Software, wenn der Host sich in der Lokation Main_Lab befindet. Die Autorisierungssphäre TPM übernimmt mit der zweiten Regel alle Einstufungen zwischen Insecure und Secured ausschließlich von den (übergeordneten) Autorisierungssphären, die für diese Hosts verantwortlich sind.

**Beispielregeln 4.7**

$$\begin{aligned}
\mathsf{derhost_level}(Netzwerk, h, Secured) \leftarrow & \mathsf{assigned}(h, Software)\&\mathsf{locationof}(h, Main_Lab)\& \\
& \mathsf{host_level}(Software, h, Secured) \\
\mathsf{derhost_level}(TPM, h, tp) \leftarrow & \mathsf{assigned}(h, as)\&\mathsf{sublevel}(Insecure, tp)\& \\
& \mathsf{sublevel}(tp, Secured)\&\mathsf{host_level}(as, h, tp)\& \\
& \mathsf{subsphereeq}(TPM, as)
\end{aligned}$$

Diese Einstufung der Host ist auch für die Einstufung der Verbindungen zwischen den Knoten wichtig, da diese die Endpunkte der Kommunikationskanäle darstellen.

## 4.3 Vertrauenswürdigkeit der Verbindung

Eine weitere wichtige Komponente in Abbildung 4.1 und sicherlich die am leichtesten angreifbare, sind die Kommunikationsverbindungen. Diese können jedoch mit kryptographischen Mitteln entsprechend abgesichert werden.

Dabei ist es nicht sinnvoll, die Kommunikationskanäle im Vorfeld zu benennen. Eine Namensgebung in der Form "Verbindung zwischen Knoten A und B" wäre auf keinen Fall sinnvoll, da es zum einen eine enorme Zahl von solchen Bezeichnungen geben müsste, die unter Umständen nie gebraucht werden. Zum anderen ist bei der Verwendung im Datalog Kontext nicht garantiert, dass dies tatsächlich der verwendete Kommunikationskanal ist. Daher wird nur $\mathcal{V}_{\mathcal{C}}$ als Menge von **Variablen über Kommunikationskanäle** eingeführt.

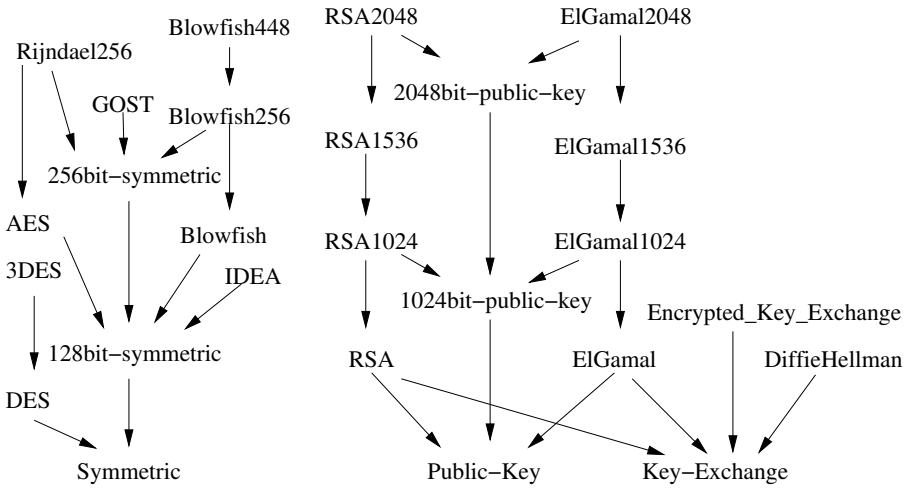



Abbildung 4.5: Beispiel für die Sicherheitslevel innerhalb einer Organisation

Um die Sicherheit der Kommunikationskanäle einstufen zu können, werden die verwendeten kryptographischen Verfahren[11] berücksichtigt. Dabei sollte neben dem verwendeten Protokoll, wie zum Beispiel "DES" oder "IDEA", bei solchen mit variabler Schlüssellänge, wie "ElGamal", "RSA" oder "Blowfish", auch diese Schlüssellänge bei der Bezeichnung mit angegeben werden, sodass dann das kryptographische Verfahren als ElGamal024, RSA2048 oder Blowfish448 bezeichnet[12] wird. Darüber hinaus ist es für Verfahren mit symmetrischem Schlüssel von Bedeutung das Austauschverfahren festzustellen. Schon bei der Hierarchie der Betriebssysteme (vgl. Abbildung 4.4) werden allgemeine Bezeichner zur Charakterisierung von bestimmten Teilen der Hierarchie verwendet. Dies kann bei der Hierarchie der Sicherheitsstufen eingesetzt werden, um zum Beispiel alle 128bit Verschlüsselungsverfahren oder alle Public–Key Verfahren mit 2048bit Schlüssellänge zusammenzufassen. Alle in einer Organisation bekannten kryptographischen Verfahren sind in der Menge $\mathcal{SL}$ enthalten. Wie schon angedeutet wird diese Menge der Sicherheitsstufen mittels einer Halbordnung $<_{\mathcal{SL}}$ strukturiert, die als Hierarchie der Sicherheitsstufen bezeichnet wird. Ein Beispiel für eine solche Hierarchie der Sicherheitsstufen findet sich in Abbildung 4.5.

Da Public-Key Verfahren auch zum Schlüsselaustausch verwendet werden können, ist eine Aufteilung nach Sicherheitsstufen für symmetrische und asymmetrische, sowie Verfahren zum Schlüsselaustausch nicht sinnvoll.

Im folgenden Abschnitt werden die nötigen Prädikate definiert, mit denen sich die Verwendung dieser kryptographischen Verfahren beschreiben lässt.

[11] Eine Übersicht über diese Verfahren bietet [Sch96b, Kap. 19-21] .

[12] ElGamal und RSA wurden in [ElG85] bzw. [RSA78]. vorgestellt, Blowfish in [Sch94] sowie AES/Rjindal in [DR00].

### 4.3.1 Eigenschaften von Kommunikationskanälen

Um festzustellen, welche Eigenschaften eine Kommunikationsverbindung hat, muss zunächst im Autorisierungssystem diese Kommunikationsverbindung zwischen zwei Hosts bestimmt werden. Wie bereits im Abschnitt 4.3 erläutert, wird diese Kommunikationsverbindung nicht explizit benannt, sondern nur eine Variable mit der aktuellen Kommunikationsverbindung zwischen zwei Hosts belegt. Da in dieser Arbeit davon ausgegangen wird, dass zwischen den Hosts in dem Modell aus Abbildung 4.1 zumindest für die Dauer des Zugriffes, und somit auch während der Autorisierung eine eindeutige Verbindung besteht, kann dies durch das folgende Prädikat bestimmt werden.

**Definition 4.16 (Kommunikationskanal)**

**channel(**$h_1, h_2, ch$**)**

ist ein dreistelliges Prädikat, dass genau dann gültig ist, wenn zwischen den beiden Hosts $h_1, h_2 \in \mathcal{HT}$ ein offener Kommunikationskanal $ch \in \mathcal{V_C}$ besteht, über den die Zugriffsanfragen kommen.

Die Informationen über die Kommunikationsverbindung können während des Verbindungsaufbaus gesammelt werden und stehen dann während der Autorisierung zur Verfügung. So bereiten auch mehrere Kommunikationsverbindungen zwischen zwei Hosts, die aufgrund von Zugriffsanfragen unterschiedlicher Subjekte gleichzeitig bestehen, keine Probleme.

Bei der Abfrage der Eigenschaften für die Kommunikationskanäle werden nicht nur Variablen benötigt, die den verwendeten Kanal bezeichnen, sondern auch Variablen über die Sicherheitsstufen, die in der Menge $\mathcal{V_{SL}}$ enthalten sind. Bei der Definition der Kontext Feature Prädikate werden Terme über die Sicherheitsstufen $(\mathcal{SLT})$, die aus dieser Menge der Variablen und den bekannten Sicherheitsstufen $\mathcal{SL}$ bestehen, verwendet ($\mathcal{SLT} = \mathcal{V_{SL}} \cup \mathcal{SL}$).

Da angenommen wird, dass jeder Knoten auch über mehrere Möglichkeiten der Authentifizierung[13] verfügt, und sei es nur durch unterschiedliche Schlüssellängen, ist diese Information auch bei der Autorisierung von Bedeutung. Mit dem folgenden Kontext Feature Prädikat kann die verwendete Authentifizierungsmethode für einen bestimmten Kommunikationskanal festgestellt werden.

**Definition 4.17 (Authentifizierungsmethode)**

**authenticated(**$ch, ac$**)**

ist ein binäres Prädikat, das genau dann gültig ist, wenn bei der Authentifizierung der Kommunikationspartner des Kanals $ch \in \mathcal{V_C}$ die Methode $ac \in \mathcal{SLT}$ verwendet wird.

In Abbildung 4.5 wird ein Beispiel für die Menge an möglichen Authentifizierungsmethoden gegeben, wobei das Element Public-Key dazu genutzt werden kann, eine beliebige Authentifizierungsmethode zu akzeptieren.

Da asymmetrische Verfahren, die für die Authentifizierung genutzt werden, zu langsam für die Verschlüsselung großer Datenmengen sind, ist es sinnvoll, den Kommunikationskanal mittels eines effizienteren symmetrischen Verfahrens zu sichern. In Abbildung 4.5 sind verschiedene symmetrische Verfahren[14] aufgeführt. Ähnlich wie beim Prädikat zur Authentifizierung kann auch die Verschlüsselungsmethode für einen Kanal bestimmt werden.

[13] Ein Überblick über verschiedene asymmetrischen Verschlüsselungsverfahren, die zur Authentifizierung eingesetzt werden können, findet sich in [Sch96b, Kap.19–21]. Eine Übersicht über weitere Authentifizierungsmethoden wird in [Eck04, Kap. 10] präsentiert.

[14] Eine Übersicht zu symmetrischen Verschlüsselungsverfahren findet sich in [Sch96b, Kap 12-14].

**Definition 4.18 (Verschlüsselungsmethode)**

$$\textbf{encryption}(ch, en)$$

ist ein binäres Prädikat, das genau dann gültig ist, wenn für den Kanal $ch \in \mathcal{V}_{\mathcal{C}}$ die Methode $en \in \mathcal{SLT}$ zur Verschlüsselung genutzt wurde.

Für symmetrische Verfahren müssen beide Kommunikationspartner über einen gemeinsamen Schlüssel verfügen. Der Austausch dieses Schlüssels ist anfällig für die Korruption (vgl. [Eck04, Seite 408ff]) und sollte daher auch bei der Vertraulichkeitseinschätzung in Betracht gezogen werden, wozu das folgende Prädikat genutzt werden kann.

**Definition 4.19 (Schlüsselaustausch)**

$$\textbf{key\_exchanged}(ch, ke)$$

ist ein binäres Prädikat, das genau dann gültig ist, wenn der symmetrische Schlüssel, der für den Kanal $ch \in \mathcal{V}_{\mathcal{C}}$ genutzt wird, mittels des Protokolls $ke \in \mathcal{SLT}$ ausgetauscht wurde.

Falls kein symmetrischer Schlüssel für eine Kommunikationsverbindung verwendet wird[15] und damit auch kein Schlüsselaustausch stattgefunden hat, ist dieses Prädikat für keine Belegung des Parameters $ke$ gültig.

In der Beispielhierarchie in Abbildung 4.5 war Key-Exchange das kleinste Element (gemäß $<_{\mathcal{SL}}$) für einen Teilbaum, der alle möglichen Schlüsselaustauschverfahren enthält.

Mit diesen Prädikaten kann in der folgenden Beispielregel festgelegt werden, dass ein Leserecht gewährt wird, wenn zwischen dem Host $h$, auf dem das Subjekt eingeloggt ist, und dem Host $h'$, auf dem das Objekt gespeichert ist, ein Kommunikationskanal besteht, bei dem sich die Hosts mittels RSA bei einer Schlüssellänge von 1024bit authentifiziert haben. Zudem wurde über DiffieHellman zwischen den Hosts ein symmetrischer Schlüssel ausgetauscht, der für die Verschlüsselung des Kommunikationskanals per IDEA eingesetzt wird.

**Beispielregel 4.8**

$$\begin{aligned}\mathsf{read}(s, o) \leftarrow & \mathsf{login}(s, h) \& \mathsf{storage}(o, h') \& \mathsf{channel}(h, h', ch) \& \\ & \mathsf{authenticated}(ch, RSA1024) \& \mathsf{encrypted}(ch, IDEA) \& \\ & \mathsf{key\_exchanged}(ch, DiffieHellman)\end{aligned}$$

Bei dieser Regel wird nur auf einzelne Sicherheitsstufen zurückgegriffen ,ohne die Hierarchie der Sicherheitsstufen aus Abbildung 4.5 zu nutzen.

### 4.3.2 Abfrage der Sicherheitsstufen

Bestimmte Zugriffe sollen nur dann erlaubt werden, wenn mindestens eine bestimmte Sicherheitsstufe erreicht wird. Alternativ können sie auch verboten werden, wenn eine Sicherheitsstufe unterschritten wird. Hierzu müssen Bedingungen an die Beziehungen in der Hierarchie der Sicherheitslevel (vgl. Abbildung 4.5) geknüpft werden können. Diese können mit den drei Prädikate ausgedrückt werden, die im Folgenden definiert werden.

**Definition 4.20 (direkte Beziehungen zwischen Sicherheitsstufen)**

$$\textbf{securein}(sl_1, sl_2)$$

ist ein binäres Prädikat, das bei $(sl_1, sl_2) \in <_{\mathcal{SL}}$ für die beiden Sicherheitsstufen $sl_1, sl_2 \in \mathcal{SLT}$ gilt.

[15] sondern trotz der Performanceeinbußen ein asymmetrisches Verfahren

Wenn bei den Sicherheitsstufen ein bestimmtes Verfahren wie RSA mit einer beliebigen Schlüssellänge zugelassen werden soll, ist es wichtig, auch die indirekte Beziehung zwischen den Sicherheitsstufen abfragen zu können.

**Definition 4.21 (indirekte Beziehung zwischen Sicherheitsstufen)**
Die indirekte Beziehung zwischen Sicherheitsstufen kann über die folgenden Regeln für $sl_1, sl_2 \in \mathcal{SLT}$ abgefragt werden, wobei $sl' \in \mathcal{V}_{\mathcal{SL}}$:

$$\textbf{subsecure}(sl_1, sl_2) \leftarrow \mathsf{securein}(sl_1, sl_2)$$
$$\textbf{subsecure}(sl_1, sl_2) \leftarrow \mathsf{subsecure}(sl_1, sl')\&\mathsf{securein}(sl', sl_2)$$

Das Beispiel für die Hierarchie der Sicherheitsstufen in Abbildung 4.5 enthält viele Elemente, wie RSA, ElGamal, 1024bit-public-key, 128bit–symmetric etc., die zur Kategorisierung der Verfahren dienen. Um ab einer bestimmten Sicherheitsstufe einschließlich in eine Zugriffsregel mit einbeziehen zu können, wird ein weiteres Prädikat subsecureeq benötigt, das ähnlich der Definition 3.16 von subroleeq(), definiert wird. Hierzu wird zunächst noch ein weiteres Prädikat benötigt, das die Gleichheit von Sicherheitsstufen abfragt.

**Definition 4.22 (Gleichheit Sicherheitsstufen)**

$$\textbf{equal\_secure}(sl_1, sl_2)$$

ist ein binäres Prädikat, das genau dann gültig ist, wenn die beiden Sicherheitsstufen $sl_1, sl_2 \in \mathcal{SLT}$ die gleiche Belegung haben.

Somit kann aufbauend auf den letzten drei definierten Kontext Feature Prädikaten, das folgende Prädikat definiert werden.

**Definition 4.23 (erweiterte Beziehung zwischen Sicherheitsstufen)**
Die indirekte Beziehung zwischen Sicherheitsstufen kann über die folgenden Regeln für $sl_1, sl_2 \in \mathcal{SLT}$ abgefragt werden, wobei $sl' \in \mathcal{V}_{\mathcal{SL}}$:

$$\textbf{subsecureeq}(sl_1, sl_2) \leftarrow \mathsf{equal\_secure}(sl_1, sl_2)$$
$$\textbf{subsecureeq}(sl_1, sl_2) \leftarrow \mathsf{securein}(sl_1, sl_2)$$
$$\textbf{subsecureeq}(sl_1, sl_2) \leftarrow \mathsf{subsecure}(sl_1, sl')\&\mathsf{securein}(sl', sl_2)$$

Im Gegensatz zur Beispielregel 4.8 können nun eine ganze Reihe von Sicherheitsstufen abgedeckt werden. So wird durch die folgende Regel Lesezugriff gewährt, wenn zwischen den beteiligten Hosts[16] ein Kommunikationskanal besteht, der folgende Eigenschaften aufweist: Die Authentifizierung erfolgte mittels RSA bei einer Schlüssellänge von mindestens 1024bit, der Kanal ist mit einem symmetrischen Verfahren verschlüsselt, dessen Schlüssellänge 128bit beträgt, der Schlüssel wurde mittels DiffieHellman oder Encrypted-Key-Exchange ausgetauscht.

**Beispielregel 4.9**

$$\begin{aligned}\mathsf{read}(s, o) \leftarrow &\mathsf{login}(s, h)\&\mathsf{storage}(o, h')\&\mathsf{channel}(h, h', ch)\& \\ &\mathsf{authenticated}(ch, au)\&\mathsf{subsecureeq}(RSA1024, au)\& \\ &\mathsf{encrypted}(ch, en)\&\mathsf{securein}(128bit - symmetric, en)\& \\ &\mathsf{key\_exchanged}(ch, ex)\&\mathsf{securein}(Key - Exchange, ex)\&\neg\mathsf{subsecure}(Public - Key, ex)\end{aligned}$$

Falls weitere Schlüsselaustauschverfahren in die Hierarchie in Abbildung 4.5 eingefügt werden, sind diese auch mit der obigen Regel abgedeckt, sofern sie nicht gleichzeitig zu den Public-Key Verfahren gehören.

Die in diesem Abschnitt 4.3 vorgestellten Prädikate dienen der Einstufung der Kommunikationskanäle. Die Menge $\mathcal{CF}_{\mathcal{C}}$ enthält alle Kontext Feature Prädikate für Kommunikationskanäle aus den Definitionen 4.16 bis 4.23.

[16] an dem das Subjekt eingeloggt ist, bzw. auf dem das Objekt gespeichert ist

### 4.3.3 Einstufung von Kommunikationskanälen

Mit diesen Kontext Feature Prädikaten für Kommunikationskanäle kann für eine Autorisierungssphäre eine Vertrauenseinstufung spezifiziert werden. Dabei können zusätzlich zu diesen Kontext Feature Prädikaten auch die Vertrauensstufen der Hosts mit einbezogen werden, da es keinen Sinn macht, eine stark kryptographisch abgesicherte Leitung zu fordern, wenn gleichzeitig der Host als "Insecure" eingestuft wird.

Durch das folgende Prädikat kann eine bestimmte Vertrauensstufe eines Kommunikationskanals durch eine Autorisierungssphäre spezifiziert werden.

**Definition 4.24 (Vertrauenstufe Kommunikationskanal)**
Die Regel für Vertrauenstufen von Kommunikationskanälen hat die folgende Form:

$$\textbf{channel\_level(}as, h_1, h_2, t\textbf{)} \leftarrow L_1 \& \ldots \& L_n$$

wobei $as \in \mathcal{AST}$ die Autorisierungssphäre ist, die festlegt, dass den Kommunikationskanal zwischen den Hosts $h_1, h_2 \in \mathcal{HT}$ die Vertrauensstufe $t \in \mathcal{TLT}$ zugesprochen wird, wenn alle Prädikate $L_i \in \mathcal{F} \cup \mathcal{CF}_{\mathcal{TL}} \cup \mathcal{CF}_{\mathcal{H}} \cup \mathcal{CF}_{\mathcal{C}} \cup \{\text{host\_level}, \text{derhost\_level}\}$ (mit $n \in \mathbb{N}$ und $1 \leq i \leq n$) gültig sind. Alle Regeln vom Typ host und derhost müssen aus der gleichen Autorisierungssphäre $as$ oder aus einer übergeordneten Autorisierungssphäre $as'$ sein ($as <_{\mathcal{AS}} as'$).

Somit kann für die Autorisierungssphäre Netzwerk die folgende Einstufung eines Kommunikationskanals als Secured spezifiziert werden. Dieses ähnelt dem Beispiel aus Regel 4.9, jedoch ist nun ein beliebiges Authentifizierungsverfahren mit mindestens 1024bit Länge zugelassen und eine Verschlüsselung des Kommunikationskanals mit einem symmetrischen Verfahren mit mindestens 128bit Schlüssellänge, wobei dieser Schlüssel nicht mittels eines Public-Key Verfahrens ausgetauscht wurde.

**Beispielregel 4.10**

$$\begin{aligned}\text{channel\_level}(Netzwerk, h_1, h_2, Secured) \leftarrow &\text{channel}(h_1, h_2, ch)\& \\ &\text{derhost\_level}(Netzwerk, h_1, Secured)\& \\ &\text{derhost\_level}(Netzwerk, h_2, Secured)\& \\ &\text{authenticated}(ch, au)\& \\ &\text{subsecureeq}(1024bit - public - key, au)\& \\ &\text{encrypted}(ch, en)\&\text{subsecure}(128bit - symmetric, en)\& \\ &\text{key\_exchanged}(ch, ex)\&\text{securein}(Key - Exchange, ex)\& \\ &\neg\text{subsecure}(Public - Key, ex)\end{aligned}$$

Darüber hinaus kann die Einstufung eines Kommunikationskanals in eine Vertrauensstufe in einer bestimmten Autorisierungssphäre auch spezifiziert werden, indem auf die anderen direkten Einstufungen zurückgegriffen wird.

**Definition 4.25 (abgeleitete Vertrauenstufe Kommunikationskanal)**
Die Regel für die abgeleitete Vertrauenstufe eines Kommunikationskanals hat die folgende Form:

$$\textbf{derchannel\_level(}as, h_1, h_2, t\textbf{)} \leftarrow L_1 \& \ldots \& L_n$$

wobei $as \in \mathcal{AST}$ die Autorisierungssphäre ist, die festlegt, dass dem Kommunikationskanal zwischen Host $h_1, h_2 \in \mathcal{HT}$ die Vertrauensstufe $t \in \mathcal{TLT}$ zugesprochen wird, wenn alle Prädikate $L_i \in \{\mathcal{F} \cup \mathcal{CF}_{\mathcal{TL}} \cup \mathcal{CF}_{\mathcal{H}} \cup \cup\mathcal{CF}_{\mathcal{C}} \cup \{\text{host\_level}, \text{derhost\_level}, \text{channel\_level}\}$ und $L_n \in \{\text{channel\_level}\}$ (mit festem $n \in \mathbb{N}$ und $1 \leq i < n$) gültig sind. Alle Regeln vom Typ host, derhost und channel müssen aus der gleichen Autorisierungssphäre $as$ oder aus einer übergeordneten Autorisierungssphäre $as'$ sein ($as <_{\mathcal{AS}} as'$).

Mit der folgenden Regel übernimmt die Autorisierungssphäre Netzwerk alle Einstufungen unterhalb von Secured zu einer beliebigen übergeordneten Autorisierungssphäre, sofern gleichzeitig auch die Hosts von Netzwerk indirekt unterhalb der Vertrauensstufe Secured eingeordnet sind.

**Beispielregel 4.11**

$$\begin{aligned}\mathsf{derchannel\_level}(Netzwerk, h_1, h_2, tp_c) \leftarrow & \mathsf{subsphere}(Netzwerk, as)\&\mathsf{sublevel}(tp_c, Secured)\& \\ & \mathsf{sublevel}(tp_1, Secured)\&\mathsf{sublevel}(tp_2, Secured)\& \\ & \mathsf{derhost\_level}(Netzwerk, h_1, tp_1)\& \\ & \mathsf{derhost\_level}(Netzwerk, h_2, tp_2)\& \\ & \mathsf{channel\_level}(as, h_1, h_2, tp_c)\end{aligned}$$

Die drei Variablen über Vertrauensstufen $tp_1, tp_2$ und $tp_c$ sind nötig, damit der Kommunikationskanal und die beiden Hosts, jeweils in drei verschiedene Vertrauensstufen eingeordnet werden können.

## 4.4 Vertrauenswürdigkeit der Applikationen

Wenn es möglich ist, bestimmte Daten durch eine unsichere Applikation verarbeiten zu lassen, sind die Anstrengungen bezüglich eines sicheren Kanals oder sicheren Hosts umsonst. Daher muss bei einer Zugriffsanfrage auch die Applikation bei der Entscheidung mitberücksichtigt werden. Hierzu muss die Menge der Applikationen $\mathcal{AP}$ bekannt sein. Eine Hierarchie über diese Applikationen zu bilden, hätte höchstens den Sinn, die verschiedenen Versionen einer Applikation zu organisieren. In dieser Arbeit wird darauf allerdings verzichtet. Eine entsprechende Hierarchie $<_{\mathcal{AP}}$ könnte ähnlich der für Betriebssysteme im Abschnitt 4.2.3 definiert werden und die entsprechenden Prädikate für die Vergleiche würden analog zu denen in den Definitionen 4.11 und 4.12 definiert werden.

### 4.4.1 Eigenschaften von Applikationen

Für die Definition der Kontext Feature Prädikate für Applikationen werden Terme über Applikationen benötigt, die sich aus der Menge der Variablen über Applikationen $\mathcal{V}_{\mathcal{AP}}$ und den bekannten Applikationen zusammensetzen ($\mathcal{APT} = \mathcal{V}_{\mathcal{AP}} \cup \mathcal{AP}$).

In dem Modell, das in Abbildung 4.1 skizziert ist, wird von drei Hosts ausgegangen, die an einer Zugriffsanfrage beteiligt sind. Der Host, an dem das Subjekt $s$ eingeloggt ist, kann über das Kontext Feature Prädikat $\mathsf{login}(s, h)$ bestimmt werden, der Host auf dem das Objekt $o$ gespeichert ist über $\mathsf{storage}(o, h)$. Mit dem folgenden Kontext Feature Prädikat wird der Host bestimmt, auf dem die Applikation läuft, die im Auftrag des Subjektes $s$ auf das Objekt $o$ zugreifen soll.

**Definition 4.26 (Ausführung der Applikation)**
Das dreistellige Prädikat

$$\textbf{execution}(app, s, h)$$

ist genau dann gültig, wenn die Applikation $app \in \mathcal{APT}$ auf dem Host $h \in \mathcal{HT}$ für das Subjekt $s \in \mathcal{ST}$ ausgeführt wird.

Neben der Einstufung in eine bestimmte Vertrauensstufe für eine laufende Applikation ist es auch nötig, die Applikation unabhängig von der verteilten Umgebung einzustufen. So können Applikationen aus fragwürdigen Quellen von solchen, bei denen ein entsprechendes Review stattgefunden hat, unterschieden werden.

**Definition 4.27 (Vertrauen in die Quellen der Applikation)**
Das dreistellige Prädikat

$$\textbf{application\_trusted}(as, app, t)$$

ist genau dann gültig, wenn die Applikation $app \in \mathcal{APT}$ von der Autorisierungssphäre $as \in \mathcal{AST}$ die Vertrauensstufe $t \in \mathcal{TLT}$ erhalten hat.

Die Menge $\mathcal{CF}_{\mathcal{APP}}$ enthält die beiden Kontext Feature Prädikate für Applikationen aus den Definitionen 4.26 und 4.27.

### 4.4.2 Einstufung von Applikationen

Mit den Einstufungen der Applikationen, die auf einem Host laufen und im Auftrag eines Subjektes auf ein Objekt zugreifen wollen, in eine bestimmte Vertrauensstufe können alle Komponenten, die im Modell in Abbildung 4.1 vorgestellt wurden, abgedeckt werden. Dabei kann auf die Einstufungen für Hosts und Kommunikationskanäle zurückgegriffen werden, die mit den zuvor in den Abschnitten 4.2.4 und 4.3.3 definierten Kontext Trust Prädikaten spezifiziert sind.

Das folgende Kontext Trust Prädikat ermöglicht die Festlegung einer bestimmten Vertrauensstufe durch eine Autorisierungssphäre für eine laufende Applikation.

**Definition 4.28 (Vertrauenstufe Applikation)**
Die Regel für die Vertrauenstufe einer Applikation hat die folgende Form:

$$\textbf{application\_level(}as, s, app, o, t\textbf{)} \leftarrow L_1 \& \ldots \& L_n$$

wobei $as \in \mathcal{AST}$ die Autorisierungssphäre ist, die festlegt, dass der Applikation $app \in \mathcal{APT}$ die von Subjekt $s \in \mathcal{ST}$ ausgeführt wird und auf das Objekt $o \in \mathcal{OT}$ zugreift die Vertrauensstufe $t \in \mathcal{TLT}$ zugesprochen wird, wenn alle Prädikate $L_i \in \mathcal{F} \cup \mathcal{CF}_{\mathcal{H}} \cup \mathcal{CF}_{\mathcal{C}} \cup \mathcal{CF}_{\mathcal{APP}} \cup \{\text{host\_level}, \text{derhost\_level}, \text{channel\_level}, \text{derchannel\_level}\}$ (mit festem $n \in \mathbb{N}$ und $1 \leq i \leq n$) gültig sind. Alle Regeln vom Typ host, derhost, channel und derchannel müssen aus der gleichen Autorisierungssphäre $as$ oder aus einer übergeordneten Autorisierungssphäre $as'$ sein ($as <_{\mathcal{AS}} as'$).

Somit kann die Autorisierungssphäre Netzwerk spezifizieren, unter welchen Umständen eine Applikation als Secured eingestuft wird.

**Beispielregel 4.12**

$$\begin{aligned}\text{application\_level}(Netzwerk, s, app, o, Secured) \leftarrow\ & \text{login}(s, h_s) \& \text{storage}(o, h_o) \& \text{execution}(app, s, h_a) \& \\ & \text{application\_trusted}(Network, app, Trusted) \& \\ & \text{channel\_level}(Netzwerk, h_s, h_a, Secured) \& \\ & \text{channel\_level}(Netzwerk, h_a, h_s, Secured)\end{aligned}$$

Darüber hinaus kann sich die Einstufung einer Applikation in eine Vertrauensstufe in einer bestimmten Autorisierungssphäre auch von der Einstufung in einer anderen Autorisierungssphäre abgeleiten, was durch folgendes Prädikat ausgedrückt wird.

**Definition 4.29 (abgeleitete Vertrauenstufe Applikation)**
Die Regel für die abgeleitete Vertrauenstufe einer Applikation hat die folgende Form:

$$\textbf{derapplication\_level(}as, s, app, o, t\textbf{)} \leftarrow L_1 \& \ldots \& L_n$$

wobei $as \in \mathcal{AST}$ die Autorisierungssphäre ist, die festlegt, dass der Applikation $app \in \mathcal{APT}$ die von Subjekt $s \in \mathcal{ST}$ ausgeführt wird und auf das Objekt $o \in \mathcal{OT}$ zugreift, die Vertrauensstufe $t \in \mathcal{TLT}$ zugesprochen wird, wenn alle Prädikate $L_i \in \mathcal{F} \cup \mathcal{CF}_{\mathcal{H}} \cup \mathcal{CF}_{\mathcal{C}} \cup \mathcal{CF}_{\mathcal{APP}} \cup \{\text{host\_level}, \text{derhost\_level}, \text{channel\_level}, \text{derchannel\_level}\}$ und $L_n \in \{\text{application\_level}\}$ (mit festem $n \in \mathbb{N}$ und $1 \leq i < n$) gültig sind. Alle Regeln vom Typ host, derhost, channel, derchannel und application müssen aus der gleichen Autorisierungssphäre $as$ oder aus einer übergeordneten Autorisierungssphäre $as'$ sein ($as <_{\mathcal{AS}} as'$).

Eine Möglichkeit für die Autorisierungssphäre Netzwerk wäre es auch die Vertrauensstufe der Applikation von der Einstufung der Autorisierungssphäre abhängig zu machen, auf dem diese Applikation läuft, sofern diese Autorisierungssphäre übergeordnet ist.

| | Terme | | Variablen | | Mengen | Halbordnung |
|---|---|---|---|---|---|---|
| Vertrauensstufe | $\mathcal{TLT}$ | = | $\mathcal{V}_{\mathcal{TL}}$ | $\cup$ | $\mathcal{TL}$ | $<_{\mathcal{TL}}$ |
| Lokalität | $\mathcal{LT}$ | = | $\mathcal{V}_{\mathcal{L}}$ | $\cup$ | $\mathcal{L}$ | $<_{\mathcal{L}}$ |
| Host | $\mathcal{HT}$ | = | $\mathcal{V}_{\mathcal{H}}$ | $\cup$ | $\mathcal{H}$ | |
| Betriebssystem | $\mathcal{OST}$ | = | $\mathcal{V}_{\mathcal{OS}}$ | $\cup$ | $\mathcal{OS}$ | $<_{\mathcal{OS}}$ |
| Sicherheitsstufe | $\mathcal{SLT}$ | = | $\mathcal{V}_{\mathcal{SL}}$ | $\cup$ | $\mathcal{SL}$ | $<_{\mathcal{SL}}$ |
| Kommunikationskanal | | | $\mathcal{V}_{\mathcal{C}}$ | | | |
| Applikation | $\mathcal{APT}$ | = | $\mathcal{V}_{\mathcal{AP}}$ | $\cup$ | $\mathcal{AP}$ | |

Tabelle 4.1: Terme und dazugehörige Variablenmengen und Mengen für Lokalitäten

**Beispielregel 4.13**

$$\begin{aligned}\mathsf{derapplication\_level}(Netzwerk, s, app, o, tp) \leftarrow &\mathsf{login}(s, h_s)\&\mathsf{storage}(o, h_o)\&\mathsf{execution}(app, s, h_a)\&\\ &\mathsf{assigned}(h_a, as)\&\mathsf{application\_trusted}(as, app, tp)\&\\ &\mathsf{subsphereeq}(Netzwerk, as)\end{aligned}$$

Die direkten und indirekten Einstufungen der Applikationen sind der Hauptansatzpunkt, um bei den Zugriffspolitiken die verteilte Umgebung einer Ausführung zu berücksichtigen. Darüber hinaus kann zum Beispiel bei Schreiboperationen eine höhere Sicherheit des Hosts gefordert werden, auf dem das Objekt gespeichert werden soll.

Alle Regeln zur Einstufung von Hosts, Kommunikationskanälen und Applikationen, die in einer Autorisierungssphäre $as$ spezifiziert werden, können auch in den entsprechenden Regeln der untergeordneten Autorisierungssphären $as'$ verwendet werden ($as' <_{\mathcal{AS}} as$). Dies gilt auch für die in den folgenden Kapiteln definierten Prädikate. Daher wird für die verteilte Auswertung angenommen, dass diese Regeln an die entsprechenden Hosts der untergeordneten Autorisierungssphären repliziert werden.

Für die weitere Definition der Prädikate im nächsten Kapitel werden alle **Kontext Feature Prädikate**, die in diesem Kapitel definiert sind, in einer Menge $\mathcal{CF}$ zusammengefasst:

$$\mathcal{CF} = \mathcal{CF}_{\mathcal{TL}} \cup \mathcal{CF}_{\mathcal{H}} \cup \mathcal{CF}_{\mathcal{C}} \cup \mathcal{CF}_{\mathcal{APP}}$$

Analog wird eine Menge $\mathcal{CL}$ definiert, die alle **Kontext Trust Prädikate** aus diesem Kapitel (vgl. Definitionen 4.14, 4.15, 4.24, 4.25, 4.28 sowie 4.29) enthält:

$$\begin{aligned}\mathcal{CL} = \{&\mathsf{host\_level}(), \mathsf{derhost\_level}(), \mathsf{channel\_level}(), \mathsf{derchannel\_level}(),\\ &\mathsf{application\_level}(), \mathsf{derapplication\_level}()\}\end{aligned}$$

Alle Mengen, Terme und Halbordnungen, die in diesem Kapitel definiert wurden, sind zur Übersicht in der Tabelle 4.1 zusammengefasst.

## 4.5 Weitere Beispiele für Vertrauensstufen

Neben den Eigenschaften die von der Autorisierungssphäre Netzwerk in den Beispielregeln 4.7, 4.10, 4.11, 4.12 und 4.13 werden im Folgenden einige Beispielregeln präsentiert, mit denen in den Autorisierungssphären Buchhaltung und Crypto Einstufungen für Vertrauensstufen vorgenommen werden. Auf diese Beispielregeln wird in den folgenden Kapiteln 5 und 7 zurückgegriffen.

In den folgenden Beispielregeln wird durch die Autorisierungssphäre Buchhaltung festgelegt, unter welchen Bedingungen ein Zugriff mit der Vertrauensstufe Privacy versehen wird. Für die beteiligten Hosts ist dabei ihr Standort basierend auf dem Beispiel in Abbildung 4.3 wichtig, da dieser entsprechende rechtliche Regelungen nach sich zieht (vgl.[BDS03]). Es wird davon ausgegangen, dass ab einem gewissen Grad an Verschlüsselung der Kommunikationsverbindung, die Sicherheit der personenbezogenen Daten gewährleistet werden kann.

**Beispielregeln 4.14**

$$\begin{aligned}
\mathsf{host_level}(Buchhaltung, h, Privacy) \leftarrow & \mathsf{sublocationof}(h, safe-harbor)\\
\mathsf{host_level}(Buchhaltung, h, Privacy) \leftarrow & \mathsf{sublocationof}(h, EU)\\
\mathsf{channel_level}(Buchhaltung, h_1, h_2, Privacy) \leftarrow & \mathsf{channel}(h_1, h_2, ch)\&\\
& \mathsf{host_level}(Buchhaltung, h_1, Privacy)\&\\
& \mathsf{host_level}(Buchhaltung, h_2, Privacy)\&\\
& \mathsf{authenticated}(ch, au)\&\mathsf{encrypted}(ch, en)\&\\
& \mathsf{subsecureeq}(1024bit-public-key, au)\&\\
& \mathsf{subsecure}(128bit-symmetric, en)\&\\
& \mathsf{key_exchanged}(ch, DiffieHellamn)\\
\mathsf{application_level}(Buchhaltung, s, app, o, Privacy) \leftarrow & \mathsf{login}(s, h_s)\&\\
& \mathsf{host_level}(Buchhaltung, h_s, Privacy)\&\\
& \mathsf{storage}(o, h_o)\&\mathsf{execution}(app, s, h_a)\&\\
& \mathsf{host_level}(Buchhaltung, h_o, Privacy)\&\\
& \mathsf{host_level}(Buchhaltung, h_a, Privacy)\&\\
& \mathsf{channel_level}(Buchhaltung, h_s, h_a, Privacy)\&\\
& \mathsf{channel_level}(Buchhaltung, h_a, h_s, Privacy)
\end{aligned}$$

In der Autorisierungssphäre Crypto werden beispielsweise die folgenden Regeln definiert, um den Zugriff als Crypto_Secured einzustufen. Für die Einstufung der Hosts ist zum einen das verwendete Betriebssystem gemäß Abbildung 4.4 von Bedeutung, zum anderen der Standort oder die verwendete Hardware und die zugeordnete Autorisierungssphäre.

**Beispielregeln 4.15**

$$\begin{aligned}
\mathsf{host_level}(Crypto, h, Crypto_Secured) \leftarrow & \mathsf{sublocationof}(h, Development)\&\mathsf{host_os}(h, os)\&\\
& \mathsf{subos}(Unix, os)\&\mathsf{subos}(Unstable, os)\\
\mathsf{host_level}(Crypto, h, Crypto_Secured) \leftarrow & \mathsf{host\text{-}arch}(h, TPM)\&\mathsf{assigned}(h, \mathsf{Crypto})\\
& \&\mathsf{host_os}(h, os)\&\mathsf{subos}(Unix, os)\&\\
& \neg\mathsf{subos}(Unstable, os)\\
\mathsf{channel_level}(Crypto, h_1, h_2, Crypto_Secured) \leftarrow & \mathsf{channel}(h_1, h_2, ch)\&\\
& \mathsf{host_level}(Crypto, h_1, Crypto_Secured)\&\\
& \mathsf{host_level}(Crypto, h_2, Crypto_Secured)\&\\
& \mathsf{authenticated}(ch, au)\&\mathsf{encrypted}(ch, en)\&\\
& \mathsf{subsecureeq}(2048bit-public-key, au)\&\\
& \mathsf{subsecure}(256bit-symmetric, en)\&\\
& \mathsf{key_exchanged}(ch, Encrypted-Key-Exchange)\\
\mathsf{application_level}(Crypto, s, app, o, Crypto_Secured) \leftarrow & \mathsf{login}(s, h_s)\&\mathsf{storage}(o, h_o)\&\\
& \mathsf{execution}(app, s, h_a)\&\\
& \mathsf{host_level}(Crypto, h_s, Crypto_Secured)\&\\
& \mathsf{host_level}(Crypto, h_o, Crypto_Secured)\&\\
& \mathsf{host_level}(Crypto, h_a, Crypto_Secured)\&\\
& \mathsf{channel_level}(Crypto, h_s, h_a, Crypto_Secured)\&\\
& \mathsf{channel_level}(Crypto, h_a, h_s, Crypto_Secured)
\end{aligned}$$

Für den Zugriff auf personenbezogene Daten werden in Crypto ähnliche Bedingungen festgelegt wie in Buchhaltung.

**Beispielregeln 4.16**

$$
\begin{aligned}
\mathsf{host\_level}(Crypto, h, Privacy) \leftarrow & \mathsf{sublocationof}(h, Company) \\
\mathsf{channel\_level}(Crypto, h_1, h_2, Privacy) \leftarrow & \mathsf{channel}(h_1, h_2, ch)\& \\
& \mathsf{host\_level}(Crypto, h_1, Privacy)\& \\
& \mathsf{host\_level}(Crypto, h_2, Privacy)\& \\
& \mathsf{authenticated}(ch, au)\&\mathsf{encrypted}(ch, en)\& \\
& \mathsf{subsecureeq}(1024bit - public - key, au)\& \\
& \mathsf{subsecure}(128bit - symmetric, en)\& \\
& \mathsf{key\_exchanged}(ch, DiffieHellamn) \\
\mathsf{application\_level}(Crypto, s, app, o, Privacy) \leftarrow & \mathsf{login}(s, h_s)\& \\
& \mathsf{host\_level}(Crypto, h_s, Privacy)\& \\
& \mathsf{storage}(o, h_o)\&\mathsf{execution}(app, s, h_a)\& \\
& \mathsf{host\_level}(Crypto, h_o, Privacy)\& \\
& \mathsf{host\_level}(Crypto, h_a, Privacy)\& \\
& \mathsf{channel\_level}(Crypto, h_s, h_a, Privacy)\& \\
& \mathsf{channel\_level}(Crypto, h_a, h_s, Privacy)
\end{aligned}
$$

An welche Bedingungen eine bestimmte Vertrauensstufe geknüpft wird, kann somit frei von den jeweiligen Autorisierungssphären einer Organisation festgelegt werden. Durch die Verwendung von entsprechenden Einstufungen der übergeordneten Autorisierungssphäre muss die Kompetenz hierzu nicht in jeder Autorisierungssphäre vorhanden sein.

# Teil II

# Dezentrale Spezifizierung von Zugriffsstrategien

# Kapitel 5

# Lokale Zugriffsstrategien

Ein wesentlicher Beitrag des vorgestellten Autorisierungssystems ist es, die Zugriffsstrategien dezentral zu spezifizieren und nicht durch einen zentralen Security Officer wie in [JSS97, JSSB97, JSSS01] vorgestellt, da dieser nicht über genügend Detailwissen verfügen kann, um für jede Organisationseinheit die optimalen Zugriffsstrategien zu entwickeln. Diese Zugriffsstrategien können dezentral durch sogenannte Autorisierungsteams definiert werden, die für eine Autorisierungssphäre und die entsprechend zugeordneten Organisationseinheiten (vgl. Abschnitt 3.3) verantwortlich sind. Jede Organisationseinheit mit einer eigenen Autorisierungssphäre zu versehen, in der lokale Zugriffsstrategien definiert werden, ist problemlos möglich. Nur ist dies aus organisatorischer Sicht nicht sinnvoll, da dann in jeder Organisationseinheit auch das entsprechende Personal vorhanden sein muss, das kompetent genug ist, die Zugriffsstrategien zu formulieren.

Die lokale Zugriffsstrategie einer Autorisierungssphäre wird mittels dreier Prädikate spezifiziert, die auf denen in [JSS97, JSSB97, JSSS01] aufbauen. Die Erweiterung, die in [WL01] das erste Mal vorgestellt wurde und in der vorliegenden Arbeit detaillierter behandelt wird, ermöglicht jedoch eine dezentrale Definition durch ein Autorisierungsteam und eine Zuordnung zu einer Autorisierungssphäre $as$. Mit diesen Prädikaten kann für ein Subjekt $s$, das mittels einer Applikation $app$ auf ein Objekt $o$ mit der Methode $a$ zugreifen will, ein Verbot oder eine Erlaubnis spezifiziert werden. Wobei die Feature Prädikate aus Kapitel 3 dazu dienen, die Rollen des Subjektes und die Objekte, sowie die entsprechenden Organisationseinheiten und Autorisierungssphären genauer zu spezifizieren. Ergänzend spezifizieren die Kontext Feature Prädikate und Kontext Trust Prädikate aus Kapitel 4 die verteilte Umgebung und die durchführende Applikation. Von diesen direkten Zugriffsprivilegien können für Benutzer mit anderen (übergeordneten) Rollen indirekte Zugriffsprivilegien abgeleitet werden.

Da das vorgestellte Autorisierungssystem hybride[1] Zugriffsprivilegien erlaubt, kann es innerhalb einer Autorisierungssphäre zu sogenannten **Widersprüchen** kommen. Diese treten dann auf, wenn für ein bestimmtes Subjekt der Zugriff auf ein Objekt mittels einer Applikation zum einen erlaubt, gleichzeitig aber auch verboten wird. Diese Widersprüche[2] müssen innerhalb einer Autorisierungssphäre gelöst werden. Hierzu wird ein drittes Prädikat (do) definiert, mit denen zum einen Widersprüche in den direkten und indirekten Zugriffsprivilegien aufgelöst werden können. Zum anderen können Abschlüsse (Closure) der Zugriffsstrategien spezifiziert werden, indem bisher undefinierte Zugriffe automatisch explizit erlaubt oder verboten werden. Dabei bedeutet explizit in diesem Zusammenhang, dass eine Regel aufgestellt wird, die diese Erlaubnis bzw. dieses Verbot spezifiziert. Die Zugriffsprivilegien, die durch Regeln dieser drei Prädikate geformt werden, werden im Folgenden als **lokale Zugriffsstrategien** eine Autorisierungssphäre bezeichnet. Dabei bezieht

[1] Im Gegensatz zu offenen Systemen in denen nur Verbote bzw. geschlossenen Systemen in denen nur Erlaubnisse definiert werden können.

[2] Als Konflikte werden in dieser Arbeit nur gegensätzliche Aussagen, die unter Berücksichtigung von Zugriffsstrategien aus zwei verschiedenen Autorisierungssphären zustande kommen, bezeichnet. Demgegenüber ist von Inkonsistenzen die Rede, wenn eine Autorisierungssphäre gegensätzliche eigene Entscheidungen an andere Autorisierungssphären weitergibt.

sich *lokal* darauf, dass die entsprechenden Regeln von *einem* Autorisierungsteam spezifiziert werden, nicht jedoch darauf, dass sie sich nur auf bestimmte Rollen oder Objekte beziehen. Bei der Vererbung und Zugriffsanfragen zwischen Autorisierungssphären wird auf diese lokalen Zugriffsstrategien zurückgegriffen, um das Ergebnis für eine Zugriffsanfrage zu ermitteln. Dabei dient das dritte Prädikat (do) als Einstiegspunkt für die Auswertung der **lokalen Zugriffsstrategien**. Im Abschnitt 5.1 werden die drei Prädikate für die lokalen Zugriffsstrategien definiert und Beispiele für ihre Verwendungsmöglichkeiten gegeben.

Durch die freie hybride Spezifizierung dieser lokalen Zugriffsstrategien kann für eine bestimmte Zugriffsanfrage sowohl eine do Regel gültig sein, die den Zugriff erlaubt, als auch eine, die dies verbietet. Dieses Phänomen ist als **Inkonsistenz** bekannt (vgl. u.a. [JSSS01, WJ03]) und würde auch bei der Vererbung oder Zugriffen zwischen Autorisierungssphären verhindern, dass eindeutige Auswertungen möglich sind. Mit der Einführung eines weiteren Prädikates könnte eine automatische Auflösung dieser Inkonsistenzen (bevorzugt zu einem Zugriffsverbot) erreicht werden, jedoch würde damit nur über die fehlerhafte Spezifizierung hinweggetäuscht. Daher wird in dieser Arbeit der Ansatz verfolgt, die Inkonsistenzen zu entdecken und gegebenenfalls die Auswertung einer Zugriffsanfrage mit einem Fehler abzubrechen. Im Abschnitt 5.2 wird dieses Vorgehen im Detail vorgestellt.

## 5.1 Prädikate für lokale Zugriffsstrategien

In den vorangegangen beiden Kapiteln (vgl. Abschnitt 3.1 und Einleitung vom Kapitel 4) wurde beispielhaft ein Prädikat $\mathsf{read}(s, o) \leftarrow L_i, L_i \in \{\mathcal{F} \cup \mathcal{CF} \cup \mathcal{CTL}\}$ genutzt, mit dem genau dann ein Lesezugriff auf ein Objekt $o$ durch ein Subjekt $s$ erlaubt wird, wenn alle Prädikate im Rumpf für diese Belegung von $s$ und $o$ gültig sind. Die Zugriffsanfragen im vorliegenden Ansatz beziehen sich auf ein Subjekt $s$, das mittels einer Applikation *app* auf ein Objekt $o$ mit der Zugriffsmethode $a$ zugreifen will. Dementsprechend müssen die Prädikate, mit denen die Zugriffsprivilegien spezifiziert werden, statt der zwei Parameter aus dem Beispielprädikat, diese vier Informationen als Parameter enthalten. Zusätzlich wird aufgrund der dezentralen Definition der Zugriffsprivilegien auch noch die Autorisierungssphäre *as* mit angegeben, deren Autorisierungsteam dieses Zugriffsprivileg definiert hat. Die Feature Prädikate ($\mathcal{F}$) und Kontext Feature Prädikate ($\mathcal{CF}$) sowie Kontext Trust Prädikate ($\mathcal{CTL}$) aus den vorangegangen Kapiteln 3 und 4 dienen dazu Subjekte, Applikationen und Objekte genau zu beschreiben.

Mit dem folgenden Access Prädikat können Zugriffsprivilegien direkt vergeben oder entzogen werden.

**Definition 5.1 (Grundregel)**
Die Grundregel hat die Form

$$\textbf{cando}(as, s, app, o, \langle sign \rangle a) \leftarrow L_1 \& \ldots \& L_n$$

wobei $as \in \mathcal{AS}$ die Autorisierungssphäre bezeichnet, die diese Regel aufgestellt hat. Falls alle Prädikate $L_i \in \mathcal{F} \cup \mathcal{CF} \cup \mathcal{CL}$ (mit festem $n \in \mathbb{N}$ und $1 \leq i \leq n$) mit einer entsprechenden Belegung für ein Subjekt $s \in \mathcal{V}_S$, das mittels einer Applikation $app \in \mathcal{V}_{\mathcal{AP}}$ auf ein Objekt $o \in \mathcal{V}_{\mathcal{O}}$ mit der Methode $a \in AC$ zugreifen will, gültig sind, wird diese Anfrage abhängig von $\langle sign \rangle$ direkt erlaubt oder verboten.

Mit diesem Prädikat können grundlegende Zugriffsrechte definiert werden, deren Gültigkeitsbereich durch die Prädikate im Regelrumpf spezifiziert wird. Wenn die Rollenhierarchie aus Abbildung 3.5 auf Seite 32 und die Hierarchie der Objekttypen aus Abbildung 3.6 auf Seite 35 zugrunde gelegt wird[3], wird durch die folgende Beispielregel von der Autorisierungssphäre Netzwerk allen Subjekten, die in einer Organisationseinheit unterhalb von Netzwerk die Rolle Programmer innehaben, Lese- bzw. Schreibzugriff auf Objekte vom Type Program_Source gewährt. Die verwendete Applikation wird von Netzwerk als Confirmed eingestuft.

[3]Dies ist auch für alle weiteren Beispiele der Fall.

**Beispielregeln 5.1**

$$\begin{aligned}
\mathsf{cando}(Netzwerk, s, app, o, +read) \leftarrow & \mathsf{typeof}(o, Program_Source)\& \\
& \mathsf{unitof}(o, u)\&\mathsf{role}(s, u, Programmer)\&\mathsf{unitin}(u, Netzwerk)\& \\
& \mathsf{application_trusted}(Netzwerk, app, Confirmed) \\
\mathsf{cando}(Netzwerk, s, app, o, +write) \leftarrow & \mathsf{typeof}(o, Program_Source)\& \\
& \mathsf{unitof}(o, u)\&\mathsf{role}(s, u, Programmer)\&\mathsf{unitin}(u, Netzwerk)\& \\
& \mathsf{application_trusted}(Netzwerk, app, Confirmed)
\end{aligned}$$

Eine alternative Vorgehensweise wird in der Autorisierungssphäre Crypto durch die folgende Beispielregel festgelegt, bei der die Vertrauensstufe der Applikation und verteilten Umgebung mitangegeben werden.

**Beispielregel 5.2**

$$\begin{aligned}
\mathsf{cando}(Crypto, s, app, o, +read) \leftarrow & \mathsf{typeof}(o, UML_Model)\&\mathsf{unitof}(o, u)\& \\
& \mathsf{role}(s', u, Developer)\&\mathsf{unitin}(u, Crypto)\&signature(o, s')\& \\
& \mathsf{role}(s, u, Programmer)\&\mathsf{unitin}(u, Crypto)\& \\
& \mathsf{application_level}(Crypto, s, app, o, Crypto_Secured)
\end{aligned}$$

Demnach wird jedem Subjekt, das die Rolle Programmer in einer Organisationseinheit unterhalb von Crypto innehat, dann Leserecht gewährt, wenn dieses Objekt die Signatur eines Subjektes aufweist, das in der gleichen Organisationseinheit die Rolle Developer innehat. Darüber hinaus muss die Applikation und der gesamte verteilte Zugriff in Crypto als Crypto_Secure eingestuft werden (vgl. Beispielregel 4.15 auf Seite 62).

Eine der Grundideen der rollenbasierten Zugriffskontrolle ist die **Weitergabe** von Zugriffsprivilegien entlang der Rollenhierarchie zu vererben (vgl. Abschnitt 3.4). Dies kann über sogenannte Ableitungsregeln geschehen.

**Definition 5.2 (Ableitungsregel)**
Die Ableitungsregel hat die Form

$$\textbf{dercando}(as, s, app, o, \langle sign\rangle a) \leftarrow L_1\& \ldots \&L_n$$

wobei $as \in \mathcal{AS}$ die Autorisierungssphäre bezeichnet, die diese Regel aufgestellt hat. Falls alle Prädikate $L_i \in \mathcal{F} \cup \mathcal{CF} \cup \mathcal{CL} \cup \{cando\}$ und $L_n \in \{cando\}$ (mit festem $n \in \mathbb{N}$ und $1 \leq i < n$) mit einer entsprechenden Belegung für Subjekt $s \in \mathcal{V}_S$, das mittels einer Applikation $app \in \mathcal{V}_{\mathcal{AP}}$ auf ein Objekt $o \in \mathcal{V}_{\mathcal{O}}$ mit der Methode $a \in AC$ zugreifen will, gültig sind, wird die Anfrage abhängig von$\langle sign\rangle$ indirekt erlaubt bzw. verboten. Alle Regeln vom Typ cando müssen von der gleichen Autorisierungssphäre $as$ sein.

Die Forderung, dass mindestens eine cando Prädikat enthalten ist, zielt darauf ab, dass nicht dercando Regeln geschaffen werden, die eigentlich nicht benötigt werden. Zudem müssen sämtliche Prädikate vom Typ cando von der gleichen Autorisierungssphäre definiert worden sein, wie die dercando Regel. Die Gründe hierfür liegen in der Vererbung der Regeln entlang der Autorisierungssphäre. Sollten beliebige cando Regeln aus anderen Autorisierungssphären referenziert werden können, würde es zu ungewünschten Seiteneffekten kommen, da dann nicht nur die Zugriffsregeln der angegebenen Autorisierungssphären verwendet würden, sondern implizit auch diejenigen aus weiteren Autorisierungssphären, auf die mit dercando referenziert wird.
Ausgehend von den obigen Regeln können somit wie folgt Zugriffsrechte weitergegeben werden.

**Beispielregeln 5.3**

$$\begin{aligned}
\mathsf{dercando}(Netzwerk, s, app, o, +read) \leftarrow & \mathsf{role}(s, u, Developer)\&\mathsf{subuniteq}(u, Netzwerk)\& \\
& \mathsf{unitof}(o, u)\&\mathsf{subtypeof}(o, Program_Executable)\& \\
& \mathsf{role}(s', u, Programmer)\& \\
& \mathsf{application_trusted}(Netzwerk, app, Confirmed)\& \\
& \mathsf{cando}(Netzwerk, s', app, o, +read) \\
\mathsf{dercando}(Netzwerk, s, app, o, +write) \leftarrow & \mathsf{role}(s, u, Developer)\&\mathsf{subuniteq}(u, Netzwerk)\& \\
& \mathsf{unitof}(o, u)\&\mathsf{subtypeof}(o, Program_Executable)\& \\
& \mathsf{role}(s', u, Programmer)\& \\
& \mathsf{application_trusted}(Netzwerk, app, Confirmed)\& \\
& \mathsf{cando}(Netzwerk, s', app, o, +write)
\end{aligned}$$

Die beiden obigen Regeln werden jeweils mit einer konkreten Belegung für das Subjekt $s$, die Applikation *app* und das Objekt $o$ bei einer Anfrage für einen Lese- bzw. Schreibzugriff ausgewertet. Damit alle Prädikate im Rumpf gültig werden, muss das Subjekt die Rolle Developer in einer Organisationseinheit $u$ innehaben, die mit Netzwerk identisch oder dieser Organisationseinheit untergeordnet ist. Das Objekt $o$ muss zur selben Organisationseinheit $u$ gehören und der Typ des Objektes kann unterhalb von Program_Executable einschließlich liegen. Außerdem muss für ein Subjekt $s'$, das die Rolle Programmer in derselben Organisationseinheit $u$ innehat, ein Lese- bzw. Schreibzugriff auf dasselbe Objekt $o$ durch die Autorisierungssphäre Netzwerk direkt erlaubt sein. In Datalog wird $s'$ als freie Variable bezeichnet, d.h. das bei der Auswertung des Prädikates verschiedene Belegungen für $s'$ probiert werden, bis eine Passende gefunden wird, bei der alle Prädikate gültig sind. Falls es keine Belegung für $s'$ gibt für die dies der Fall ist, wird auch diese Regel als nicht gültig angesehen. Das Prädikat kann allerdings noch über andere Regeln gültig werden, sofern bei einer Regel alle Prädikate im Rumpf gültig sind.

Ein wichtiger Aspekt ist hierbei, dass sich diese Ableitungsregeln aufgrund des Prädikates $\mathsf{role}(s', u, Programmer)$ auf die zugewiesenen Rollen von Subjekt $s'$ beziehen und nicht auf die aktive Rolle $\mathsf{activ\_role}(s', u, Programmer)$. Da ansonsten die Ableitung von Zugriffsprivilegien von den im Augenblick der Anfrage aktivierten Rollen abhängig ist, sollte das Prädikat activ_role() nicht in den Grundregeln (cando) und Ableitungsregeln (dercando) verwendet werden, außer dieser Effekt soll gerade erreicht werden.
Analog hierzu sollten in diesen Grundregeln (cando) und Ableitungsregeln (dercando) zunächst auch keine **Kontext Trust Prädikate** verwendet werden, die sich auf den Host beziehen, auf denen das Subjekt eingeloggt ist. Diese Bedingungen würden sich nicht auf das Subjekt beziehen, das tatsächlich die Zugriffsanfrage durchführt.

Ein Problem kann auftreten, wenn in einer Autorisierungssphäre *as* der Zugriff mittels Methode $a$ für eine bestimmte Kombination von Subjekt $s$ und Objekt $o$ zum einen direkt verboten wird ($\mathsf{cando}(as, s, app, o, -a)$), zum anderen eine Ableitungsregel $\mathsf{dercando}(as, s, app, o, +a)$ existiert, die diesen erlaubt. Das gleiche Problem wäre natürlich auch im umgekehrten Fall gegeben[4]. Als konkretes Beispiel wird angenommen, dass mit der folgenden Beispielregel 5.4 der Schreibzugriff auf Objekte vom Typ Program_Library für Subjekte mit der Rolle Developer direkt verboten ist.

**Beispielregel 5.4**

$$\begin{aligned}
\mathsf{cando}(Netzwerk, s, app, o, -write) \leftarrow & \mathsf{typeof}(o, Program_Library)\& \\
& \mathsf{unitof}(o, u)\&\mathsf{role}(s, u, Developer)\&\mathsf{unitin}(u, Netzwerk)\& \\
& \mathsf{application_trusted}(Netzwerk, app, tp)\&\mathsf{sublevel}(hostile, tp)
\end{aligned}$$

[4] $\mathsf{cando}(as, s, app, o, +a)$, $\mathsf{dercando}(as, s, app, o, -a)$

Um den dadurch entstandenen Widerspruch zwischen direkten und indirekten Zugriffsprivilegien zu lösen, werden die folgende Auflösungsregel definiert.

**Definition 5.3 (Auflösungsregel)**
Die Auflösungsregel hat die Form

$$\textbf{do(}as, s, app, o, \langle sign \rangle a\textbf{)} \leftarrow L_1 \& \ldots \& L_n$$

wobei $as \in \mathcal{AS}$ die Autorisierungssphäre bezeichnet, die diese Regel aufgestellt hat. Falls alle Prädikate $L_i \in \mathcal{F} \cup \mathcal{CF} \cup \mathcal{CL} \cup \{cando, dercando\}$ und $L_n \in \{cando \cup dercando\}$ (mit festem $n \in \mathbb{N}$ und $1 \leq i < n$) mit einer entsprechenden Belegung für das Subjekt $s \in \mathcal{V}_\mathcal{S}$, das mittels einer Applikation $app \in \mathcal{V}_{\mathcal{AP}}$ auf ein Objekt $o \in \mathcal{V}_\mathcal{O}$ mit der Methode $a \in AC$ zugreifen will, gültig sind, wird die Anfrage abhängig von $\langle sign \rangle$ zu einer Erlaubnis bzw. einem Verbot aufgelöst. Alle Regeln vom Typ cando oder dercando müssen von der gleichen Autorisierungssphäre $as$ sein.

Mittels Auflösungsregeln ist es möglich, innerhalb einer Autorisierungssphäre ein konsistentes Regelwerk zu erhalten. Der durch die Beispielregeln 5.3 im Zusammenhang mit Regel 5.1 auftretende Widerspruch mit Beispielregel 5.4 kann z.B. mit den folgenden Beispielregeln aufgelöst werden. Zudem werden mittels Feature Trust Prädikaten die Bedingungen für den Zugriff in einer verteilten Umgebung miteinbezogen.

**Beispielregeln 5.5**

$$\begin{aligned}
\mathsf{do}(Netzwerk, s, app, o, +read) \leftarrow & \mathsf{unitof}(o, u)\ \&\mathsf{unitin}(u, Netzwerk)\&\mathsf{role}(s, u, Developer)\& \\
& \mathsf{application_level}(Netzwerk, s, app, o, Secured)\& \\
& \mathsf{cando}(Netzwerk, s, app, o, +read) \\
\mathsf{do}(Netzwerk, s, app, o, +read) \leftarrow & \mathsf{unitof}(o, u)\ \&\mathsf{unitin}(u, Netzwerk)\&\mathsf{role}(s, u, Developer)\& \\
& \mathsf{application_level}(Netzwerk, s, app, o, Secured)\& \\
& \neg\mathsf{cando}(Netzwerk, s, app, o, -read)\& \\
& \mathsf{dercando}(Netzwerk, s, app, o, +read)
\end{aligned}$$

Die erste Beispielregel bewirkt, dass für einen Benutzer in der Rolle Developer in einer Untergruppe von Netzwerk ein Leserecht auf ein Objekt in dieser Untergruppe gegeben wird, wenn dieses direkt erlaubt ist. Die zweite Beispielregel besagt, dass es dem Benutzer gewährt wird, wenn er das Leserecht indirekt erhält und kein direktes Verbot dagegen spricht. Damit sind jedoch noch nicht alle möglichen Kombinationen von erlaubenden und verweigernden cando und dercando Regeln abgedeckt.

Im Abschnitt 1.3 wurde als ein Beispiel für Zugriffe zwischen Autorisierungssphären die inhaltliche Prüfung der Stundenzettel in der Buchhaltung angegeben. Die entsprechenden Regeln könnten in Buchhaltung bei denen ein Subjekt in einer übergeordneten Rolle der Lesezugriff auf Stundenzettel von Subjekten in untergeordneten Rollen gewährt wird, wie folgt aussehen, indem die Zuordnung des Stundenzettels über den Besitzer erfolgt.

**Beispielregeln 5.6**

$$\begin{aligned}
\mathsf{cando}(Buchhaltung, s, app, o, +read) \leftarrow & \mathsf{owner}(o, s_o)\&\mathsf{role}(s_o, u, r_o)\&\mathsf{typeof}(o, Stundenzettel)\& \\
& \mathsf{role}(s, u, r)\&\mathsf{subrole}(r_o, r)\& \\
& \mathsf{application_trusted}(Buchhaltung, app, Trusted) \\
\mathsf{do}(Buchhaltung, s, app, o, +read) \leftarrow & \mathsf{typeof}(o, Stundenzettel)\& \\
& \mathsf{application_level}(Buchhaltung, s, app, o, Privacy)\& \\
& \mathsf{cando}(Buchhaltung, s, app, o, +read)\& \\
& \neg\mathsf{dercando}(Buchhaltung, s, app, o, -read)
\end{aligned}$$

In der Autorisierungssphäre Crypto sollte der Zugriff auf fremde Stundenzettel nur von Subjekten in der Rolle Project-Manager möglich sein. Dementsprechend sehen die Regeln wie folgt aus.

**Beispielregeln 5.7**

$$\begin{aligned}
\mathsf{cando}(Crypto, s, app, o, +read) \leftarrow\ & \mathsf{owner}(o, s_o)\&\mathsf{role}(s_o, u, r_o)\mathsf{typeof}(o, Stundenzettel)\& \\
& \mathsf{role}(s, u, \mathsf{ProjectManager})\&\mathsf{subrole}(r_o, \mathsf{ProjectManager})\& \\
& \mathsf{application_trusted}(Crypto, app, Trusted) \\
\mathsf{do}(Crypto, s, app, o, +read) \leftarrow\ & \mathsf{typeof}(o, Stundenzettel)\& \\
& \mathsf{application_level}(Crypto, s, app, o, Privacy)\& \\
& \mathsf{cando}(Crypto, s, app, o, +read)\& \\
& \neg\mathsf{dercando}(Crypto, s, app, o, -read) \\
\mathsf{do}(Crypto, s, app, o, -read) \leftarrow\ & \mathsf{typeof}(o, Stundenzettel)\& \\
& \neg\mathsf{cando}(Crypto, s, app, o, +read)
\end{aligned}$$

Mit diesen drei Prädikaten können in der vorliegenden Arbeit[5] die lokalen Zugriffsstrategien einer Autorisierungssphäre spezifiziert werden.

## 5.2 Inkonsistenzen bei lokalen Zugriffsstrategien

Eine Schwierigkeit bei der Definition von hybriden Zugriffsstrategien ist die Konsistenz der Regeln, die durch die Spezifizierung einer expliziten Erlaubnis und eines expliziten Verbotes für dieselbe Kombination aus Subjekt, Applikation und Objekt aufgehoben wird. Daher muss für die lokalen Zugriffsstrategien gelten:

**Vereinbarung 5.1**
In einer Autorisierungssphäre $as$ ist für ein Subjekt $s$, das durch eine Applikation $app$ mit der Zugriffsmethode $a$ auf ein Objekt $o$ zugreifen will, nie sowohl das Prädikat $\mathsf{do}(as, s, app, o, +a)$ als auch $\mathsf{do}(as, s, app, o, -a)$ gültig.

Bei komplexeren und umfangreicheren Zugriffsstrategien können sich die Gültigkeitsbereiche der verschiedenen do Regeln überlappen, ohne dass dieses sofort bemerkt wird. Dies könnte zu einem nicht–deterministischen Verhalten bei der Auswertung der Zugriffsanfragen führen. Um Inkonsistenzen innerhalb einer lokalen Zugriffsstrategie erkennen und eine entsprechende Auswertung mit einem Fehler abbrechen zu können wird die folgende Regel eingeführt.

**Definition 5.4 (lokale Inkonsistenzregeln)**
Die lokale Inkonsistenzregel hat die folgende Form

$$\textbf{inconsistent-do}(as, s, app, o, a) \leftarrow \mathsf{do}(as, s, app, o, +a)\&\mathsf{do}(as, s, app, o, -a)$$

wobei $as \in \mathcal{AS}$ die Autorisierungssphäre bezeichnet, in der diese Regel ausgewertet wird. Falls die Regel mit einer entsprechenden Belegung für $s \in \mathcal{V}_\mathcal{S}$, $app \in \mathcal{V}_{\mathcal{AP}}$ und $o \in \mathcal{V}_\mathcal{O}$ mit Methode $a \in \mathcal{V}_\mathcal{A}$ gültig ist, wird die Zugriffsanfrage mit einem Fehler abgebrochen.

Der Datalog–Interpreter für die Auswertung der Zugriffsanfragen müsste dann so realisiert werden, dass er bei jeder ersten Auswertung von $\mathsf{do}(as, s, app, o, +a)$ oder $\mathsf{do}(as, s, app, o, -a)$ mit einer neuen Belegung für $as, s, app, o, a$ automatisch auch $\mathsf{inconsistent\text{-}do}(as, s, app, o, a)$ auswertet

[5] Auf die Definition des Prädikates decide, dass in [WL01] eingeführt wurde, um interaktiv durch Einzelfallentscheidung eines verantwortlichen Benutzers den Zugriff zu gewähren, wird in dieser Arbeit verzichtet. Die Entscheidungsgrundlagen können auch durch entsprechende Regeln modelliert und bei Bedarf jederzeit angepasst werden.

und mit einer Fehlermeldung abbricht, wenn dieses Prädikat gültig ist. Alternativ könnte ein Aufruf dieses Prädikates in jede Regel hinzugefügt werden, die auch ein do enthält. Da dieses Prädikat aber auch in Regeln auftauchen kann, die von Autorisierungsteams spezifiziert werden (siehe Def. 6.2), lässt sich dieser Aufruf nicht garantieren.

Diese lokalen Zugriffsstrategien können an untergeordnete Autorisierungssphären vererbt werden. Wie dies geschieht wird im nächsten Kapitel erläutert. Darüber hinaus werden die lokalen Zugriffsstrategien auch bei Zugriffsanfragen zwischen Autorisierungssphären ausgewertet.
Für die dezentrale Auswertung der Zugriffsanfragen wird angenommen, dass die lokalen Zugriffsstrategien, sofern nötig, in den untergeordneten Autorisierungssphären durch Replizierung bekannt gegeben werden. Weitere Details hierzu folgen am Ende des nächsten Kapitels.

# Kapitel 6

# Vererbung von Zugriffsstrategien

Die lokalen Zugriffsstrategien aus dem vorangegangenen Kapitel können völlig autonom vom zuständigen Autorisierungsteam festgelegt werden. Aus organisationstheoretischer Sicht ist eine völlig autonome Spezifizierung der Zugriffsregeln überhaupt nicht wünschenswert. "Allerdings bleibt eine Restverantwortung bei der delegierenden Stelle. Ansonsten käme es im Zuge der Enthierarchisierung zu einer vollständigen Unternehmensauflösung." [PDF99, S.223] Um dem entgegen zu wirken, wurde bereits in [WL01, WL03b, WL04a] die Möglichkeit einer Autorisierungssphäre vorgestellt, ihre Zugriffsstrategien an untergeordnete Autorisierungssphären zu vererben. Diese untergeordneten Autorisierungssphären müssen die geerbten Zugriffsstrategien befolgen. In der vorliegenden Arbeit wird dieser Ansatz noch erweitert, sodass eine korrekte Durchsetzung dieser vererbten Zugriffsstrategien nachgewiesen werden kann. Die Vererbung der Zugriffsstrategien nutzt die hierarchische Strukturierung einer Organisation aus und folgt der Hierarchie der Autorisierungssphären (vgl. Abschnitt 3.3).

Diese Vererbung soll es einem Autorisierungsteam ermöglichen, bestimmte Zugriffsstrategien in untergeordneten Autorisierungssphären durchzusetzen. Dies kann aus verschiedenen Gründen nötig sein z.B.:

- Rechtliche Grundlagen verlangen eine bestimmte Einschränkung auf den Zugriff der Daten (Datenschutz, Personalakten etc.).
- Bestimmte Daten sind zur Sicherung der betrieblichen Aufgaben/Fortbestand nur einem eingeschränktem Personenkreis zugänglich (z. B. "Coca-Cola Rezept", Planungsdaten).
- Betriebliche Abläufe werden durch Zugriffsrechte koordiniert und sollen in (Teilen) der Organisation durchgesetzt werden.

Während die ersten beiden Punkte nahe legen, dass die entsprechenden Zugriffsstrategien für alle untergeordneten Autorisierungssphären gelten, kann eine Vererbung von bestimmten Betriebsabläufen nur an eingeschränkte Teile der Organisation sinnvoll sein. Eine detaillierte Vorstellung dieses **Vererbungsprinzips** erfolgt im Abschnitt 6.1.

Das Autorisierungsteam kann mittels spezieller **Vererbungsregeln** festlegen, welche lokalen Zugriffsstrategien in bestimmten untergeordneten Autorisierungssphären und/oder Organisationseinheiten gelten sollen. Darüber hinaus können auch direkt Zugriffsprivilegien mit diesen Vererbungsregeln vergeben werden, wodurch eine schnellere Auswertung einer Zugriffsanfrage ermöglicht wird. Im Abschnitt 6.2 wird das Prädikat für die Vererbungsregeln definiert und die obigen Beispiele in entsprechende Regeln umgesetzt.

Wenn eine Organisationsstruktur kein reines Einliniensystem[1] aufweist und zudem hybride Zugriffsstrategien spezifiziert werden, kann es bei der Vererbung der Regeln zu sogenannten **Vererbungskonflikten** kommen. Diese entstehen genau dann, wenn von zwei übergeordneten Autorisierungssphären, die nicht miteinander in Beziehung stehen, für eine bestimmte Zugriffsanfrage einmal ein Verbot und zum anderen eine Erlaubnis vererbt wird. Im Abschnitt 6.3 werden

[1] Dies schließt nicht nur Mehrliniensysteme, sondern auch Einliniensysteme mit Stabshierarchie ein.

die Details zur Entstehung von Vererbungskonflikten betrachten und verschiedene Ansätze für Lösungsstrategien diskutiert.

Bei der Vorstellung der generellen Idee der Vererbung in [WL01] wurde dieses durch ein einziges zusätzliches Prädikat realisiert[2], dessen Regeln in einer übergeordneten Autorisierungssphäre festgelegt wurden. Dabei mussten diese Regeln eine bestimmte Form haben, damit eine Vererbung stattfand. Im Abschnitt 6.4 wird eine Umsetzung des Vererbungsprinzips mittels eines zusätzlichen Prädikates und entsprechenden Regeln vorgestellt, die das Vererbungsprinzip nachweislich korrekt umsetzt, ohne dass irgendein Autorisierungsteam dies unterbinden kann.

## 6.1 Vererbungsprinzip

Die Grundidee des vorliegenden Ansatzes ist es innerhalb einer Organisation die Zugriffsprivilegien möglichst dezentral in den Autorisierungssphären durch gemeinschaftlich verantwortliche Autorisierungsteams spezifizieren zu lassen. Jedoch sind diese Autorisierungssphären nicht völlig autonom und fast alle Organisationen ab einer bestimmten Größe weisen eine hierarchische Strukturierung auf. Diese spiegelt sich zum einen in der Rollenhierarchie wieder und kann wie in [SCFY96][NO96] vorgeschlagen wurde, zur Vererbung der Zugriffsprivilegien entlang der Rollenhierarchie genutzt werden. Dies hat zur Folge, dass eine übergeordnete Rolle alle Zugriffsprivilegien einer untergeordneten Rolle erhält.
Bei einer dezentralen Spezifizierung der Zugriffsprivilegien ist es sinnvoll, dass zusätzlich[3] die Möglichkeit besteht, entlang der Hierarchie der Autorisierungssphären bestimmte Zugriffsstrategien durchzusetzen. Diese wird im vorliegenden Ansatz als **Vererbung von Zugriffsstrategien** bezeichnet.

Damit soll erreicht werden, dass einerseits explizite Verbote und Zugriffe in der gesamten Organisation durchgesetzt werden, um z.B. Datenschutz oder Firmengeheimnisse zu sichern. Es kann beispielsweise von der Geschäftsleitung festgelegt werden, dass allen Subjekten in einer beliebigen Rolle unterhalb von Senior Project-Manager der Zugriff auf Objekte, bei denen das Attribut Classification den Wert Secret hat, verboten wird. Wenn dieses an alle untergeordneten Autorisierungssphären vererbt wird, kann dadurch der Zugriff auf diese Art der Objekte entsprechend eingeschränkt werden.
Auf der anderen Seite kann es sinnvoll sein, dass eine Autorisierungssphäre nur für bestimmte Teile der Organisation Zugriffsstrategien vorgibt. So kann die Autorisierungssphäre wie Software ihre lokalen Zugriffsstrategien für bestimmte Rollen wie Developer, Programmer und Tester auf Objekte bestimmten Typs wie Program_Source oder UML_File an die untergeordneten Autorisierungssphären vererben. Damit müssen sich die untergeordneten Autorisierungssphären nur um bestimmte Detailfälle kümmern, die bisher nicht mit den Zugriffsregeln abgedeckt sind.
In beiden Fällen muss sichergestellt sein, dass für die Autorisierungsteams der untergeordneten Autorisierungssphären keine Möglichkeit besteht, die so geerbten Zugriffsprivilegien zu umgehen und somit auch die Bestimmungen zum Datenschutz und zur Sicherung der Firmengeheimnisse zu unterlaufen.

Eine generelle Vererbung aller lokalen Regeln an alle untergeordneten Autorisierungssphären würde die Auswertung erheblich aufwendiger machen, da dann auch alle lokalen Zugriffsregeln auf gültige Prädikate überprüft werden müssen. Damit würde auch ein Vorteil dieses dezentralen Autorisierungsansatzes aufgeben, der darauf beruht, dass der Regelsatz einer Autorisierungssphären, gegenüber einem zentralen unternehmensweiten Satz von Zugriffsregeln, relativ klein bleiben kann. Um nicht alle lokalen Zugriffsstrategien der übergeordneten Autorisierungssphären auswerten zu müssen, kann statt dessen das Autorisierungsteam festlegen, welche lokalen Zugriffsstrategien in welchen untergeordneten Autorisierungssphären gelten sollen.

[2] Dieses Prädikat grant() wurde auch für die Konfliktlösung zwischen Autorisierungssphären eingesetzt. In dieser Funktion wird es auch in dieser Arbeit im Abschnitt 7.3 vorgestellt.

[3] Im Abschnitt 5.1 wurde mit den Beispielregeln 5.3 gezeigt, wie diese Vererbung entlang der Rollenhierarchie in einer Autorisierungssphäre spezifiziert werden kann.

Dabei würden Regeln, die den Datenschutz und Betriebsgeheimnisse betreffen, von der Autorisierungssphäre Geschäftsleitung an alle untergeordneten Autorisierungssphären vererbt. Diese Regeln werden dann in den untergeordneten Autorisierungssphären gültig und die entsprechenden Verpflichtungen zum Datenschutz wären erfüllt. Eine andere Vorgehensweise muss bei den betrieblichen Abläufen, zum Beispiel wie zuvor für Software vorgestellt, gelten. Hier erscheint es aus organisatorischer Sicht sinnvoll, dass die Vererbung nicht an alle untergeordneten Autorisierungssphären erfolgt, sondern nur an die direkten Nachfolger. Diese können dann entscheiden, welche betrieblichen Abläufe bei ihren nachfolgenden Autorisierungssphären sinnvoll sind und diese dann gezielt weitervererben, während überflüssige Regeln nicht weitergegeben werden und damit nicht bindend sind.

Neben dieser Einschränkung wie "Tief" die Regeln vererbt werden sollen, ist auch eine Einschränkung in der "Breite" sinnvoll. Dies würde bedeuten, dass die Erbregeln nur in bestimmten nachgeordneten Pfaden angewendet werden, in allen anderen jedoch nicht.

In der Beispielorganisation aus Abbildung 3.3 könnte die Autorisierungssphäre Entwicklung festlegen, dass alle von dieser Autorisierungssphäre definierten Zugriffsstrategien bezüglich Dokumenten, die eine VDE Zertifizierung betreffen[4], von allen Autorisierungssphären, die zu Hardware oder einer darunter angesiedelten Autorisierungssphäre gehören, beachtet werden müssen. In der Autorisierungssphäre Software werden beim Zugriff auf diese Art von Objekten, die Vorgaben nicht vererbt und sind damit nicht bindend.

Die Vererbung, sowohl in der Tiefe als auch in der Breite einzuschränken, sollte auch möglich sein. Dann würden beispielsweise die erwähnten Regeln für die VDE Zertifizierung nur an die Autorisierungssphäre Hardware vererbt werden. Insgesamt erscheint es sinnvoll, dass eine gezielte Vererbung von bestimmten Teilen der Regeln, die in einer Autorisierungssphäre definiert sind, möglich ist. Die Autorisierungssphären, an denen die Regeln vererbt werden, können außerdem eingeschränkt werden.

Eine Vererbung von Zugriffsstrategien an eine untergeordnete Autorisierungssphäre $as_l$ von einer Autorisierungssphäre $as_h$ macht nur Sinn, wenn sichergestellt werden kann, dass bei einer entsprechenden Zugriffsanfrage bezüglich eines aktiven Subjektes $s$ und/oder[5] Objektes $o$ dieser untergeordneten Autorisierungssphäre, die entsprechenden Zugriffsprivilegien durchgesetzt werden. Durch die dezentrale Spezifizierung aller Zugriffsstrategien besteht die Möglichkeit, dass in der untergeordneten Autorisierungssphäre ebenfalls Zugriffsregeln aufgestellt wurden, die ebenfalls auf diese Zugriffsanfrage zutreffend wären. Ebenso könnte eine weitere Autorisierungssphäre $as'$ mit $as_l <_{\mathcal{AS}} as' <_{\mathcal{AS}} as_h$ auch Zugriffsstrategien an $as_l$ vererben. Die Frage ist wie mit diesen Möglichkeiten umgegangen werden soll.

Theoretisch besteht die Möglichkeit bei jeder neuen Spezifizierung einer lokalen Zugriffsregel oder einer Vererbungsregel zu überprüfen, ob für alle Kombinationen aus Subjekt $s$, Applikation $app$ und Objekt $o$ sowie der Zugriffsmethode $a$ nicht bereits eine Zugriffsregel vererbt wird. Dieses ist sehr aufwendig, da alle möglichen Kombinationen betrachtet werden müssen. Entsprechend überfordert wäre auch ein Autorisierungsteam, das dieses stets bei der Spezifizierung ausschließen müsste. Wobei im letzteren Fall noch nicht geklärt ist, wie verhindert werden kann, dass ein Autorisierungsteam böswillig Zugriffsregeln aufstellen kann, mit denen geerbte Zugriffsstrategien unterlaufen würden.

Die beste Vorgehensweise ist es daher, dafür zu sorgen, dass bei einer Zugriffsanfrage für die in einer Autorisierungssphäre mehre Zugriffsstrategien vererbt werden, die der obersten vererbenden Autorisierungssphäre durchgesetzt werden. Falls eine untergeordnete Autorisierungssphäre eine Regel aufstellt, die bereit von einer geerbten Regel aus einer übergeordnete Autorisierungssphäre abgedeckt wird, wird diese nicht berücksichtigt. Dadurch wird erreicht, dass eine untergeordnete Autorisierungssphäre niemals die Zugriffsstrategien einer übergeordneten Autorisierungssphäre unterlaufen kann. Insgesamt lässt sich diese Vorgehensweise wie folgt zusammenfassen:

[4] Dies könnte durch ein freies Attribut oder durch ein Objekttyp wie "Abnahmeprotokoll VDE" gekennzeichnet sein.

[5] Auch bei Zugriffen zwischen Autorisierungssphären, wie sie im Kapitel 7 erläutert werden, soll weiterhin eine Vererbung von Zugriffsstrategien durchgesetzt werden.

**Definition 6.1 (Vererbungsprinzip)**
Das Autorisierungsteam jeder Autorisierungssphäre $as$ kann gezielt festlegen, für welches Subjekt $s$, das mittels einer Applikation $app$ auf ein Objekt $o$ mit der Methode $a$ zugreifen will, in einer untergeordneten Autorisierungssphären $as_l$ ($as_l <_{\mathcal{AS}} as$) die eigenen (lokalen) Zugriffsprivilegien gelten sollen. Dabei können die Subjekte über (aktive) Rollen in Organisationseinheiten, die Applikationen bezüglich ihrer Vertrauensstufe und die Objekte hinsichtlich ihres Typs oder anderer Attribute eingeschränkt werden. Sofern keine übergeordnete Autorisierungssphäre $as_h$ ($as <_{\mathcal{AS}} as_h$) ebenfalls Zugriffsprivilegien für die gleiche Kombination $s, app, o, a$ an diese untergeordnete Autorisierungssphäre $as_l$ vererbt, werden die Zugriffsregeln von $as$ berücksichtigt.

Dieses Vererbungsprinzip wird in den folgenden Abschnitten des Kapitels umgesetzt. Die Spezifizierung welche Zugriffsstrategien an untergeordnete Autorisierungssphären weitergegeben werden, erfolgt durch sogenannte **Vererbungsregeln**. Die Umsetzung dieses Vererbungsprinzips wird durch sogenannte **Erbregeln** ebenfalls in Datalog modelliert. Dadurch kann auch bewiesen werden, dass diese Erbregeln das Vererbungsprinzip korrekt umsetzen[6].

## 6.2 Vererbungsregel

Die Vererbungsregeln ermöglichen dem Autorisierungsteam einer Autorisierungssphäre $as$ festzulegen für welche Subjekte $s$, die mittels einer Applikation $app$ auf ein Objekt $o$ mit Methode $a$ zugreifen wollen, welche Zugriffsstrategien angewendet und damit vererbt werden sollen. Durch die Feature Prädikate und Kontext Feature Prädikate sowie Kontext Trust Prädikate kann der Gültigkeitsbereich der Vererbungsregeln eingeschränkt werden, insbesondere können auch die Autorisierungssphären eingeschränkt werden, zu denen die Organisationseinheiten gehören, in denen die Rolle aktiviert ist oder die dem Objekt zugeordnet ist.
Bei den Zugriffsstrategien können die lokalen Zugriffsstrategien miteinbezogen werden oder auch Zugriffsprivilegien direkt mittels Vererbungsregeln festgelegt werden. Für eine schrittweise Vererbung kann auf die lokalen Zugriffsstrategien der übergeordneten Autorisierungssphären zurückgegriffen werden.

**Definition 6.2 (Vererbungsregel)**
Die Vererbungsregel hat die Form

$$\textbf{bequeath(}as, s, app, o, \langle sign \rangle a\textbf{)} \leftarrow L_1 \& \ldots \& L_n$$

wobei $as \in \mathcal{AS}$ die Autorisierungssphäre bezeichnet, die die Vererbung festlegt. Falls alle Prädikate $L_i \in \mathcal{F} \cup \mathcal{CF} \cup \mathcal{CL} \cup \{do\}$ (mit festem $n \in \mathbb{N}$ und $1 \leq i < n$) mit einer entsprechenden Belegung für ein Subjekt $s \in \mathcal{V}_{\mathcal{S}}$, das mittels einer Applikation $app \in \mathcal{V}_{\mathcal{AP}}$ auf ein Objekt $o \in \mathcal{V}_{\mathcal{O}}$ mit einer Methode $a \in \mathcal{AC}$ zugreifen will, gültig sind, wird der Zugriff gemäß $\langle sign \rangle$ mittels Vererbung erlaubt oder verboten.
Alle Regeln vom Typ do müssen aus der gleichen Autorisierungssphäre $as$ oder aus einer übergeordneten Autorisierungssphäre $as'$ sein ($as <_{\mathcal{AS}} as'$).

Die Vererbungsregeln werden bei einer Zugriffsanfrage für eine konkrete Belegung eines Subjektes $s$, einer Applikation $app$, eines Objektes $o$ und der entsprechenden Methode $a$ ausgewertet. Für alle Kombinationen $s, app, o$ die die Bedingungen, die mit den Feature Prädikaten, Kontext Feature Prädikaten oder Kontext Trust Prädikaten spezifiziert werden, nicht erfüllen [7], ist es nicht

[6] Im Abschnitt 6.3 zeigt sich, dass das Vererbungsprinzip nur dann umgesetzt werden kann, wenn kein Vererbungskonflikt vorliegt.

[7] Für eine schnelle Auswertung versucht der Prototyp des Interpreters [Hal05] zunächst alle *EPred* der Datalog-Sprache auszuwerten. Dies sind insbesondere fast alle Feature Prädikate und fast alle Kontext Feature Prädikate[8]. Da diese direkt ausgewertet werden können, wird festgestellt, ob das entsprechende Subjekt oder Objekt sowie die Applikation zu dem durch die Feature und Kontext Feature Prädikate definierten Bereich gehört. Entsprechend wird eine eventuell enthaltenes do() Prädikat gar nicht erst ausgewertet und somit die "lokalen Zugriffsstrategien" aus Kapitel 5 nicht vererbt.

von Bedeutung, ob das enthaltene Prädikat do() gültig ist. Die entsprechende Regel wird nicht gültig.

Durch die Vererbung von lokalen Zugriffsstrategien, die von einer übergeordneten Autorisierungssphäre $as'$ definiert werden, lässt sich bei der zuvor erwähnten schrittweisen Vererbung an die direkten Nachfolger auch eine Weitervererbung durch diese Nachfolger realisieren. Falls nur die lokalen Regeln der aktuellen Autorisierungssphäre im Rumpf des Prädikates bequeath() zugelassen sind, wäre eine schrittweise Vererbung nicht möglich. Da das do() Prädikat nicht verpflichtend ist, lassen sich auch direkte Zugriffsstrategien definieren, die bisher nicht durch lokale Regeln spezifiziert sind. Hierdurch wird der Auswertungspfad entsprechend verkürzt.

### 6.2.1 Beispiele für Vererbungsregeln

In der Entwicklungsabteilung werden die Zugriffsprivilegien für Objekte, die mit einer VDE-Zertifizierung im Zusammenhang stehen, als lokale Zugriffsstrategien spezifiziert. Dabei wird angenommen, dass diese Objekte durch ein entsprechendes Attribut gekennzeichnet werden. Die Zugriffsstrategien bezüglich der Leserechte werden mit den folgenden Vererbungsregeln an die Autorisierungssphäre Hardware und den ihr untergeordneten Autorisierungssphären vererbt und gelten damit dort für Zugriffe auf Objekte mit dem entsprechenden Attribut.

**Beispielregeln 6.1**

$$\begin{aligned}
\mathsf{bequeath}(Entwicklung, s, app, o, +read) \leftarrow\ & \mathsf{attribute}(o, Zertifizierung, =, VDE)\&\mathsf{unitof}(o, u)\& \\
& \mathsf{sphereof}(u, as)\&\mathsf{subsphereeq}(as, Hardware)\& \\
& \mathsf{do}(Entwicklung, s, app, o, +read) \\
\mathsf{bequeath}(Entwicklung, s, app, o, -read) \leftarrow\ & \mathsf{attribute}(o, Zertifizierung, =, VDE)\&\mathsf{unitof}(o, u)\& \\
& \mathsf{sphereof}(u, as)\&\mathsf{subsphereeq}(as, Hardware)\& \\
& \mathsf{do}(Entwicklung, s, app, o, -read)
\end{aligned}$$

Durch die Feature Prädikate werden die obigen Vererbungsregeln nur dann gültig, wenn das Objekt ein entsprechendes Attribut aufweist und zu einer Organisationseinheit der Autorisierungssphären Hardware oder einer zu Hardware untergeordneten Autorisierungssphären gehört. Für andere Objekte, die dieses Attribut mit dem entsprechenden Wert nicht haben und für solche, die Organisationseinheiten in anderen Autorisierungssphären zugeordnet sind, gilt die obige Regel nicht und die lokalen Zugriffsstrategien von Entwicklung werden somit nicht vererbt.
Wenn in den lokalen Zugriffsstrategien von Entwicklung eine Erlaubnis oder ein Verbot für einen Lesezugriff spezifiziert ist, ist die erste bzw. zweite Regel gültig.

Beim obigen Beispiel werden die lokalen Zugriffsstrategien einer Autorisierungssphäre ausgewertet. Im folgenden Beispiel wird ein Leseverbot direkt durch die Vererbungsregeln spezifiziert, ohne dass lokale Zugriffsstrategien ausgewertet werden. Dadurch kann der Interpreter schneller zu einem Ergebnis kommen.
Zur Bewahrung von Firmengeheimnissen wird von der Autorisierungssphäre Geschäftsleitung festlegt, dass allen Subjekten in einer beliebigen Rolle unterhalb von Senior Project-Manager der Lesezugriff auf ein Objekt, dessen Attribut Classification den Wert Secret hat, verboten wird.

**Beispielregeln 6.2**

$$\begin{aligned}
\mathsf{bequeath}(Geschaeftsleitung, s, app, o, -read) \leftarrow\ & \mathsf{activ_role}(s, r, u)\& \\
& \mathsf{subrole}(r, SeniorProject-Manager)\& \\
& \mathsf{attribute}(o, Classification, =, Secret)
\end{aligned}$$

[8] Lediglich die indirekten und erweiterten Feature Prädikate wie subunit(), subsphere(), subrole() und subtype() und bei den Kontext Feature Prädikaten u.a. sublocation(), subos() und subsecure() sind keine EPred, da sie rekursiv über feste Regeln definiert sind.

Da in den Beispielregeln keine Feature Prädikate auftauchen, die die Organisationseinheit $u$ einschränken ebenso wenig die entsprechenden Autorisierungssphären der Rolle und des Objektes $o$, wird das entsprechende Verbot in allen untergeordneten Autorisierungssphären durchgesetzt. Da keine lokalen Zugriffsstrategien in Form von do Regeln ausgewertet werden müssen, kommt der Interpreter schneller zu einem Ergebnis.

Wenn die lokalen Zugriffsstrategien, die in der Autorisierungssphäre Software definiert sind, auch uneingeschränkt in allen untergeordneten Autorisierungssphären gelten sollen, reicht es aus, die beiden folgenden Regeln aufzustellen.

**Beispielregeln 6.3**

$$\mathsf{bequeath}(Software, s, app, o, +a) \leftarrow \mathsf{do}(Software, s, app, o, +a)$$
$$\mathsf{bequeath}(Software, s, app, o, -a) \leftarrow \mathsf{do}(Software, s, app, o, -a)$$

Diese Vererbungsregel würde in allen (untergeordneten) Autorisierungssphären ausgewertet, da keine Feature Prädikate enthalten sind, die irgendwelche Einschränkungen vornehmen. Nur wenn die lokalen Zugriffsstrategien von Software keine Erlaubnis und kein Verbot auf die konkrete Zugriffsanfrage liefern, würden gemäß dem Vererbungsprinzip die Vererbungsregeln oder lokalen Zugriffsstrategien der untergeordneten Autorisierungssphären ausgewertet werden.
Die Variable $a \in \mathcal{V}_{\mathcal{AC}}$ bewirkt, dass jede Zugriffsmethode vererbt wird. Falls die Vererbung nur für Lese- und Schreibzugriffe erfolgen soll, müssen beiden Regeln mit der entsprechenden Belegung explizit aufgeführt werden[9].

Wenn auf die Beispielregel 6.3 verzichtet wird, kann eine schrittweise Vererbung der Zugriffsstrategien von Software an Netzwerk und von Netzwerk an VPN durch die folgenden Regeln erreicht werden.

**Beispielregeln 6.4**

$$\begin{aligned}
\mathsf{bequeath}(\mathsf{Software}, s, app, o, +a) \leftarrow & \mathsf{activ_role}(s, u, r) \& \mathsf{unitof}(o, u) \& \mathsf{sphereof}(u, \mathsf{Netzwerk}) \& \\
& \mathsf{do}(\mathsf{Software}, s, app, o, +a) \\
\mathsf{bequeath}(\mathsf{Software}, s, app, o, -a) \leftarrow & \mathsf{activ_role}(s, u, r) \& \mathsf{unitof}(o, u) \& \mathsf{sphereof}(u, \mathsf{Netzwerk}) \& \\
& \mathsf{do}(\mathsf{Software}, s, app, o, -a) \\
\mathsf{bequeath}(\mathsf{Netzwerk}, s, app, o, +a) \leftarrow & \mathsf{activ_role}(s, u, r) \& \mathsf{unitof}(o, u) \& \mathsf{sphereof}(u, \mathsf{VPN}) \& \\
& \mathsf{do}(\mathsf{Software}, s, app, o, +a) \\
\mathsf{bequeath}(\mathsf{Netzwerk}, s, app, o, -a) \leftarrow & \mathsf{activ_role}(s, u, r) \& \mathsf{unitof}(o, u) \& \mathsf{sphereof}(u, \mathsf{VPN}) \& \\
& \mathsf{do}(\mathsf{Software}, s, app, o, -a)
\end{aligned}$$

Die ersten beiden Regeln werden von dem Autorisierungsteam der Autorisierungssphäre Software spezifiziert. Aufgrund der Feature Prädikate in dieser Regel werden die lokalen Zugriffsstrategien von Software nur an Netzwerk vererbt, wenn das Objekt $o$ auf dem sich die Anfrage bezieht aus einer Organisationseinheit der Autorisierungssphäre Netzwerk stammt und eine Rolle des Subjektes dort aktiviert ist.
Die dritte und vierte Beispielregel wird in der Autorisierungssphäre Netzwerk spezifiziert und legt durch die Feature Prädikate fest, dass die lokalen Zugriffsstrategien von Software auch bei

[9] Alternative zu Beispielregel 6.3 bei der nur Lese und Schreibreichte vererbt werden.

$$\mathsf{bequeath}(Software, s, app, o, +read) \leftarrow \mathsf{do}(Software, s, app, o, +read)$$
$$\mathsf{bequeath}(Software, s, app, o, -read) \leftarrow \mathsf{do}(Software, s, app, o, -read)$$
$$\mathsf{bequeath}(Software, s, app, o, +write) \leftarrow \mathsf{do}(Software, s, app, o, +write)$$
$$\mathsf{bequeath}(Software, s, app, o, -write) \leftarrow \mathsf{do}(Software, s, app, o, -write)$$

Zugriffsanfragen gelten sollen, bei denen die aktive Rolle und das Objekt in einer Organisationseinheit der Autorisierungssphäre VPN sind.
Obwohl die vererbten Zugriffsregeln eigentlich von Software stammen, kann dieses Vorgehen auch zu einem Vererbungskonflikt in VPN führen, da die Vererbungsregeln, die in VPN gültig sind, nicht von Software spezifiziert sind, sondern von Netzwerk.

Alternativ kann die Autorisierungssphäre Software die Vererbung auch auf bestimmte Rollen wie Developer, Programmer und Tester sowie auf Objekte bestimmten Typs, die gemäß der Hierarchie der Objekttypen aus Abbildung 3.6 zu einem Software_Project gehören, festlegen. Für diese Subjekte und Objekte greifen die folgenden Regeln und vererben die Zugriffsstrategien der Autorisierungssphäre Software.

**Beispielregeln 6.5**

$$\begin{aligned}\mathsf{bequeath}(Software, s, app, o, +a) \leftarrow\ & \mathsf{activ\_role}(s, u, r)\&\mathsf{unitof}(o, u)\& \\ & \mathsf{subroleeq}(r, Developer)\&\mathsf{subtypeof}(o, Software\_Project)\& \\ & \mathsf{do}(Software, s, app, o, +a) \\ \mathsf{bequeath}(Software, s, app, o, -a) \leftarrow\ & \mathsf{activ\_role}(s, u, r)\&\mathsf{unitof}(o, u)\& \\ & \mathsf{subroleeq}(r, Developer)\&\mathsf{subtypeof}(o, Software\_Project)\& \\ & \mathsf{do}(Software, s, app, o, -a)\end{aligned}$$

Diese Beispielregeln sind wieder in allen untergeordneten Autorisierungssphären gültig. Durch das Hinzufügen von weiteren Feature Prädikaten wie $\mathsf{sphereof}(u, Netzwerk)$ kann dies auch eingeschränkt werden.
Welche Vorgehensweise die Richtige ist, hängt ausschließlich von den konkreten Gegebenheiten einer Organisation ab.

### 6.2.2 Inkonsistenzen bei Vererbungsregeln

Ähnlich wie bei den lokalen Zugriffsstrategien kann es auch bei den Vererbungsregeln aufgrund der Definition von hybriden Zugriffsstrategien zu Inkonsistenz kommen. Daher muss für die Vererbungsregeln gelten:

**Vereinbarung 6.1**
In einer Autorisierungssphäre $as$ ist für ein Subjekt $s$, das durch eine Applikation $app$ mit der Zugriffsmethode $a$ auf ein Objekt $o$ zugreifen will, nie sowohl das Prädikat $\mathsf{bequeath}(as, s, app, o, +a)$ und $\mathsf{bequeath}(as, s, app, o, -a)$ gültig.

Analog zu der Vereinbarung 5.1 und der daraus resultierenden lokalen Inkonsistenzenregel (Def. 7.7) wird die folgende Regel definiert, um Inkonsistenzen innerhalb der Vererbungsregeln zu erkennen und eine entsprechende Auswertung mit einem Fehler abbrechen zu können.

**Definition 6.3 (Inkonsistenzregeln Vererbung)**
Die Inkonsistenzregel zur Vererbung hat die folgende Form

$$\begin{aligned}\textbf{inconsistent-bequeath}(as, s, app, o, a) \leftarrow\ & \mathsf{bequeath}(as, s, app, o, +a)\& \\ & \mathsf{bequeath}(as, s, app, o, -a)\end{aligned}$$

wobei $as \in \mathcal{AS}$ die Autorisierungssphäre bezeichnet, in der diese Regel ausgewertet wird. Falls die Regel mit einer entsprechenden Belegung für $s \in \mathcal{V}_{\mathcal{S}}$, $app \in \mathcal{V}_{\mathcal{AP}}$ und $o \in \mathcal{V}_{\mathcal{O}}$ mit Methode $a \in \mathcal{V}_{\mathcal{A}}$ gültig ist, wird die Zugriffsanfrage mit einem Fehler abgebrochen.

Der Datalog–Interpreter für die Auswertung der Zugriffsanfragen müsste dann so realisiert werden, dass er bei jeder ersten Auswertung eines Prädikates $\mathsf{bequeath}(as, s, app, o, +a)$ oder $\mathsf{bequeath}(as, s, app, o, -a)$ automatisch auch $\mathsf{inconsistent\text{-}bequeath}(as, s, app, o, a)$ auswertet und

mit einer Fehlermeldung abbricht, wenn dieses Prädikat gültig ist. Alternativ könnte ein Aufruf dieses Prädikates in jede Regel hinzugefügt werden, die auch ein bequeath enthält. Da dieses Prädikat nur in Regeln[10] auftaucht, die automatisch spezifiziert werden, wäre dies eine weitere Möglichkeit, die bei der Realisierung verfolgt werden kann.

## 6.3 Konflikte bei der Vererbung

Im vorangegangenen Abschnitt wurde definiert, wie eine Autorisierungssphäre gezielt festlegen kann für welche Subjekte, die auf bestimmte Objekte mittels einer Applikation mit einer Methode zugreifen wollen, die eigenen Zugriffsstrategien vererbt werden. Dabei sind alle drei Parameter Subjekt, Applikation und insbesondere Objekt nicht fix, sondern können mittels Feature Prädikaten sowie Kontext Feature Prädikaten und Kontext Trust Prädikaten spezifiziert werden. Bei Subjekten bezieht sich dies auf die aktive Rolle, die in einem bestimmten Bereich der Rollenhierarchie liegt (vgl. Abschnitt 3.4). Wohingegen bei Objekten sowohl dessen Objekttyp in einem bestimmten Bereich der Hierarchie der Objekttypen liegt, als auch die Attribute des Objektes einen bestimmten Wert haben. Darüber hinaus können bei Objekten weitere Eigenschaften wie Signatur, Besitzer oder Verzeichnis (siehe Abschnitt 3.5) Ausschlag gebend sein. Dabei kann bei Objekten auch eine Kombination von Eigenschaften geändert werden. Durch die Kontext Feature Prädikate kann festgelegt werden, dass die Applikation eine bestimmte Vertrauenswürdigkeit haben muss (siehe Abschnitt 4.4). Darüber hinaus können mittels Kontext Trust Prädikaten, die Zugriffsprivilegien in einer verteilten Umgebung an bestimmte Sicherheitsstandards gekoppelt werden, wie es im Kapitel 4 beschrieben ist.

Solange die Halbordnung über die Autorisierungssphären ($<_{\mathcal{AS}}$), entlang derer die Vererbung erfolgt, eine Baumstruktur aufweist, wird der höheren Autorisierungssphäre immer der Vorzug gegeben. Sobald eine Autorisierungssphäre mehr als einen direkten Vorgänger hat, kann es bei hybriden Zugriffsstrategien Vererbung zu sogenannten Konflikten kommen. Da diese Konstruktion der Halbordnung jedoch aus organisatorischen Gründen sinnvoll ist, wie im Abschnitt 3.3 gezeigt, müssen diese Vererbungskonflikte detailliert betrachtet werden.

### 6.3.1 Auftreten und Erkennen von Vererbungskonflikten

In Abbildung 6.1 wird eine mögliche Hierarchie der Autorisierungssphären skizziert, dabei sollen die Autorisierungssphären $as_{h1}$ und $as_{h2}$ ein explizites Verbot und eine explizite Erlaubnis definieren und festlegen, dass diese an alle untergeordneten Autorisierungssphären vererbt werden. Diese beiden Autorisierungssphären liegen auf getrennten Pfaden und sind damit gemäß der Halbordnung $<_{\mathcal{AS}}$ nicht vergleichbar. Wenn sich diese Pfade in einer Autorisierungssphäre $as$ wieder vereinen, kann es hier und in allen Autorisierungssphären $as''$, die unterhalb von $as$ liegen, zum Konflikt kommen.

In dieser Abbildung[11] deutet eine ausgefüllte Pfeilspitze mit durchgehender Linie an, dass es auf jeden Fall eine untergeordnete Autorisierungssphäre gibt, eine nicht ausgefüllte Pfeilspitze mit einer gestrichelten Linie deutet an, dass beliebig viele weitere Autorisierungssphären dort liegen können aber auch keine weitere.
Die Schreibweise $\mathsf{AS_{h1}}$ und AS in der Abbildung soll deutlich machen, dass es sich um konkrete Autorisierungssphären handelt. Bei der folgenden Definition werden allerdings Variablen über Autorisierungssphären verwendet.

**Definition 6.4 (Vererbungskonflikt)**
In einer Autorisierungssphäre $as$ liegt genau dann ein Vererbungskonflikt vor, wenn in zwei gemäß der Halbordnung $<_{\mathcal{AS}}$ übergeordneten Autorisierungssphären $as_{h1}$ und $as_{h2}$ jeweils Vererbungsregeln definiert sind, die für das gleiche Subjekt $s$, das auf ein Objekt $o$ über eine Applikation $app$ mit der

[10] Das Prädikat bequeath() kann gemäß Definition 6.5 nur in inherit() Regeln verwendet werden. Siehe auch Abbildung 8.1.
[11] sowie in den Abbildungen 7.1 bis 7.4 im Kapitel 7.1

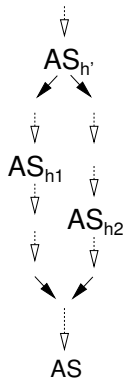



Abbildung 6.1: Auftreten eines Vererbungskonfliktes

Methode $a$ zugreifen will, sowohl ein Verbot und eine Erlaubnis geben ist. Dabei stehen die beiden Autorisierungssphären $as_{h1}$ und $as_{h2}$ gemäß der Halbordnung $<_{\mathcal{AS}}$ nicht zueinander in Beziehung. Es gilt also $as <_{\mathcal{AS}} as_{h1}$, $as <_{\mathcal{AS}} as_{h2}$, $as_{h1} \not<_{\mathcal{AS}} as_{h2}$, $as_{h2} \not<_{\mathcal{AS}} as_{h1}$. In einer gemeinsamen übergeordneten Autorisierungssphäre $as'_h$ mit $as <_{\mathcal{AS}} as_{h1} <_{\mathcal{AS}} as_{h'}, as <_{\mathcal{AS}} as_{h2} <_{\mathcal{AS}} as_{h'}$ wird für diese Zugriffsanfrage keine passende Vererbungsregel spezifiziert.

Wie in der Abbildung 6.1 angedeutet, kann die Autorisierungssphäre $as$, in der ein Vererbungskonflikt auftritt, gemäß $<_{\mathcal{AS}}$ die oberste Autorisierungssphäre sein, die sowohl unterhalb von $as_{h1}$ und $as_{h2}$ liegt, als auch beliebig tiefer.

Eine sofortige Erkennung eines Vererbungskonfliktes bei der Spezifizierung einer neuen Regel oder der Einfügung einer neuen Kante in der Autorisierungshierarchie ist schwierig, da ab der Autorisierungssphäre $as$, in der ein Vererbungskonflikt auftreten kann, alle möglichen Kombinationen von Subjekt, Applikation und Objekt, die in dieser Autorisierungssphäre verwendet werden, getestet werden müssten. Dies ist für die zu $as$ untergeordneten Autorisierungssphären $as'$ mit den dort verwendeten Kombinationen für $s, app, o$ fortzusetzen.

Daher ist es sinnvoll, nur beim Auswerten einer Zugriffsanfrage zu prüfen, ob ein Vererbungskonflikt für den konkreten Fall von Subjekt, Applikation und Objekt vorliegt. Dieses Vorgehen wird bei der Umsetzung des Vererbungsprinzips im folgenden Abschnitt 6.4 realisiert. Im Falle eines Vererbungskonfliktes kann diese Umsetzung nicht funktionieren. In diesem Fall wird die Anfrage mit einem Fehler abgebrochen und die Autorisierungsteams der beteiligten Autorisierungssphären müssen eine Lösung finden.

### 6.3.2 Lösungsstrategien für Vererbungskonflikte

In Abbildung 6.1 wurde skizziert, dass sich die beiden Autorisierungssphären $as_{h1}$ und $as_{h2}$, die den Vererbungskonflikt auslösen, nicht durch die Relation $<_{\mathcal{AS}}$ vergleichen lassen. Eine Möglichkeit für eine automatische Auflösung der Vererbungskonflikte wäre es, die Autorisierungssphären indirekt zu vergleichen. Der Abstand, d.h. die Anzahl der Autorisierungssphären, die zwischen $as$ und $as_{h1}$ bzw. $as$ und $as_{h2}$ liegen, könnte bei der Suche nach einer Konfliktlösung eine Rolle spielen. Alternativ kann auch der Abstand zur ersten gemeinsamen übergeordneten Autorisierungssphäre

$as_{h'}$ herangezogen werden. Dies wäre mittels Feature Prädikaten theoretisch abfragbar[12], jedoch könnte die Zahl der Autorisierungssphären nicht unbedingt mit der Zahl der Organisationseinheiten, die dazwischen liegen identisch sein und müssten dann unter Umständen durch weitere Feature Prädikate[13] auch miteinbezogen werden.

Diese automatische Auflösung des Vererbungskonfliktes anhand des Kriteriums "Abstand" erscheint aber nicht (immer) wünschenswert, wenn davon ausgegangen wird, dass die Autorisierungsteams die Regeln bewusst so spezifiziert haben. An Stelle dessen sollte, wie am Ende von Abschnitt 6.3.1 formuliert, die Auswertung abgebrochen werden und eine Fehlermeldung an die Autorisierungsteams aller betroffenen Autorisierungssphären $as_{h1}$ und $as_{h2}$ sowie an die Autorisierungssphäre $as$ in der der Vererbungskonflikt unmittelbar festgestellt wurde, gegeben werden. Zwischen den Teams dieser Autorisierungssphäre muss dann eine mögliche Lösungsstrategie entwickelt werden. Diese Lösung resultiert in neuen oder modifizierten Zugriffsregeln, die dann wieder an alle untergeordneten Autorisierungssphären repliziert werden.

Die möglichen Lösungsansätze lassen sich wie folgt skizzieren: Das Vererbungsprinzip (Def. 6.1) sieht vor, dass jede Autorisierungssphäre festlegen kann, für welche Subjekte, Applikationen und Objekte Regeln vererbt werden sollen und an welche Autorisierungssphäre diese Vererbung erfolgt. Zudem kann in der Autorisierungssphäre, in der ein Vererbungskonflikt auftritt, versucht werden diesen zu vermeiden, in dem dort andere Rollen oder Objekttypen verwendet werden. Letztlich könnte eine übergeordnete Autorisierungssphäre, die in Abbildung 6.1 als $as_{h'}$ bezeichnet wird, eine Regel für das entsprechende Tripel aus Subjekt, Applikation und Objekt vererben, sodass in den beiden untergeordneten Autorisierungssphären $as_{h1}$ oder $as_{h2}$ die Vererbungsregeln ignoriert werden. Im Detail können diese Ansätze wie folgt umgesetzt werden.

### Änderung des Scopes

Durch die Vererbungsregel bequeath() kann mittels der Feature Prädikate genau festgelegt werden, für welche Subjekte, in bestimmten aktiven Rollen in einer Organisationseinheit, und für welche Objekte, die einen bestimmten Typ oder eine andere Eigenschaft aufweisen, die Regeln vererbt werden sollen. Darüber hinaus können durch Kontext Feature Prädikate die Applikationen beschrieben werden und mittels Kontext Trust Level Prädikaten die Eigenschaften der verteilten Umgebung bestimmt werden (vgl. Def. 6.2 auf Seite 78).

Bei den Beispielregeln 6.5 in Abschnitt 6.2 wurden zwei Vererbungsregeln aufgestellt, sodass alle von der Autorisierungssphäre Software aufgestellten Regeln bezüglich Objekte, deren Objekttyp gemäß der Hierarchie in Abbildung 3.6 unterhalb von Software_Project liegen oder für Subjekte mit der aktiven Rolle Developer, Programmer oder Tester vererbt werden. Wenn dies dann in der Autorisierungssphäre TPM zu Vererbungskonflikten führt, da die Regeln einen großen Bereich an Objekttypen betreffen, ist es ein Ansatz, diesen einzuschränken. Entsprechend werden mit den folgenden Beispielregeln als ein Lösungsansatz nur Regeln vererbt, die sich auf die Dokumentation beziehen, wobei Fehlermeldungen (Bug_Report) ausgenommen sind. Dazu würde in den Beispielregeln 6.5 das Prädikat subtype$(o, Software_Project)$ durch die Prädikate subtypeof$(o, Documentation)$& $\neg$typeof$(o, Bug_Report)$ ersetzt.

[12] Es lässt sich ein Feature Prädikat largerdistanc$(as, as_1, as_2)$, über die folgenden beiden Regeln definieren, das dann gültig ist, wenn zwischen $as$ und $as_1$ mehr Autorisierungssphären liegen, als zwischen $as$ und $as_2$.

$$\text{largerdistance}(as, as_1, as_2) \leftarrow \text{spherein}(as, as_X)\&\text{subsphere}(as_X, as_1)\&\text{spherein}(as, as_2)$$

$$\text{largerdistance}(as, as_1, as_2) \leftarrow \text{spherein}(as_1, as'_1)\&\text{subsphere}(as_2, as'_2)\&\text{largerdistance}(as, as'_1, as'_2)$$

Dieses Prädikat stellt fest, welche übergeordnete Autorisierungssphäre $as_1$ und $as_2$ "weiter entfernt" ist. Für untergeordnete Autorisierungssphären müssten lediglich die Reihenfolge der Parameter bei spherein und subsphere vertauscht werden.

[13] Analog zu largerdistance kann auch ein Feature Prädikat definiert werden, das mit Hilfe von unitin() und subunit() den Abstand zwischen den Organisationseinheiten feststellt.

**Beispielregeln 6.6**

$$\begin{aligned}
\mathsf{bequeath}(Software, s, app, o, +a) \leftarrow & \mathsf{activ_role}(s, u, r) \& \mathsf{unitof}(o, u) \& \mathsf{subroleeq}(r, Developer) \& \\
& \mathsf{subtypeof}(o, Documentation) \& \neg \mathsf{typeof}(o, Bug_Report) \& \\
& \mathsf{do}(Software, s, app, o, +a) \\
\mathsf{bequeath}(Software, s, app, o, -a) \leftarrow & \mathsf{activ_role}(s, u, r) \& \mathsf{unitof}(o, u) \& \mathsf{subroleeq}(r, Developer) \& \\
& \mathsf{subtypeof}(o, Documentation) \& \neg \mathsf{typeof}(o, Bug_Report) \& \\
& \mathsf{do}(Software, s, app, o, -a)
\end{aligned}$$

Die Regeln für Objekte mit diesem Typ müssen auch nicht für alle Subjekte, deren aktive Rolle unterhalb von Developer einschließlich liegt, vererbt werden. So könnten die Zugriffsstrategien für die Objekte vom Typ Evaluation_Report nur an Subjekte in der aktiven Rolle Tester vererbt werden. Die Zugriffsstrategien für Objekte vom Typ UML_File kann auch nur an Developer und Programmer weitergegeben werden. Bei Letzteren könnte dann auch noch eingeschränkt werden, dass nur die Leserechte vererbt werden. Dies alles wird durch die folgenden Regeln spezifiziert.

**Beispielregeln 6.7**

$$\begin{aligned}
\mathsf{bequeath}(Software, s, app, o, +a) \leftarrow & \mathsf{activ_role}(s, u, Tester) \& \mathsf{unitof}(o, u) \& \\
& \mathsf{typeof}(o, Evaluation_Report) \& \\
& \mathsf{do}(Software, s, app, o, +a) \\
\mathsf{bequeath}(Software, s, app, o, -a) \leftarrow & \mathsf{activ_role}(s, u, Tester) \& \mathsf{unitof}(o, u) \& \\
& \mathsf{typeof}(o, Evaluation_Report) \& \\
& \mathsf{do}(Software, s, app, o, -a) \\
\mathsf{bequeath}(Software, s, app, o, +read) \leftarrow & \mathsf{activ_role}(s, u, r) \& \mathsf{unitof}(o, u) \& \\
& \mathsf{subroleeq}(r, Developer) \& \mathsf{subroleeq}(Programmer, r) \& \\
& \mathsf{typeof}(o, UML_File) \& \\
& \mathsf{do}(Software, s, app, o, +read) \\
\mathsf{bequeath}(Software, s, app, o, -read) \leftarrow & \mathsf{activ_role}(s, u, r) \& \mathsf{unitof}(o, u) \& \\
& \mathsf{subroleeq}(r, Developer) \& \mathsf{subroleeq}(Programmer, r) \& \\
& \mathsf{typeof}(o, UML_File) \& \\
& \mathsf{do}(Software, s, app, o, -read)
\end{aligned}$$

Mit dieser Lösungsstrategie würden sich zwar die Zahl der bequeath() Regeln erhöhen, allerdings wäre der Scope, d.h. der Bereich von Subjekten und Objekten, für die Regeln vererbt werden, wesentlich detaillierter definiert.

Wenn jedoch die Rollen in den Organisationseinheiten konkret vorgegeben sind, ebenso die Objekteigenschaften nicht weiter eingeschränkt werden sollen und die verteilte Umgebung der Applikationen so zutreffend ist, muss geprüft werden, ob dies auch für die untergeordnete Autorisierungssphäre zutreffend ist.

### Vererbungstiefe einschränken

Wenn die Zugriffsstrategien aus organisatorischen Gründen[14] nur an bestimmte Autorisierungssphären vererbt werden sollen, kann dies in den bequeath() Regeln mit Hilfe der in Abschnitt 3.3 definierten Feature Prädikate modelliert werden, in dem mit dem Feature Prädikat sphereof() die verantwortliche Autorisierungssphäre für das Objekte festgestellt wird[15]. Diese Autorisierungssphäre kann dann auch mit der vererbenden Autorisierungssphäre in Bezug gesetzt werden. Die

[14] wie zu Beginn des Kapitels 6 dargestellt

[15] Falls möglicherweise abweichend, kann ebenso die Autorisierungssphäre festgestellt werden, in der die aktive Rolle des Subjektes liegt.

dadurch formulierten Bedingungen legen fest, an welche Autorisierungssphären Zugriffsstrategien vererbt werden sollen. Dabei gibt es eine Reihe von Möglichkeiten, wie dies geschehen kann.

Das genaue Gegenteil zu den bisherigen Beispielen, in denen die Zugriffsstrategien an alle untergeordneten Autorisierungssphären vererbt werden, ist die Vererbung an eine einzige Autorisierungssphäre. Durch die folgende Beispielregel vererbt Software alle Zugriffsstrategien ausschließlich[16] an die Autorisierungssphäre Netzwerk. Nur wenn die Organisationseinheit $u$, in der das Subjekt $s$ eine (beliebige) Rolle aktiviert hat und zu der auch das Objekt, auf das zugegriffen werden soll, gehört, der Autorisierungssphäre Netzwerk zugeordnet ist, werden die entsprechenden lokalen Regeln von Software ausgewertet. Dadurch werden, wie im Abschnitt 6.2 beschrieben, die Zugriffsstrategien vererbt.

**Beispielregel 6.8**

$$\mathsf{bequeath}(Software, s, app, o, +a) \leftarrow \mathsf{activ\_role}(s, u, r) \& \mathsf{unitof}(o, u) \& \mathsf{sphereof}(u, Netzwerk) \& \mathsf{do}(Software, s, app, o, +a)$$

Außerdem kann die Vererbung auch ab einer bestimmten Autorisierungssphäre in der Hierarchie abgebrochen werden, indem wie in der folgenden Regel dargestellt, eine negative Bedingung eingeführt wird. Falls die Vererbung an mehren Stellen unterbunden werden soll, können hier auch weitere negierte Prädikate eingefügt werden.

**Beispielregel 6.9**

$$\mathsf{bequeath}(Entwicklung, s, app, o, +a) \leftarrow \mathsf{activ\_role}(s, u, r) \& \mathsf{unitof}(o, u) \& \mathsf{sphereof}(u, as) \& \neg\mathsf{subsphere}(as, Crypto) \& \mathsf{do}(Entwicklung, s, app, o, +a)$$

Diese Vorgehensweisen lassen sich auch kombinieren. So könnten die lokalen Regeln an bestimmte Zweige der nachfolgenden Autorisierungssphären vererbt werden, jedoch nicht an eine bestimmte Autorisierungssphäre, um so die Vererbungskonflikte zu vermeiden.

### Verwendete Rollen, Applikationen und Objekteigenschaften

Da Vererbungskonflikte dann auftreten, wenn für eine Belegung von Subjekt, Applikation und Objekt zwei gegensätzliche Regeln für dieselbe Zugriffsmethode definiert wurden, wäre es theoretisch eine weitere Möglichkeit, in der untergeordneten Autorisierungssphäre andere Rollen und Objekte mit anderen Eigenschaften zu verwenden, als solche, die durch die geerbten Regeln abgedeckt sind. Dies wäre dann weniger eine Lösungs- sondern eine Vermeidungsstrategie.

Dabei ist aus organisatorischer Sicht zu prüfen, ob eine neue Rolle sinnvoll ist und in die Rollenhierarchie eingefügt werden sollte. Gleiches gilt auch für einen neuen Typ für Objekte. Bei beiden Hierarchien müssen die neuen Elemente in Beziehung zu bereits existierenden Elementen stehen (vgl. Abschnitte 3.4 und 3.5). Wenn die Vererbungsregeln sich jedoch durch Feature Prädikate wie subrole() bzw. subtype() auf eine ganze Menge von Rollen bzw. Objekttypen beziehen, kann es gut sein, dass die neuen Elemente ebenfalls durch diese Regeln abgedeckt werden. Analog können mit dem Kontext Feature Prädikate application_trusted() andere Applikationen verwendet werden (vgl. Abschnitt 4.4.1). Im Kapitel 9 wird gezeigt, wie gezielt festgelegt werden kann, wer an welchen Stellen in den verschiedenen Hierarchien neue Elemente einfügen kann.

### Übergeordnete Autorisierungssphäre

Wenn sich die betroffenen Autorisierungssphären nicht auf eine Lösung des Vererbungskonfliktes einigen können, besteht noch die Möglichkeit, dass die übergeordnete Autorisierungssphäre eine vorgibt. In der Skizze in Abbildung 6.1 wäre dies die Autorisierungssphäre $\mathsf{AS}_{\mathsf{h}'}$, die eine entsprechende bequeath() Regel formulieren könnte, die dann zunächst nicht nur für die Autorisierungssphäre AS bindend wäre, sondern auch für $\mathsf{AS}_{\mathsf{h1}}$ und $\mathsf{AS}_{\mathsf{h2}}$. Dies hätte zur Folge, dass die

[16] Sofern es keine weiteren bequeath() Regeln gibt.

Vererbungsregeln der beiden untergeordneten Autorisierungssphären, die im Vererbungskonflikt standen nun gemäß Lemma 6.5 bzw. 6.6 nicht beachtet werden.

Diese Vererbungsregeln können von der übergeordneten Autorisierungssphäre $\mathsf{AS}_{\mathsf{h}'}$ nach dem folgenden Grundschema so spezifiziert werden, dass sie gezielt für die Autorisierungssphäre gültig sind, in der der Vererbungskonflikt auftritt. Eine Einbeziehung der Vererbungsregeln der untergeordneten Autorisierungssphäre $\mathsf{AS}_{\mathsf{h1}}$ oder $\mathsf{AS}_{\mathsf{h2}}$ in die Vererbungsregeln von $\mathsf{AS}_{\mathsf{h}'}$ ist nicht vorgesehen, dies würde den Weg auf dem Zugriffsregeln vererbt werden wesentlich komplexer und unübersichtlicher machen, da die Zugriffsregeln dann sowohl von übergeordneten als auch von untergeordneten Autorisierungssphären stammen können. Die Umsetzung des Vererbungsprinzips als ein geschichtetes Datalog–Programm wäre noch komplexer[17]. Außerdem lassen sich, wie sich im nächsten Abschnitt 6.4 zeigen wird, die Autorisierungssphären von denen ein Vererbungskonflikt ausgeht mittels Datalog nicht direkt identifizieren.

Daher kann von der Autorisierungssphäre $as_{h'}$ nur die Lösung direkt an die Autorisierungssphäre $as$ vererbt werden, in der der Vererbungskonflikt aufgetreten ist.

**Beispielregeln 6.10**

$$\mathsf{bequeath}(AS_{h'}, s, app, o, \pm a) \leftarrow \mathsf{unitof}(o, u) \& \mathsf{sphereof}(u, as) \& \mathsf{subsphereeq}(as, AS)$$

Hierbei muss $\pm a$ durch die konkrete Zugriffsmethode ersetzt werden und mit dem entsprechenden Vorzeichen, je nachdem ob ein entsprechendes Verbot oder Erlaubnis vererbt werden soll. Außerdem sollte mittels weiterer Feature Prädikate diese Regel auf die Rollen und Objekte beschränkt werden, bei denen der Vererbungskonflikt gelöst werden soll. Wenn diese Lösung des Vererbungskonfliktes nur auf die Autorisierungssphäre $as$ beschränkt sein soll und nicht auch in allen untergeordneten Autorisierungssphären $as'$ (vgl. Abbildung 6.1), muss $\mathsf{sphereof}(u, as) \& \mathsf{subsphereeq}(as, AS)$ durch $\mathsf{sphereof}(u, AS)$ ersetzt werden.

## 6.4 Umsetzung des Vererbungsprinzips

In den vorangegangenen Abschnitten wurde das Vererbungsprinzip konzeptionell vorgestellt und dessen Möglichkeiten aufgezeigt. Einige Probleme und Einschränkungen, wie die Einbeziehung Vererbungsregeln anderer Autorisierungssphären, treten erst dann auf, wenn eine Umsetzung des Vererbungsprinzips genauer betrachtet wird. An Stelle einer Referenzimplementierung des Vererbungsprinzips in einer Programmiersprache wie Java soll dies, wie auch die gesamte Auswertung einer Zugriffsanfrage, mittels Datalog–Prädikate und fest vorgegebenen Regeln erfolgen. Hierdurch kann die korrekte Umsetzung des Vererbungsprinzips — unter der notwendigen Einschränkung, dass kein Vererbungskonflikt vorliegt — im Abschnitt 6.4.3 nachgewiesen werden. Wie diese Vererbungskonflikte mittels eines Datalog–Prädikates entdeckt werden können, wird im Abschnitt 6.4.2 gezeigt. Zunächst wird im folgenden Abschnitt ein Prädikat definiert, mit dem die Erbregeln spezifiziert werden, die das Vererbungsprinzip umsetzen.

### 6.4.1 Erbregeln

Nach den obigen Beispielen für mögliche Vererbungsregeln muss nun durch eine entsprechende Spezifizierung der Erbregeln sichergestellt werden, dass diese in den untergeordneten Autorisierungssphären immer berücksichtigt werden. Dazu werden in jeder Autorisierungssphäre Erbregeln aufgerufen, deren Inhalt von dem Autorisierungsteam nicht verändert werden kann, sondern abhängig, ob eine oder zwei übergeordnete Autorisierungssphären vorhanden sind, gemäß vordefinierter Regelsätze in das Regelwerk der Autorisierungssphäre eingefügt werden. Dies erfolgt

[17] Das Prädikat bequeath() muss in diesem Fall ähnlich wie inherit() im Abschnitt 8.2 auch umgeformt werden. Um dann dynamisch, d.h. mittels Variablen auf Vererbungsregeln einer beliebigen Autorisierungssphäre zugreifen zu können, müsste ein weiteres Prädikat eingeführt werden, das ähnlich wie local() (vgl. Definition 7.5 auf Seite 108 und dessen Verwendung in den Beispielregeln 7.6 auf Seite 109) definiert ist.

beim Anlegen einer neuen Autorisierungssphäre oder bei einer Änderung der Beziehung zwischen Autorisierungssphären in diesem Sinne automatisch.

Die so spezifizierten Erbregeln stellen sicher, dass im Falle einer gültigen Vererbungsregel in einer übergeordneten Autorisierungssphäre, das Prädikat entsprechend gültig ist. Bei einer Zugriffsanfrage wird der erstmalige Aufruf dieses Prädikates auch durch Regeln gewährleistet, die weder von dem lokalen Autorisierungsteam in dessen Autorisierungssphäre vererbt werden soll, noch von übergeordneten Autorisierungsteams beeinflusst werden kann, da diese in gleicher Weise automatisch erzeugt werden (vgl. Regelsätze 7.4 und 7.5).

**Definition 6.5 (Erbregel)**
Die Erbregel hat die Form

$$\textbf{inherit(}as, s, app, o, \langle sign\rangle a\textbf{)} \leftarrow L_1 \& \ldots \& L_n$$

wobei $as \in \mathcal{AS}$ die Autorisierungssphäre bezeichnet, in der diese Regel automatisch spezifiziert wird. Falls alle Prädikate $L_i \in \{spherein, equal\_sphere, inherit, bequeath\}$ (mit festem $n \in \mathbb{N}$ und $1 \leq i < n$) mit einer entsprechenden Belegung für $s \in \mathcal{V}_{\mathcal{S}}$, $app \in \mathcal{V}_{\mathcal{AP}}$ und $o \in \mathcal{V}_{\mathcal{O}}$ gültig sind, wird der Zugriff mit Methode $a \in AC$ gemäß $\langle sign\rangle$ gewährt oder verboten. Alle Regeln vom Typ bequeath und inherit müssen aus einer Autorisierungssphäre $as'$ sein, die direkter Vorgänger von $as$ gemäß der Hierarchie $<_{\mathcal{AS}}$ ist $((as, as') \in <_{\mathcal{AS}})$.

Beim Anlegen einer Autorisierungssphäre sowie bei einer Änderung der direkten Vorgänger werden die folgenden Regelsätze automatisch durch das Autorisierungssystem erzeugt. Mit diesen Regelsätzen kann die Durchsetzung der Vererbungsregeln garantiert werden. Die entsprechenden Beweise werden im Abschnitt 6.4.3 vorgestellt.

Dabei wird unterschieden, ob die Autorisierungssphäre eine direkt übergeordnete Autorisierungssphäre oder zwei direkt übergeordnete Autorisierungssphären hat[18]. Im einfachen Fall bei einer übergeordneten Autorisierungssphäre werden die folgenden Regeln erzeugt:

**Regeln 6.11 (Erbregeln bei einer übergeordneten Autorisierungssphäre)**

$$\begin{aligned}
&\mathsf{inherit}(AS, s, app, o, +a) \leftarrow \mathsf{spherein}(AS, as_{h1}) \& \mathsf{inherit}(as_{h1}, s, app, o, +a)\\
&\mathsf{inherit}(AS, s, app, o, -a) \leftarrow \mathsf{spherein}(AS, as_{h1}) \& \mathsf{inherit}(as_{h1}, s, app, o, -a)\\
&\mathsf{inherit}(AS, s, app, o, +a) \leftarrow \mathsf{spherein}(AS, as_{h1}) \& \mathsf{bequeath}(as_{h1}, s, app, o, +a) \&\\
&\qquad\qquad \neg\mathsf{inherit}(as_{h1}, s, app, o, +a) \& \neg\mathsf{inherit}(as_{h1}, s, app, o, -a)\\
&\mathsf{inherit}(AS, s, app, o, -a) \leftarrow \mathsf{spherein}(AS, as_{h1}) \& \mathsf{bequeath}(as_{h1}, s, app, o, -a) \&\\
&\qquad\qquad \neg\mathsf{inherit}(as_{h1}, s, app, o, +a) \& \neg\mathsf{inherit}(as_{h1}, s, app, o, -a)
\end{aligned}$$

Falls eine weitere Autorisierungssphäre hinzukommen, müssen die vorangegangen Regeln durch die Folgenden ersetzt werden.

[18] Im Abschnitt 2.5 wurde gezeigt, dass maximal zwei übergeordnete Autorisierungssphären sinnvoll vorkommen können.

**Regeln 6.12 (Erbregeln bei zwei übergeordneten Autorisierungssphären)**

$$\begin{aligned}
\mathsf{inherit}(AS, s, app, o, +a) \leftarrow & \mathsf{spherein}(AS, as_{h1}) \& \mathsf{spherein}(AS, as_{h2}) \& \neg \mathsf{equal_sphere}(as_{h1}, as_{h2}) \& \\
& \neg \mathsf{bequeath}(as_{h2}, s, app, o, -a) \& \neg \mathsf{inherit}(as_{h2}, s, app, o, -a) \& \\
& \mathsf{inherit}(as_{h1}, s, app, o, +a) \\
\mathsf{inherit}(AS, s, app, o, -a) \leftarrow & \mathsf{spherein}(AS, as_{h1}) \& \mathsf{spherein}(AS, as_{h2}) \& \neg \mathsf{equal_sphere}(as_{h1}, as_{h2}) \& \\
& \neg \mathsf{bequeath}(as_{h2}, s, app, o, +a) \& \neg \mathsf{inherit}(as_{h2}, s, app, o, +a) \& \\
& \mathsf{inherit}(as_{h1}, s, app, o, -a) \\
\mathsf{inherit}(AS, s, app, o, +a) \leftarrow & \mathsf{spherein}(AS, as_{h1}) \& \mathsf{spherein}(AS, as_{h2}) \& \neg \mathsf{equal_sphere}(as_{h1}, as_{h2}) \& \\
& \neg \mathsf{inherit}(as_{h1}, s, app, o, +a) \& \neg \mathsf{inherit}(as_{h1}, s, app, o, -a) \& \\
& \neg \mathsf{inherit}(as_{h2}, s, app, o, +a) \& \neg \mathsf{inherit}(as_{h2}, s, app, o, -a) \& \\
& \mathsf{bequeath}(as_{h1}, s, app, o, +a) \& \neg \mathsf{bequeath}(as_{h2}, s, app, o, -a) \\
\mathsf{inherit}(AS, s, app, o, -a) \leftarrow & \mathsf{spherein}(AS, as_{h1}) \& \mathsf{spherein}(AS, as_{h2}) \& \neg \mathsf{equal_sphere}(as_{h1}, as_{h2}) \& \\
& \neg \mathsf{inherit}(as_{h1}, s, app, o, +a) \& \neg \mathsf{inherit}(as_{h1}, s, app, o, -a) \& \\
& \neg \mathsf{inherit}(as_{h2}, s, app, o, +a) \& \neg \mathsf{inherit}(as_{h2}, s, app, o, -a) \& \\
& \mathsf{bequeath}(as_{h1}, s, app, o, -a) \& \neg \mathsf{bequeath}(as_{h2}, s, app, o, +a)
\end{aligned}$$

Die beiden Variablen $as_{h1}$ und $as_{h2}$ können mit verschiedenen Belegungen versehen werden. Bei den obigen Regeln und genau zwei übergeordneten Autorisierungssphären spielt es keine Rolle, welche Kombinationen zuerst getestet werden [19]. Die beide Regelsätze 6.11 und 6.12 enthalten Negationen des Regelkopfes. Während dies zur Formulierung der Zugriffsregeln notwendig ist, stellt es für die Evaluierung durch einen Interpreter ein Problem dar, da bei einer geschichteten Datalog–Sprache nur Negationen von Prädikaten im Rumpf enthalten sein dürfen, die auf einer tieferen Schicht liegen. Da jedoch nur inherit Prädikate der nächsthöheren Autorisierungssphäre negiert werden, kann in Abschnitt 8.2 eine Möglichkeit vorgestellt werden, wie die vorliegenden Regeln in ein geschichtetes Datalog–Programm umgewandelt werden können. Die Eigenschaften der Regeln, die im Folgenden näher untersucht werden, sind dadurch nicht beeinträchtigt.

### 6.4.2 Erkennen von Vererbungskonflikten

Gemäß der Definition 6.5 von inherit wird für jede übergeordneten Autorisierungssphäre der entsprechende Regelsatz 6.11 oder 6.12 ausgewertet. Letzterer wird erzeugt, wenn es zwei oder mehr übergeordnete Autorisierungssphären gibt (vgl. Abschnitt 6.4.1). Die Definition der Konfliktregel muss dieser Vorgehensweise Rechnung tragen und entsprechend der Definition 6.5 aufgebaut sein.

**Definition 6.6 (Konfliktregel)**
Die Konfliktregel hat die folgende Form

$$\mathbf{conflict}(as, s, app, o, a) \leftarrow L_1 \& \ldots \& L_n$$

wobei $as \in \mathcal{AS}$ die Autorisierungssphäre bezeichnet, in der ein Konflikt unmittelbar auftritt, falls alle Prädikate $L_i \in \{spherein, equal\_sphere, bequeath, inherit\}$ und $L_n \in \{inherit\}$ (mit festem $n \in I\!N$ und $1 \leq i < n$) mit einer entsprechenden Belegung für $s \in \mathcal{V}_S$, $app \in \mathcal{V}_{\mathcal{AP}}$ und $o \in \mathcal{V}_O$ mit Methode $a \in AC$ gültig sind. Alle Regeln vom Typ inherit müssen aus einer Autorisierungssphäre $as'$ sein, die direkter Vorgänger von $as$ gemäß der Hierarchie $<_{\mathcal{AS}}$ ist $(as <_{\mathcal{AS}} as')$ .

Da der Interpreter selbst nicht unterscheiden kann, ob der Regelsatz 6.11 oder 6.12 ausgewertet wird, es also nur eine oder mehrere übergeordnete Autorisierungssphären gibt, muss nach jeder

[19] Der Interpreter sollte zur schnelleren Auswertung so arbeiten, dass zuerst die inherit() Prädikate ausgewertet werden. Damit wird der Auswertungspfad zunächst bis zur obersten Autorisierungssphäre führen. Wenn dort keine bequeath() Regel vorliegt, die gültig ist, geht der Auswertungspfad ein Schritt zurück, und die nächst tiefere Autorisierungssphäre wird betrachtet. Sobald eine bequeath() Regel gültig ist, greift auf allen darunterliegenden Autorisierungssphären die erste bzw. zweite Regel und die bequeath() Prädikate werden nicht mehr ausgewertet.

Auswertung von inherit in einer Autorisierungssphäre getestet werden, ob ein Konflikt vorliegt. Dies erfolgt über die folgende Regel:

**Regel 6.13**

$$\begin{aligned}
\mathsf{conflict}(AS,s,app,o,a) \leftarrow & \mathsf{spherein}(AS,as_{h1})\&\mathsf{spherein}(AS,as_{h2})\&\neg\mathsf{equal_sphere}(as_{h1},as_{h2})\& \\
& \mathsf{inherit}(as_{h1},s,app,o,+a)\&\mathsf{inherit}(as_{h2},s,app,o,-a) \\
\mathsf{conflict}(AS,s,app,o,a) \leftarrow & \mathsf{spherein}(AS,as_{h1})\&\mathsf{spherein}(AS,as_{h2})\&\neg\mathsf{equal_sphere}(as_{h1},as_{h2})\& \\
& \neg\mathsf{inherit}(as_{h1},s,app,o,+a)\&\neg\mathsf{inherit}(as_{h1},s,app,o,-a)\& \\
& \mathsf{bequeath}(as_{h1},s,app,o,+a)\&\mathsf{inherit}(as_{h2},s,app,o,-a) \\
\mathsf{conflict}(AS,s,app,o,a) \leftarrow & \mathsf{spherein}(AS,as_{h1})\&\mathsf{spherein}(AS,as_{h2})\&\neg\mathsf{equal_sphere}(as_{h1},as_{h2})\& \\
& \neg\mathsf{inherit}(as_{h1},s,app,o,+a)\&\neg\mathsf{inherit}(as_{h1},s,app,o,-a)\& \\
& \mathsf{bequeath}(as_{h1},s,app,o,-a)\&\mathsf{inherit}(as_{h2},s,app,o,+a) \\
\mathsf{conflict}(AS,s,app,o,a) \leftarrow & \mathsf{spherein}(AS,as_{h1})\&\mathsf{spherein}(AS,as_{h2})\&\neg\mathsf{equal_sphere}(as_{h1},as_{h2})\& \\
& \neg\mathsf{inherit}(as_{h1},s,app,o,+a)\&\neg\mathsf{inherit}(as_{h1},s,app,o,-a)\& \\
& \neg\mathsf{inherit}(as_{h2},s,app,o,+a)\&\neg\mathsf{inherit}(as_{h2},s,app,o,-a)\& \\
& \mathsf{bequeath}(as_{h1},s,app,o,+a)\&\mathsf{bequeath}(as_{h2},s,app,o,-a)
\end{aligned}$$

Wenn die Autorisierungssphäre $AS$ nur eine übergeordnete Autorisierungssphäre hat, wird es keine gültige Belegung für die beiden Variablen $as_{h1}$ und $as_{h2}$ gemäß der Feature Prädikate $\mathsf{spherein}(AS,as_{h1})\&\mathsf{spherein}(AS,as_{h2})\&\neg\mathsf{equal\_sphere}(as_{h1},as_{h2})$ geben und die Auswertung wird abgebrochen. Somit ist der Aufwand für die Auswertung dieser Regel, die u.U. bei jeder Autorisierungssphäre erfolgen muss[20], nicht sehr groß. Durch die Auswertung der inherit Regel ist bereits bekannt, ob die Prädikate $\mathsf{inherit}(as_{h1},s,app,o,+a)$ und $\mathsf{inherit}(as_{h2},s,app,o,-a)$ gültig sind, wobei eine entsprechende Vertauschung der Zustimmung und Ablehnung beim Auswerten keine Rolle spielt[21].

Falls das Prädikat conflict gemäß des Regelsatzes 6.13 nicht gültig ist, gibt es keinen Vererbungskonflikt und die übergeordneten Autorisierungssphären haben keine Zugriffsstrategien für die vorliegende Kombination aus Subjekt, Applikation und Objekt mit der entsprechenden Zugriffsmethode vererbt. Ansonsten bricht die Auswertung der Anfrage mit einer Fehlermeldung ab und der Vererbungskonflikt muss aufgelöst werden.

### 6.4.3 Korrektheit der Umsetzung

Im Folgenden soll bewiesen werden, dass die beiden Regelsätze 6.11 und 6.12 auf Seite 88 das Vererbungsprinzip gemäß Definition 6.1 umsetzen. Hierzu werden zunächst alle Möglichkeiten von gültigen Vererbungsregeln in verschiedenen Autorisierungssphären betrachtet. Es wird angenommen, dass in einer beliebigen Autorisierungssphäre $as \in \mathcal{AS}$ die Prädikate $\mathsf{inherit}(as,s,app,o,+a)$ und $\mathsf{inherit}(as,s,app,o,-a)$ ausgewertet werden.

Um die Beschreibung der indirekten Beziehung zwischen zwei Autorisierungssphären $as_l$ und $as_h$ kompakter darstellen zu können, wird die folgende Notation[22] eingeführt:
$as_l \ll_{\mathcal{AS}} as_h := (as_l, as_h) \notin <_{\mathcal{AS}} \wedge as_l <_{\mathcal{AS}} as_h$

Als erste Möglichkeit wird gezeigt, dass eine Erlaubnis nur dann geerbt werden kann, wenn eine entsprechende Vererbungsregel gültig ist.

[20] Eventuell lässt sich der Interpreter an dieser Stelle noch anpassen. Eine andere Möglichkeit der Beschleunigung wäre es in den Autorisierungssphären, die nur einen Vorgänger haben eine Dummyregel der Form $\mathsf{conflict}(AS,s,app,o,a) \leftarrow FALSE$ zu erzeugen.

[21] Der Cache des Interpreters [Hal05] arbeitet mit konkreten Werten.

[22] Wie in Abschnitt 3.2 für Organisationseinheiten dargestellt, gilt auch für die Relation $<_{\mathcal{AS}}$, dass mit $(as_l, as_h) \in <_{\mathcal{AS}}$ eine $as_l$ direkt zu $as_h$ untergeordnet ist, während mit $as_l <_{\mathcal{AS}} as_h$ sowohl die direkte als auch die transitive Beziehung bezeichnet wird.

**Lemma 6.1**
Es sei $as \in \mathcal{AS}$ beliebig. Wenn $\forall as_h \in \mathcal{AS}$ mit $as <_{\mathcal{AS}} as_h$ das Prädikat bequeath$(as_h, s, app, o, +a)$ nicht gültig ist, dann ist gemäß der Regelsätze 6.11 und 6.12 inherit$(as, s, app, o, +a)$ nicht gültig.

**Beweis zu Lemma 6.1**
Angenommen das Prädikat inherit$(as, s, app, o, +a)$ wäre gültig.
Dann muss die erste oder dritte Regel aus Regelsatz 6.11 bzw. 6.12 gültig sein. Die dritte Regel ist genau dann gültig, wenn in der übergeordneten Autorisierungssphäre $as_{h1}$ bzw. den übergeordneten Autorisierungssphären $as_{h1}, as_{h2}$ keine inherit() Regel gültig ist, jedoch das Prädikat bequeath$(as_{h1}, s, app, o, +a)$. Dies ist laut Lemma aber nicht der Fall.

Die erste Regel ist genau dann gültig wenn in einer übergeordneten Autorisierungssphäre $as_h$ das Prädikat inherit$(as_{h1}, s, app, o, +a)$ gültig ist. Da auch in dieser Autorisierungssphäre aus den gleichen Gründen wie zuvor nie die dritte Regel gültig sein kann, wird ein weiterer Schritt rekursiv nach oben gegangen. In der obersten Autorisierungssphäre ist das Prädikat spherein$(AS, as_{h1})$ nicht gültig. Somit kann auch die erste Regel niemals gültig sein. □

Analog zur Erlaubnis wird auch ein Verbot nur dann geerbt, wenn eine entsprechende Vererbungsregel gültig ist.

**Lemma 6.2**
Es sei $as \in \mathcal{AS}$ beliebig. Wenn $\forall as_h \in \mathcal{AS}$ mit $as <_{\mathcal{AS}} as_h$ das Prädikat bequeath$(as_h, s, app, o, -a)$ nicht gültig ist, dann ist gemäß der Regelsätze 6.11 und 6.12 inherit$(as, s, app, o, -a)$ nicht gültig.

**Beweis zu Lemma 6.2**
Der Beweis erfolgt analog zu dem von Lemma 6.1 nur mit verändertem Vorzeichen bei der Zugriffsmethode. □

Als zweite Möglichkeit wird der Fall betrachtet, dass genau eine Vererbungsregel in einer relevanten Autorisierungssphäre gültig ist. Alle Vererbungsregeln in Autorisierungssphären die nicht oberhalb von $as$ gemäß der Relation $<_{\mathcal{AS}}$ liegen, müssen nicht betrachtet werden.

**Lemma 6.3**
Es sei $as \in \mathcal{AS}$ beliebig. Wenn nur in der Autorisierungssphäre $as_h$ mit $as <_{\mathcal{AS}} as_h$ das Prädikat bequeath$(as_h, s, app, o, +a)$ gültig ist und $\forall as'_h \in \mathcal{AS}, as < as'_h : \neg$bequeath$(as'_h, s, app, o, -a)$, dann ist gemäß der Regelsätze 6.11 und 6.12 das Prädikat inherit$(as, s, app, o, +a)$ gültig.

**Beweis zu Lemma 6.3**
Angenommen, das Prädikat inherit$(as, s, app, o, +a)$ wäre nicht gültig.
Es sei $as$ direkt $as_h$ untergeordnet d.h. $(as, as_h) \in <_{\mathcal{AS}}$. Da in keiner Autorisierungssphäre das Prädikat bequeath$(as'_h, s, app, o, -a)$ gültig ist, kann gemäß Lemma 6.2 inherit$(as_h, s, app, o, -a)$ auch nicht gültig sein. Da nur in $as_h$ das Prädikat bequeath$(as'_h, s, app, o, +a)$ gültig ist, kann gemäß Lemma 6.1 das Prädikat inherit$(as_h, s, o.app, +a)$ nicht gültig sein. Damit wäre aber die dritte Regel aus Regelsatz 6.11 bzw. 6.12 erfüllt.

Es seien $as$ nur indirekt $as_h$ untergeordnet ($as \ll_{\mathcal{AS}} as_h$). Dann wird die erste Regel solange rekursiv aufgerufen, bis zur Autorisierungssphäre $as'$, die $as_h$ direkt untergeordnet ist. Hier gilt dann wieder wie oben die dritte Regel. □

Für den Fall $as \ll_{\mathcal{AS}} as_h$ spielt es keine Rolle, ob es eine oder mehrere "Pfade" zwischen den Autorisierungssphären gibt. Für die Gültigkeit der Regel ist eine Möglichkeit ausreichend.
Für den Fall das genau eine Vererbungsregeln mit einem expliziten Verbot vorliegt, lässt sich die Korrektheit analog beweisen.

**Lemma 6.4**
Es sei $as \in \mathcal{AS}$ beliebig. Wenn nur in der Autorisierungssphäre $as_h$ mit $as <_{\mathcal{AS}} as_h$ das Prädikat bequeath$(as_h, s, app, o, -a)$ gültig ist und $\forall as'_h \in \mathcal{AS}, as < as'_h : \neg$bequeath$(as'_h, s, app, o, +a)$, dann ist gemäß der Regelsätze 6.11 und 6.12 das Prädikat inherit$(as, s, app, o, -a)$ gültig.

**Beweis zu Lemma 6.4**
Der Beweis erfolgt analog zu dem von Lemma 6.3 nur mit verändertem Vorzeichen bei der Zugriffsmethode und vertauschen der Lemma 6.1 und 6.2. □

In den beiden folgenden Lemmata wird gezeigt, dass eine Vererbungsregel einer übergeordneten Autorisierungssphäre stets durchgesetzt wird, auch wenn eine untergeordnete Autorisierungssphäre ebenfalls eine Vererbungsregel definiert hat. Zur besseren Lesbarkeit wird im Beweis nicht immer darauf eingegangen, dass $\mathsf{bequeath}(as'_h, s, app, o, +a)$ oder $\mathsf{bequeath}(as'_h, s, app, o, -a)$ gültig ist, sondern statt dessen die Notation $\mathsf{bequeath}(as'_h, s, app, o, \pm a)$ verwendet.

**Lemma 6.5**
Es sei $as \in \mathcal{AS}$ beliebig. Wenn es eine Autorisierungssphäre $as_h$ mit $as <_{\mathcal{AS}} as_h$ gibt, für die das Prädikat $\mathsf{bequeath}(as_h, s, app, o, +a)$ gültig ist und für alle anderen Autorisierungssphären $as'_h$ mit $as <_{\mathcal{AS}} as'_h$, für die das Prädikat $\mathsf{bequeath}(as'_h, s, app, o, \pm a)$ gültig ist, gilt $as'_h <_{\mathcal{AS}} as_h$. Dann ist gemäß der Regelsätze 6.11 und 6.12 $\mathsf{inherit}(as, s, app, o, +a)$ gültig.

**Beweis zu Lemma 6.5**
Angenommen das Prädikat $\mathsf{inherit}(as, s, app, o, +a)$ wäre nicht gültig.

1. Es seien $(as, as'_h), (as'_h, as_h) \in <_{\mathcal{AS}}$.
   Aufgrund der Annahme, dass $\forall as'_h, as <_{\mathcal{AS}} as'_h$ für die $\mathsf{bequeath}(as'_h, s, app, o, \pm a)$ gilt, ist gemäß Lemma 6.1 und 6.2 weder $\mathsf{inherit}(as_h, s, app, o, +a)$ noch $\mathsf{inherit}(as_h, s, app, o, -a)$ gültig. Für eine eventuell vorhanden Autorisierungssphäre $as_{h'}$ mit $(as'_h, as_{h'}) \in <_{\mathcal{AS}}$ gilt das gleich, außerdem kann gemäß Annahme $\mathsf{bequeath}(as_{h'}, s, app, o, \pm a)$ nicht gelten. Somit ist für $as'_h$ auf jeden Fall die dritte Regel aus Regelsatz 6.11 bzw. 6.12 gültig. Für eine eventuell vorhandene Autorisierungssphäre $as'$ mit $(as, as'), (as', as_h) \in <_{\mathcal{AS}}$ folgt aufgrund der gleichen Argumentation $\mathsf{inherit}(as', s, app, o, +a)$, unabhängig davon, ob das Prädikat $\mathsf{bequeath}(as', s, app, o, \pm a)$ gilt. Somit ist für $as$ auf jeden Fall die erste Regel aus Regelsatz 6.11 bzw. 6.12 erfüllt (ggf. auch mit beiden Belegungen für $as_{h1}$).

2. Es sei $as <_{\mathcal{AS}} as'_h \ll_{\mathcal{AS}} as_h$ und $\nexists as''_h : as <_{\mathcal{AS}} as''_h <_{\mathcal{AS}} as_h$ für die das Prädikat $\mathsf{bequeath}(as''_h, s, app, o, \pm a)$ gilt.
   Dann ist analog zur Argumentation im Punkt 1 für ein $as'$ mit $(as', as_h) \in <_{\mathcal{AS}}$ und $as'_h <_{\mathcal{AS}} as'$ die dritte Regel aus Regelsatz 6.11 bzw. 6.12 erfüllt. Für jede $as''$ mit $as'_h <_{\mathcal{AS}} as''' <_{\mathcal{AS}} as''$ ist damit auch $\mathsf{inherit}(as'', s, app, o, +a)$ aufgrund der ersten Regel aus Regelsatz 6.11 bzw. 6.12 gültig. Aus demselben Grund gilt dann auch $\mathsf{inherit}(as'_h, s, app, o, +a)$. Für alle eventuell vorhandenen $as'''$ mit $as <_{\mathcal{AS}} as''' <_{\mathcal{AS}} as'_h$ gilt dann ebenfalls aufgrund der ersten Regel $\mathsf{inherit}(as''', s, app, o, +a)$. Dementsprechend ist auch $\mathsf{inherit}(as, s, app, o, +a)$ gültig.

3. Es sei $as <_{\mathcal{AS}} as''_h <_{\mathcal{AS}} as'_h <_{\mathcal{AS}} as_h$ mit $\mathsf{bequeath}(as''_h, s, app, o, \pm a)$ aber $\nexists as'''_h, as <_{\mathcal{AS}} as''_h \not<_{\mathcal{AS}} as'_h$ für die $\mathsf{bequeath}(as'''_h, s, app, o, \pm a)$ gilt.
   Gemäß der Argumentation in Punkt 2 gilt $\mathsf{inherit}(as'_h, s, app, o, +a)$. Für eine Autorisierungssphäre $as'$ mit $(as''_h, as') \in <_{\mathcal{AS}}$ und $as' <_{\mathcal{AS}} as'_h$ gilt dann gemäß Punkt 2 auch $\mathsf{inherit}(as', s, app, o, +a)$. Entsprechend ist für $as'_h$ auch die erste Regel aus Regelsatz 6.11 bzw. 6.12 erfüllt und es gilt $\mathsf{inherit}(as''_h, s, app, o, +a)$. Für beliebige weitere Autorisierungssphären $as^*_h$ mit $as <_{\mathcal{AS}} as^*_h <_{\mathcal{AS}} as''_h$ für die $\mathsf{bequeath}(as^*_h, s, app, o, \pm a)$ gültig ist, gilt analog zur oberen Argumentation auch, dass $\mathsf{inherit}(as^*_h, s, app, o, +a)$ gilt. Ebenso gilt das Prädikat $\mathsf{inherit}(as, s, app, o, +a)$.

4. Es sei $as <_{\mathcal{AS}} as'_h <_{\mathcal{AS}} as_h$ und $as <_{\mathcal{AS}} as''_h <_{\mathcal{AS}} as_h$ mit $\mathsf{bequeath}(as''_h, s, app, o, \pm a)$, aber $as'_h \not<_{\mathcal{AS}} as''_h$ und $as''_h \not<_{\mathcal{AS}} as'_h$. Es sei $as'$ die oberste Autorisierungssphäre für die gilt $as' <_{\mathcal{AS}} as'_h \wedge as' <_{\mathcal{AS}} as''_h$. Für $as'_l$ und $as'_r$ gilt $(as', as'_l) \in <_{\mathcal{AS}} \wedge as'_l <_{\mathcal{AS}} as'_h$ und entsprechend $(as', as'_r) \in <_{\mathcal{AS}} \wedge as'_r <_{\mathcal{AS}} as''_h$.
   Gemäß Punkt 3 gilt sowohl für $as'_l$ als auch $as'_r$ das Prädikat $\mathsf{inherit}(as'_l, s, app, o, +a)$ und

$\mathsf{inherit}(as'_r, s, app, o, +a)$. Damit ist in der Autorisierungssphäre $as'$ die erste Regel aus Regelsatz 6.12 für beide Belegungen der Variable $as_{h1}$ gültig. Für den Fall, dass $as'$ nicht mit $as$ identisch ist, kann analog zum Punkt 2 gezeigt werden, dass auch $\mathsf{inherit}(as, s, app, o, +a)$ gilt.

Für weitere $as^*_h$ mit $as <_{\mathcal{AS}} as^*_h <_{\mathcal{AS}} as'_h$ und $as <_{\mathcal{AS}} as^*_h <_{\mathcal{AS}} as''_h$ für die das Prädikat $\mathsf{bequeath}(as^*_h, s, app, o, \pm a)$ gültig ist, folgt analog zur oberen Argumentation auch, dass das Prädikat $\mathsf{inherit}(as^*_h, s, app, o, +a)$ gilt. Daraus folgt dann entsprechend $\mathsf{inherit}(as, s, app, o, +a)$. □

**Lemma 6.6**
Es sei $as \in \mathcal{AS}$ beliebig. Wenn es eine Autorisierungssphäre $as_h$ mit $as <_{\mathcal{AS}} as_h$ gibt, für die das Prädikat $\mathsf{bequeath}(as_h, s, app, o, -a)$ gültig ist und für alle anderen Autorisierungssphären $as'_h$ mit $as <_{\mathcal{AS}} as'_h$, für die das Prädikat $\mathsf{bequeath}(as'_h, s, app, o, \pm a)$ gültig ist, gilt $as'_h <_{\mathcal{AS}} as_h$. Dann ist gemäß der Regelsätze 6.11 und 6.12 $\mathsf{inherit}(as, s, app, o, -a)$ gültig.

**Beweis zu Lemma 6.6**
Der Beweis erfolgt analog zu dem von Lemma 6.5 nur mit verändertem Vorzeichen bei der Zugriffsmethode. □

Eine weitere Möglichkeit ist, dass zwei übergeordnete Autorisierungssphären, die gemäß $<_{\mathcal{AS}}$ nicht zueinander in Beziehung stehen, beide eine Erlaubnis vererben, bzw. beide ein Verbot.

**Lemma 6.7**
Es sei $as \in \mathcal{AS}$ beliebig. Wenn es zwei Autorisierungssphäre $as_h$ und $as'_h$ mit $as <_{\mathcal{AS}} as_h$, $as <_{\mathcal{AS}} as'_h$, $as_h \not<_{\mathcal{AS}} as'_h$ $as'_h \not<_{\mathcal{AS}} as_h$ gibt, für die $\mathsf{bequeath}(as_h, s, app, o, +a)$ und $\mathsf{bequeath}(as'_h, s, app, o, +a)$ gültig sind, und für alle anderen Autorisierungssphären $as'$ mit $as <_{\mathcal{AS}} as'$, für die das Prädikat $\mathsf{bequeath}(as'_h, s, app, o, \pm a)$ gültig ist, gilt $as' <_{\mathcal{AS}} as_h \vee as' <_{\mathcal{AS}} as'_h$, dann ist gemäß der Regelsätze 6.11 und 6.12 $\mathsf{inherit}(as, s, app, o, +a)$ gültig.

**Beweis zu Lemma 6.7**
Es sei $as^*$ die oberste Autorisierungssphäre für die $as <_{\mathcal{AS}} as^* <_{\mathcal{AS}} as_h$ und $as <_{\mathcal{AS}} as^* <_{\mathcal{AS}} as'_h$ gilt. Angenommen das Prädikat $\mathsf{inherit}(as^*, s, app, o, a)$ wäre nicht gültig.

Da somit $as^*$ zwei übergeordnete Autorisierungssphären hat, wird der Regelsatz 6.12 ausgewertet.

1. Es sei $(as^*, as') \in <_{\mathcal{AS}} \wedge as' <_{\mathcal{AS}} as'_h$ sowie $(as^*, as'') \in <_{\mathcal{AS}} \wedge as'' <_{\mathcal{AS}} as''_h$.
   Damit gelten entsprechend der Lemmata 6.3 bzw. 6.5 die Prädikate $\mathsf{inherit}(as', s, app, o, +a)$ und $\mathsf{inherit}(as'', s, app, o, +a)$. Demnach ist die erste Regel in Regelsatz 6.12 mit beiden möglichen Belegungen für die Variable $as_{h1}$ gültig,

2. Es sei $(as^*, as'_h) \in <_{\mathcal{AS}}$ sowie $(as^*, as'') \in <_{\mathcal{AS}} \wedge as'' <_{\mathcal{AS}} as''_h$.
   Damit gilt aufgrund der Lemmata 6.1 und 6.2 weder das Prädikat $\mathsf{inherit}(as'_h, s, app, o, +a)$ noch $\mathsf{inherit}(as'_h, s, app, o, -a)$. Analog zu Punkt 1 gilt $\mathsf{inherit}(as'', s, app, o, +a)$. Demnach ist die erste Regel in Regelsatz 6.12 gültig,

   Analog kann der Fall $(as^*, as') \in <_{\mathcal{AS}} \wedge as' <_{\mathcal{AS}} as'_h$ sowie $(as^*, as''_h) \in <_{\mathcal{AS}}$ betrachtet werden.

3. Es sei $(as^*, as'_h) \in <_{\mathcal{AS}}$ sowie $(as^*, as''_h) \in <_{\mathcal{AS}}$.
   Damit ist aufgrund der Lemmata 6.1 und 6.2 keine $\mathsf{inherit}$ Regel in $as'_h$ oder $as''_h$ gültig. Allerdings ist dann die dritte Regel im Regelsatz 6.12 mit beiden Belegungen für die Variable $as_{h1}$ gültig.

Falls $as$ die oberste Autorisierungssphäre ist, sodass gilt $as <_{\mathcal{AS}} as_h \wedge as <_{\mathcal{AS}} as'_h$, gilt die gleiche Argumentation wie oben an Stelle von $as^*$ mit $as$. Ansonsten ist analog zu Lemma 6.3 bzw. 6.5 auch $\mathsf{inherit}(as, s, app, o, +a)$ gültig.

□

**Lemma 6.8**
Es sei $as \in \mathcal{AS}$ beliebig. Wenn es zwei Autorisierungssphäre $as_h$ und $as'_h$ mit $as <_{\mathcal{AS}} as_h$, $as <_{\mathcal{AS}} as'_h$, $as_h \not<_{\mathcal{AS}} as'_h$ $as'_h \not<_{\mathcal{AS}} as_h$ gibt, für die bequeath$(as_h, s, app, o, -a)$ und bequeath$(as'_h, s, app, o, -a)$ gültig sind, sowie für alle anderen Autorisierungssphären $as'$ mit $as <_{\mathcal{AS}} as'$, für die das Prädikat bequeath$(as'_h, s, app, o, \pm a)$ gültig ist, gilt $as' <_{\mathcal{AS}} as_h \vee as' <_{\mathcal{AS}} as'_h$, dann ist gemäß der Regelsätze 6.11 und 6.12 inherit$(as, s, app, o, -a)$ gültig.

**Beweis zu Lemma 6.8**
Der Beweis erfolgt analog zu dem von Lemma 6.7 nur mit verändertem Vorzeichen bei der Zugriffsmethode und mit Verweis auf die Lemmata 6.4 und 6.6 an Stelle von 6.3 und 6.5 □

Ein Problem tritt in dem Fall auf, wenn in zwei direkt übergeordnete Autorisierungssphären die nicht gemäß der Relation $<_{\mathcal{AS}}$ in Beziehung stehen, sowoahl bequeath$(as_1, s, app, o, +a)$ als auch bequeath$(as_2, s, app, o, -a)$ gültig ist. Dies wird als Vererbungskonflikt aufgefasst, der im Abschnitt 6.3 betrachtet wurde. Dies muss bei den korrekten Umsetzungen des Vererbungsprinzips ebenfalls berücksichtigt werden. Danach wird im Abschnitt 6.4.3 der Beweise, dass die beiden Regelsätze 6.11 und 6.12 das Vererbungsprinzip aus Definition 6.1 umsetzen, durch den Satz 7.2 abgeschlossen.

Im Fall eines Vererbungskonfliktes wird die gesamte Auswertung abgebrochen und es muss eine Lösungsstrategie gesucht werden. Wenn jedoch kein Vererbungskonflikt vorliegt, wird das Vererbungsprinzip gemäß Definition 6.1 durch die Auswertung der inherit() Regeln korrekt umgesetzt.

Im Falle eines Vererbungskonfliktes ist in der Autorisierungssphäre $as_x$, in der dieser auftritt, keine Erbregel gültig. Zusätzlich wird, wie im Abschnitt 6.4.2 beschrieben, die weitere Auswertung abgebrochen.

**Lemma 6.9**
Es sei $as \in \mathcal{AS}$ beliebig. Wenn es zwei Autorisierungssphäre $as_h$ und $as'_h$ mit $as <_{\mathcal{AS}} as_h$, $as <_{\mathcal{AS}} as'_h$, $as_h \not<_{\mathcal{AS}} as'_h$ $as'_h \not<_{\mathcal{AS}} as_h$ gibt, für die bequeath$(as_h, s, app, o, +a)$ und bequeath$(as'_h, s, app, o, -a)$ gültig sind, und es $\nexists as''_h, as_h <_{\mathcal{AS}} as''_h, as'_h <_{\mathcal{AS}} as''_h$ mit bequeath$(as'_h, s, app, o, \pm a)$, dann ist aufgrund der Auswertung des Regelsatzes 6.12 und der gleichzeitigen Auswertung des Regelsatzes 6.13 in mindestens einer Autorisierungssphäre $as^\dagger$ mit $as <_{\mathcal{AS}} as^\dagger <_{\mathcal{AS}} as_h$ und $as <_{\mathcal{AS}} as^\dagger <_{\mathcal{AS}} as'_h$ das Prädikat conflict$(as^\dagger, s, app, o, a)$ gültig. Dies kann auch direkt für $as$ der Fall sein.

Über die Anzahl der Autorisierungssphären $as^*$ mit $as^\dagger <_{\mathcal{AS}} as^* <_{\mathcal{AS}} as_h$ oder $as^\dagger <_{\mathcal{AS}} as^* <_{\mathcal{AS}} as'_h$ in denen bequeath$(as^*, s, app, o, \pm a)$ gilt wird keine Annahme gemacht.

**Beweis zu Lemma 6.9**
Angenommen das Prädikat conflict$(as^\dagger, s, app, o, a)$ wäre nicht gültig. Es sei $as^\dagger$ die oberste Autorisierungssphäre für die $as <_{\mathcal{AS}} as^\dagger <_{\mathcal{AS}} as_h$ und $as <_{\mathcal{AS}} as^\dagger <_{\mathcal{AS}} as'_h$ gilt. Da somit $as^\dagger$ zwei übergeordnete Autorisierungssphären hat, wird der Regelsatz 6.12 ausgewertet und somit auch stets überprüft, ob conflict$(as^\dagger, s, app, o, a)$ gemäß des Regelsatzes 6.13 gilt.

1. Es sei $(as^\dagger, as') \in <_{\mathcal{AS}} \wedge as' <_{\mathcal{AS}} as'_h$ sowie $(as^\dagger, as'') \in <_{\mathcal{AS}} \wedge as'' <_{\mathcal{AS}} as''_h$.
   Damit gilt entsprechend der Lemmata 6.3 bzw. 6.5 inherit$(as', s, app, o, +a)$. Ebenso gilt entsprechend der Lemmata 6.4 bzw. 6.6 inherit$(as'', s, app, o, -a)$. Demnach ist keine Regel in Regelsatz 6.12 gültig, jedoch gilt die erste Regel aus Regelsatz 6.13.

2. Es sei $(as^\dagger, as'_h) \in <_{\mathcal{AS}}$ sowie $(as^\dagger, as'') \in <_{\mathcal{AS}} \wedge as'' <_{\mathcal{AS}} as''_h$.
   Damit gilt aufgrund der Lemmata 6.1 und 6.2 weder das Prädikat inherit$(as'_h, s, app, o, +a)$ noch inherit$(as'_h, s, app, o, -a)$. Analog zu Punkt 1 gilt inherit$(as'', s, app, o, -a)$. Daher ist keine Regel in Regelsatz 6.12 gültig, jedoch gilt die zweite Regel aus Regelsatz 6.13.

3. Es sei $(as^\dagger, as') \in <_{\mathcal{AS}} \wedge as' <_{\mathcal{AS}} as'_h$ sowie $(as^\dagger, as''_h) \in <_{\mathcal{AS}}$.
   Dies ist analog zum vorigen Punkt 2 nur mit veränderten Vorzeichen bei der Zugriffsmethode. Damit ist keine Regel in Regelsatz 6.12 gültig, jedoch gilt die dritte Regel aus Regelsatz 6.13.

4. Es sei $(as^{\dagger}, as'_h) \in <_{\mathcal{AS}}$ sowie $(as^{\dagger}, as''_h) \in <_{\mathcal{AS}}$.
   Damit ist aufgrund der Lemmata 6.1 und 6.2 keine inherit Regel in $as'_h$ oder $as''_h$ gültig. Daher ist keine Regel in Regelsatz 6.12 gültig, jedoch gilt die vierte Regel aus Regelsatz 6.13.

Falls $as$ die oberste Autorisierungssphäre ist, sodass gilt $as <_{\mathcal{AS}} as_h \wedge as <_{\mathcal{AS}} as'_h$, gilt die gleiche Argumentation wie oben an Stelle von $as^{\dagger}$ mit $as$.

□

Obwohl in der Autorisierungssphäre $as^{\dagger}$ keine Regel aus Regelsatz 6.12 gültig ist, wird bei einer dritten (oder noch mehr) Autorisierungssphären das Prädikat $\mathsf{inherit}(as, s, app, o, +a)$ bzw. $\mathsf{inherit}(as, s, app, o, -a)$ nicht gültig werden, da zur Auswertung alle zu $as$ übergeordneten Autorisierungssphären überprüft werden und damit wird auch für zwei Autorisierungssphären in der entsprechenden Autorisierungssphäre $as^{\dagger}$ das $\mathsf{conflict}(as^{\dagger}, s, app, o, a)$ gültig. In diesem Moment bricht die Auswertung wie in Abschnitt 6.4.2 ab.

**Satz 6.1 (Umsetzung des Vererbungsprinzips)**
Das Vererbungsprinzip gemäß Definition 6.1 wird bei der Auswertung von $\mathsf{inherit}(as, s, app, o, +a)$ und $\mathsf{inherit}(as, s, app, o, -a)$ gemäß der Regelsätze 6.11 und 6.12 in der Autorisierungssphäre $as$ umgesetzt. Im Falle eines Vererbungskonfliktes wird dies durch den Regelsatz 6.13 erkannt.

**Beweis zu Satz 6.1**
In den Lemmata 6.1 bzw. 6.2 wurde gezeigt, dass das Prädikat $\mathsf{inherit}(as, s, app, o, +a)$ bzw. $\mathsf{inherit}(as, s, app, o, -a)$ nicht fälschlicherweise gültig wird. In den Lemmata 6.3 bis 6.8 wurde für alle grundlegenden Konstellationen von ein oder mehr Autorisierungssphären $as_h$ mit $as <_{\mathcal{AS}} as_h$, in denen das Prädikat $\mathsf{bequeath}(as_h, s, app, +a)$ bzw. $\mathsf{bequeath}(as_h, s, app, -a)$ gültig ist gezeigt, dass das $\mathsf{inherit}(as, s, app, o, +a)$ bzw. $\mathsf{inherit}(as, s, app, o, -a)$ gemäß dem Vererbungsprinzip gültig ist.
Für den Fall eines Vererbungskonfliktes wurde in Lemma 6.9 gezeigt, dass für mindestens eine Autorisierungssphäre $as^{\dagger}$ mit $as <_{\mathcal{AS}} as^{\dagger}$ das Prädikat $\mathsf{conflict}(as^{\dagger}, s, app, o, a)$ gültig ist und dadurch die Auswertung abgebrochen wird. □

Der Aufruf der Prädikate $\mathsf{inherit}(as, s, app, o, +a)$ und $\mathsf{inherit}(as, s, app, o, -a)$ in jeder Autorisierungssphäre $as$ wird durch weitere Regeln[23] sichergestellt, die nicht von den lokalen Autorisierungsteams geändert werden können. Diese Regeln werden in den Abschnitten 7.2.2 und 7.3 vorgestellt. Somit wird das Vererbungsprinzip in jeder Autorisierungssphäre korrekt umgesetzt.

Für die dezentrale Auswertung wird angenommen, dass alle Erbregeln und Vererbungsregeln einer Autorisierungssphäre an den zuständigen Knoten aller untergeordneten Autorisierungssphären repliziert werden. Falls bei den Erbregeln auch auf die lokalen Zugriffsstrategien zurückgegriffen wird — hierzu reicht ein Prädikat $\mathsf{do}()$ im Rumpf einer Erbregel aus[24] — müssen diese Regelsätze auch an alle untergeordneten Autorisierungssphäre repliziert werden.

[23] siehe Regeln 7.4 auf Seite 105 und Regeln 7.5 auf Seite 108
[24] vgl. Definition 6.2 und Beispielregeln 6.3

# Kapitel 7

# Zugriffe zwischen Autorisierungssphären

Moderne Organisationen strukturieren ihre internen Abläufe zumeist prozessorientiert, da dies von den aktuellen Normen zum Qualitätsmanagement [ISO00b] bzw. deren Anforderungen gemäß [ISO00a] gefordert wird. Die in [Seg03] vorgestellten Maßnahmen zur Verbesserung der Unternehmensabläufe bzw. Qualitätsverbesserung (unabhängig, ob in produzierenden oder dienstleistenden Unternehmen) zielen darauf ab, dass von verschiedenen Stellen[1], die (Teil–)Ergebnisse kontrolliert werden. Diese Stellen sind dabei nicht zwingend die Vorgesetzten, sondern beliebige andere Stellen. Zur Kontrolle wird dabei auch auf die "Daten–"Objekte der anderen Organisationseinheiten zugegriffen.
Daraus ergibt sich, dass über die Grenzen der Abteilungen hinweg Zugriffe auf Datenobjekte notwendig sind. Für die Autorisierung bedeutet dies, dass ein Subjekt, dessen aktive Rolle in einer Organisationseinheit liegt, die zu einer Autorisierungssphäre $as_1$ gehört, auf ein Objekt zugreifen kann, für das eine Organisationseinheit verantwortlich ist, die zu einer anderen Autorisierungssphäre $as_2$ gehört.

Bei diesen Zugriffen nur die Zugriffsstrategien einer Autorisierungssphäre auszuwerten, wird dem dezentralen Ansatz nicht gerecht. Daher werden bei einer entsprechenden Anfrage beide Zugriffsstrategien berücksichtigt, um sowohl die entsprechenden Arbeitsabläufe für die Rolle, als auch die Vorgaben der für das Objekt verantwortlichen Autorisierungssphären zu berücksichtigen. Durch die hybride Autorisierung kann es, ähnlich wie bei den Vererbungskonflikten, bei einer konkreten Zugriffsanfrage zu unterschiedlichen Ergebnissen kommen, sodass zum einen der Zugriff erlaubt wird und gleichzeitig von der anderen Autorisierungssphäre der Zugriff verboten wird. Dies wird als **Koordinierungskonflikte** bezeichnet und im Abschnitt 7.1 genauer betrachtet.

Diese Koordinierungskonflikte müssen aufgelöst werden, ohne dass einer einzigen Autorisierungssphäre die Aufgabe einer zentrale Koordinierungsstelle zufällt. Eine Berücksichtigung von Details zu den Arbeitsabläufen aus Organisationseinheiten anderer Autorisierungssphären, wie sie bei der Auflösung von Vererbungskonflikten im Abschnitt 6.3.2 vorgestellt wurde, wäre bei einem dezentralen Ansatz nicht angebracht. Hierzu würden die Autorisierungssphären Detailwissen über die internen Arbeitsabläufe und die daraus resultierenden Zugriffsstrategien benötigen. Dies ist bei einer gemeinsamen untergeordneten Autorisierungssphäre, wie sie für übergreifende Projekte oder eine Matrixorganisation eingerichtet werden, noch vertretbar. Dagegen ist für die oben vorgestellten Arten von Zugriffen, eine Koordinierung durch eine übergeordnete Autorisierungssphäre basierend allein auf den Ergebnissen der untergeordneten Autorisierungssphären sinnvoller. Zudem ist aus organisatorischer Sicht diese übergeordnete Autorisierungssphäre für den Informationsfluss zwischen den Organisationseinheiten verantwortlich [PDF99, Rob01].

[1] In diesem Ansatz entspricht eine Stelle einer bestimmten Rolle in einer Organisationseinheit, bzw. für die Zugriffsanfrage dem Subjekt, das diese Rolle in einer Organisationseinheit aktiviert hat.

Während bei einem allgemeinen Einliniensystem die verantwortliche Autorisierungssphäre noch einfach zu ermitteln ist, wird dies schon bei Stabshierarchien und besonders bei Mehrliniensystemen komplexer. Im Abschnitt 7.2 wird festgelegt, welche Autorisierungssphäre grundsätzlich verantwortlich ist und gezeigt, wie diese ermittelt werden kann.

Eine individuelle Auflösung eines Koordinierungskonfliktes für eine bestimmte Anfrage würde das Autorisierungsteam der verantwortlichen Autorisierungssphäre völlig unnötig belasten. Daher muss die Möglichkeit bestehen, grundsätzliche Vorgehensweisen zur Auflösung eines Koordinierungskonfliktes zu spezifizieren. Das im Abschnitt 7.3 definierte Prädikat grant ermöglicht die Spezifizierung von **Entscheidungsregeln** in der jeweils verantwortlichen Autorisierungssphäre. Wobei verschiedene Vorgehensweisen abhängig von Rolle, Art der Objekte etc. möglich sind. Dabei muss auch darauf geachtet werden, dass keine Inkonsistenzen in diesen Entscheidungsregeln entstehen bzw. diese gegebenenfalls entdeckt werden. Zur weiteren Diskussion dieser Vorgehensweise werden abschließend allgemeine Strategien zur Koordinierung vorgestellt.

## 7.1 Koordinierungskonflikt

Der Zugriff zwischen Organisationseinheiten wurde im Abschnitt 1.3 am Beispiel der Objekte vom Typ Stundenzettel motiviert, auf die die Verantwortlichen aus den Softwareprojekten zur Kontrolle lesend zugreifen müssen. Dabei formuliert die Autorisierungssphäre Buchhaltung, die für die Zugriffsregeln auf die Objekte über die entsprechende Organisationseinheit zuständig ist, nur allgemeine Regeln. Danach darf ein Subjekt in einer übergeordneten Rolle die Stundenzettel einer untergeordneten Rolle einsehen, wie es in den Beispielregeln 5.6 auf Seite 71 dargestellt ist. Die Autorisierungssphäre, die für die Organisationseinheit zuständig ist, in der die Rolle aktiviert ist, kennt die Details der Arbeitsabläufe. In den Beispielregeln 5.7 darf im Projekt Crypto nur ein Project-Manager auf die Stundenzettel lesend zugreifen, während einem Developer dieses Leserecht explizit verweigert wird.
Daher werden in dieser Arbeit die Zugriffsstrategien für beide Organisationseinheiten berücksichtigt, sodass zum einen nur bestimmte Objekte freigegeben werden können, zum anderen die Privilegien ausschließlich an bestimmte Rollen geknüpft werden können. Wenn diese beiden Organisationseinheiten zur gleichen Autorisierungssphäre gehören, müssen dies sowohl die Vererbungsregeln (vgl. Kapitel 6) der übergeordneten Autorisierungssphären als auch die lokalen Zugriffsregeln (vgl. Kapitel 5) entsprechend berücksichtigen.

Wenn die beiden Organisationseinheiten jedoch zu unterschiedlichen Autorisierungssphären gehören, müssen jeweils für beide Autorisierungssphären die Vererbungsregeln und, sofern diese nicht zutreffen, die entsprechenden lokalen Zugriffsregeln ausgewertet werden. Dabei kann es vorkommen, dass in der einen Autorisierungssphäre der Zugriff erlaubt wird, während in der anderen Autorisierungssphäre dieser Zugriff verboten wird. Dies wird in dieser Arbeit als Koordinierungskonflikt bezeichnet.

**Definition 7.1**
Ein **Koordinierungskonflikt** liegt bei einer Zugriffsanfrage zwischen Autorisierungssphären genau dann vor, wenn bei einer Zugriffsanfrage eines Subjektes $s$, das durch eine Applikation $app$ auf ein Objekt $o$ mit der Methode $a$ zugreifen will, die lokalen und geerbten Zugriffsregeln der Autorisierungssphäre, in der das Subjekt eine Rolle aktiviert hat, nicht zur selben Entscheidung kommen, wie bei gleicher Belegung die lokalen und geerbten Zugriffsregeln der Autorisierungssphäre der Organisationseinheit, zu der das Objekt gehört.

Dabei liegt auch dann ein Koordinierungskonflikt vor, wenn eine der beiden Autorisierungssphären zu keinem Ergebnis kommt. Der Unterschied zum Vererbungskonflikt (vgl. Definition 6.4) liegt darin, dass der Koordinierungskonflikt nicht in einer konkreten Autorisierungssphäre auftritt, sondern nur bei Zugriff zwischen Autorisierungssphären. Bei den Lösungsstrategien für Vererbungskonflikte aus Abschnitt 6.3.2 müssen Informationen aus den Arbeitsabläufen der beiden

Organisationseinheiten ausgetauscht werden. Dies soll gerade bei einer dezentralen Administrierung vermieden werden.
Daher wird von einer übergeordneten Autorisierungssphäre ein allgemeines Vorgehen bei Koordinierungskonflikten festgelegt, wobei sich die Vorgehensweisen an bestimmte Bedingungen knüpfen lassen, die sich mittels Feature Prädikaten sowie Kontext Feature Prädikaten und Kontext Trust Leveln festlegen lassen.

Zunächst wird im folgenden Abschnitt 7.2 dargestellt, welche Autorisierungssphären für die Koordinierung verantwortlich sind. Da dies bei komplexeren Organisationsstrukturen durchaus mehrere sein können, muss die koordinierende Autorisierungssphäre ermittelt werden.

## 7.2 Koordinierende verantwortliche Autorisierungssphäre

In den Fällen, in denen die Zugriffsanfrage zwischen zwei Organisationseinheiten von den entsprechen Autorisierungssphäre zu unterschiedlichen Ergebnissen kommt, wird eine übergeordnete Autorisierungssphäre hinzugezogen, die für die Koordinierung dieser Konflikte verantwortlich sein soll. Dabei ist zu klären, in welcher Autorisierungssphäre die Entscheidung in solch einem Fall getroffen werden soll. Eine ausschließliche Ansiedlung dieser Entscheidungen in der obersten Autorisierungssphäre einer Organisation würde diese unnötig belasten. In dieser Arbeit wird dabei die Vorgehensweise gewählt, dass die Autorisierungssphäre, die zu beiden beteiligten Autorisierungssphären übergeordnet ist, **verantwortlich** für die Entscheidung ist, ob und unter welchen Bedingungen ein Zugriff erlaubt bzw. verboten wird.

Für die weitere Diskussion wird **die Relation $<_{\mathcal{AS}}$ als gerichteter Graph aufgefasst**. Zur einfacheren Orientierung basiert die Argumentation auch auf diesen Graph.

**Definition 7.2 (Koordinierende und verantwortliche Autorisierungssphären)**
Eine Autorisierungssphäre $as$ ist genau dann für die Auflösung eines Koordinierungskonfliktes bei einer Zugriffsanfrage eines Subjektes mit aktiver Rolle in einer Organisationseinheit der Autorisierungssphäre $as_r$ auf ein Objekt in einer Organisationseinheit der Autorisierungssphäre $as_o$ **verantwortlich**, wenn sich die zwei Pfade von $as_r$ und $as_o$ bei dieser Autorisierungssphäre zum ersten Mal treffen.
Die Autorisierungssphäre $as$ ist die **koordinierende** Autorisierungssphäre, wenn sie verantwortlich ist und – sofern vorhanden – für alle anderen verantwortlichen Autorisierungssphären $\{as_1, \ldots, as_n\} \setminus as$ dieser Autorisierungssphäre $as$ untergeordnet sind, $\forall as_x = \{as_1, \ldots, as_n\} \setminus as : as_x <_{\mathcal{AS}} as$.

Wenn es sich bei der Hierarchie der Autorisierungssphären $<_{\mathcal{AS}}$ um ein Einlinien-System handelt, ist die unterste Autorisierungssphäre, der sowohl $as_r$ als auch $as_o$ untergeordnet sind verantwortlich. Bezogen auf das Beispiel aus der Einleitung zu diesem Kapitel ist dies der Zugriff durch ein Subjekt in der Rolle Programmer in der Organisationseinheit IPv6 auf ein Objekt in der Organisationseinheit Routing, das technische Spezifikationen der Hardware enthält (vgl. Abbildung 3.1 auf Seite 24). Die Entscheidung sollte in der Autorisierungssphäre Entwicklung getroffen werden, da diese sowohl der Autorisierungssphäre Netzwerk als auch Hardware übergeordnet ist.

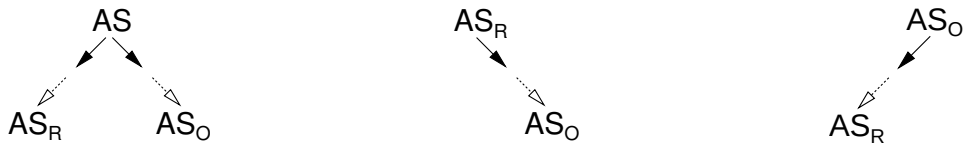
Bei einem Mehrliniensystem kann es allerdings mehrere Pfade von der obersten Autorisierungssphäre bis zu der Autorisierungssphäre $as_r$, in dem die Rolle aktiviert ist und der Autorisierungssphäre $as_o$ zu der das Objekt gehört, geben. Jedes Mal, wenn diese Pfade sich bei einer Autorisierungssphäre $as_x$ treffen, kann ein Koordinierungskonflikt auftreten.

### 7.2.1 Ermittlung der verantwortlichen Autorisierungssphären

In Abbildung 7.1 sind die einfachen Konstellationen, die in einem Einliniensystem ohne Stabshierarchie (vgl. Abschnitt 2.5.1) ausschließlich anzutreffen sind, schematisch dargestellt[2].

In den Abbildungen 7.1 bis 7.4 hat das Subjekt stets eine Rolle in einer Organisationseinheit der Autorisierungssphäre $\mathsf{AS_R}$ aktiviert und will einen Zugriff auf ein Objekt in einer Organisationseinheit der Autorisierungssphäre $\mathsf{AS_O}$.

[2]Die Bedeutung der Linien ist indentisch mit der im vorangegangenen Kapitel.




(a) übergeordnete Autorisierungssphäre

(b) Autorisierungssphäre der aktiven Rolle übergeordnet

(c) Autorisierungssphäre des Objektes übergeordnet

Abbildung 7.1: Beziehung zwischen den unmittelbar verantwortlichen Autorisierungssphären

Neben der Konstellation in Abbildung 7.1(a), die die angesprochen Beispiele aus der Organisation in Abbildung 3.2 schematisch wiedergibt, müssen auch die beiden folgenden Abbildungen betrachtet werden, um die verantwortliche Autorisierungssphäre zu finden. Zum einen kann die Autorisierungssphäre der Organisationseinheit, in der das Subjekt eine Rolle aktiviert hat, zu der Autorisierungssphäre, zu der Organisationseinheit das Objekt gehört, übergeordnet sein. Dies ist in Abbildung 7.1(b) skizziert. Zum anderen kann die Anordnung genau umgekehrt sein, wie in Abbildung 7.1(c) dargestellt. In diesen beiden Fällen der Entscheidung der übergeordneten Autorisierungssphäre den Vorzug zu geben, ist eine naheliegende Möglichkeit. Um das Autorisierungssystem aber möglichst flexibel zu gestalten, sollen die beiden Autorisierungssphären $AS_R$ bzw. $AS_O$ als *verantwortliche Autorisierungssphäre* identifiziert werden. Diese können dann die angesprochene Vorgehensweise — bei der der eigenen Entscheidung der Vorzug gegeben wird — aber auch jede andere spezifizieren.

Die Identifizierung einer verantwortlichen Autorisierungssphäre, in der die gegensätzlichen Entscheidungen koordiniert werden müssen, erfolgt wiederum mit Hilfe eines Prädikates.

**Definition 7.3 (Feststellung einer verantwortlichen Autorisierungssphäre)**
Die Regel zur Feststellung einer verantwortlichen Autorisierungssphäre hat folgende Form

$$\textbf{responsible}(as, as_r, as_o) \leftarrow L_1 \& \ldots \& L_n$$

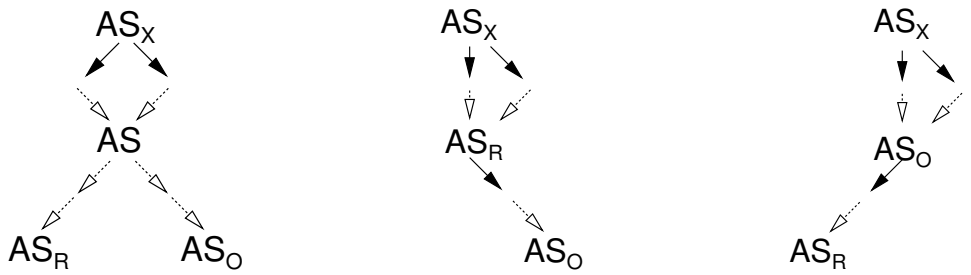
wobei $as \in \mathcal{AS}$ die Autorisierungssphäre bezeichnet, die genau dann für die Koordinierung zwischen den Zugriffsregeln aus der Autorisierungssphäre $as_r \in \mathcal{AST}$ und den Zugriffsregeln der Autorisierungssphäre $as_o \in \mathcal{AST}$ verantwortlich ist, falls alle Prädikate $L_i \in \{\mathsf{subsphere}, \mathsf{spherein}, \mathsf{equal\_sphere}, \mathsf{responsible}\}$ (mit festem $n \in \mathbb{N}$ und $1 \leq i < n$) gültig sind.

Dabei sind $as_r$ und $as_o$ beim ersten Aufruf jeweils mit der Autorisierungssphäre der aktiven Rolle bzw. des Objektes belegt. Mit diesem Prädikat können die folgenden Regeln aufgestellt werden, die jeweils bei einer Konstellation aus Abbildung 7.1 gültig sind.

**Regeln 7.1**

$$\begin{aligned}
\mathsf{responsible}(as, as_r, as_o) \leftarrow & \mathsf{subsphereeq}(as_r, as'_r) \& \mathsf{spherein}(as'_r, as) \& \neg \mathsf{subsphereeq}(as_r, as'_o) \& \\
& \mathsf{subsphereeq}(as_o, as'_o) \& \mathsf{spherein}(as'_o, as) \& \neg \mathsf{subsphereeq}(as_o, as'_r) \& \\
& \neg \mathsf{equal_sphere}(as'_r, as'_o) \\
\mathsf{responsible}(as, as_r, as_o) \leftarrow & \mathsf{subsphere}(as_r, as_o) \& \mathsf{equal_sphere}(as_o, as) \\
\mathsf{responsible}(as, as_r, as_o) \leftarrow & \mathsf{subsphere}(as_o, as_r) \& \mathsf{equal_sphere}(as_r, as)
\end{aligned}$$

Die Regeln aus Regelsatz 7.1 werden jeweils mit der Belegung für eine zu prüfende Autorisierungssphäre $as$ aufgerufen. Dabei ist die Variable $as_r$, mit der Autorisierungssphäre, in der das Subjekt eine Rolle aktiviert hat — im Folgenden kurz die **Autorisierungssphäre der Rolle** genannt—, belegt. Während die Variable $as_o$ die Belegung mit der Autorisierungssphäre erhält, zu der Organisationseinheit das Objektes gehört — im Folgenden kurz die **Autorisierungssphäre des Objektes** genannt.




(a) übergeordnete Autorisierungssphäre (b) Autorisierungssphäre der aktiven Rolle übergeordnet (c) Autorisierungssphäre des Objektes übergeordnet

Abbildung 7.2: Beispiele für nicht verantwortliche Autorisierungssphären $\mathsf{AS_x}$

**Lemma 7.1**
Eine Autorisierungssphäre $as$ ist für die Koordinierung eines Zugriffes eines Subjektes mit aktiver Rolle in einer Organisationseinheit der Autorisierungssphäre $as_r$ auf ein Objekt in einer Organisationseinheit der Autorisierungssphäre $as_o$ verantwortlich, wenn es zwei kreuzungsfreie Pfade gemäß der Relation $<_{\mathcal{AS}}$ von $as$ zu $as_r$ und von $as$ zu $as_o$ gibt oder es einen Pfad zu einer der Autorisierungssphären $as_r$ oder $as_o$ gibt und die andere mit $as$ identisch ist. In diesen Fällen ist eine Regel aus Regelsatz 7.1 gültig.

**Beweis zu Lemma 7.1**
Angenommen keine Regel aus Regelsatz 7.1 wäre in den angegebenen Fällen gültig.

Die erste Regel in Regelsatz 7.1 ist genau dann gültig, wenn $as$ zwei untergeordnete Autorisierungssphären hat, wobei der einen $as_r$ aber nicht $as_o$ untergeordnet ist und der anderen $as_o$ aber nicht $as_r$ untergeordnet ist. Dies bedeutet, dass die Pfade kreuzungsfrei sein müssen.
Die zweite Regel ist genau dann gültig, wenn die Autorisierungssphäre $as$ mit $as_r$ identisch ist und der Autorisierungssphäre $as_o$ untergeordnet ist, d.h. ein Pfad zu $as_o$ existiert.
Entsprechend ist die dritte Regel genau dann gültig, wenn $as$ mit $as_o$ identisch ist und der Autorisierungssphäre $as_r$ untergeordnet ist. □

Die Regeln sind also jeweils bei einer Konstellation aus Abbildung 7.1 gültig. Die Autorisierungssphäre $AS_x$ ist bei den Konstellationen[3] in Abbildung 7.2 deshalb nicht für die Koordinierung verantwortlich, da alle Pfade zu einer einzigen Autorisierungssphäre laufen, die bereits als verantwortlich identifiziert wurde. Damit ist in $AS_x$ keine Koordinierung notwendig. Die Prädikate $\neg\mathsf{subsphereeq}(as_o, as'_r)$ und $\neg\mathsf{subsphereeq}(as_r, as'_o)$ in der ersten Regel sind deshalb nötig, um bei einer Konstellation wie in Abbildung 7.2 nicht die Autorisierungssphäre $AS_x$ mit dieser Regel als verantwortlich zu identifizieren, da in diesem Fall die Wege bis $as_r$ bzw. $as_o$ nicht kreuzungsfrei sind. In Abbildung 7.2(a) ist $\mathsf{AS}$ in beiden Wegen enthalten, und in 7.2(b) bzw. 7.2(c) ist $as_o$ bzw. $as_r$ auch ein innerer Knoten auf dem anderen Pfad.

Solange die Organisation streng hierarchisch gegliedert ist, es sich also um ein Einliniensystem ohne eine Stabshierarchie handelt, können nur die Konstellationen aus Abbildung 7.1 auftreten. Bei einer Organisationsstruktur mit einem Mehrliniensystem oder auch schon bei einem Einliniensystem mit Stabshierarchie, wenn die Stäbe jeweils einer eigenen Autorisierungssphäre zugeordnet sind, (vgl. Abschnitt 2.5.2) kann es komplexere Strukturen bei der Hierarchie der Autorisierungssphären geben, als in der Beispielorganisation in Abbildung 3.2 dargestellt. Dabei kann neben den in Abbildung 7.1 dargestellten einfachen Beziehungen und der daraus resultierenden verantwortlichen Autorisierungssphäre auch eine Autorisierungssphäre deshalb für die Koordinierung verantwortlich sein, um alle möglichen "Pfade" zu der Autorisierungssphäre der Rolle bzw. zu der

[3]Die gestrichelten Pfeile unterhalb von $\mathsf{AS}$ deuten an, dass auch die Möglichkeit besteht, dass entweder die Autorisierungssphäre der Rolle oder die Autorisierungssphäre des Objektes mit dieser Autorisierungssphäre identisch sein kann.

Autorisierungssphäre des Objektes abzudecken. Diese verantwortlichen Autorisierungssphären liegen oberhalb einer Autorisierungssphäre, die mittels einer der Regeln aus Regelsatz 7.1 identifiziert wurden.


(a) Autorisierungssphäre der aktiven Rolle übergeordnet

(b) Autorisierungssphäre des Objektes übergeordnet

Abbildung 7.3: Beziehung zwischen den verantwortlichen Autorisierungssphären

Aus diesem Grund ist die Autorisierungssphäre AS in Abbildung 7.3(a) für die Koordinierung verantwortlich. Zum einen ist eine direkt untergeordnete Autorisierungssphäre $\mathsf{AS}'_{\mathsf{R}}$ der Autorisierungssphäre der Rolle ($\mathsf{AS}_{\mathsf{R}}$) übergeordnet oder mit dieser identisch. Zum anderen ist die Autorisierungssphäre AS′ untergeordnet und steht gemäß der Teilordnung $<_{\mathcal{AS}}$ nicht in Beziehung [4] zur Autorisierungssphäre $\mathsf{AS}'_{\mathsf{R}}$. Wenn diese Konstellation vorliegt, ist die erste der beiden folgenden Regeln gültig, analog für Abbildung 7.3(b) die zweite Regel.

**Regeln 7.2**

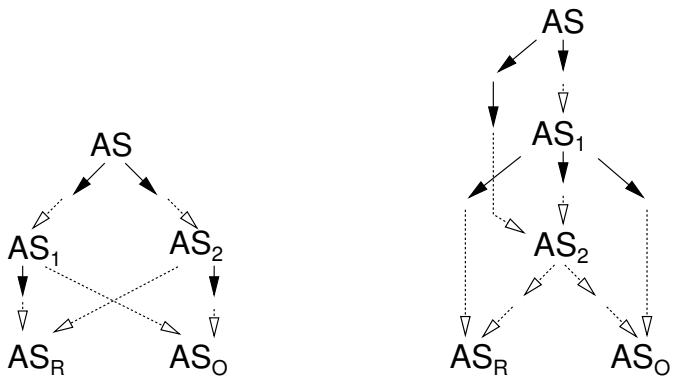
$$\begin{aligned}
\mathsf{responsible}(as, as_r, as_o) \leftarrow\ & \mathsf{spherein}(as'_r, as)\&\mathsf{spherein}(as'_{ro}, as)\&\neg\mathsf{equal_sphere}(as'_{ro}, as'_r)\& \\
& \mathsf{subsphereeq}(as', as'_{ro})\&\mathsf{responsible}(as', as_r, as_o)\& \\
& \mathsf{subsphereeq}(as_r, as'_r)\&\neg\mathsf{subsphereeq}(as_o, as'_r) \\
\mathsf{responsible}(as, as_r, as_o) \leftarrow\ & \mathsf{spherein}(as'_o, as)\&\mathsf{spherein}(as'_{ro}, as)\&\neg\mathsf{equal_sphere}(as'_{ro}, as'_o)\& \\
& \mathsf{subsphereeq}(as', as'_{ro})\&\mathsf{responsible}(as', as_r, as_o)\& \\
& \mathsf{subsphereeq}(as_o, as'_o)\&\neg\mathsf{subsphereeq}(as_r, as'_o)
\end{aligned}$$

**Lemma 7.2**
Eine Autorisierungssphäre $as$ ist für die Koordinierung eines Zugriffes eines Subjektes mit aktiver Rolle in einer Organisationseinheit der Autorisierungssphäre $as_r$ auf ein Objekt in einer Organisationseinheit der Autorisierungssphäre $as_o$ verantwortlich, wenn es zwei kreuzungsfreie Pfade gemäß der Relation $<_{\mathcal{AS}}$ gibt, wobei der eine von $as$ zu $as_r$ führt und von keinem Knoten dieses Pfades ein Pfad zu $as_o$ führt, während der andere zu einer verantwortlichen Autorisierungssphäre führt. Analog ist $as$ auch verantwortlich, wenn der erste Pfad von $as$ zu $as_o$ führt und es von keinem Knoten dieses Pfades einen Pfad zu $as_r$ gibt. In diesen beiden Fällen ist eine Regel aus Regelsatz 7.2 gültig.

**Beweis zu Lemma 7.2**
Angenommen keine der Regeln aus Regelsatz 7.2 wäre in den angegebenen Fällen gültig.

Die erste Regel im Regelsatz 7.2 ist genau dann gültig, wenn $as$ zwei untergeordnete Autorisierungssphären hat, wobei die eine zu $as_r$ aber nicht zu $as_o$ übergeordnet ist und die andere zu einer verantwortlichen Autorisierungssphäre übergeordnet ist. Damit müssen die beiden Pfade, die von den untergeordneten Autorisierungssphären ausgehen, kreuzungsfrei sein.
Analog ist die zweite Regel genau dann gültig, wenn die eine Autorisierungssphäre zu $as_o$ übergeordnet ist, aber nicht zu $as_r$. □

[4] Bei der Koordinierung von gegensätzlichen Entscheidungen, werden in Abschnitt 7.2.2 nur Aufrufe von Regeln untergeordneter Autorisierungssphären zugelassen, um so eine strukturierte Auswertung zu ermöglichen und Zirkel bei den Aufrufen zu verhindern.




(a) übergeordnete Autorisierungssphäre (b)

Abbildung 7.4: Komplexe Beziehung zwischen den verantwortlichen Autorisierungssphären

Die Identifizierung von mehreren verantwortlichen Autorisierungssphären in einer komplexen Hierarchie stellt kein Problem dar, da zur Koordinierung der Entscheidung der Autorisierungssphäre der Rolle und der Autorisierungssphäre des Objektes die **oberste** verantwortliche Autorisierungssphäre herangezogen wird. Bevor diese in Abschnitt 7.2.2 ermittelt wird, müssen noch die Konstellationen aus Abbildung 7.4 berücksichtigt werden.

Bei der Konstellation in Abbildung 7.4(a) werden sowohl die Autorisierungssphäre $\mathsf{AS_1}$ als auch $\mathsf{AS_2}$ durch den Regelsatz 7.1 als verantwortliche Autorisierungssphären identifiziert. Dabei besteht auch die Möglichkeit, dass entweder $\mathsf{AS_1}$ mit der Autorisierungssphäre des Objektes oder $\mathsf{AS_2}$ mit der Autorisierungssphäre der Rolle identisch ist[5]. Sowohl $\mathsf{AS_1}$ als auch $\mathsf{AS_2}$ sind der Autorisierungssphäre $\mathsf{AS}$ unabhängig voneinander untergeordnet. Daher ist die Autorisierungssphäre $\mathsf{AS}$ für die Koordinierung verantwortlich.

Eine weitere Möglichkeit besteht darin, dass die Autorisierungssphären $\mathsf{AS_1}$ und $\mathsf{AS_2}$ auch zueinander in Beziehung stehen, wie in Abbildung 7.4(b) zu sehen ist. Unabhängig davon, ob diese Beziehung besteht, wird die Autorisierungssphäre $\mathsf{AS}$ durch die folgende Regel als verantwortliche Autorisierungssphäre identifiziert.

**Regeln 7.3**

$$\begin{aligned}
\mathsf{responsible}(as, as_r, as_o) \leftarrow\ & \mathsf{subsphereeq}(as_1, as_1')\&\mathsf{spherein}(as_1', as)\&\mathsf{responsible}(as_1, as_r, as_o)\& \\
& \mathsf{subsphereeq}(as_2, as_2')\&\mathsf{spherein}(as_2', as)\&\mathsf{responsible}(as_2, as_r, as_o)\& \\
& \neg\mathsf{equal_sphere}(as_1', as_2')\&\neg\mathsf{subsphereeq}(as_2, as_1')\&\neg\mathsf{subsphereeq}(as_1, as_2') \\
\mathsf{responsible}(as, as_r, as_o) \leftarrow\ & \mathsf{subsphereeq}(as_1, as_1')\&\mathsf{spherein}(as_1', as)\&\mathsf{responsible}(as_1, as_r, as_o)\& \\
& \mathsf{subsphere}(as_2, as_2')\&\mathsf{spherein}(as_2', as)\&\mathsf{responsible}(as_2, as_r, as_o)\& \\
& \mathsf{subsphere}(as_2, as_1)\&\neg\mathsf{equal_sphere}(as_1', as_2')\&\neg\mathsf{subsphereeq}(as_1, as_2')
\end{aligned}$$

**Lemma 7.3**
Eine Autorisierungssphäre $as$ ist für die Koordinierung eines Zugriffes eines Subjektes mit aktiver Rolle in einer Organisationseinheit der Autorisierungssphäre $as_r$ auf ein Objekt in einer Organisationseinheit der Autorisierungssphäre $as_o$ verantwortlich, wenn es zwei kreuzungsfreie Pfade gemäß der Relation $<_{\mathcal{AS}}$ gibt, die zu zwei ebenfalls für denselben Zugriff verantwortlichen Autorisierungssphären führen und nicht in Beziehung zueinander stehen. Alternativ gibt es von $as$ zwei kreuzungsfreie Pfade zu verantwortlichen Autorisierungssphären, die miteinander in Beziehung stehen, wobei es auf dem Pfad

[5]Der umgekehrte Fall ($\mathsf{AS_1} = \mathsf{AS_R}$ bzw. $\mathsf{AS_2} = \mathsf{AS_O}$) ändert nichts grundsätzliches an dieser Konstellation. Falls sowohl $\mathsf{AS_1} = \mathsf{AS_O}$ als auch $\mathsf{AS_2} = \mathsf{AS_R}$ gilt, liegt allerdings wieder eine der beiden Konstellationen aus Abbildung 7.3 vor, abhängig davon ob $\mathsf{AS_R} <_{\mathcal{AS}} \mathsf{AS_O}$ oder $\mathsf{AS_O} <_{\mathcal{AS}} \mathsf{AS_R}$.

zur unteren dieser beiden Autorisierungssphären mindestens eine weitere Autorisierungssphäre gibt, die nicht verantwortlich ist.
In diesem Fällen ist eine der Regeln aus Regelsatz 7.3 gültig.

**Beweis zu Lemma 7.3**
Angenommen keine der Regeln aus Regelsatz 7.2 wäre in den angegebenen Fällen gültig.

Die erste Regel im Regelsatz 7.2 ist genau dann gültig, wenn $as$ zwei untergeordnete Autorisierungssphären hat, die entweder auch eine verantwortliche Autorisierungssphäre sind, oder jeweils zu einer entsprechend verantwortlichen Autorisierungssphäre übergeordnet sind, ohne dabei übergeordnet zu der anderen Autorisierungssphäre zu sein. Die zweite Regel ist genau dann gültig, wenn es zwei zu $as$ untergeordnete Autorisierungssphären gibt, wobei die Erste ebenfalls verantwortlich sein kann oder zu zwei verantwortlichen Autorisierungssphäre übergeordnet ist. Die andere Autorisierungssphäre ist nur zur unteren der beiden verantwortlichen Autorisierungssphäre übergeordnet. □

Es gibt Konstellationen, in denen beide Regeln aus Regelsatz 7.3 gültig sein können. Weitere komplexere Hierarchien lassen sich als Kombinationen aus den bisher vorgestellten Konstellationen darstellen, sodass keine weiteren Regeln zur Identifizierung einer verantwortlichen Autorisierungssphäre benötigt werden.

**Satz 7.1**
Das Prädikat responsible$(as, as_r, as_o)$ ist gemäß der Regelsätze 7.1 bis 7.3 genau für die Autorisierungssphäre $as$ gültig, die gemäß der Definition 7.2 verantwortlich ist.

**Beweis zu Satz 7.1**
Ein Koordinierungskonflikt kann immer dann auftreten, wenn sich die Pfade von der Autorisierungssphäre, in dem die Rolle aktiviert ist und der zu der das Objekt gehört, zum ersten Mal in einer Autorisierungssphäre $as$ treffen. In den Lemmata 7.1 bis 7.3 wurden gezeigt, dass für alle möglichen Konstellationen von $as$ stets mindestens eine der Regeln aus den Regelsätzen 7.1 bis 7.3 gültig ist.

Die tatsächliche Koordinierung erfolgt in der obersten Autorisierungssphäre, die mittels dieser Regeln als verantwortlich identifiziert wurde.

### 7.2.2 Ermittlung der koordinierenden Autorisierungssphäre

Die koordinierende Autorisierungssphäre $as$ für die Zugriffsanfragen zwischen zwei Autorisierungssphären $as_r$ und $as_o$ ist stets die oberste Autorisierungssphäre in der responsible$(as, as_r, as_o)$ gültig ist. Dies kann durch den folgenden Prädikatausdruck dargestellt werden:

$$as \in \mathcal{AS}, \nexists as' \in \mathcal{AS} : \text{responsible}(as, as_r, as_o) \& \text{responsible}(as', as_r, as_o) \& \text{subsphere}(as, as')$$

Die Negierung des Existenzquantors ist in Datalog nicht vorgesehen, sodass diese Bedingung nicht direkt umgesetzt werden kann. Da eine doppelte Implementierungen der Funktionalität des Autorisierungssystems vermieden werden soll, wird im Folgenden eine Möglichkeit entwickelt, wie diese oberste verantwortliche Autorisierungssphäre auch mit Hilfe von Datalog Regeln ermittelt werden kann. Hierzu wird ein weiteres Prädikat coordinate eingeführt. Die Suche nach der koordinierenden Autorisierungssphäre erfolgt dann nach folgendem Schema:
Die mit dem Prädikat coordinate gebildeten Regeln prüfen zunächst beginnend bei der obersten Autorisierungssphäre, ob bereits mit dieser Belegung für $as$ das Prädikat responsible gemäß der Regeln aus dem vorangegangenen Abschnitt 7.2 gültig ist. Sollte die Autorisierungssphäre $as$ nicht verantwortlich sein, wird das Prädikat coordinate für eine untergeordnete Autorisierungssphäre geprüft. Dieser Aufruf mit einer untergeordneten Autorisierungssphäre erfolgt solange bis eine verantwortliche Autorisierungssphäre gefunden ist.

Falls das Prädikat responsible für die aktuelle Autorisierungssphäre gilt, so ist dies auch die oberste verantwortliche und damit koordinierende Autorisierungssphäre[6]. Sollten in dieser Autorisierungssphäre bereits passende Zugriffsregeln für das Subjekt $s$, das mittels der Applikation $app$ und der Methode $a$ auf das Objekt $o$ zugreifen will, geerbt werden (vgl. Kapitel 6), so müssen diese übernommen werden. Ansonsten werden die Regeln, die in der aktuellen Autorisierungssphäre zur Koordinierung definiert sind, ausgewertet. Das entsprechende Prädikat (grant) wird zur besseren Übersicht im nächsten Abschnitt 7.3 definiert und zudem einige Beispiele für mögliche Vorgehensweise bei der Koordinierung vorgestellt.
Das Prädikat zur Auffindung der koordinierenden Autorisierungssphäre ist wie folgt definiert.

**Definition 7.4 (Koordinierungsregel)**
Die Koordinierungsregel hat die Form

$$\textbf{coordinate(}as, s, app, o, \langle sign\rangle a\textbf{)} \leftarrow L_1 \& \ldots \& L_n$$

wobei $as \in \mathcal{AST}$ die Autorisierungssphäre bezeichnet, die überprüft wird. Falls alle Prädikate $L_i \in \{\text{active\_role}, \text{unitof}, \text{sphereof}, \text{spherein}, \text{subsphereeq}, \text{responsible}, \text{inherit}, \text{grant}, \text{coordinate}\}$ (mit festem $n \in I\!N$ und $1 \leq i < n$) mit einer entsprechenden Belegung für $s \in \mathcal{V}_{\mathcal{S}}$, $app \in \mathcal{V}_{\mathcal{AP}}$ und $o \in \mathcal{V}_{\mathcal{O}}$ gültig sind, wird der Zugriff mit Methode $a \in AC$ gemäß $\langle sign\rangle$ explizit gewährt oder verboten. Alle Regeln vom Typ coordinate werden mit einer direkt untergeordneten Autorisierungssphäre $as'$ aufgerufen $((as', as) \in <_{\mathcal{AS}})$. Die Regeln vom Typ inherit sowie grant sind von derselben Autorisierungssphäre $as$.

Der folgende Regelsatz setzt die oben beschriebene Vorgehensweise zur Auffindung der koordinierenden Autorisierungssphäre um. Dieser ist systemweit in jeder Autorisierungssphäre definiert und kann nicht von den Autorisierungsteams verändert werden. Wenn ein Koordinierungskonflikt auftritt, werden in der obersten Autorisierungssphäre $\mathsf{AS_{TOP}}$ die beiden Prädikate $\text{coordinate}(AS_{TOP}, s, app, o, +a)$ und $\text{coordinate}(AS_{TOP}, s, app, o, -a)$ ausgewertet. Abhängig von den geerbten Regeln in der koordinierenden Autorisierungssphäre oder von den Entscheidungsregeln der koordinierenden Autorisierungssphäre ist maximal eines der Prädikate gültig.

**Regeln 7.4**

$$\begin{aligned}
\text{coordinate}(as, s, app, o, +a) \leftarrow\ & \text{activ_role}(s, u_r, r) \& \text{sphereof}(u_r, as_r) \& \\
& \text{unitof}(o, u_o) \& \text{sphereof}(u_o, as_o) \& \text{responsible}(as, as_r, as_o) \& \\
& \text{inherit}(as, s, app, o, +a) \\
\text{coordinate}(as, s, app, o, -a) \leftarrow\ & \text{activ_role}(s, u_r, r) \& \text{sphereof}(u_r, as_r) \& \\
& \text{unitof}(o, u_o) \& \text{sphereof}(u_o, as_o) \& \text{responsible}(as, as_r, as_o) \& \\
& \text{inherit}(as, s, app, o, -a)
\end{aligned}$$

$$\begin{aligned}
\text{coordinate}(as, s, app, o, +a) \leftarrow\ & \text{activ_role}(s, u_r, r) \& \text{sphereof}(u_r, as_r) \& \\
& \text{unitof}(o, u_o) \& \text{sphereof}(u_o, as_o) \& \text{responsible}(as, as_r, as_o) \& \\
& \neg\text{inherit}(as, s, app, o, +a) \& \neg\text{inherit}(as, s, app, o, -a) \& \\
& \text{grant}(as, s, app, o, +a) \\
\text{coordinate}(as, s, app, o, -a) \leftarrow\ & \text{activ_role}(s, u_r, r) \& \text{sphereof}(u_r, as_r) \& \\
& \text{unitof}(o, u_o) \& \text{sphereof}(u_o, as_o) \& \text{responsible}(as, as_r, as_o) \& \\
& \neg\text{inherit}(as, s, app, o, -a) \& \neg\text{inherit}(as, s, app, o, -a) \& \\
& \text{grant}(as, s, app, o, -a)
\end{aligned}$$

[6]Dies ist unabhängig davon, ob der Interpreter einen Breiten- oder Tiefendurchlauf bei der Auswertung der Prädikate durchführt.

$$\begin{aligned}
\mathsf{coordinate}(as, s, app, o, +a) \leftarrow & \mathsf{activ_role}(s, u_r, r)\&\mathsf{sphereof}(u_r, as_r)\& \\
& \mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_o, as_o)\&\neg\mathsf{responsible}(as, as_r, as_o)\& \\
& \mathsf{subsphereeq}(as', as_r)\&\mathsf{subsphereeq}(as', as_o)\& \\
& \mathsf{spherein}(as', as)\&\mathsf{coordinate}(as', s, app, o, +a) \\
\mathsf{coordinate}(as, s, app, o, -a) \leftarrow & \mathsf{activ_role}(s, u_r, r)\&\mathsf{sphereof}(u_r, as_r)\& \\
& \mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_o, as_o)\&\neg\mathsf{responsible}(as, as_r, as_o)\& \\
& \mathsf{subsphereeq}(as', as_r)\&\mathsf{subsphereeq}(as', as_o)\& \\
& \mathsf{spherein}(as', as)\&\mathsf{coordinate}(as', s, app, o, -a)
\end{aligned}$$

Innerhalb einer Autorisierungssphäre $as$ kann höchstens eine Regel gültig sein. Nur wenn keine der ersten vier Regeln gültig ist, insbesondere dann wenn das Prädikat $\mathsf{responsible}(as, as_r, as_o)$ nicht gültig ist, kann das Prädikat coordinate für eine untergeordnete Autorisierungssphäre $as'$ eventuell gültig sein und wird entsprechend ausgewertet.

**Satz 7.2**
Ausgehend von der obersten Autorisierungssphäre werden die Prädikate $\mathsf{coordinate}(as, s, app, o, +a)$ und $\mathsf{coordinate}(as, s, app, o, -a)$ solange rekursiv aufgerufen, bis die oberste verantwortliche Autorisierungssphäre erreicht ist.
Wenn für die oberste verantwortliche Autorisierungssphäre eine Vererbungsregel gültig ist, wird das coordinate Prädikat für die oberste Autorisierungssphäre mit dem entsprechenden Vorzeichen gültig.
Falls keine Vererbungsregel für die oberste verantwortliche Autorisierungssphäre gültig ist, wird für diese Autorisierungssphäre das Prädikat grant ausgewertet und das Prädikat coordinate für die oberste Autorisierungssphäre wird entsprechend dem Vorzeichen gültig.
Falls es keine gültige Vererbungsregel und kein gültiges grant Prädikat gibt, bleibt auch das Prädikat coordinate für die oberste Autorisierungssphäre ungültig.

**Beweis zu Satz 7.2**

1. Angenommen die Prädikate $\mathsf{coordinate}(as, s, app, o, +a)$ bzw. $\mathsf{coordinate}(as, s, app, o, -a)$ würden nicht genau so lange aufgerufen, bis die oberste verantwortliche Autorisierungssphäre erreicht ist.

   Die ersten vier Regeln im Regelsatz 7.4 können nur gültig sein, wenn $\mathsf{responsible}(as, as_r, as_o)$ auch gültig ist. Gemäß Satz 7.1 ist die genau für die Autorisierungssphäre $as$ der Fall nach Definition 7.2 verantwortlich für die Koordinierung sind. Daher werden bei der Auswertung der fünften und sechsten Regel die Prädikate coordinate für die untergeordneten Autorisierungssphären ausgewertet, die sowohl auf dem Pfad zur Autorisierungssphäre der Rolle und zur Autorisierungssphäre des Objektes liegen.
   Wenn es mehr als einen Pfad von der aktuellen Autorisierungssphäre zu denen der Rolle und des Objektes gibt, kann ein beliebiger Pfad zuerst ausgewertet werden, da eine verantwortliche Autorisierungssphäre erst erreicht wird, wenn sich diese beiden Pfade wieder treffen. Ansonsten würde eine Konstellation wie in Abbildung 7.4(b) vorliegen (falls sich die Pfade nicht mehr Treffen würden, läge die Konstellation aus Abbildung 7.4(a) vor), dann wäre jedoch die aktuelle Autorisierungssphäre gemäß Lemma 7.3 auch verantwortlich.
   Der rekursive Aufruf von coordinate() in der fünften und sechsten Regel wird nicht fortgesetzt, wenn die aktuelle Autorisierungssphäre verantwortlich ist.

2. Angenommen für die oberste verantwortliche Autorisierungssphäre wäre eine Vererbungsregel gültig, aber das entsprechende Prädikat $\mathsf{coordinate}(AS_{TOP}, s, app, o, +a)$ bzw. das Prädikat $\mathsf{coordinate}(AS_{TOP}, s, app, o, -a)$ auf der obersten Autorisierungssphäre wird nicht gültig.

   Gemäß Punkt 1 wird das Prädikat coordinate solange rekursiv aufgerufen, bis die oberste verantwortliche Autorisierungssphäre erreicht ist. Die fünfte und sechste Regel aus Regelsatz 7.4 sind für diese Autorisierungssphäre niemals gültig. In den ersten vier Regeln wird das Prädikat $\mathsf{inherit}(as, s, app, o, +a)$ bzw. $\mathsf{inherit}(as, s, app, o, -a)$ ausgewertet. Gemäß Satz

6.1 ist dieses Prädikat gültig, wenn eine Vererbungsregel einer übergeordneten Autorisierungssphäre für die aktuelle Autorisierungssphäre und Zugriffsanfrage $(s, app, o, \pm a)$ gültig ist. Entsprechend wird entweder die erste oder zweite Regel aus Regelsatz 7.4 gültig. Damit werden die entsprechenden coordinate Prädikate der übergeordneten Autorisierungssphären entlang des rekursiven Aufrufs gültig. Dadurch wird letztlich das coordinate Prädikat der obersten Autorisierungssphäre mit dem gleichen Vorzeichen gültig.

3. Angenommen für die oberste verantwortliche Autorisierungssphäre würde bei einer gültigen Vererbungsregel das Prädikat grant ausgewertet.

   Die dritte und vierte Regel aus Regelsatz 7.4 können nur gültig sein, wenn die Erbregel sowohl für die Erlaubnis als auch für das Verbot für diese Autorisierungssphäre nicht gültig sind, gemäß Satz 6.1 liegt dann keine gültige Vererbungsregel für diese Autorisierungssphäre und der Zugriffsanfrage $(s, app, o, \pm a)$ vor.

4. Angenommen für die oberste verantwortliche Autorisierungssphäre würde keine gültige Vererbungsregel vorliegen und bei einem gültigen Prädikat grant für diese Autorisierungssphäre und Zugriffsanfrage würde das entsprechende coordinate Prädikat auf der obersten Autorisierungssphäre nicht gültig.

   Analog zu Punkt 3 sind beide Prädikate $\mathsf{inherit}(as, s, app, o, -a)$ und $\mathsf{inherit}(as, s, app, o, -a)$ nicht gültig und die aktuelle Autorisierungssphäre ist gemäß Punkt 1 die oberste verantwortliche Autorisierungssphäre. Dann wird das Prädikat $\mathsf{coordinate}(as, s, app, o, +a)$ bzw. $\mathsf{coordinate}(as, s, app, o, -a)$ abhängig davon gültig, ob ein Prädikat $\mathsf{grant}(as, s, app, o, +a)$ bzw. $\mathsf{grant}(as, s, app, o, +a)$ gültig wird. Analog zu Punkt 2 wird dann auch das entsprechende Prädikat coordinate für die oberste Autorisierungssphäre gültig.

5. Angenommen für die oberste verantwortliche Autorisierungssphäre würde keine gültige Vererbungsregel vorliegen und kein gültiges Prädikat grant für diese Autorisierungssphäre und Zugriffsanfrage, aber das entsprechende coordinate Prädikat auf der obersten Autorisierungssphäre würde gültig sein.

   In Punkt 1 wurde gezeigt, dass das coordinate() Prädikat aufgrund der fünften und sechsten Regel aus Regelsatz 7.4 rekursiv bis zur obersten verantwortlichen Autorisierungssphäre aufgerufen wird.
   Die erste und zweite Regel kann nur gültig werden, wenn die Erbregel für diese Autorisierungssphäre gültig ist. Da angenommen wird, dass keine gültige Vererbungsregel für diese Autorisierungssphäre vorliegt, kann gemäß Satz 6.1 auch keine Erbregel gültig sein.
   Die dritte oder vierte Regel ist nur gültig, wenn ein entsprechendes grant Prädikat gültig ist. Dies ist gemäß Annahme nicht der Fall.
   Damit ist auf der obersten verantwortlichen Autorisierungssphäre keines der beiden Prädikat $\mathsf{coordinate}(as, s, app, o, +a)$ bzw. $\mathsf{coordinate}(as, s, app, o, -a)$ gültig. Somit sind auch alle Regeln entlang des rekursiven Aufrufs nicht gültig, einschließlich der obersten Autorisierungssphäre.

□

Bevor im Abschnitt 8.1 der Beginn der Auswertung einer Zugriffsanfrage beschrieben wird, muss noch beschrieben werden, wie die oberste verantwortliche und damit koordinierende Autorisierungssphäre entsprechende Regeln festlegen bzw. spezifizieren kann, mit denen die Entscheidungen bei einem Koordinierungskonflikt modelliert werden können.

## 7.3 Vorgehen der koordinierenden Autorisierungssphäre

Die koordinierende Autorisierungssphäre kann auf der einen Seite völlig unabhängig von der Autorisierungssphäre der Rolle und des Objektes entscheiden und sich auch über deren jeweiligen

geerbten Regeln hinwegsetzen. Allerdings werden die vererbten Regeln von übergeordneten Autorisierungssphären der koordinierenden Autorisierungssphäre durch die Regeln im Regelsatz 7.4 auf jeden Fall durchgesetzt.

Auf der anderen Seite sollte die koordinierende Autorisierungssphäre auch direkt auf die Entscheidungen, die in den jeweiligen Autorisierungssphären bezüglich der aktuellen Zugriffsanfrage getroffen werden, zurückgreifen können und diese in den eigenen Regeln berücksichtigen. Eine automatische Auflösung des Koordinierungskonfliktes, etwa danach welche Autorisierungssphäre oder Organisationseinheit "höher" innerhalb der Organisationsstruktur liegt, ist aus den gleichen Gründen wie beim Vererbungskonflikt (siehe Beginn des Abschnittes 6.3.2) nicht sinnvoll.

Um bei der Einbeziehung der Ergebnisse der Autorisierungssphäre der Rolle und des Objektes nicht weiter Schritt für Schritt die gesamte Hierarchie der Autorisierungssphären durchlaufen zu müssen, wird automatisch der Regelsatz 7.5 in jeder Autorisierungssphäre definiert. Diese Regeln können dann von jeder übergeordneten Autorisierungssphäre aufgerufen werden. Damit wird sichergestellt, dass die Entscheidungen der geerbten Regeln aus Kapitel 6 und die lokalen Regeln aus Kapitel 5 für die übergeordneten koordinierenden Autorisierungssphären zur Verfügung gestellt werden[7]. Ein direkter Zugriff auf die Erbregeln einer beliebigen, variablen Autorisierungssphäre ist, wie in Abschnitt 8.2 gezeigt wird, nicht möglich. Daher dienen die Regeln, die mit dem folgenden Prädikat definiert werden, als lokaler Einstieg. Diese Regeln werden beim Anlegen einer Autorisierungssphäre automatisch angelegt und können von keinem Autorisierungsteam modifiziert werden[8]. Durch die Auswertung der inherit Prädikate der übergeordneten Autorisierungssphäre(n) wird sichergestellt, dass vererbte Zugriffsstrategien auch beachtet werden (vgl. Abschnitt 6.4.1).

**Definition 7.5 (Lokaler Einstieg)**
Die Regel für den lokalen Einstieg hat die Form

$$\textbf{local(}as, s, app, o, \langle sign\rangle a\textbf{)} \leftarrow L_1 \& \ldots \& L_n$$

wobei $as \in \mathcal{AST}$ die Autorisierungssphäre bezeichnet, in der diese Regel automatisch definiert wird. Falls alle Prädikate $L_i \in \{inherit, do\}$ (mit festem $n \in \mathbb{N}$ und $1 \leq i < n$) mit einer entsprechenden Belegung für $s \in \mathcal{V}_{\mathcal{S}}$, $app \in \mathcal{V}_{\mathcal{AP}}$ und $o \in \mathcal{V}_{\mathcal{O}}$ gültig sind, wird der Zugriff mit Methode $a \in AC$ gemäß $\langle sign\rangle$ explizit gewährt oder verboten. Alle Regeln vom Typ inherit und do sind von derselben Autorisierungssphäre $as$.

Da gemäß dem Vererbungsprinzip (Def. 6.1) der Spezifikation der übergeordneten Autorisierungssphären der Vorzug gegeben werden muss, wird zunächst überprüft, ob eine gültige Erbregel vorliegt. Nur wenn dies nicht der Fall ist, werden die entsprechenden lokalen Zugriffsstrategien berücksichtigt. Dabei hat das Autorisierungsteam der entsprechenden Autorisierungssphäre keinen Einfluss auf dieses Vorgehen. Die folgenden Regeln sind in jeder Autorisierungssphäre automatisch definiert. Dies ist wichtig, da das Autorisierungsteam sonst in der Lage wäre, die Vererbung in Teilen oder komplett unberücksichtigt zu lassen und somit die Durchsetzung von bestimmten Zugriffsstrategien zu verhindern.

**Regeln 7.5**

$$\begin{aligned}
&\mathsf{local}(AS, s, app, o, +a) \leftarrow \mathsf{inherit}(AS, s, app, o, +a)\\
&\mathsf{local}(AS, s, app, o, -a) \leftarrow \mathsf{inherit}(AS, s, app, o, -a)\\
&\mathsf{local}(AS, s, app, o, +a) \leftarrow \neg\mathsf{inherit}(AS, s, app, o, -a) \& \neg\mathsf{inherit}(AS, s, app, o, -a) \&\\
&\qquad\qquad \mathsf{do}(AS, s, app, o, +a)\\
&\mathsf{local}(AS, s, app, o, -a) \leftarrow \neg\mathsf{inherit}(AS, s, app, o, -a) \& \neg\mathsf{inherit}(AS, s, app, o, -a) \&\\
&\qquad\qquad \mathsf{do}(AS, s, app, o, -a)
\end{aligned}$$

[7] Gleichzeitig werden diese Regeln auch genutzt, wenn die Organisationseinheit des Subjektes und die Organisationseinheit des Objektes zur gleichen Autorisierungssphäre gehören, oder die Organisationseinheiten identisch sind. Mehr dazu folgt im Abschnitt 8.1.

[8] Lediglich bei der Umwandlung zu einem geschichteten Datalog–Programm, die in Abschnitt 8.2 beschrieben wird, werden diese Regeln geändert.

Die Schreibweise des Terms $AS$ verdeutlicht, dass es sich hierbei nicht um eine Variable handelt, sondern in jeder Autorisierungssphäre der konkrete Bezeichner steht. Wenn die ersten beiden Regeln vom Interpreter zuerst ausgewertet werden, wird für die Negierungen in der dritten und vierten Regel keine unnötige Rechenzeit verwendet, da dann bereits festgestellt wurde, dass es für $\mathsf{inherit}(AS, s, app, o, +a)$ bzw. $\mathsf{inherit}(AS, s, app, o, -a)$ keine gültige Regel gibt. Somit werden nun die lokalen Zugriffsstrategien ausgewertet.

Wie diese Regeln zum lokalen Einstieg genutzt werden, obliegt der koordinierenden Autorisierungssphäre. Dort können mit Hilfe der Entscheidungsregeln verschiedene Zugriffsstrategien spezifiziert werden.

**Definition 7.6 (Entscheidungsregel)**
Die Entscheidungsregel hat die Form

$$\mathbf{grant(}as, s, app, o, \langle sign\rangle a\mathbf{)} \leftarrow L_1 \& \ldots \& L_n$$

wobei $as \in \mathcal{AS}$ die Autorisierungssphäre bezeichnet, die diese Regel aufgestellt hat. Falls alle Prädikate $L_i \in \mathcal{F} \cup \mathcal{CF} \cup \mathcal{CL} \cup \{local\}$ (mit festem $n \in I\!N$ und $1 \leq i < n$) mit einer entsprechenden Belegung für $s \in \mathcal{V}_{\mathcal{S}}$, $app \in \mathcal{V}_{\mathcal{AP}}$ und $o \in \mathcal{V}_{\mathcal{O}}$ gültig sind, wird der Zugriff mit Methode $a \in AC$ gemäß $\langle sign\rangle$ explizit gewährt oder verboten. Alle Regeln vom Typ local müssen von einer untergeordneten Autorisierungssphäre $as'$ sein ($as' <_{\mathcal{AS}} as$) sein.

In den vorangegangenen Veröffentlichungen zu dieser Arbeit, insbesondere [WL01], wurde ausschließlich das Prädikat grant dazu eingesetzt, eine Vererbung und Koordinierung der lokalen Zugriffsregeln der verschiedenen Autorisierungssphären zu spezifizieren. Diese konnte dann von einzelnen Autorisierungsteams unter Umständen unterlaufen werden. Daher wurden in der vorliegenden Arbeit weitere Prädikate wie inherit() oder coordinate() eingeführt, die dies verhindern. Das Prädikat grant wird nun ausschließlich für die Definition der Entscheidungsregeln bei der Koordinierung eines Zugriffs zwischen zwei Autorisierungssphären eingesetzt. Durch die Einschränkung, dass lediglich das Prädikat local() aus anderen Autorisierungssphären in den Entscheidungsregeln verwendet werden darf, wird sichergestellt, dass alle an untergeordnete Autorisierungssphären vererbten Zugriffsstrategien auch bei der Entscheidung mit berücksichtigt werden[9].

Die Feature Prädikate und Kontext Feature Prädikate können dazu eingesetzt werden, unterschiedliche Entscheidungen zu treffen. Die Einschränkung auf lokale Einstiegsregeln von untergeordneten Autorisierungssphären verhindert, dass das vorgesehen Vorgehen durch die koordinierende Autorisierungssphäre durchbrochen werden kann. Allerdings kann durch diese Definition nicht verhindert werden, dass an Stelle der Autorisierungssphäre der Rolle und des Objektes die Regeln für den lokalen Einstieg anderer, untergeordnete Autorisierungssphären aufgerufen werden.

**Beispielregeln 7.6**

$$\begin{aligned}
\mathsf{grant}(Entwicklung, s, app, o, -write) \leftarrow\ & \mathsf{activ_role}(s, u_r, r) \& \mathsf{sphereof}(u_r, as_r) \& \\
& \mathsf{unitof}(o, u_o) \& \mathsf{sphereof}(u_o, as_o) \& \\
& \mathsf{subtypeof}(o, Documentation) \& \\
& \mathsf{local}(as_r, s, app, o, +write) \& \\
& \mathsf{local}(as_o, s, app, o, -write) \\
\mathsf{grant}(Entwicklung, s, app, o, +write) \leftarrow\ & \mathsf{activ_role}(s, u_r, r) \& \mathsf{sphereof}(u_r, as_r) \& \\
& \mathsf{unitof}(o, u_o) \& \mathsf{sphereof}(u_o, as_o) \& \\
& \mathsf{subtypeof}(o, Documentation) \& \\
& \neg\mathsf{local}(as_r, s, app, o, -write) \& \\
& \mathsf{local}(as_o, s, app, o, +write)
\end{aligned}$$

[9] Alternativ könnte die Definition auch so abgeändert werden, dass zusätzlich auch nur lokale Zugriffsstrategien mittels des Prädikates do() aufgerufenen werden können. Ob dies sinnvoll ist, müsste sich erst in einer größeren praktischen Untersuchung zeigen.

$$\begin{aligned}\mathsf{grant}(Entwicklung, s, app, o, -write) \leftarrow\ & \mathsf{activ\_role}(s, u_r, r)\&\mathsf{sphereof}(u_r, as_r)\& \\ & \mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_o, as_o)\& \\ & \mathsf{subtypeof}(o, Documentation)\& \\ & \mathsf{local}(as_r, s, app, o, -write)\& \\ & \mathsf{local}(as_o, s, app, o, -write)\end{aligned}$$

Mit den obigen Regeln koordiniert die Autorisierungssphäre Entwicklung zwischen untergeordneten Autorisierungssphären ein Schreibzugriff auf Objekte, die einen Objekttyp unterhalb von Documentation haben (vgl. Abbildung 3.6). Falls die Autorisierungssphäre der Rolle einen Schreibzugriff gewährt, jedoch die Autorisierungssphäre des Objektes diesen ablehnt, erfolgt kein Schreibzugriff. Sofern die Autorisierungssphäre des Objektes den Schreibzugriff gewährt und die der Rolle diesen nicht ablehnt (eine explizite Zustimmung ist nicht erforderlich) kann ein Schreibzugriff erfolgen. Falls beide Autorisierungssphären den Schreibzugriff ablehnen, wird dieser auch von Entwicklung übernommen.

Für den Zugriff auf Objekte vom Typ Stundenzettel kann in der Autorisierungssphäre Geschäftsleitung die folgende Regelung bezüglich des Lesezugriffs festgelegt werden. Wobei der Lesezugriff auch dann verweigert wird, wenn in eine der beiden beteiligten Autorisierungssphären keine gültige local() Regel vorliegt.

**Beispielregeln 7.7**

$$\begin{aligned}\mathsf{grant}(Geschäftsleitung, s, app, o, +read) \leftarrow\ & \mathsf{activ\_role}(s, u_r, r)\&\mathsf{sphereof}(u_r, as_r)\& \\ & \mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_o, as_o)\& \\ & \mathsf{typeof}(o, Stundenzettel)\& \\ & \mathsf{local}(as_r, s, app, o, +read)\& \\ & \mathsf{local}(as_o, s, app, o, +read) \\ \mathsf{grant}(Geschäftsleitung, s, app, o, -read) \leftarrow\ & \mathsf{activ\_role}(s, u_r, r)\&\mathsf{sphereof}(u_r, as_r)\& \\ & \mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_o, as_o)\& \\ & \mathsf{typeof}(o, Stundenzettel)\& \\ & \neg\mathsf{local}(as_r, s, app, o, +read)\& \\ & \mathsf{local}(as_o, s, app, o, +read)\end{aligned}$$

$$\begin{aligned}\mathsf{grant}(Geschäftsleitung, s, app, o, -read) \leftarrow\ & \mathsf{activ\_role}(s, u_r, r)\&\mathsf{sphereof}(u_r, as_r)\& \\ & \mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_o, as_o)\& \\ & \mathsf{typeof}(o, Stundenzettel)\& \\ & \mathsf{local}(as_r, s, app, o, +read)\& \\ & \neg\mathsf{local}(as_o, s, app, o, +read)\end{aligned}$$

Diese Beispiele sollen auch deutlich machen, dass alle möglichen Kombinationen von Zugriffserlaubnissen und Ablehnungen betrachtet werden müssen. So fehlt in den Beispielregeln 7.6 unter anderem eine Regelung für den Fall einer Ablehnung durch die Autorisierungssphäre der Rolle und einer Zustimmung durch die Autorisierungssphäre des Objektes. In den Beispielregeln 7.7 gibt es keine Regelung für den Fall, dass es in beiden Autorisierungssphären keine gültig local() Regel gibt. Dies muss bei der Auswertung der Zugriffsanfragen berücksichtigt werden. Im Abschnitt 7.3.2 werden weitere Strategien zur Koordinierung vorgestellt.

Für die dezentrale Auswertung, die im nächsten Kapitel genauer erläutert wird, wird angenommen, dass analog zu den Vererbungs- und Erbregeln alle Koordinierungsregeln und Entscheidungsregeln einer Autorisierungssphären an den zuständigen Knoten aller untergeordneten Autorisierungssphären repliziert werden. Dabei muss sichergestellt sein, dass die Koordinierungsregeln einer Autorisierungssphäre keine Inkonsistenzen aufweisen.

### 7.3.1 Inkonsistenzen bei Entscheidungsregeln

Ähnlich wie bei den lokalen Zugriffsstrategien und Vererbungsregeln kann es auch bei den Entscheidungsregeln aufgrund der Definition von hybriden Zugriffsstrategien zu Inkonsistenz kommen. Daher muss für die Entscheidungsregeln gelten:

**Vereinbarung 7.1**
In einer Autorisierungssphäre $as$ ist für ein Subjekt $s$, das durch eine Applikation $app$ mit der Zugriffsmethode $a$ auf ein Objekt $o$ zugreifen will, nie sowohl das Prädikat grant$(as, s, app, o, +a)$ als auch grant$(as, s, app, o, -a)$ gültig.

Analog zu der Vereinbarung 5.1 und 6.1 bzw. der daraus resultierenden Inkonsistenzenregel (Def. 5.4 und 6.3) wird die folgende Regel definiert, um Inkonsistenzen innerhalb der Entscheidungsregeln zu erkennen und eine entsprechende Auswertung mit einem Fehler abbrechen zu können.

**Definition 7.7 (Inkonsistenzregeln Entscheidungsregeln)**
Die Inkonsistenzregel zur Entscheidung hat die folgende Form

$$\textbf{inconsistent-grant}(as, s, app, o, a) \leftarrow \text{grant}(as, s, app, o, +a) \& \text{grant}(as, s, app, o, -a)$$

wobei $as \in \mathcal{AS}$ die Autorisierungssphäre bezeichnet, in der diese Regel ausgewertet wird. Falls die Regel mit einer entsprechenden Belegung für $s \in \mathcal{V}_{\mathcal{S}}$, $app \in \mathcal{V}_{\mathcal{AP}}$ und $o \in \mathcal{V}_{\mathcal{O}}$ mit Methode $a \in \mathcal{V}_{\mathcal{A}}$ gültig ist, wird die Zugriffsanfrage mit einem Fehler abgebrochen.

Wie auch bei den vorangegangenen Inkonsistenzregeln müsste der Datalog–Interpreter für die Auswertung der Zugriffsanfragen dann so realisiert werden, dass er bei jeder ersten Auswertung eines Prädikates grant$(as, s, app, o, +a)$ oder grant$(as, s, app, o, -a)$ automatisch auch das Prädikat inconsistent-grant$(as, s, app, o, a)$ auswertet und mit einem Fehler abbricht, wenn dieses Prädikat gültig ist. Alternativ könnte ein Aufruf dieses Prädikates in jeder Regel hinzugefügt werden, die auch ein grant enthält. Da dieses Prädikat nur in Regeln auftaucht, die automatisch generiert werden, wäre dies eine weitere Möglichkeit.

### 7.3.2 Strategien zur Koordinierung

Bei der Spezifizierung von Entscheidungsregeln muss das Autorisierungsteam der verantwortlichen Autorisierungssphäre darauf achten, auch alle Kombinationen von Erlaubnissen bzw. Verboten sowie auch undefinierte Fälle zu berücksichtigen, die als mögliche Ergebnisse von der Autorisierungssphäre der Rolle und des Objektes vorliegen können. Wenn diese beiden Autorisierungssphären zum selben Ergebnis kommen, d.h. Erlaubnis oder Verbot, werden die Entscheidungsregeln nicht ausgewertet. Bei allen anderen möglichen Kombinationen sollte eine passende Entscheidungsregel vorliegen[10].

Zur Auflösung der Koordinierungskonflikte werden im Folgenden einige mögliche Vorgehensweisen aufgezeigt, die als typische Strategien für die Autorisierungsteams der verantwortlichen Autorisierungssphäre herangezogen werden können. Die hierzu formulierten **Entscheidungsregeln** beziehen sich auf die Entscheidungen der Autorisierungssphäre, die für Organisationseinheit in der die Rolle des Subjektes aktiviert ist, verantwortlich ist und die der Autorisierungssphäre, die für die Organisationseinheit des Objektes verantwortlich ist.
Mittels Feature Prädikaten $\mathcal{F}$ sowie Kontext Feature Prädikaten $\mathcal{CF}$ und Kontext Trust Prädikaten $\mathcal{CL}$ (vgl. Kap. 3 und 4) können die Bedingungen, unter denen bestimmte Entscheidungsregeln angewendet werden sollen, spezifiziert werden. So lässt sich deren Anwendung auf Zugriffe zwischen bestimmten Bereichen von Autorisierungssphären beschränken. Ebenso kann die Anwendung vom

[10]Falls keine passende Entscheidungsregel vorliegt, wird, wie im Abschnitt 8.1 erläutert wird, die Zugriffsanfrage verboten.

Typ der Objekte oder den aktiven Rollen abhängig gemacht werden.
Für die weitere Definition und Analyse der Entscheidungsregeln und der Regelsätze, die diese realisieren, sind die genauen Feature Prädikate sowie Kontext Feature Prädikate und Kontext Trust Prädikate nicht von Bedeutung, sie werden daher als `Scope` zusammengefasst. Bei der Verwendung einer der folgenden Regelsätze wird davon ausgegangen, dass *der* `Scope` *für alle Regeln identisch ist.*

### Verbot oder Erlaubnis bevorzugen

Eine aus der Literatur bekannte Vorgehensweise ist "*permission takes precedence*" bzw. "*denial takes precedence*". Im Falle eines Konflikts wird dabei einer Erlaubnis bzw. einem Verbot der Vorzug gegeben.
Die beiden Regelsätze 7.8 und 7.9 zeigen die Entscheidungsregeln für "*permission takes precedence*", durch eine Vertauschung von "$+a$" zu "$-a$" und umgekehrt wird daraus "*denial takes precedence*". Die beiden Regelsätze unterscheiden sich darin, ob auch in den Fällen, in denen eine der beiden Autorisierungssphären zu keinem Ergebnis kommt, die Entscheidungsregel ein Ergebnis liefern soll.

**Regeln 7.8**

$$\mathsf{grant}(\mathsf{AS}, s, app, o, +a) \leftarrow \mathsf{local}(as_1, s, app, o, +a)\&\texttt{Scope}$$
$$\mathsf{grant}(\mathsf{AS}, s, app, o, +a) \leftarrow \mathsf{local}(as_2, s, app, o, +a)\&\texttt{Scope}$$
$$\mathsf{grant}(\mathsf{AS}, s, app, o, -a) \leftarrow \mathsf{local}(as_1, s, app, o, -a)\&\neg\mathsf{local}(as_2, s, app, o, +a)\&\texttt{Scope}$$
$$\mathsf{grant}(\mathsf{AS}, s, app, o, -a) \leftarrow \mathsf{local}(as_2, s, app, o, -a)\&\neg\mathsf{local}(as_1, s, app, o, +a)\&\texttt{Scope}$$

Die beiden ersten Regeln sorgen dafür, dass sobald eine der beiden Autorisierungssphären eine Erlaubnis ausspricht, diese auch weitergegeben wird. Da die Entscheidung der anderen Autorisierungssphäre nicht berücksichtigt wird, kann diese ein Verbot ausgesprochen haben oder keine gültige $\mathsf{local}()$ Regel haben[11]. Die andere Autorisierungssphäre sollte aber dennoch durch entsprechende Feature Prädikate festgelegt werden. Im letzteren Fall wurde für die angefragte Kombination $s, app, o, a$ weder eine Zugriffsstrategie vererbt noch lokal definiert. Die dritte oder vierte Regel ist genau dann gültig, wenn in einer der beiden Autorisierungssphären ein Verbot gilt und die andere zu keiner Erlaubnis kommt.

**Regeln 7.9**

$$\mathsf{grant}(\mathsf{AS}, s, app, o, +a) \leftarrow \mathsf{local}(as_1, s, app, o, +a)\&\mathsf{local}(as_2, s, app, o, -a)\&\texttt{Scope}$$
$$\mathsf{grant}(\mathsf{AS}, s, app, o, +a) \leftarrow \mathsf{local}(as_1, s, app, o, -a)\&\mathsf{local}(as_2, s, app, o, +a)\&\texttt{Scope}$$

Wenn der Regelsatz 7.9 verwendet wird, wird durch die Entscheidungsregeln nur dann eine Erlaubnis spezifiziert, wenn in der einen Autorisierungssphäre eine Erlaubnis vorliegt und in der anderen Autorisierungssphäre ein Verbot. Falls in mindestens einer der beiden Autorisierungssphären kein Ergebnis vorliegt, ist keine der beiden Regeln gültig[12].

### Autorisierungssphäre bevorzugen

Eine weitere Strategie bei der Auflösung von Koordinierungskonflikten ist es, einer der beiden Autorisierungssphären den Vorzug zu geben. Dies wird in den beiden folgenden Regelsätzen umgesetzt, wobei im Regelsatz 7.11 nur dann eine Entscheidung spezifiziert wird, wenn in beiden Autorisierungssphären ein Ergebnis vorliegt. Welcher Autorisierungssphäre der Vorzug gegeben wird, kann zum einen direkt bestimmt werden, indem $as_1$ durch eine konkrete Bezeichnung ersetzt wird. Zum anderen kann $as_1$ auch als Variable verwendet werden, deren Belegung durch die Feature Prädikate $\mathcal{F}$ im `Scope` festgelegt wird.

[11] Falls die zweite Autorisierungssphäre ebenfalls eine Erlaubnis ausspricht, wird die Entscheidungsregel erst gar nicht aufgerufen (siehe erste Regel aus Regelsatz 8.2 auf Seite 118).

[12] Damit kommt es dann zur Defaultauflösung, die im Regelsatz 8.5 auf Seite 119 spezifiziert ist.

**Regeln 7.10**

$$\mathsf{grant}(\mathsf{AS}, s, app, o, +a) \leftarrow \mathsf{local}(as_1, s, app, o, +a)\&\mathtt{Scope}$$
$$\mathsf{grant}(\mathsf{AS}, s, app, o, -a) \leftarrow \mathsf{local}(as_1, s, app, o, -a)\&\mathtt{Scope}$$
$$\mathsf{grant}(\mathsf{AS}, s, app, o, +a) \leftarrow \mathsf{local}(as_2, s, app, o, +a)\&\neg\mathsf{local}(as_1, s, app, o, -a)\&\mathtt{Scope}$$
$$\mathsf{grant}(\mathsf{AS}, s, app, o, -a) \leftarrow \mathsf{local}(as_2, s, app, o, -a)\&\neg\mathsf{local}(as_1, s, app, o, +a)\&\mathtt{Scope}$$

In der dritten und vierten Regel kann das Prädikat $\neg\mathsf{local}(as_1, s, app, o, +a)$ bzw. das Prädikat $\neg\mathsf{local}(as_1, s, app, o, -a)$ deshalb ausgelassen werden, weil ansonsten gar kein Koordinierungskonflikt vorläge und die Entscheidungsregeln gar nicht aufgerufen würden.

**Regeln 7.11**

$$\mathsf{grant}(\mathsf{AS}, s, app, o, +a) \leftarrow \mathsf{local}(as_1, s, app, o, +a)\&\mathsf{local}(as_2, s, app, o, -a)\&\mathtt{Scope}$$
$$\mathsf{grant}(\mathsf{AS}, s, app, o, -a) \leftarrow \mathsf{local}(as_1, s, app, o, -a)\&\mathsf{local}(as_2, s, app, o, +a)\&\mathtt{Scope}$$

Die obigen Regeln entsprechen vom Aufbau dem Regelsatz 7.9, nur dass hier die Entscheidungen der Autorisierungssphäre $as_1$ immer der Vorzug gegeben wird.

**Weitere Autorisierungssphäre hinzuziehen**

Neben diesen Strategien, die nur die Entscheidungen aus der Autorisierungssphäre der Rolle und des Objektes miteinbeziehen, ist eine weitere Möglichkeit zur Auflösung von Koordinierungskonflikten, eine zusätzliche Autorisierungssphäre $as_3$ hinzuzuziehen. Die Entscheidung wird dann abhängig von dieser Autorisierungssphäre getroffen, wobei nur auf lokale Zugriffsstrategien und an diese Autorisierungssphäre vererbte Zugriffsstrategien zurückgegriffen werden kann.

Die beiden folgenden Regelsätze unterscheiden sich darin, dass beim zweiten Regelsatz nur im Konfliktfall die dritte Autorisierungssphäre hinzugezogen wird, während beim Ersten eine der beiden Autorisierungssphären auch undefiniert sein kann.
Zu bemerken ist noch, dass durch den Regelsatz 7.12 eine Entscheidung der zusätzlichen Autorisierungssphäre, die nicht mit mindestens einer der beiden anderen Autorisierungssphären übereinstimmt, nicht übernommen wird. Wenn also beispielsweise die Autorisierungssphäre $as_1$ ein Verbot ausspricht und keine Entscheidung von $as_2$ vorliegt, führt eine Erlaubnis durch $as_3$ dazu, dass keine der unteren Entscheidungsregeln greift. Sollte dies jedoch gewünscht sein, müsste der Regelsatz anders spezifiziert werden. In diesem Fall liefe es darauf hinaus, dass immer die Entscheidungen der dritten Autorisierungssphäre übernommen werden[13].

**Regeln 7.12**

$$\begin{aligned}\mathsf{grant}(\mathsf{AS}, s, app, o, +a) \leftarrow{}& \mathsf{local}(as_1, s, app, o, +a)\&\neg\mathsf{local}(as_2, s, app, o, +a)\&\\ & \mathsf{local}(as_3, s, app, o, +a)\&\mathtt{Scope}\\ \mathsf{grant}(\mathsf{AS}, s, app, o, +a) \leftarrow{}& \neg\mathsf{local}(as_1, s, app, o, +a)\&\mathsf{local}(as_2, s, app, o, +a)\&\\ & \mathsf{local}(as_3, s, app, o, +a)\&\mathtt{Scope}\\ \mathsf{grant}(\mathsf{AS}, s, app, o, -a) \leftarrow{}& \mathsf{local}(as_1, s, app, o, -a)\&\neg\mathsf{local}(as_2, s, app, o, -a)\&\\ & \mathsf{local}(as_3, s, app, o, -a)\&\mathtt{Scope}\\ \mathsf{grant}(\mathsf{AS}, s, app, o, -a) \leftarrow{}& \neg\mathsf{local}(as_1, s, app, o, -a)\&\mathsf{local}(as_2, s, app, o, -a)\&\\ & \mathsf{local}(as_3, s, app, o, -a)\&\mathtt{Scope}\end{aligned}$$

[13] Wenn auch bei einer undefinierten Entscheidung der beiden anderen Autorisierungssphären ein Ergebnis geliefert werden soll, sieht der Regelsatz dann wie folgt aus:

$$\mathsf{grant}(\mathsf{AS}, s, app, o, +a) \leftarrow \mathsf{local}(as_3, s, app, o, +a)\&\mathsf{Scope}$$
$$\mathsf{grant}(\mathsf{AS}, s, app, o, -a) \leftarrow \mathsf{local}(as_3, s, app, o, -a)\&\mathsf{Scope}$$

Die erste oder dritte Regel ist genau dann gültig, wenn die Autorisierungssphären $as_1$ und $as_3$ zum selben Ergebnis kommen. Wenn die Ergebnisse von $as_2$ und $as_3$ übereinstimmen, ist die zweite oder vierte Regel gültig.

Beim folgenden Regelsatz werden die Ergebnisse der dritten Autorisierungssphäre nur dann übernommen, wenn ein Koordinierungskonflikt vorliegt. Falls eine der beiden Autorisierungssphären zu keinem Ergebnis kommt, gibt es auch keine gültige Entscheidungsregel.

**Regeln 7.13**

$$\begin{aligned}
\mathsf{grant}(\mathsf{AS}, s, app, o, +a) \leftarrow & \mathsf{local}(as_1, s, app, o, +a) \& \mathsf{local}(as_2, s, app, o, -a) \& \\
& \mathsf{local}(as_3, s, app, o, +a) \& \mathit{Scope} \\
\mathsf{grant}(\mathsf{AS}, s, app, o, +a) \leftarrow & \mathsf{local}(as_1, s, app, o, -a) \& \mathsf{local}(as_2, s, app, o, +a) \& \\
& \mathsf{local}(as_3, s, app, o, +a) \& \mathit{Scope} \\
\mathsf{grant}(\mathsf{AS}, s, app, o, -a) \leftarrow & \mathsf{local}(as_1, s, app, o, +a) \& \mathsf{local}(as_2, s, app, o, -a) \& \\
& \mathsf{local}(as_3, s, app, o, -a) \& \mathit{Scope} \\
\mathsf{grant}(\mathsf{AS}, s, app, o, -a) \leftarrow & \mathsf{local}(as_1, s, app, o, -a) \& \mathsf{local}(as_2, s, app, o, +a) \& \\
& \mathsf{local}(as_3, s, app, o, -a) \& \mathit{Scope}
\end{aligned}$$

Dies sind einige Beispiele für ein strukturiertes Vorgehen bei der Koordinierung von Zugriffsstrategien. Die Ansätze hierzu stammen aus den grundlegenden Arbeiten zu Kompositionsoperatoren [BVS00, BVS02]. In diesen wurden allerdings nur geschlossene Autorisierungssysteme betrachtet. Da in dem vorliegenden Ansatz auch hybride Zugriffsprivilegien spezifiziert werden können, mussten die Auflösungsstrategien entsprechend erweitert werden. Ein Ausbau dieser Auflösungsstrategien zu hybriden Kompositionsoperatoren kann Teil der weiterführenden Arbeiten sein.

# Kapitel 8

# Dezentrale Auswertung von Zugriffsanfragen

Der vorgestellte Ansatz ermöglicht eine vollständige dezentrale Auswertung der Zugriffsstrategien. Dabei werden nur die Regeln der direkt "betroffenen" Autorisierungssphären und die geerbten Regeln übergeordnete Autorisierungssphären ausgewertet. Im Falle einer Zugriffsanfrage zwischen Autorisierungssphären müssen noch die Entscheidungsregeln der verantwortlichen Autorisierungssphäre ausgewertet werden.

Neben den lokalen Zugriffsstrategien, den Vererbungsregeln und Entscheidungsregeln, die von den Autorisierungsteams der entsprechenden Autorisierungssphären spezifiziert werden, sind zur Umsetzung der dezentralen Autorisierung weitere Prädikate definiert. Mittels der Regeln, die mit diesen Prädikaten spezifiziert werden, z.B. **Erbregeln** oder **Koordinierungsregeln**, wird sichergestellt, dass die festgelegten Abläufe wie Vererbungsprinzip oder beim Zugriff zwischen Autorisierungssphären, nicht durch Regelspezifikationen der Autorisierungsteams unterlaufen werden können. Die korrekte Umsetzung der Vererbung und Koordinierung von Zugriffen durch die entsprechenden Datalog–Regeln wurde in den Abschnitten 6.4.3 bzw. 7.2.1 und 7.2.2 nachgewiesen. Durch die Einführung dieser zusätzlichen Prädikate und der Spezifizierung der dazugehörigen Regeln wird die gesamte Autorisierung einheitlich mittels Datalog modelliert, sodass keine detaillierten Annahmen über die Implementierung des Autorisierungssystems (z.B. in Java) gemacht werden müssen. Vom Datalog–Interpreter wird angenommen, dass er auf die mittels Feature Prädikaten und Kontext Feature Prädikaten abfragbaren Fakten effizient zurückgreifen kann[1]. Außerdem müssen vom Interpreter zusätzliche Prädikate zur Konsistenzprüfung und zur Erkennung von Vererbungskonflikten ausgewertet werden[2].

Im Abschnitt 8.1 wird gezeigt, wie am entsprechenden Host einer Autorisierungssphäre mit der Auswertung begonnen wird. Dabei wird zunächst festgestellt, ob bei der vorliegenden Zugriffsanfrage die Autorisierungssphären der Rolle und Objektes identisch sind. In diesem Fall kann die Anfrage über die Regeln zum lokalen Einstieg direkt entschieden werden. Dabei wird davon ausgegangen, dass diese Anfragen an den für die Autorisierung zuständigen Host innerhalb dieser Autorisierungssphäre gestellt werden.

Bei Zugriffen zwischen Autorisierungssphären sind die entsprechenden Hosts der Autorisierungssphäre der Rolle und des Objektes involviert. Dabei kann angenommen werden, dass zunächst die Zugriffsanfrage in der Autorisierungssphäre der Rolle ausgewertet wird. Anschließend wird die Zugriffsanfrage zusammen mit dem lokalem Ergebnis d.h. der Auswertung des local() Prädikates — *unabhängig* ob eine explizite Erlaubnis, Verbot oder keine Entscheidung lokal vorliegt — an die Autorisierungssphäre des Objektes weitergeleitet. Das dortige lokale Ergebnis kann dann wieder

[1] Ein erster Prototyp wurde bereits in [Hal05] vorgestellt.

[2] Die genauen Details sind bei der Erläuterung der Konsistenzprüfung in den Abschnitten 5.2, 6.2.2 und 7.3.1 sowie bei der Erkennung von Vererbungskonflikten im Abschnitt 6.4.2 gegeben worden.

an die erste Autorisierungssphäre zurückgeschickt werden. Durch die Definition der Prädikate im folgenden Abschnitt wird sichergestellt, dass bei beiden Auswertungen das gleiche Gesamtergebnis eintritt. Dabei wird bei unterschiedlichen Entscheidungen der beiden Autorisierungssphären die koordinierende Autorisierungssphäre durch Auswertung des Prädikates coordinate() hinzugezogen.

Es ist davon auszugehen, dass eine inkonsistente d.h. nicht aktuelle Replizierung, bei einer Zugriffsanfrage ohne größeren Aufwand erkannt werden kann und im weiteren Verlauf aufgehoben wird[3].

Die Verwendung von Negationen ist in Datalog–Programmen nicht unproblematisch. Durch eine Schichtung der Prädikate kann allerdings eine deterministische Auswertung garantiert werden (vgl. Abschnitt 2.3). Aufgrund der notwendigen Definition der inherit() Prädikate und der entsprechenden Erbregeln in den Regelsätzen 6.11 und 6.12 ist diese Schichtung zunächst nicht gegeben. Im Abschnitt 8.2 wird gezeigt, wie die Regeln so umgewandelt werden können, dass ein geschichtetes Datalog–Programm vorliegt, bei dem Negationen problemlos ausgewertet werden können.

## 8.1 Beginn einer Zugriffsanfrage

In dem vorliegenden Ansatz soll die gesamte Zugriffsanfrage einschließlich der dezentralen Auswertung der Zugriffsanfragen mittels einer Datalog–Sprache spezifiziert werden. Nachdem alle Prädikate in einer Button–Up Reihenfolge definiert sind, fehlt noch das oberste Prädikat, das die Schnittstelle zwischen dem übrigen Autorisierungssystem, das die Koppelung an das Betriebssystem realisiert und den Definitionen der Zugriffsstrategien mittels einer Datalog–Sprache realisiert, darstellt. Der Datalog–Interpreter wird zur Auswertung der Regeln, die mit diesem Prädikat fest spezifiziert sind, mit einer konkreten Belegung für ein Subjekt $s$, das durch die Applikation *app* auf ein Objekt $o$ mit der Methode $a$ zugreifen will, aufgerufen. Basierend auf den Informationen, die mittels Feature Prädikaten und Kontext Feature Prädikaten abgerufen werden, werden die darauf aufbauenden Regeln ausgewertet. Die Zugriffsanfrage wird abhängig davon, ob eine Regel mit query(s, app, o, +a) oder query(s, app, o, -a) gültig ist, genehmigt oder verboten. Durch die Regelsätze 8.1 bis 8.5 wird sichergestellt, dass eines der beiden Prädikate stets gültig ist.

**Definition 8.1 (Anfrage)**
Die Anfrageregel hat die Form

$$\textbf{query(}s, app, o, \langle sign \rangle a\textbf{)} \leftarrow L_1 \& \ldots \& L_n$$

Falls alle Prädikate $L_i \in \mathcal{F} \cup \{local, coordinate\}$ (mit festem $n \in \mathbb{N}$ und $1 \leq i < n$) mit einer entsprechenden Belegung für $s \in \mathcal{V}_{\mathcal{S}}$, $app \in \mathcal{V}_{\mathcal{AP}}$ und $o \in \mathcal{V}_{\mathcal{O}}$ gültig sind, wird der Zugriff mit Methode $a \in AC$ gemäß $\langle sign \rangle$ explizit gewährt oder verboten. Alle Regeln vom Typ local und coordinate können aus beliebigen Autorisierungssphären $as \in \mathcal{AS}$ sein.

Dieses Prädikat enthält deshalb als Parameter keine Autorisierungssphäre, da die damit spezifizierten Regeln nicht unabhängig definiert werden, sondern organisationsweit identisch sind. Dies bedeutet nicht, dass alle Anfragen zu einem einzigen Host geschickt werden, der somit eine zentrale Instanz darstellen würde. Sondern die Regeln können an den Knoten, der zur Bearbeitung von Zugriffsanfragen einer Autorisierungssphäre vorgesehen ist, ausgewertet werden. Falls die Auswertung der Zugriffsanfragen in einer Autorisierungssphäre auf mehrere Knoten verteilt wird, müssen alle Regelsätze dort auch repliziert werden. Es wird dabei angenommen, dass, wie in den Kapiteln 6 und 7 beschrieben, die Erb- und Vererbungsregeln sowie die Entscheidungs- und Koordinierungsregeln und gegebenenfalls die lokalen Zugriffsregeln[4], sowohl an der definierenden Autorisierungssphäre als auch an allen untergeordneten Autorisierungssphären durch Replizierung

[3] Weitere Details über die Konsistenz der jeweiligen Replizierung werden im Abschnitt 8.1 dargestellt.

[4] Die lokalen Zugriffsregeln müssen nur dann an untergeordnete Autorisierungssphären repliziert werden, wenn wie in Abschnitt 6.2 erläutert, in den Erbregeln auf diese zurückgegriffen wird.

zur Verfügung stehen[5]. Bei einem Zugriff zwischen verschiedenen Autorisierungssphären wird angenommen, dass die Auswertung zunächst bei der Autorisierungssphäre der Rolle vorgenommen wird und die Auswertung der Prädikate $\mathsf{local}(as_o, s, app, o, +a)$ und $\mathsf{local}(as_o, s, app, o, -a)$ an die zuständige Autorisierungssphäre des Objektes geschickt wird. Sobald bekannt ist, ob eines dieser Prädikate gültig ist, kann die Zugriffsanfrage, gegebenenfalls durch Auswertung der Koordinierungsregeln der verantwortlichen Autorisierungssphäre, abgeschlossen werden. Falls die Autorisierungssphäre des Objektes dem Gesamtergebnis nicht vertraut, kann dort ebenfalls eine Zugriffsanfrage für die gleiche Belegung durchgeführt werden. Dann werden entsprechend die Auswertungen der Prädikate $\mathsf{local}(as_r, s, app, o, +a)$ und $\mathsf{local}(as_r, s, app, o, -a)$ an die Autorisierungssphäre der Rolle geschickt. Das Gesamtergebnis wird dasselbe sein, da angenommen werden kann, dass eine Inkonsistenz bei den replizierten Regelsätzen bei den Anfragen zur Auswertung der $\mathsf{local}$ Prädikate entdeckt werden kann.

Im folgenden Abschnitt werden zunächst die Regeln definiert, die bei einer Zugriffsanfrage zur Geltung kommen, wenn sowohl aktive Rolle und Objekt in der gleichen Autorisierungssphäre liegen. Anschließend wird im Abschnitt 8.1.2 ein Regelsatz spezifiziert, der bei einem Zugriff zwischen zwei Autorisierungssphären genutzt wird und bei einem Koordinierungskonflikt dafür sorgt, dass die Entscheidungsregeln der verantwortlichen Autorisierungssphäre berücksichtigt werden.

### 8.1.1 Anfragen innerhalb einer Autorisierungssphäre

Wenn die Organisationseinheit, in der die Rolle des Subjektes aktiviert ist, und die Organisationseinheit des Objektes zur gleichen Autorisierungssphäre gehören, kann die Zugriffsanfrage am entsprechenden Knoten ausgewertet werden. Es muss lediglich überprüft werden, ob von einer übergeordneten Autorisierungssphäre Zugriffsstrategien geerbt werden, die auf die vorliegende Anfrage passen.

**Regeln 8.1**

$$\begin{aligned}
\mathsf{query}(s, app, o, +a) \leftarrow\ & \mathsf{activ_role}(s, u_r, r)\&\mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_r, as)\&\mathsf{sphereof}(u_o, as)\& \\
& \mathsf{local}(as, s, app, o, +a) \\
\mathsf{query}(s, app, o, -a) \leftarrow\ & \mathsf{activ_role}(s, u_r, r)\&\mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_r, as)\&\mathsf{sphereof}(u_o, as)\& \\
& \mathsf{local}(as, s, app, o, -a) \\
\mathsf{query}(s, app, o, -a) \leftarrow\ & \mathsf{activ_role}(s, u_r, r)\&\mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_r, as)\&\mathsf{sphereof}(u_o, as)\& \\
& \neg\mathsf{local}(as, s, app, o, +a)\&\neg\mathsf{local}(as, s, app, o, -a)
\end{aligned}$$

Durch die Belegung der Variable $as$ wird sichergestellt, dass die Autorisierungssphäre, die für das Objekt zuständig ist, mit der identisch ist, in der Autorisierungssphäre auch die aktive Rolle des Benutzers liegt. Danach werden dann die $\mathsf{local}$ Regeln dieser Autorisierungssphäre ausgewertet. Wobei dort gemäß Regelsatz 7.5 zunächst überprüft wird, ob eine vererbte Zugriffsstrategie zutreffend ist.

Die dritte Regel aus Regelsatz 8.1 sorgt dafür, dass bei einer Anfrage von Subjekt $s$, das durch eine Applikation $app$ mittels einer Zugriffsmethode $a$ auf Objekt $o$ zugreifen will, zu der es keine Zugriffsstrategie in der entsprechenden Autorisierungssphäre $as$ gibt, diese Anfrage verweigert wird. Von einer Änderung des Vorzeichens im Kopf der letzten Regel in $+a$ ist abzusehen, da dies als Schlupfloch genutzt werden kann, um auf entsprechende Objekte zuzugreifen.

### 8.1.2 Anfragen zwischen Autorisierungssphären

Wenn für die Organisationseinheit der aktiven Rolle des Subjektes eine andere Autorisierungssphäre verantwortlich ist, als für die Organisationseinheit zu der das Objekt gehört, ist keine der

[5] An Stelle einer Replizierung wäre auch ein versenden einer entsprechenden Anfrage an den zuständigen Knoten möglich. Dies würde aber gerade bei der Auswertung der Vererbungsregeln aus den Regelsätzen 6.11 und 6.12 zu einem hohen Datenverkehr führen.

obigen Regeln aus Regelsatz 8.1 gültig, da es keine gültige Belegung für $as$ gibt. In den folgenden Regeln wird durch die Feature Prädikate im Teilausdruck $\mathsf{activ\_role}(s, r, u_r)\&\ \mathsf{sphereof}(u_r, as_r)\&\ \mathsf{unitof}(o, u_o)\&\ \mathsf{sphereof}(u_o, as_o)\&\ \neg\mathsf{equal\_sphere}(as_r, as_o)$ stets überprüft, dass es sich um zwei verschiedene Autorisierungssphären handelt. Damit wird sichergestellt, dass keine der Regeln aus Regelsatz 8.1 gültig sein kann.

Bei Zugriffen zwischen Autorisierungssphären werden die Zugriffsstrategien der beiden beteiligten Autorisierungssphären ausgewertet. Dabei wird durch die Auswertung des Prädikates $\mathsf{local}()$ überprüft, ob eine geerbte Regel für die Zugriffsanfrage zutreffend ist, oder eine lokale Autorisierungssphäre.
Damit bei einer Zugriffsanfrage der Interpreter ein gültiges Ergebnis liefern kann, müssen die folgenden Regelsätzen 8.2 bis 8.5 alle möglichen Kombinationen von Erlaubnissen und Verboten abdecken, die von der Autorisierungssphäre der Rolle und des Objektes, sowie im Falle eines Koordinierungskonfliktes zusätzlich noch von der koordinierenden Autorisierungssphäre, definiert werden können.

Der einfachste Fall ist im Regelsatz 8.2 abgedeckt, bei die Autorisierungssphären der Rolle und des Objektes zum gleichen Ergebnis kommen.

**Regeln 8.2**

$$\begin{aligned}
\mathsf{query}(s, app, o, +a) \leftarrow\ & \mathsf{activ_role}(s, r, u_r)\&\mathsf{sphereof}(u_r, as_r)\&\mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_o, as_o)\& \\
& \neg\mathsf{equal_sphere}(as_r, as_o)\&\mathsf{local}(as_r, s, o, +a)\&\mathsf{local}(as_o, s, o, +a) \\
\mathsf{query}(s, app, o, -a) \leftarrow\ & \mathsf{activ_role}(s, r, u_r)\&\mathsf{sphereof}(u_r, as_r)\&\mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_o, as_o)\& \\
& \neg\mathsf{equal_sphere}(as_r, as_o)\&\mathsf{local}(as_r, s, o, -a)\&\mathsf{local}(as_o, s, o, -a)
\end{aligned}$$

Eine der Regeln im Regelsatz 8.3 ist dann gültig, wenn die Autorisierungssphäre der Rolle und des Objektes nicht zum gleichen Ergebnis kommen, jedoch eine der beiden Autorisierungssphären mit der Entscheidung der koordinierenden Autorisierungssphäre übereinstimmt. Dieses wird dann entsprechend als Ergebnis der Zugriffsanfrage übernommen. Wie mit Satz 7.2 auf Seite 106 gezeigt wurde, wird das Prädikat $\mathsf{coordinate}(AS_{TOP}, s, app, o, \pm a)$ gemäß der geerbten Zugriffsstrategien der koordinierenden Autorisierungssphäre oder, falls es nicht geerbt wird, gemäß deren Entscheidungsregeln gültig.

**Regeln 8.3**

$$\begin{aligned}
\mathsf{query}(s, app, o, +a) \leftarrow\ & \mathsf{activ_role}(s, r, u_r)\&\mathsf{sphereof}(u_r, as_r)\&\mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_o, as_o)\& \\
& \neg\mathsf{equal_sphere}(as_r, as_o)\&\neg\mathsf{local}(as_r, s, o, +a)\&\mathsf{local}(as_o, s, o, +a)\& \\
& \mathsf{coordinate}(AS_{TOP}, s, app, o, +a) \\
\mathsf{query}(s, app, o, +a) \leftarrow\ & \mathsf{activ_role}(s, r, u_r)\&\mathsf{sphereof}(u_r, as_r)\&\mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_o, as_o)\& \\
& \neg\mathsf{equal_sphere}(as_r, as_o)\&\mathsf{local}(as_r, s, o, +a)\&\neg\mathsf{local}(as_o, s, o, +a)\& \\
& \mathsf{coordinate}(AS_{TOP}, s, app, o, +a) \\
\mathsf{query}(s, app, o, -a) \leftarrow\ & \mathsf{activ_role}(s, r, u_r)\&\mathsf{sphereof}(u_r, as_r)\&\mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_o, as_o)\& \\
& \neg\mathsf{equal_sphere}(as_r, as_o)\&\neg\mathsf{local}(as_r, s, o, -a)\&\mathsf{local}(as_o, s, o, -a)\& \\
& \mathsf{coordinate}(AS_{TOP}, s, app, o, -a) \\
\mathsf{query}(s, app, o, -a) \leftarrow\ & \mathsf{activ_role}(s, r, u_r)\&\mathsf{sphereof}(u_r, as_r)\&\mathsf{unitof}(o, u_o)\&\mathsf{sphereof}(u_o, as_o)\& \\
& \neg\mathsf{equal_sphere}(as_r, as_o)\&\mathsf{local}(as_r, s, o, -a)\&\neg\mathsf{local}(as_o, s, o, -a)\& \\
& \mathsf{coordinate}(AS_{TOP}, s, app, o, -a)
\end{aligned}$$

Der Regelsatz 8.4 deckt die Fälle ab, in denen mindestens in einer der beiden Autorisierungssphären der Rolle oder des Subjektes kein Ergebnis vorliegt und die andere Autorisierungssphäre nicht mit dem Ergebnis der koordinierenden Autorisierungssphäre übereinstimmt. In diesem Fall entscheidet dann die koordinierende Autorisierungssphäre über die Zugriffsanfrage.

**Regeln 8.4**

$$\begin{aligned}
\mathsf{query}(s, app, o, +a) \leftarrow & \mathsf{activ_role}(s, r, u_r) \& \mathsf{sphereof}(u_r, as_r) \& \mathsf{unitof}(o, u_o) \& \mathsf{sphereof}(u_o, as_o) \& \\
& \neg\mathsf{equal_sphere}(as_r, as_o) \& \neg\mathsf{local}(as_r, s, o, +a) \& \neg\mathsf{local}(as_r, s, o, -a) \& \\
& \neg\mathsf{local}(as_o, s, o, +a) \& \mathsf{coordinate}(AS_{TOP}, s, app, o, +a) \\
\mathsf{query}(s, app, o, +a) \leftarrow & \mathsf{activ_role}(s, r, u_r) \& \mathsf{sphereof}(u_r, as_r) \& \mathsf{unitof}(o, u_o) \& \mathsf{sphereof}(u_o, as_o) \& \\
& \neg\mathsf{equal_sphere}(as_r, as_o) \& \neg\mathsf{local}(as_o, s, o, +a) \& \neg\mathsf{local}(as_o, s, o, -a) \& \\
& \neg\mathsf{local}(as_r, s, o, +a) \& \mathsf{coordinate}(AS_{TOP}, s, app, o, +a) \\
\mathsf{query}(s, app, o, -a) \leftarrow & \mathsf{activ_role}(s, r, u_r) \& \mathsf{sphereof}(u_r, as_r) \& \mathsf{unitof}(o, u_o) \& \mathsf{sphereof}(u_o, as_o) \& \\
& \neg\mathsf{equal_sphere}(as_r, as_o) \& \neg\mathsf{local}(as_r, s, o, +a) \& \neg\mathsf{local}(as_r, s, o, -a) \& \\
& \neg\mathsf{local}(as_o, s, o, -a) \& \mathsf{coordinate}(AS_{TOP}, s, app, o, -a) \\
\mathsf{query}(s, app, o, +a) \leftarrow & \mathsf{activ_role}(s, r, u_r) \& \mathsf{sphereof}(u_r, as_r) \& \mathsf{unitof}(o, u_o) \& \mathsf{sphereof}(u_o, as_o) \& \\
& \neg\mathsf{equal_sphere}(as_r, as_o) \& \neg\mathsf{local}(as_o, s, o, +a) \& \neg\mathsf{local}(as_o, s, o, -a) \& \\
& \neg\mathsf{local}(as_r, s, o, -a) \& \mathsf{coordinate}(AS_{TOP}, s, app, o, -a)
\end{aligned}$$

Falls die Autorisierungssphären der Rolle und des Objektes beide zu keinem Ergebnis kommen, sind abhängig vom Ergebnis der Autorisierungssphäre die beiden ersten oder die beiden letzten Regeln gleichzeitig gültig. Da sich die entsprechenden Prädikate im Kopf nicht unterscheiden, ist dies nicht von Bedeutung.

Falls die koordinierende Autorisierungssphäre jedoch zu keinem Ergebnis bezüglich der Zugriffsanfrage $s, app, o, a$ kommt, wird diese im Falle eines Koordinierungskonfliktes durch den Regelsatz 8.5 grundsätzlich abgelehnt[6]. Mit der letzten Regel wird sichergestellt, dass eine Zugriffsanfrage auch dann abgelehnt wird, wenn keine der drei Autorisierungssphären zu einem Ergebnis kommt.

**Regeln 8.5**

$$\begin{aligned}
\mathsf{query}(s, app, o, -a) \leftarrow & \mathsf{activ_role}(s, r, u_r) \& \mathsf{sphereof}(u_r, as_r) \& \mathsf{unitof}(o, u_o) \& \mathsf{sphereof}(u_o, as_o) \& \\
& \neg\mathsf{equal_sphere}(as_r, as_o) \& \mathsf{local}(as_r, s, o, +a) \& \neg\mathsf{local}(as_o, s, o, +a) \& \\
& \neg\mathsf{coordinate}(AS_{TOP}, s, app, o, +a) \& \neg\mathsf{coordinate}(AS_{TOP}, s, app, o, -a) \\
\mathsf{query}(s, app, o, -a) \leftarrow & \mathsf{activ_role}(s, r, u_r) \& \mathsf{sphereof}(u_r, as_r) \& \mathsf{unitof}(o, u_o) \& \mathsf{sphereof}(u_o, as_o) \& \\
& \neg\mathsf{equal_sphere}(as_r, as_o) \& \mathsf{local}(as_r, s, o, -a) \& \neg\mathsf{local}(as_o, s, o, -a) \& \\
& \neg\mathsf{coordinate}(AS_{TOP}, s, app, o, +a) \& \neg\mathsf{coordinate}(AS_{TOP}, s, app, o, -a) \\
\mathsf{query}(s, app, o, -a) \leftarrow & \mathsf{activ_role}(s, r, u_r) \& \mathsf{sphereof}(u_r, as_r) \& \mathsf{unitof}(o, u_o) \& \mathsf{sphereof}(u_o, as_o) \& \\
& \neg\mathsf{equal_sphere}(as_r, as_o) \& \neg\mathsf{local}(as_r, s, o, +a) \& \mathsf{local}(as_o, s, o, +a) \& \\
& \neg\mathsf{coordinate}(AS_{TOP}, s, app, o, +a) \& \neg\mathsf{coordinate}(AS_{TOP}, s, app, o, -a) \\
\mathsf{query}(s, app, o, -a) \leftarrow & \mathsf{activ_role}(s, r, u_r) \& \mathsf{sphereof}(u_r, as_r) \& \mathsf{unitof}(o, u_o) \& \mathsf{sphereof}(u_o, as_o) \& \\
& \neg\mathsf{equal_sphere}(as_r, as_o) \& \neg\mathsf{local}(as_r, s, o, -a) \& \mathsf{local}(as_o, s, o, -a) \& \\
& \neg\mathsf{coordinate}(AS_{TOP}, s, app, o, +a) \& \neg\mathsf{coordinate}(AS_{TOP}, s, app, o, -a) \\
\mathsf{query}(s, app, o, -a) \leftarrow & \mathsf{activ_role}(s, r, u_r) \& \mathsf{sphereof}(u_r, as_r) \& \mathsf{unitof}(o, u_o) \& \mathsf{sphereof}(u_o, as_o) \& \\
& \neg\mathsf{equal_sphere}(as_r, as_o) \& \neg\mathsf{local}(as_r, s, o, +a) \& \neg\mathsf{local}(as_r, s, o, -a) \& \\
& \neg\mathsf{local}(as_o, s, o, +a) \& \neg\mathsf{local}(as_o, s, o, -a) \& \\
& \neg\mathsf{coordinate}(AS_{TOP}, s, app, o, +a) \& \neg\mathsf{coordinate}(AS_{TOP}, s, app, o, -a)
\end{aligned}$$

Die Regelsätze 8.2 bis 8.5 erscheinen recht umfangreich, jedoch muss beachtet werden, dass stets die gleichen Prädikate ausgewertet werden. Ob bei einer Zugriffsanfrage $\mathsf{local}(as_r, s, o, +a)$

[6]Eine alternative Vorgehensweise ist durch Änderung des Regelsatzes modellierbar. So könnte für den Fall, dass kein Ergebnis in der koordinierenden Autorisierungssphäre vorliegt und nur ein Ergebnis aus der Autorisierungssphäre der Rolle oder des Objektes vorliegt, diese übernommen werden.

oder $\mathsf{local}(as_r, s, o, -a)$ bzw. $\mathsf{local}(as_o, s, o, +a)$ oder $\mathsf{local}(as_o, s, o, -a)$ gültig sind, kann nach der Auswertung des Regelsatzes 8.2 zwischengespeichert werden. Abhängig davon muss bei einem Koordinierungskonflikt "nur noch" geprüft werden, ob $\mathsf{coordinate}(AS_{TOP}, s, app, o, +a)$ oder $\mathsf{coordinate}(AS_{TOP}, s, app, o, -a)$ gültig ist. Danach kann ein Datalog Interpreter relativ einfach die passende Regel finden.

## 8.2 Geschichtetes Datalog Regelwerk

Die Verwendung von Negationen in Datalog–Programmen ist problematisch, da unter Umständen keine stabilen Ergebnisse geliefert werden können. Ein Ansatz dies zu gewährleisten ist, wie im Abschnitt 2.3 erläutert, Negationen nur bei Prädikaten einer niedrigeren Schicht zu verwenden. Die Übersicht in Abbildung 8.1 fasst zusammen, welche Prädikate von anderen aufgerufen werden können. Dabei wurden alle Feature Prädikate sowie Kontext Feature Prädikate und Kontext Trust Prädikate zur besseren Übersichtlichkeit nicht mit aufgenommen. Bei inherit Prädikaten (vgl. Def. 6.5 und die Regelsätze 6.11 und 6.12 zur Durchsetzung) findet ein Aufruf des gleichen Prädikates auch mit Negation statt. Diese Negationen sind zwingend notwendig, um festzustellen, ob keine Zugriffsregeln von einer übergeordneten Autorisierungssphäre vererbt werden. In den Regeln von coordinate werden zwar ebenfalls andere coordinate Prädikate aufgerufen, jedoch niemals mit einer Negation (vgl. Regelsatz 7.4). Daher müssen diese Regeln nicht besonders betrachtet werden. Gleiches gilt für das Prädikat responsible, da in den entsprechenden Regeln 7.1, 7.2 und 7.3 ebenfalls keine Negationen desselben Prädikates vorkommen. Eine Reihe von Feature Prädikaten enthalten andere Feature Prädikate, jedoch ergeben sich dadurch keine Kreise, sodass hierbei keine Probleme bei der Schichtung entstehen.

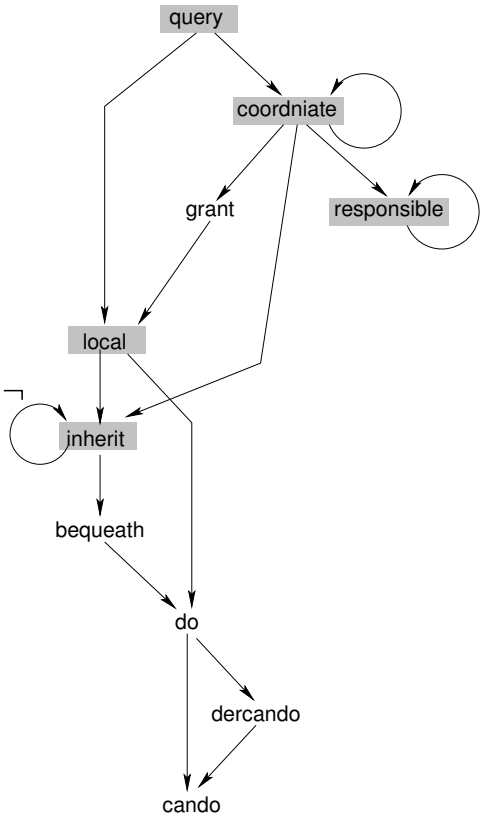



Abbildung 8.1: Schichtung der Prädikate

Die nötige Schichtung der Aufrufe kann unter Ausnutzung der Hierarchie der Autorisierungs-

sphären $<_{\mathcal{AS}}$ erreicht werden. Um ein geschichtetes Datalog–Programm zu erhalten, bei dem dann Negationen kein Problem sind, wird eine automatische Übersetzung vorgestellt, mit der die inherit Prädikate geschichtet werden. Dieses Vorgehen lehnt sich an eine Methode an, bei der eine Algebra in ein logisches Programm überführt wird und in [BVS00, S. 170ff] [BVS02, S.16ff] vorgestellt wird.

**Definition 8.2 (Übersetzung des Regelwerks)**
Ein Regelwerk $R$ wird in ein geschichtetes Datalog–Programm $P$ überführt, indem für jede Autorisierungssphäre mit der Bezeichnung $AS \in \mathcal{AS}$ ein eigenes Prädikat für die Erbregel mit dem Kopf $\mathsf{inherit}_{AS}$ definiert wird. Jedes Prädikat der Form $\mathsf{inherit}(AS, s, app, o, \langle sign \rangle a)$ (mit $s \in \mathcal{ST}, app \in \mathcal{APT}, o \in \mathcal{OT}, a \in \mathcal{AT}$) wird übersetzt nach $\mathsf{inherit}_{\mathsf{AS}}(AS, s, app, o, \langle sign \rangle a)$. Alle anderen Prädikate bleiben unverändert.

Im Regelsatz 6.12 auf Seite 88 wurden die Variablen $as_{h1}$ und $as_{h2}$ verwendet, allerdings wurde stets nur getestet, ob für die Belegung der Variable $as_{h1}$ ein gültiges Prädikat vorliegt. Nach der Übersetzung können keine Variablen mehr verwendet werden, daher müssen beide möglichen Belegungen als einzelne Regeln auftauchen. In allen Autorisierungssphären mit zwei direkten Vorgängern werden daher zusätzlich zum Regelsatz 6.12 die folgenden Regeln definiert und insbesondere bei der automatischen Übersetzung berücksichtigt.

Die Aussagen des Satzes 6.1 und der Lemmata 6.1 bis 6.9 bleiben auch mit dieser Ergänzung erhalten, da lediglich die beiden möglichen Belegungen explizit aufgeschrieben wurden. Wenn es, wie im Punkt 4 des Beweises von Lemma 6.5 erwähnt, bei beiden möglichen Belegungen der Variable eine gültige Regel gibt, werden nun nach der Übersetzung entsprechend zwei verschiedene Regeln gültig. Da diese den gleichen Kopf haben, hat es keine weitere Auswirkung auf die weitere Auswertung.

**Regeln 8.6 (Ergänzung der Erbregeln 6.12)**

$$\begin{aligned}
\mathsf{inherit}(AS, s, app, o, +a) \leftarrow\ & \mathsf{spherein}(AS, as_{h1}) \& \mathsf{spherein}(AS, as_{h2}) \& \neg \mathsf{equal_sphere}(as_{h1}, as_{h2}) \& \\
& \neg \mathsf{bequeath}(as_{h1}, s, app, o, -a) \& \neg \mathsf{inherit}(as_{h1}, s, app, o, -a) \& \\
& \mathsf{inherit}(as_{h2}, s, app, o, +a) \\
\mathsf{inherit}(AS, s, app, o, -a) \leftarrow\ & \mathsf{spherein}(AS, as_{h1}) \& \mathsf{spherein}(AS, as_{h2}) \& \neg \mathsf{equal_sphere}(as_{h1}, as_{h2}) \& \\
& \neg \mathsf{bequeath}(as_{h1}, s, app, o, +a) \& \neg \mathsf{inherit}(as_{h1}, s, app, o, +a) \& \\
& \mathsf{inherit}(as_{h2}, s, app, o, -a) \\
\mathsf{inherit}(AS, s, app, o, +a) \leftarrow\ & \mathsf{spherein}(AS, as_{h1}) \& \mathsf{spherein}(AS, as_{h2}) \& \neg \mathsf{equal_sphere}(as_{h1}, as_{h2}) \& \\
& \neg \mathsf{inherit}(as_{h1}, s, app, o, +a) \& \neg \mathsf{inherit}(as_{h1}, s, app, o, -a) \& \\
& \neg \mathsf{inherit}(as_{h2}, s, app, o, +a) \& \neg \mathsf{inherit}(as_{h2}, s, app, o, -a) \& \\
& \mathsf{bequeath}(as_{h2}, s, app, o, +a) \& \neg \mathsf{bequeath}(as_{h1}, s, app, o, -a) \\
\mathsf{inherit}(AS, s, app, o, -a) \leftarrow\ & \mathsf{spherein}(AS, as_{h1}) \& \mathsf{spherein}(AS, as_{h2}) \& \neg \mathsf{equal_sphere}(as_{h1}, as_{h2}) \& \\
& \neg \mathsf{inherit}(as_{h1}, s, app, o, +a) \& \neg \mathsf{inherit}(as_{h1}, s, app, o, -a) \& \\
& \neg \mathsf{inherit}(as_{h2}, s, app, o, +a) \& \neg \mathsf{inherit}(as_{h2}, s, app, o, -a) \& \\
& \mathsf{bequeath}(as_{h2}, s, app, o, -a) \& \neg \mathsf{bequeath}(as_{h1}, s, app, o, +a)
\end{aligned}$$

Als ein Nachteil ist einzuwenden, dass nun die Änderung der Autorisierungssphäre durch Einfügen eines neuen Elementes oder Veränderung der Relationen in $<_{\mathcal{AS}}$ eine erneute Übersetzung zur Folge hat. Neben den inherit() und conflict() Regeln sind davon nur die coordinate() und local() Regeln[7] betroffen. Die Übersetzung lässt sich durch eine Textersetzung und ein anschließendes Neueinlesen der Regeln durch den Interpreter realisieren. Sie kann offline und in jeder Autorisierungssphäre unabhängig durchgeführt werden. Die neu übersetzten Regeln können dann

[7] vgl. Regelsätze 6.11 und 6.12 auf Seite 88, Regelsatz 6.13 auf Seite 90 sowie Regelsatz 7.4 auf Seite 105 und Regelsatz 7.5 auf Seite 108

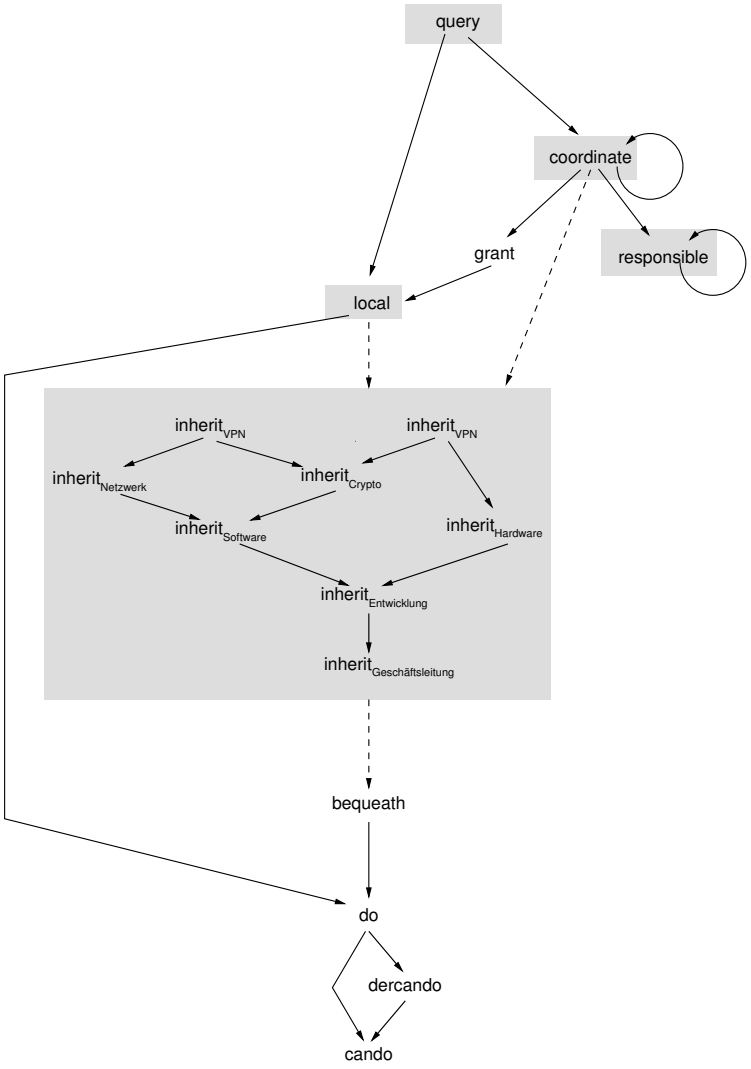



Abbildung 8.2: Ausschnitt aus der Schichtung der Prädikate im Datalog–Programm für die Beispielorganisation

an alle unter– und übergeordneten Autorisierungssphären repliziert werden. Bei Änderungen in der Hierarchie der Autorisierungssphären müssen nur die inherit() Regeln der untergeordneten Autorisierungssphären gegebenenfalls ersetzt werden, wenn sich die Anzahl der direkt übergeordneten Autorisierungssphären ändert[8]. Zusätzlich muss überprüft werden, an welchen Autorisierungssphären nun eine Replizierung durchgeführt werden muss, und es muss eine entsprechende Benachrichtigung durchgeführt werden. An allen untergeordneten Autorisierungssphären müssen die neuen Regeln bekannt gegeben werden. Dies kann auch noch weitere Regeln umfassen, die nun zum ersten Mal in diesen Autorisierungssphären repliziert werden.

Letztendlich lässt sich aus organisatorischer Sicht noch anmerken, dass solche Änderungen im Aufbau einer Organisation äußerst selten auftreten, oder im Falle einer durchgreifenden Umstrukturierung auf einen relativ kurzen Zeitraum beschränkt sind.

Das gemäß Definition 8.2 erzeugte Datalog–Programm $P$ ist geschichtet. Negationen des vor-

[8]Bei einer direkt übergeordneten Autorisierungssphäre wird der Regelsatz 6.11 verwendet. Bei zwei direkt übergeordneten Autorisierungssphären der Regelsatz 6.12 wie im Abschnitt 6.4.1 auf den Seiten 87ff erläutert wird.

mals gleichen inherit Prädikates beziehen sich aufgrund der Definition 6.5 nach der Übersetzung nur noch auf (übersetzte) inherit() Prädikate, die auf einer tieferen Schicht liegen. Dabei ist die Schichtung der übersetzten inherit() Prädikate eine Spiegelung der Hierarchie der Autorisierungssphären.

Für die Autorisierungssphären der Beispielorganisation aus Abbildung 3.2 ist die geschichtete Version der Datalog–Sprache in Abbildung 8.2 zu sehen. Dabei weist der direkte Pfeil, wie schon bei Abbildung 8.1 erläutert, auf die Prädikate, die im Rumpf enthalten sein können.
Der gestrichelte Pfeil von local auf den grau unterlegten Bereich der inherit Prädikate deutet an, dass von diesem Prädikat Pfeile zu allen Prädikaten in diesem Bereich eingezeichnet werden müssten. Ebenso müssten von allen Prädikaten aus diesem grau unterlegten Bereich Pfeile auf das Prädikat bequeath zeigen. Alle diese Pfeile zu berücksichtigen, hätten die Abbildung jedoch zu unübersichtlich gemacht.

Der Pfeil vom Prädikat $\text{inherit}_{\text{Geschäftsleitung}}$ zum Prädikat bequeath() ist aus organisatorischer Sicht relativ sinnlos, da die Geschäftsleitung als oberste Ebene nur ihre eigenen Erbregeln berücksichtigen kann. An dieser Stelle bei der Übersetzung des Regelwerks eine Ausnahmeregelung zu machen, ist nicht notwendig, da für eine Autorisierungssphäre, die keine übergeordneten Autorisierungssphären hat, keine inherit Regeln gültig werden können.

Somit wird die Auswertung einer Anfrage in dem vorgestellten Ansatz zur dezentralen rollenbasierten Autorisierung komplett durch ein geschichtetes Datalog–Programm modelliert. Dessen Korrektheit für die wichtigen Aspekte der Vererbung und Koordinierung konnten nachgewiesen werden.
Insbesondere zur Lösung von Vererbungskonflikten müssen administrative Prozesse modelliert werden können. Darüber hinaus müssen aber auch alle anderen Elemente dieses Autorisierungssystems administriert werden. Wie dies genau geschieht, wird im folgenden Teil behandelt.

# Teil III

# Dezentrale Administrierung der Zugriffskontrolle

# Kapitel 9

# Administrierungssprache

Besonders bei den möglichen Lösungsansätzen für Vererbungskonflikte und den Strategien bei Koordinierungskonflikten wird deutlich, dass das Regelwerk auf keinen Fall starr ist und dauerhaft festgelegt wurde. Statt dessen kommt es immer wieder zu Anpassungen der Regeln, um das Regelwerk für neue Situationen zu adaptieren, aber auch bestehende Ungenauigkeiten — als solche kann man jede Art von Konflikten zwischen den Zugriffsstrategien der verschiedenen Autorisierungssphäre auch charakterisieren – zu bereinigen.
Neben der Administrierung der Zugriffsregeln, die einen wichtigen Aspekt bei der Zugriffskontrolle ausmachen, müssen auch die anderen Elemente des Autorisierungssystems wie Rollen, Organisationseinheiten, Objekt–Typen, Sicherheitseinstufungen und Autorisierungssphären administriert werden. Hinzu kommen neben diesen Mengen auch die entsprechenden Hierarchien über diese Elemente.

Dabei wird die Verantwortung für die Administrierung nicht, wie bei den Ansätzen zu ARBAC97 bzw. ARBAC99 [1], den einzelnen Subjekten in speziellen Administrierungsrollen überlassen. Statt dessen ist eine Gruppe von Subjekten in bestimmten Rollen einer Organisationseinheit für die verschiedenen Administrierungsaufgaben verantwortlich. Die Zusammensetzung der Gruppen kann je nach Administrierungsaufgabe unterschiedlich sein und wurde bereits als Autorisierungsteam im Abschnitt 3.3 eingeführt.

Der Verzicht auf spezielle Administrierungsrollen wird deshalb gewählt, um näher an den Strukturen realer Organisationen zu sein, bei denen z.B. ein Manager oder Abteilungsleiter auch für die Zuordnung der Mitarbeiter zu bestimmten Tätigkeiten und damit zu den entsprechenden Rollen (mit) verantwortlich[2] ist. Somit werden diesen Rollen neben den "normalen" Privilegien bzw. Zugriffsmethoden, wie sie im Abschnitt 3.6 vorgestellt wurden, weitere Privilegien eingeräumt.

Besonders bei komplexen Regelwerken in großen Organisationen wäre eine einzelne Person überfordert. Daher bietet sich die Ausarbeitung von Regeländerungen und anderen Administrierungsvorgängen in einem Autorisierungsteam an, um zudem auch eine wechselseitige Kontrolle zu erreichen.

Dieser Ansatz wurde bereits in [WL03b, WL04a] vorgestellt, in diesem und den folgenden Kapiteln 10 und 11 wird eine überarbeitete Form detaillierter präsentiert. Dabei beschränkt sich dieses Kapitel auf die Administrierungssprache, die wieder als Datalog–Sprache definiert ist. Der folgende Abschnitt 9.1 geht zunächst auf deren grundlegende Struktur ein. Anschließend zeigt Abschnitt 9.2 die Besonderheiten bei der Auswertung der Regeln auf. Danach wird im Abschnitt 9.3 beispielhaft einen Teil der Administrierungsprädikate vorgestellt, mit denen ein Administrierungsprozess initiiert werden kann und somit einzelne Vorschläge für die Veränderung/Administrierung des Autorisierungssystems erstellt werden können. Mit Hilfe von Partizipationsprädikaten, die im

[1] u.a. [SBC+97, SM99]. Weiteren Erläuterungen und eine Literaturübersicht findet sich im Abschnitt 2.2.

[2] In vielen Organisationen kann ein Manager einer Arbeitsgruppe nicht alleine z.B. über die Zuordnung eines neuen Mitarbeiters zu seinem Team entscheiden, sondern seine Vorgesetzen oder die Manager des bisherigen Teams, sowie u.U. die Personalabteilung sind ebenfalls involviert.

Abschnitt 9.4 definiert werden, können weitere Subjekte in den Administrierungsprozess involviert werden.

## 9.1 Grundstruktur der Administrierungssprache

Wie in der Übersicht der Kapitelstruktur schon erwähnt, wird die Administrierung ebenfalls durch eine Datalog–Sprache realisiert. Hierzu werden zwei neue Arten von Prädikaten definiert, *Administrierungsprädikate* und *Partizipationsprädikate*. Dabei wird der Administrierungsprozess in zwei Schritte gliedert. Im ersten Schritt wird mit Hilfe von Administrierungsprädikaten festgelegt, welche Subjekte welche Art von Veränderungen vorschlagen dürfen. Anschließend muss dieser Vorschlag durch eine Gruppe von Subjekten bestätigt werden, was mit Hilfe von Partizipationsprädikaten modelliert wird. Diese Trennung zwischen Initiierung und Bestätigung erscheint aus zwei Gründen sinnvoll: Zum einen kann davon ausgegangen werden, dass unterschiedliche Arten von Änderungen, die aber den gleichen "Bereich" betreffen, z.B. Hinzufügungen und Löschen von Regeln einer bestimmten Autorisierungssphäre, oder auch die Administrierung der Rollenhierarchie in einem bestimmten Abschnitt (Erzeugung und Löschen von Kanten, neue Rollen in diesem Bereich etc.) von denselben Subjekten bestätigt werden. Daher ist es einfacher, diese Bestätigung durch ein Partizipationsprädikat einzubinden, anstatt in jedem Administrierungsprädikat die Bestätigungen extra zu modellieren.
Zum anderen können Änderungen bei der Bestätigung (z.B. Veränderung der Gruppe) einfacher durchgeführt werden, da nur der Rumpf der Regel des entsprechenden Partizipationsprädikates verändert werden muss.
Die Grundstruktur[3] der Administrierungsregeln sieht wie folgt aus:

$$\begin{aligned} admin(as, s_i, \ldots) &\leftarrow L_i \& participate(as, ip, s \ldots) \\ participate(as, ip, s \ldots) &\leftarrow L_i \end{aligned}$$

Wobei $admin(as, s_i, \ldots)$ für ein beliebiges Administrierungsprädikat steht, welches gemäß der Autorisierungssphäre $as \in \mathcal{AS}$ festlegt, dass $s_i$ das Subjekt ist, das den administrativen Prozess initiieren darf. Die weiteren Parameter sind abhängig von den administrativen Prozessen und werden im Abschnitt 9.3 detailliert erläutert. Die Subjekte, die einen administrativen Prozess initiieren können, werden mit Hilfe von Feature Prädikaten $L_i \in \mathcal{F}$ bestimmt. An dieser Stelle erscheint es nicht sinnvoll auch Kontext Feature Prädikate ($\mathcal{CF}$) oder Kontext Trust Prädikate ($\mathcal{CL}$) aus Kapitel 4 zuzulassen, da davon auszugehen ist, dass die administrativen Prozesse nur über fest definierte, vertrauenswürdige Verbindungen von entsprechenden Hosts durchgeführt werden können. Andernfalls müsste der Ansatz so erweitert werden, dass auch spezifiziert werden kann, unter welchen Bedingungen Subjekte an der Bestätigung beteiligt werden dürfen und was geschehen soll, wenn sich diese Bedingungen ändern.

Das Prädikat $participate(as, ip, s \ldots)$ stellt ein beliebiges Partizipationsprädikat dar, welches festlegt, in welcher Form und von welchen Subjekten $s$ die Bestätigung in der Autorisierungssphäre $as \in \mathcal{AST}$ unter der Bezeichnung $ip \in \mathcal{IP}_{as}$ eingeholt wird. Bei einigen Partizipationsprädikaten sind noch weitere Parameter nötig, wie sich bei den Definitionen 9.17 bis 9.20 zeigen wird. Bei der Definition der Administrierungsprädikate wird noch die Menge der Partizipationsprädikate $\mathcal{PP}$ benötigt, die im Abschnitt 9.4 definiert sind.

Die Administrierungsprädikate aus Abschnitt 9.3 verweisen stets auf ein eindeutig bestimmtes Partizipationsprädikat einer Autorisierungssphäre. Um nicht innerhalb einer Autorisierungssphäre auf unterschiedliche Gruppen von Beteiligten bei der gleichen Art der Bestätigung — also je Partizipationsprädikat — beschränkt zu sein, erhalten die Partizipationsprädikate einen zusätzlichen Parameter, dessen Konstante als Unterscheidungsmerkmal dient.
Diese Konstanten können zum Beispiel als Bezeichner für Partizipationsprozesse Hinweise auf die Beteiligten wie "Developer-Net" oder "Manager-Develop" oder auf den Admin–Prozess, bei dem es

[3]Durch die Änderung der Reihenfolge dieser Prädikate gegenüber [WL03b, WL04a] bleibt die Reihenfolge der Auswertung von IPreds dieselbe wie bei der Auswertung von Zugriffsanfragen.

eingesetzt werden soll, wie "Assign-sub-Manager" enthalten. Um unnötige Konflikte mit doppelten Bezeichnern zu vermeiden, sind diese lokal innerhalb einer Autorisierungssphäre definiert. Die Menge $\mathcal{IP}_{as}$ enthält alle in einer Autorisierungssphäre $as$ definierten Bezeichner. Über die Menge der Bezeichner einer Autorisierungssphäre sind keine Variable zulässig, da dann unter Umständen mehrere Partizipationsprädikate bei der Auswertung der Regeln als relevant angesehen würden, was genau vermieden werden soll.

Die an der Bestätigung beteiligten Subjekte $s$ werden ebenfalls mit Hilfe von Feature Prädikaten $L_i \in \mathcal{F}$ bestimmt. Wie dies genau geschieht, wird im folgenden Abschnitt 9.2 erläutert. Die bisher intuitiv beschriebenen Autorisierungsteams können wie folgt definiert werden.

**Definition 9.1 (Autorisierungsteam)**
Zum Autorisierungsteam einer Autorisierungssphäre $as \in \mathcal{AS}$ gehören alle Subjekte, die in einer gültigen Belegung für die Variable $s_i$ der initiierenden Subjekte bei dem Administrierungsprädikat oder einer Variable über Subjekte bei den Partizipationsprädikaten $s$ liegen.

Bei den Partizipationsprädikaten wird es neben der Variable $s$ auch noch weitere Variablen geben, die beteiligte Subjekte angeben. Wie zu Beginn des Kapitels erläutert, verzichtet der hier vorgestellte Ansatz auf explizite Administrierungsrollen, sodass die Autorisierungsteams nur implizit definieren werden können. Wie die Administrierungsprädikat und Partizipationsprädikate ausgewertet werden, wird im folgenden Abschnitt erläutert.

## 9.2 Besonderheiten bei der Auswertung

Gegenüber der Auswertung der Autorisierungsregeln, dessen Prototyp in [Hal05] vorgestellt wird, müssen bei der Auswertung der Administrierungsprädikate und Partizipationsprädikate die folgenden Änderungen vorgenommen werden.

Bei den Administrierungsprädikaten wird zunächst überprüft, ob das Subjekt überhaupt berechtigt ist, diesen Administrierungsvorgang zu initiieren, bevor während der Auswertung der Partizipationsprädikate die Reaktionen der weiteren Beteiligten eingeholt werden. Demgegenüber werden bei den Autorisierungsregeln die Prädikate so ausgewertet, dass möglichst schnell ein Ergebnis erzielt werden kann. Bei der Auswertung der Rümpfe der Administrierungsprädikate ist dieses Vorgehen deswegen nicht geeignet, weil so unter Umständen die Partizipationsprädikate betrachtet werden, obwohl eines der Feature Prädikate im Rumpf der Administrierungsprädikate nicht gültig ist. Somit würden die Subjekte, die im Rumpf des Partizipationsprädikates durch die Feature Prädikate festgelegt werden, völlig unnötig benachrichtigt, was für die Akzeptanz eines solchen Systems nicht förderlich ist.

Der zweite Unterschied besteht genau in der Ermittlung aller beteiligten Subjekte für die Bestätigung. Sowohl bei den Autorisierungsregeln, als auch bei der Auswertung der Administrierungsprädikate liegt eine konkrete Belegung für alle Variablen im Prädikatskopf vor. Freie Variablen können lediglich im Rumpf vorkommen. In diesem Fall muss überprüft werden, ob es für diese Variablen eine einzige gültige Belegung gibt. Sobald diese gefunden ist, können die entsprechenden Prädikate als gültig betrachtet werden.
Bei Partizipationsprädikaten sind alle Variablen im Prädikatkopf, die sich auf Subjekte beziehen, stets freie Variablen. Darüber hinaus reicht es nicht aus, eine einzige gültige Belegung zu finden, sondern durch die Feature Prädikate sollen gerade **alle** Beteiligte beschrieben werden. Dies hat zur Folge, dass hier alle gültigen Belegungen der Variable gefunden werden müssen.

Bei der Auswertung der Partizipationsprädikate kann ohne Probleme festgestellt werden, in welchen Kontext sie benutzt wurden, sodass die Subjekte über die Details des Administrierungsprozesses, zu dem sie eine Bestätigung abgeben sollen, ohne Probleme informiert werden können. In dieser Arbeit wird nicht näher darauf eingegangen, wie die entsprechenden Subjekte benachrichtigt werden und ihre Reaktion zurückliefern. Denkbar wäre eine Benachrichtigung per Mail oder auch SMS und einer entsprechenden Reaktion über eine Web–Applikation.

## 9.3 Administrationsprädikate

Wie zu Beginn des Kapitels erwähnt, umfasst das vorgestellte Administrierungssystem eine ganze Reihe von Elementen, die alle über die im Kapitel 3 vorgestellten Feature Prädikate und die im Kapitel 4 vorgestellten Kontext Feature Prädikate abgefragt werden können. Da die in Tabelle 9.1 aufgeführten Elemente alle sehr ähnlich behandelt werden, sollen beispielhaft die Administrierungsprädikate für Rollen und deren Hierarchie vorgestellt werden. Außerdem wird noch auf einige Besonderheiten bei den Hierarchien von Organisationseinheiten und Autorisierungssphären eingegangen. Zudem werden die Administrierungsprädikate für die Autorisierungsregeln und die Vertrauensregeln erläutert.

| | | |
|---|---|---|
| Unit | $\mathcal{U}$ | $<_{\mathcal{U}}$ |
| Autorisierungssphären | $\mathcal{AS}$ | $<_{\mathcal{AS}}$ |
| Rollen | $\mathcal{R}$ | $<_{\mathcal{R}}$ |
| Subjekte | $\mathcal{S}$ | |
| Objekttypen | $\mathcal{T}$ | $<_{\mathcal{T}}$ |
| Vertrauensstufe | $\mathcal{TL}$ | $<_{\mathcal{TL}}$ |
| Lokalität | $\mathcal{L}$ | $<_{\mathcal{L}}$ |
| Host | $\mathcal{H}$ | |
| Betriebssystem | $\mathcal{OS}$ | $<_{\mathcal{OS}}$ |
| Sicherheitsstufe | $\mathcal{SL}$ | $<_{\mathcal{SL}}$ |
| Applikation | $\mathcal{AP}$ | $<_{\mathcal{AP}}$ |
| Objekte | $\mathcal{O}$ | $<_{\mathcal{O}}$ |
| Kommunikationskanal | $(V_C)$ | |
| access-methods | $\mathcal{AC}$ | |

Tabelle 9.1: Elemente der Administrierungssprache

Anzumerken ist, dass es keine Administrierungsprädikate für Objekte gibt, da das Anlegen und Löschen von Objekten durch das Autorisierungssystem geregelt wird, auch ergibt sich die Objekthierarchie durch die Struktur des Filesystems. Kommunikationskanäle werden, wie in Abschnitt 4.3 erläutert, nicht direkt benannt, sondern lediglich über Variablen referenziert. Daher ist hierzu keine direkte Administrierung nötig/möglich. Im Teil 2 wurde stets von einer Menge von Zugriffsmethoden gesprochen, ohne dass diese Menge vollständig benannt wurde. Allerdings wird davon ausgegangen, dass die Zugriffsmethoden unveränderlich sind und somit nicht administriert werden.

### 9.3.1 Rollen und Rollenhierarchie

Die Administrierung von Rollen und deren Hierarchien gehört gemäß RBAC–Standard zur Kernfunktionalität eines Administrierungssystems (vgl. [FKC03]). Zunächst werden in diesem Abschnitt die dazugehörigen Prädikate diskutiert. Die Zuweisung von Subjekten zu Rollen wird zusammen mit den anderen Zuweisungen im folgenden Abschnitt 9.3.2 behandelt.

In der Vereinbarung 3.2 wird festgelegt, dass jede Rolle zu einer anderen in Beziehung stehen muss. Um dies zu erfüllen, wird beim Anlegen einer Rolle bereits eine andere existierende Rolle mit angeben, zu der die neue Rolle gemäß der Rollenhierarchie $<_{\mathcal{R}}$ in Beziehung steht. Bei eingeschränkten Rollenhierarchien, wie sie in Abschnitt 3.4 besprochen wurden, wird nur eines der beiden folgenden Prädikate verwendet. Bei einer Baumstruktur mit nur einem Vorgänger nur create-lower-role und bei einer invertierten Baumstruktur mit nur einem Nachfolger create-upper-role. Bei einer allgemeinen Rollenhierarchie kann neben diesem auch das nächste Prädikat sinnvoll sein.

**Definition 9.2 (Anlegen einer untergeordneten Rolle)**
Eine Regel zum Anlegen einer untergeordneten Rolle hat die folgende Form

$$\textbf{create-lower-role}(as, s_i, r, h_r) \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V}_\mathcal{S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as \in \mathcal{AS}$ das Anlegen einer neuen Rolle $r \notin \mathcal{R}$, die der Rolle $h_r \in \mathcal{RT}$ direkt untergeordnet werden soll, vorschlagen darf, falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind. Die Rolle $r$ wird angelegt und eine Kante $r <_\mathcal{R} h_r$ zur Rollenhierarchie $<_\mathcal{R}$ hinzugefügt, wenn die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist.

Analog zur vorangegangenen Definition kann bei invertierten, eingeschränkten oder allgemeinen Rollenhierarchien eine übergeordnete Rolle definiert werden.

**Definition 9.3 (Anlegen einer übergeordneten Rolle)**
Eine Regel zum Anlegen einer übergeordneten Rolle hat die folgende Form

$$\textbf{create-upper-role}(as, s_i, r, l_r) \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V}_\mathcal{S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as \in \mathcal{AS}$ das Anlegen einer neuen Rolle $r \notin \mathcal{R}$, die der Rolle $l_r \in \mathcal{RT}$ direkt übergeordnet werden soll, vorschlagen darf, falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind. Die Rolle $r$ wird angelegt und eine Kante $l_r <_\mathcal{R} r$ zur Rollenhierarchie $<_\mathcal{R}$ hinzugefügt, wenn die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist.

Bei einer allgemeinen Rollenhierarchie können neben der ursprünglichen Kante noch Zusätzliche eingefügt werden, wie es in der Abbildung 3.5 auf Seite 32 zu sehen ist. Die entsprechenden Regeln können mit dem folgenden Prädikat modelliert werden.

**Definition 9.4 (Anlegen einer neuen Kante in der Rollenhierarchie)**
Eine Regel zur Erzeugung einer neuen Kante in der Rollenhierarchie hat die folgende Form

$$\textbf{edge-addition-role}(as, s_i, h_r, l_r) \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V}_\mathcal{S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as \in \mathcal{AS}$ die Erzeugung einer neuen Kante in der Rollenhierarchie zwischen den beiden Rollen $h_r, l_r \in \mathcal{RT}$, vorschlagen darf, falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind und zuvor gilt $h_r \not<_\mathcal{R} l_r$. Die neue Kante $l_r <_\mathcal{R} h_r$ wird in die Rollenhierarchie eingefügt, wenn zudem die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist.

Die Bedingung $h_r \not<_\mathcal{R} l_r$ stellt sicher, dass keine Zyklen in die Rollenhierarchie eingefügt werden können. Zusätzliche Kanten wie in Abbildung 3.5 von Senior Project-Manager über Test-Coordinator nach Tester sind aber möglich.

Wesentlich kritischer ist das Löschen einer Kante in der Rollenhierarchie. Diese Thematik wurde in [NO94] und [SBM99] diskutiert. Für das hier vorgestellte System muss also sichergestellt sein, dass durch das Löschen einer Kante keine Rollen betroffen sind, die in den Autorisierungsregeln durch die Feature Prädikate $\mathsf{role}(s, u, r)$ sowie $\mathsf{rolein}(r_1, r_2)$ bzw. $\mathsf{subrole}(r_1, r_2)$ referenziert werden. Während dies für diesen expliziten Fall noch relativ einfach zu überprüfen ist, muss bei freien Variablen die Menge der möglichen Belegungen berechnet werden und geprüft werden, dass diese Menge durch das Löschen der Kante nicht verändert wird. Wenn dies der Fall wäre, müsste das Löschen der Kante verhindert werden. Dies ist eine strengere Forderung als in der Administrierung des Standard-RBAC [SBM99, S. 132] bei der nur transitive Kanten gelöscht werden können.

**Definition 9.5 (Löschen einer Kante in der Rollenhierarchie)**
Eine Regel zum Löschen einer Kante in der Rollenhierarchie hat die folgende Form

$$\textbf{edge-deletion-role}(as, s_i, h_r, l_r) \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V_S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as \in \mathcal{AS}$ das Löschen der Kante in der Rollenhierarchie zwischen den beiden Rollen $h_r, l_r \in \mathcal{RT}$ vorschlagen darf, falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind und zuvor gilt $l_r <_{\mathcal{R}} h_r$, wobei überprüft wird, dass diese Kante nicht mehr referenziert wird. Die Kante $l_r <_{\mathcal{R}} h_r$ wird aus der Rollenhierarchie entfernt, wenn zudem die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist.

Eine ähnliche Problematik ergibt sich auch beim Löschen einer Rolle. Aus diesem Grund werden Rollen nur deaktiviert, sodass sie nicht mehr zugewiesen bzw. aktiviert werden können.

**Definition 9.6 (Deaktivierung einer Rolle)**
Eine Regel zum Deaktivieren einer Rolle hat die folgende Form

$$\textbf{deaktivate-role}(as, s_i, r) \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V_S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as \in \mathcal{AS}$ das Deaktivieren einer Rolle $r \in \mathcal{R}$ vorschlagen darf, falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind, und die Deaktivierung in Kraft tritt, wenn die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist.

Für alle weiteren Elemente, die in Tabelle 9.1 aufgeführt sind, müssen ebenfalls entsprechende Administrationsprädikate zum Anlegen und zum Deaktivieren und zur Verwaltung der entsprechenden Hierarchien, also zum Anlegen und Löschen der Kanten, definiert werden. Beim Ändern der Hierarchie der Autorisierungssphären muss die im Abschnitt 8.2 diskutierte Übersetzung des Datalog–Programmes beachtet werden. Dies geschieht analog zu den in diesem Abschnitt vorgestellten Administrationsprädikaten für Rollen und deren Hierarchien.

### 9.3.2 Zuweisungen von Rollen und Objekten

Neben Administrierung der verschiedenen Elemente und deren Hierarchien gehört zu den Administrierungsaufgaben auch die Zuweisung einer Rolle in einer Organisationseinheit bzw. Unit zu einem bestimmten Benutzer. Außerdem muss noch die Zuweisung eines Besitzers und einer Organisationseinheit für ein bestimmtes Objekt betrachtet werden.

Das Anlegen von neuen Objekten wird nicht als administrativen Vorgang aufgefasst, sondern analog zu verschiedenen Filesystemen als Schreiboperation auf das übergeordnete Objekt[4]. Ähnliches gilt für das Löschen von Objekten, das als Schreiboperation auf das Objekt selbst und das übergeordnete Objekt gesehen werden kann[5].

Die zu Beginn des Kapitels erwähnte Rollenzuweisung, wird in einer größeren dynamischen Organisation einer der häufigeren Administrationsvorgänge sein. Das Recht zur Initiierung eines entsprechenden Administrationsvorganges kann mit dem folgenden Prädikat vergeben werden.

**Definition 9.7 (Rollenzuweisung)**
Eine Zuweisungsregel hat die folgende Form

$$\textbf{assign}(as, s_i, s, r, u) \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V_S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as \in \mathcal{AS}$ die Zuweisung der Rolle $r \in \mathcal{RT}$ in Unit $u \in \mathcal{UT}$ für das Subjekt $s \in \mathcal{V_S}$ vorschlagen darf. Falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind und die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist, wird die Zuweisung durchgeführt.

Wesentlich kritischer ist das Löschen einer Rollenzuweisung, da damit unter Umständen Zugriffsregeln, die vorangegangene Zugriffe mittels done() bzw. done_since() Prädikaten (vgl. die Definitionen 3.27 und 3.28 im Abschnitt 3.6) verweisen, nicht mehr gültig wären. Daher werden

[4] Alternativ könnte man noch eine zusätzliche Zugriffsoperation *"create"* definieren, die dies detaillierter abdeckt.
[5] Hier könnte alternativ auch als Gegenstück zur Operation *"create"* eine Operation *"delete"* einführt werden.

Rollenzuweisungen nicht gelöscht, sondern lediglich für ungültig erklärt. Diese werden dann nur noch bei der Auswertung von done() und done_since() Prädikaten berücksichtigt.

Der Entzug einer Rolle kann über das folgende Prädikat initiiert werden.

**Definition 9.8 (Rollenentzug)**
Eine Entzugsregel hat die folgende Form

$$\textbf{withdraw}(as, s_i, s, r, u) \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V_S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as \in \mathcal{AS}$ den Entzug der Rolle $r \in \mathcal{RT}$ in Unit $u \in \mathcal{UT}$ für das Subjekt $s \in \mathcal{V_S}$ vorschlagen darf. Falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind und die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist, wird der Entzug der Rolle durchgeführt.

Die Zugriffsprivilegien für ein Objekt sind stets an eine Rolle in einer Organisationseinheit geknüpft, unter Umständen kann dies auch eine Reihe von Rollen, eine Reihe von Organisationseinheiten oder eine Kombination davon sein. Bei der Auswertung einer Regel, aufgrund derer das Anlegen eines neuen Objektes schließlich genehmigt wird, wird wie im Abschnitt 8.1 erläutert, stets die aktive Rolle und deren Organisationseinheit bestimmt. Dieser Organisationseinheit wird auch das neue Objekt zugeordnet.

Obwohl es in manchen Organisationen sicherlich der Fall sein kann, dass diese Zuweisung sich häufiger ändert, wenn das Objekt und die darin enthaltenen Inhalte einen bestimmten Bearbeitungsprozess durchlaufen. Im gewählten Szenario könnten dies zum Beispiel Objekte sein, die die Dokumentation zu den Programmen enthalten, die zunächst einer Arbeitsgruppe in der Entwicklungsabteilung zugeordnet sind, nach dem Release aber der Vertriebsabteilung zugeordnet werden. Bei einer Versicherung könnten in einem Objekt die Daten zu einem Schadensfall enthalten sein, der Besitzer wäre dann zunächst der Versicherungsvertreter vor Ort, später wäre es dann der Sachbearbeiter, der die Ansprüche prüft.

**Definition 9.9 (Unitänderung)**
Eine Unitzuweisungsregel hat die folgende Form

$$\textbf{change-unit}(as, s_i, o, u_o, u_n) \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V_S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as \in \mathcal{AS}$ die Unitzuweisung für Objekt $o \in \mathcal{OT}$ von der alten Organisationseinheit $u_o \in UT$ zur neuen Organisationseinheit $u_n \in UT$ vorschlagen darf. Falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind und die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist, wird die zugehörige Unit von $o$ entsprechend auf $u_n$ geändert.

Bei dieser Regel wird also implizit immer überprüft, ob unitof$(o, u_o)$ gültig ist.

Wie in Abschnitt 3.5 erläutert, hat ein Objekt stets einen Besitzer, bei der Erstellung eines Objektes ist dies automatisch das Subjekt, das diese Operation durchführt. Im Gegensatz zur Bestimmung der Organisationseinheit ist dies relativ einfach, da das Subjekt, das ein neues Objekt anlegt, nicht umständlich bestimmt werden muss.

**Definition 9.10 (Besitzerwechsel)**
Eine Regel zum Wechsel des Besitzers hat die folgende Form

$$\textbf{change-owner}(as, s_i, o, s_o, s_n) \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V_S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as \in \mathcal{AS}$ den Wechsel des Besitzers für Objekt $o \in \mathcal{OT}$ vom Subjekt $s_o \in ST$, zu Subjekt $s_n \in ST$ vorschlagen darf. Falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind und die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist, wird ist $s_n$ der neue Besitzer von $o$.

Bei dieser Regel wird also implizit auch immer geprüft, ob $\mathsf{owner}(o, s_o)$ gültig ist.

Für die Zuordnung der Hosts zu bestimmten Lokationen, Betriebssystemen, verantwortlicher Autorisierungssphäre oder Hardware–Architekturen (vgl. Def. 4.5, 4.10, 4.13, 4.9) müssen auch entsprechende Administrationsprädikate definiert werden. Gleiches gilt für das Vertrauen in die Quellen eine Applikation (vgl. Def. 4.27). Diese entsprechenden Administrationsprädikate unterscheiden sich nicht wesentlich von denen der zuvor definierten Prädikaten zur Zuweisung einer Rolle bzw. den Wechsel von Unit oder Besitzer, daher seien sie nur kurz in Tabelle 9.2 zusammengefasst.

| Administrationsprädikat | |
|---|---|
| **create-host**$(as_i, s_i, h, l, os, ar, as)$ | Gemäß der Autorisierungssphäre $as_i$ kann das Subjekt $s_i$ das Anlegen eines neuen Hosts $h$ initiieren, der an Lokation $l$ mit dem Betriebssystem $os$ und der Hardware-Architektur $ar$ von der Autorisierungssphäre $as$ verwaltet wird. |
| **delete-host**$(as_i, s_i, h, l, os, ar, as)$ | Gemäß der Autorisierungssphäre $as_i$ kann das Subjekt $s_i$ das Löschen eines neuen Hosts $h$ initiieren, der an Lokation $l$ mit dem Betriebssystem $os$ und der Hardware-Architektur $ar$ von der Autorisierungssphäre $as$ verwaltet wird. |
| **change-location**$(as_i, s_i, h, l_o, l_n)$ | Gemäß der Autorisierungssphäre $as_i$ kann das Subjekt $s_i$ den Wechsel der Lokation des Hosts $h$ von $l_o$ zu $l_n$ initiieren. |
| **change-os**$(as_i, s_i, h, os_o, os_n)$ | Gemäß der Autorisierungssphäre $as_i$ kann das Subjekt $s_i$ den Wechsel des Betriebssystems von Hosts $h$ von Lokation $os_o$ zu $os_n$ initiieren. |
| **change-arch**$(as_i, s_i, h, as_o, as_n)$ | Gemäß der Autorisierungssphäre $as_i$ kann das Subjekt $s_i$ den Wechsel der Hardware–Architektur von Hosts $h$ von $ar_o$ zu $ar_n$ initiieren. |
| **change-as**$(as_i, s_i, h, as_o, as_n)$ | Gemäß der Autorisierungssphäre $as_i$ kann das Subjekt $s_i$ den Wechsel der zugeordneten Autorisierungssphäre von Hosts $h$ von $as_o$ zu $as_n$ initiieren. |
| **create-app**$(as_i, s_i, as, app, t)$ | Gemäß der Autorisierungssphäre $as$ kann das Subjekt $s_i$ das Anlegen einer neuen Applikation $app$ initiieren, die nach Autorisierungssphäre $as$ die Vertrauensstufe $t$ hat. |
| **change-app**$(as_i, s_i, as, app, t_o, t_n)$ | Gemäß der Autorisierungssphäre $as$ kann das Subjekt $s_i$ das Ändern der Vertrauensstufe von Autorisierungssphäre $as$ für die Applikation $app$ von der Vertrauensstufe $t_n$ nach $t_n$ initiieren. |

Tabelle 9.2: Administrationsprädikate für lokale Aspekte

Das Anlegen eines neuen Benutzers ist relativ unkritisch, hier muss lediglich überprüft werden, dass die Bezeichnung für das Benutzerkonto bisher noch nicht im System vorhanden ist. Damit ein Benutzer Objekte signieren kann, wird Public–Key Infrastruktur vorausgesetzt[6], sodass es zu jedem Benutzer einen digitalen Schlüssel gibt, dessen öffentlicher Teil beim Anlegen im gesamten System verfügbar gemacht wird. Über die genaue Benutzerverwaltung werden keine weiteren Annahmen getroffen.

**Definition 9.11 (Anlegen eines Benutzers)**
Eine Regel zum Anlegen eines Benutzerkontos hat die folgende Form

$$\textbf{create-subject}(as, s_i, s) \leftarrow L_i \& P$$

[6] vgl. Abschnitt 3.5.3

wobei $s_i \in \mathcal{V}_\mathcal{S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as \in \mathcal{AS}$ das Anlegen eines neuen Subjektes $s \notin \mathcal{S}$ vorschlagen darf, falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind und das angelegt wird, wenn die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist.

Extrem kritisch wäre es einen Benutzer bzw. dessen Benutzeraccount und digitalen Schlüssel endgültig aus dem System zu löschen und zwar aus den gleichen Gründen wie das endgültige Löschen einer Rollenzuweisung. Sämtliche Signaturen an Objekten, die durch diesen Benutzer gemacht wurden nicht mehr auf ihre Gültigkeit überprüft werden könnten. Ähnliches gilt für bei der Auswertung von done() , done_since() oder owner(). Daher wird ein Benutzer bzw. der Benutzeraccount nur deaktiviert, sodass ein Einloggen in das System nicht mehr möglich ist.

**Definition 9.12 (Deaktivieren eines Benutzers)**
Eine Regel zum Deaktivieren eines Benutzerkontos hat die folgende Form

$$\textbf{delete-subject(}as, s_i, s\textbf{)} \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V}_\mathcal{S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as \in \mathcal{AS}$ das Deaktivieren des Subjektes $s \notin \mathcal{V}_\mathcal{S}$ vorschlagen darf, falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind, und die Deaktivierung durchgeführt wird, wenn die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist.

Sollte der Namensraum für Benutzerkonten irgendwann knapp werden, könnte durch ein zusätzliches Programm offline überprüft werden, ob von einem deaktivierten Benutzer noch signierte Dateien im Filesystem sind bzw. solche die ihn noch als Besitzer ausweisen, oder Aktionen, die über done() bzw. done_since(), referenziert werden. Wenn dies nicht der Fall ist, könnte das entsprechende Benutzerkonto ohne Problem endgültig gelöscht werden.

### 9.3.3 Zugriffsregeln

Um zu vermeiden für alle Prädikate, die in den Kapiteln 5 bis 7 definiert wurden und deren Regeln nicht automatisch erzeugt werden (vgl. Abbildung 8.1), ein eigenes Administrationsprädikat definieren zu müssen, wird auf diese Prädikate mittels eines Bezeichners verwiesen. Dadurch kann für alle Access Prädikate das gleiche Prädikat zum Anlegen von Regeln und ein zweites Prädikat zum Löschen verwendet werden. Die Bezeichner für Access Prädikate sind, wie in Tabelle 9.3 dargestellt, definiert. Die Menge $\mathcal{IA}$ enthält all diese Bezeichner.

| Access Prädikat | $\mathcal{IA}$ |
|---|---|
| cando | icando |
| dercando | idercando |
| do | ido |
| bequeath | ibequeath |
| grant | igrant |

Tabelle 9.3: Bezeichner für Access Prädikate

Es wurde darauf verzichtet in die Menge $\mathcal{IA}$ zusätzlich noch Elemente aufzunehmen, die auf mehrere Access Prädikate verweisen, z.B. "ialldo" das auf die Zugriffsprädikate cando, dercando und do verweisen könnte. Da die Bezeichnungen für diese Elemente nicht eindeutig zu wählen sind[7], würde dies zu unnötigen Verwirrungen führen.

Mit dem folgenden Administrationsprädikat kann festgelegt werden, welche Subjekte das Recht haben, einen Administrierungsprozess zum Hinzufügen einer neuen Zugriffsregel zu initiieren.

[7]Eine alternative Bezeichnung könnte auch "icando-dercando-do" sein oder eine beliebige Permutation der Prädikatsnamen.

**Definition 9.13 (Hinzufügen einer Zugriffsregel)**
Eine Regel zum Hinzufügen einer neuen Zugriffsregel hat die folgende Form

$$\textbf{addrule}(as_i, s_i, ia, as, sign, a) \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V}_\mathcal{S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as_i \in \mathcal{AS}$ das Hinzufügen einer neuen Zugriffsregel vom Typ $ia \in \mathcal{IA}$ für die Autorisierungssphäre $as \in \mathcal{AST}$ für die Methode $a \in AC$ vorschlagen darf, wobei diese Regel entweder erlaubend ($sign = +$) oder verbietend ($sign = -$) sein darf, falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind. Die Zugriffsregel wird dem Regelsatz der Autorisierungssphäre $as$ zugefügt, wenn die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist.

Die Unterscheidung zwischen der Autorisierungssphäre $as_i$, in der diese Regel festgelegt ist, und der Autorisierungssphäre $as$, in denen die Zugriffsregel angelegt werden soll, ist besonders bei der Auflösung von Vererbungskonflikten hilfreich.

Jede Regel soll, wie in [Hal05] beschrieben, über eine eindeutige Identifizierung verfügen, diese kann dann beim Löschen angegeben werden. Durch das folgende Prädikat kann überprüft werden, ob das initiierte Subjekt berechtigt ist, diesen Administrierungsprozess anzustoßen.

**Definition 9.14 (Entfernen von Zugriffsregeln)**
Eine Regel zum Löschen einer Zugriffsregel hat die folgende Form

$$\textbf{delrule}(as_i, s_i, ia, as, sign, a) \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V}_\mathcal{S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as_i \in \mathcal{AS}$ das Löschen einer neuen Zugriffsregel vom Typ $ia \in \mathcal{IA}$ aus der Autorisierungssphäre $as \in \mathcal{AST}$ für die Methode $a \in ACT$ vorschlagen darf, wobei diese Regel entweder erlaubend ($sign = +$) oder verbietend ($sign = -$) sein kann, falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind. Die Zugriffsregel wird aus dem Regelsatz der Autorisierungssphäre $as$ gelöscht, wenn die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist.

Beim Löschen müssen sich die Beteiligten darüber im Klaren sein, dass durch das Fehlen einer Regel eventuell bisher verwendete Auswertungspfade nicht mehr gültig sind. In diesem Ansatz wird nur das das Einfügen und Löschen, nicht aber das Ändern von Regeln als Administrierungsprozess definiert.

Bei den beiden obigen Administrationsprädikaten kann nicht bestimmt werden, welche Subjekte $s \in ST$ und Objekte $o \in OT$ von der neuen Regel oder der zu löschenden Regel betroffen sein dürfen. Die entsprechenden Parameter könnten relativ einfach auf folgender Weise in das Prädikat eingebaut werden.

$$\textbf{addrule}(as_i, s_i, ir, as, s, app, o, sign, a) \leftarrow L_i \& P$$
$$\textbf{delrule}(as_i, s_i, ir, as, s, app, o, sign, a) \leftarrow L_i \& P$$

Jedoch ergibt sich das Problem, ob die Subjekte, Applikationen und Objekte der neuen bzw. zu löschenden Zugriffsregel mit denen in diesen Administrationsregeln übereinstimmen. Es ist möglich, alle Belegungen der jeweiligen Variablen zu ermitteln, wobei über Feature Prädikate wie $\mathsf{role}(s, u, r) \& \mathsf{unitof}(o, u)$ durchaus Beziehungen zwischen den beiden Variablen erzeugt werden können. Nur wenn es sich bei der ermittelten Teilmenge von Belegungen der Variablen in den Zugriffsregeln um eine Untermenge der Belegungen, die bei den Administrationsprädikaten ermittelt wurde, handelt, wären die Bindungen für diese erweiterten Administrationsprädikate erfüllt. Dies ist jedoch sehr aufwendig, effizientere Herangehensweisen können als Teil der zukünftigen Arbeit betrachtet werden. Außerdem würde dies bedeuten, dass sich der Interpreter für diese Administrationsprädikate anders verhalten müsste, als in Abschnitt 9.2 beschrieben.

### 9.3.4 Vertrauensregeln

Zusätzlich zu den Zugriffsregeln können in den Autorisierungssphären auch Regeln bezüglich der Einstufung in Vertrauensstufen, mittels Kontext Trust Prädikaten definiert werden. Da sich deren Parameterliste von denen der Access Prädikate unterscheidet, müssen zusätzliche Administrationsprädikate jeweils zum Anlegen und Löschen der entsprechenden Regeln definiert werden.

Wie sich bei den Administrationsregeln zu den Access Prädikaten gezeigt hat, können die Belegungen der Parameter in den Regeln nicht berücksichtigt werden. Daher macht es keinen Sinn diese bei den Administrationsprädikaten mit aufzunehmen. Wie schon bei den Access Prädikaten wird daher nur ein Administrationsprädikat für das Anlegen von neuen Regeln mit Kontext Trust Prädikaten und ein zusätzliches Prädikat zum Löschen dieser Regeln angelegt.

Die Unterscheidung für welches Kontext Trust Prädikat dieser Administrationsvorgang durchgeführt werden soll, erfolgt wieder mittels Bezeichner. Die Bezeichner für Kontext Trust Prädikate $\mathcal{ICT}$ sind analog zu denen für Access Prädikate $\mathcal{IA}$ in Tabelle 9.4 definiert. Auf die Definition von zusätzlichen Elementen, die mehrere Kontext Trust Prädikate abdecken, wird wie bei $IA$ verzichtet.

| Kontext Trust Prädikat | $\mathcal{ICT}$ |
|---|---|
| host | ihost |
| derhost | iderhost |
| channel | ichannel |
| derchannel | iderchannel |
| application | iapplication |
| derapplication | iderapplication |

Tabelle 9.4: Bezeichner für Kontext Trust Prädikate

Die Rechte zur Initiierung eines Administrierungsprozesses, bei dem eine neue Vertrauensregel den bisherigen hinzugefügt werden soll, kann mit dem folgenden Administrationsprädikat modelliert werden.

**Definition 9.15 (Hinzufügen einer Vertrauensregel)**
Eine Regel zum Hinzufügen einer neuen Vertrauensregel hat die folgende Form

$$\textbf{addtrustrule}(as_i, s_i, ict, as, tl) \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V_S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as_i \in \mathcal{AS}$ das Hinzufügen einer neuen Vertrauensregel vom Typ $ict \in \mathcal{ICT}$ für die Autorisierungssphäre $as \in \mathcal{AST}$ für die Methode $tl \in \mathcal{TL}$ vorschlagen darf, falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind. Die Vertrauenssregel wird dem Regelsatz der Autorisierungssphäre $as$ zugefügt, wenn die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist.

Wie schon beim Löschen der Zugriffsregeln verfügt auch jede Vertrauensregel über eine eindeutige Identifizierung, die beim Löschen angegeben werden kann. Durch das folgende Prädikat kann überprüft werden, ob das initiierte Subjekt berechtigt ist, diesen Administrierungsprozess anzustoßen.

**Definition 9.16 (Entfernen von Vertrauensregeln)**
Eine Regel zum Löschen einer Vertrauensregel hat die folgende Form

$$\textbf{deltrustrule}(as_i, s_i, ict, as, tl) \leftarrow L_i \& P$$

wobei $s_i \in \mathcal{V_S}$ das Subjekt ist, das gemäß der Autorisierungssphäre $as_i \in \mathcal{AS}$ das Löschen einer Vertrauensregel vom Typ $ict \in \mathcal{ICT}$ aus der Autorisierungssphäre $as \in \mathcal{AST}$ für die Vertrauensstufe $tl \in \mathcal{TL}$ vorschlagen darf, falls alle Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ gültig sind. Die Vertrauenssregel wird aus dem Regelsatz der Autorisierungssphäre $as$ gelöscht, wenn die Auswertung des Partizipationsprädikates $P \in \mathcal{PP}$ positiv ist.

Mit diesen Administrationsprädikaten und den entsprechend analog definierten Administrationsprädikaten für die weiteren Elemente (vgl. Tabelle 9.1) können die Rechte zur Initiierung der entsprechenden Administrierungsprozesse modelliert werden. Auf die Verwendung von Kontext Feature Prädikaten und Kontext Trust Prädikaten in diesen Administrationsregeln wird zunächst verzichtet, da im Rahmen einer verteilten Implementierung noch zu klären ist, was das Objekt ist, auf dem sich die Zugriffsanfrage bzw. die Anfrage zur Initiierung eines Administrierungsprozesses bezieht. Die Definition von Prädikaten, die die Rechte zum Hinzufügen und Löschen von neuen Administrierungsregeln beschreiben, bleibt Teil der zukünftigen Forschungsarbeit. Hier muss vor allem eine Herangehensweise gefunden werden, die Zahl dieser Prädikate gering zu halten. Außerdem müssen diese neuen Regeln auch wieder administriert werden.

## 9.4 Partizipationsprädikate

Mit den im vorangegangen Abschnitt definierten Administrationsprädikaten können alle notwendigen Änderungen am Autorisierungssystem initiiert werden. Damit die Änderungen auch durchgeführt werden, müssen sie von anderen Subjekten, die (damit)[8] auch zum Autorisierungsteam gehören, bestätigt werden. Dies kann mit Hilfe der nun folgenden Partizipationsprädikate modelliert werden, die eine ganze Bandbreite von Bestätigungen zur Verfügung stellen. Diese reichen von der einfachen Zustimmung durch eine Person, über eine Abstimmung mit einfacher Mehrheit, bis hin zu einer Abstimmung, bei der ein Teil der Beteiligten ein Vetorecht hat.

Die einfachste Art einer Bestätigung in einem Administrationsprozess ist die Zustimmung durch einen weiteren[9] Beteiligten. Dies kann durch die folgende Entscheidungsregel modelliert werden.

**Definition 9.17 (Zustimmungsregel)**
Die Zustimmungsregel hat die Form

$$\textbf{approve}(as, ip, s) \leftarrow L_i$$

wobei $s \in \mathcal{V}_S$ das Subjekt ist, dessen Reaktion ("Zustimmung" oder "Ablehnung") als Teil eines Administrierungsprozesses in der Autorisierungssphäre $as \in \mathcal{AST}$ unter der Bezeichnung $ip \in \mathcal{IP}_{as}$ eingeholt werden soll. Die Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ dienen dazu, dieses Subjekt $s$ festzulegen. Wenn die Reaktion von Subjekt $s$ eine Zustimmung ist, wird das Prädikat als positiv ausgewertet betrachtet, wenn es eine Ablehnung ist entsprechend als negativ.
Sollte es mehr als eine gültige Belegung für das Subjekt $s$ geben oder gar keine, bricht die Auswertung mit einer Fehlermeldung ab.

Bei der Auswertung der Zustimmungsregel wird zunächst die gültige Belegung für die Variable $s$ durch Auswertung der Feature Prädikate im Rumpf bestimmt. Das Subjekt, das dadurch bestimmt ist, erhält eine Benachrichtigung, mit der es um seine Reaktion zu dem initiierten Administrierungsprozess gebeten wird. Falls diese Reaktion eine Zustimmung ist, wird das Prädikat *approve* als gültig angesehen und damit wird gemäß der Definition der Administrierungsprädikate in Abschnitt 9.3 dieses gültig und die vorgeschlagene Änderung wird entsprechend durchgeführt.

Ein Beispiel hierfür sind die folgenden beiden Regeln, bei der die Erste für die Autorisierungssphäre Netzwerk festlegt, das jedes Subjekt in der Rolle Project–Manager in einer zur Autorisierungssphäre Netzwerk gehörenden Organisationseinheit, die Zuordnung eines (anderen) Subjektes zu einer der Rolle Project–Manager untergeordneten Rolle initiieren darf. Dieser Vorschlag wird umgesetzt, wenn eine Bestätigung durch die mit Assign-sub-Manager bezeichnete Zustimmungsregel in der Autorisierungssphäre Netzwerk erfolgt. Die zweite Regel enthält genau diese Zustimmungsregel, wobei im Regelrumpf festgelegt ist, dass die Reaktion des Subjekts, welches die Rolle Direktor in der Organisationseinheit Software innehat, eingeholt werden soll.

[8] vgl. Abschnitt 3.3

[9] Rein technisch betrachtet gibt es keine Einschränkung, dass dies nicht auch das gleich Subjekte ist, das den Prozess initiiert hat. Dazu kann entweder noch ein Feature Prädikat equal_subject definiert werden, das dann negiert in den Administrierungsprädikaten verwendet wird und so dies explizit ausschließt. Eine andere Möglichkeit wäre eine implizite Verwendung und Prüfung im Interpreter.

**Beispielregel 9.1**

$$\begin{aligned}\mathsf{assign}(\mathsf{Netzwerk}, s_i, s, r, u) \leftarrow & \mathsf{role}(s_i, \mathsf{Project-Manager}, u)\& \\ & \mathsf{sphereof}(u, \mathsf{Netzwerk})\&\mathsf{subrole}(r, \mathsf{manager})\& \\ & \mathsf{approve}(\mathsf{Netzwerk}, \mathsf{Assign-sub-Manager}, s) \\ \mathsf{approve}(\mathsf{Netzwerk}, \mathsf{Assign-sub-Manager}, s) \leftarrow & \mathsf{role}(s, \mathsf{Director}, \mathsf{Software})\end{aligned}$$

Da stets von einem Administrationsteam ausgegangen wird, das gemeinschaftlich für eine Autorisierungssphäre verantwortlich ist, sollte die Zustimmung durch ein einzelnes Subjekt nicht häufig vorkommen[10]. Statt dessen wird mit dem folgenden Prädikat die Bestätigung von einer Gruppe von Subjekten eingeholt.

**Definition 9.18 (Mehrfach–Zustimmung)**
Die Regel für die mehrfache Zustimmung hat die Form

$$\textbf{multi-approve}(as, ip, s) \leftarrow L_i$$

wobei $s \in \mathcal{V}_\mathcal{S}$ die Subjekte sind, deren Reaktion ("Zustimmung" oder "Ablehnung") als Teil eines Administrierungsprozesses in der Autorisierungssphäre $as \in \mathcal{AST}$ unter der Bezeichnung $ip \in \mathcal{IP}_{as}$ eingeholt werden sollen. Die Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ dienen dazu, diese Subjekte $s$ festzulegen. Wenn alle Reaktionen der Subjekte $s$ eine Zustimmung sind, wird das Prädikat als positiv ausgewertet betrachtet, wenn es mindestens eine Ablehnung gibt, entsprechend als negativ.

Mit der folgenden Regel wird in der Autorisierungssphäre Entwicklung festgelegt, dass unter der Bezeichnung Senior-Staff die Zustimmung aller Subjekte eingeholt werden soll, die in einer Organisationseinheit in der Entwicklungsabteilung eine Rolle unterhalb von Development Manager und oberhalb Developer gemäß der Rollenhierarchie in Abbildung 3.5 auf Seite 32 innehaben.

**Beispielregel 9.2**

$$\begin{aligned}\mathsf{multi\text{-}approve}(\mathsf{Entwicklung}, \mathsf{Senior-Staff}, s) \leftarrow & \mathsf{role}(s, r, u)\&\mathsf{subunit}(u, \mathsf{Entwicklung})\& \\ & \mathsf{subrole}(\mathsf{Developer}, r)\& \\ & \mathsf{subrole}(r, \mathsf{DevelopmentManager})\end{aligned}$$

Problematisch ist bei diesem Vorgang, dass jeder Beteiligter der einmal bestimmt wurde, die Bestätigung beliebig lange aufhalten kann, in dem er einfach nicht reagiert. Daher ist es sinnvoll auch die Möglichkeit vorzusehen, einen Zeitraum anzugeben, bis wann die Reaktion eingehen soll, damit sie gewertet wird. An dieser Stelle wird darauf verzichtet, noch ein weiteres Partizipationsprädikat zu definieren. Statt dessen wird diese Festlegung eines Zeitraums bei der folgenden Definition für ein Quorum genutzt.

**Definition 9.19 (Quorum)**
Die Regel für ein Quorum hat die Form

$$\textbf{quorum}(as, ip, s, d) \leftarrow L_i$$

wobei $s \in \mathcal{V}_\mathcal{S}$ die Subjekte sein, deren Reaktionen ("Zustimmung", "Ablehnung" oder "Enthaltung") als Teil eines Administrierungsprozesses in der Autorisierungssphäre $as \in \mathcal{AST}$ unter der Bezeichnung $ip \in \mathcal{IP}_{as}$ innerhalb der Zeitspanne von $d$ Minuten eingeholt werden sollen. Die Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ dienen dazu, diese Subjekte $s$ festzulegen. Wenn nach dem Ablauf der Zeitspanne $d$ die einfache Mehrheit der Reaktion der Subjekte $s$ eine Zustimmung ist, wird das Prädikat als positiv ausgewertet betrachtet. Wenn es keine Mehrheit an Zustimmungen gibt, entsprechend als negativ.

[10] Wie sich in Abschnitt 9.2 und auch beim zugehörigen Admin–Netzwerk in Abschnitt 10.3.1 herausstellen wird, ist die Forderung, dass es nur eine einzige Belegung für die Variable $s$ geben darf, kritisch.

Bei den bisherigen Partizipationsprädikaten wurden alle Beteiligte gleich behandelt. In bestimmten Fällen erscheint es jedoch sinnvoll, einer Teilgruppe Sonderrechte einzuräumen, mit denen sie den initiierten Administrierungsprozess auf jeden Fall unterbinden kann. Dies kann mit Hilfe des folgenden Prädikates geschehen, das im Prinzip ein Quorum darstellt, jedoch bestimmten Beteiligten das Vetorecht einräumt.

**Definition 9.20 (Abstimmung)**
Die Regel für eine Abstimmung hat die Form

$$\textbf{vote(}as, ip, s, s_v, d\textbf{)} \leftarrow L_i$$

wobei $s, s_v \in \mathcal{V}_\mathcal{S}$ die Subjekte sind, deren Reaktionen ("Zustimmung", "Ablehnung" oder "Enthaltungen", für $s_v$ zusätzlich "Veto") innerhalb des Zeitraums von $d$ Minuten als Teil eines Administrierungsprozesses in der Autorisierungssphäre $as \in \mathcal{AST}$ unter der Bezeichnung $ip \in \mathcal{IP}_{as}$ eingeholt werden sollen. Die Feature Prädikate $L_i \in \mathcal{F}, 1 \leq i \leq n$ dienen dazu, die Subjekte $s$ und $s_v$ festzulegen. Wenn nach Ablauf der Zeitspanne $d$ von einem Subjekt $s_v$ ein "Veto" abgegeben wurde, ist die Auswertung des Prädikates negativ. Ansonsten ist die Auswertung positiv, wenn die einfache Mehrheit der Reaktion von den Subjekten $s$ und $s_v$ eine Zustimmung ist. Falls es keine Mehrheit an Zustimmung gibt, ist die Auswertung ebenfalls negativ.

Mit den folgenden Regeln wird in der Autorisierungssphäre VPN festgelegt, dass der Manager des Projektes VPN die Möglichkeit hat, neue Zuordnungen zu untergeordneten Rollen in diesem Projekt zu initiieren. Auf diese Zuordnung sollen innerhalb eines Zeitraums von z.B. 10 Stunden (600 Minuten) alle Developer im Projekt sowie die beiden Manager im übergeordneten Projekt Netzwerk und Crypto reagieren, wobei Letztere ein Vetorecht haben.

**Beispielregeln 9.3**

$$\begin{aligned}
\mathsf{assign}(\mathsf{VPN}, s_i, s, r, \mathsf{VPN}) \leftarrow & \mathsf{role}(s_i, \mathsf{manager}, \mathsf{VPN})\& \\
& \mathsf{subrole}(r, \mathsf{manager})\&\mathsf{vote}(\mathsf{VPN}, \mathsf{Assign}, s, s_v, 600) \\
\mathsf{vote}(\mathsf{VPN}, \mathsf{Assign}, s, s_v, 600) \leftarrow & \mathsf{role}(s, \mathsf{designer}, \mathsf{VPN})\& \\
& \mathsf{role}(s_v, \mathsf{manager}, u_v)\&
\end{aligned}$$

Die Menge der Partizipationsprädikate $\mathcal{PP}$ enthält, wie zu Beginn des Kapitels bereits erwähnt, die in diesem Abschnitt definierten Prädikate.

Nach einer erfolglosen Auswertung der Partizipationsprädikate kann nicht ohne weiteres festgestellt werden, weshalb es zu diesem Ergebnis gekommen ist. Auch können Details zum Ablauf dieses Prozesses nicht formal festgelegt werden. Im folgenden Kapitel wird mit Hilfe einer neuen Klasse von Petri–Netzen eine Möglichkeit geschaffen, diese Details formal zu spezifizieren.

# Kapitel 10

# Administrationsnetze

Bisher konnten die Vorgänge zur Administrierung des Autorisierungssystems nur in zwei Phasen aufgeteilt werden, die Initiierung eines Administrierungsvorganges und die Bestätigung durch weitere Mitglieder des verantwortlichen Administrierungsteams. Die Details bei einer Bestätigung lassen sich in den Definitionen der Prädikate nur informell beschreiben. Während Workflow–Netze weit verbreitet sind, um bestimmte Vorgänge zu modellieren, fehlen jedoch meistens die Möglichkeiten exakte Details über die Vorgehensweise und die Möglichkeit einer formalen Analyse. Petri–Netze werden schon lange dafür genutzt, um Abläufe zu modellieren und auch formal zu verifizieren. In diesem Kapitel wird eine neue Klasse von Petri–Netzen definiert, die auf die Bedürfnisse bei der Modellierung von Administrierungsvorgängen abgestimmt ist. Dazu gehören neben den beteiligten Subjekten, zeitliche Angaben wie beim Quorum oder Vote, aber auch Möglichkeiten, um die Reaktionen der Beteiligten zu repräsentieren.

Im folgenden Kapitel 11 werden die Administrationsnetze so erweitert, dass komplexe Abläufe modelliert werden können. Doch zunächst werden in Abschnitt 10.1 Administrationsnetze als eine spezielle Klasse von Petri–Netzen definiert. In Abschnitt 10.2 wird das Verhalten für einige Transitionen detaillierter erläutert, die speziell für die Modellierung von administrativen Vorgängen benötigt werden. Die Möglichkeiten von Administrationsnetzen werden in Abschnitt 10.3 demonstriert, indem inkrementell zu jedem der in Abschnitt 9.4 vorgestellten Partizipationsprädikate ein entsprechendes Administrationsnetz entwickelt wird.

## 10.1 Admin–Netze als Erweiterung von PrT–Netzen

Administrationsnetze oder kurz Admin–Netze werden als eine Erweiterung von Prädikat–Transitionsnetzen (PrT) [GL79] definiert. Zum einen werden Inhibitor Kanten [Val83] hinzugefügt, die es erlauben, auch das Fehlen eines bestimmten Tokens bei der Aktivierung mit einzubeziehen. Insbesondere ist die Koppelung von Tokens möglich, die in [SW78] vorgestellt wurde. Schließlich werden noch Zeitkonstraints zugelassen [GMMP91], die es erlauben, den Zeitraum, in dem Transitionen schalten können, einzuschränken.

Im Folgenden werden die einzelnen Elemente von Admin–Netzen definiert und mit kurzen Beispielen erläutert. Dabei wird auf bekannte Konzepte zurückgegriffen[1]. Grundlage für alle Petri–Netze ist ein gerichtetes Netz mit Stellen und Transitionen, wobei auf den Stellen sogenannte Tokens abgelegt werden können.

**Definition 10.1 (Gerichtetes Netz)**
Das Tripel $N = (S, T, F)$ ist ein gerichtetes Netz, bestehend aus einer Menge von Stellen $S$ und einer Menge von Transitionen $T$ sowie einer Flussrelation $F$. Hierbei gilt:

1. $S \cap T = \emptyset$ Die Mengen der Stellen und Transitionen sind disjunkt.

[1] vgl. [DJ01]

2. $S \cup T \neq \emptyset$ Das Netz ist nicht leer.

3. $F \subseteq S \times T \cup T \times S$. Die Flussrelation definiert Kanten von Stellen zu Transitionen und von Transitionen zu Stellen.

Für eine beliebige Transition $t$ werden alle Stellen $s \in F^{-1}(t)$ als **Eingangsstellen** von $t$ bezeichnet sowie alle Stellen $s \in F(t)$ als **Ausgangsstellen** von $t$.

Da sich RBAC und Petri–Netze bisher völlig unabhängig voneinander entwickelt haben, kommt es leider zu dieser Ähnlichkeit bei der Bezeichnung für Stellen $S$ und Subjekte $\mathcal{S}$. In dieser Arbeit wird aber nicht von der üblichen Namenskonvention abgewichen, sondern jeweils explizit angegeben, ob es sich im unmittelbaren Zusammenhang um eine Stelle $s \in S$ oder ein Subjekt $s \in \mathcal{S}$ handelt.

Die Elemente, die in diesen Netzen fließen sollen, werden meistens als Tokens, Marken, Farben bzw. gefärbte Marken/Tokens oder Objekte bezeichnet, wobei im Folgenden der Begriff **Token** verwendet wird. In der einschlägigen Literatur (z.B. [Jen96c]) kann die Bedeutung der einzelnen Tokens frei definiert werden und ergibt sich meistens aus den Zusammenhängen der jeweiligen Petri–Netze. Im Folgenden wird diese Bedeutung für Admin–Netze festgelegt.

**Definition 10.2 (Tokens)**
Für die Administrationsnetze werden die folgenden Tokens stets in einem bestimmten Kontext genutzt:

- Das Standardtoken $\bullet$ wird verwendet, um den Kontrollfluss zu modellieren.

- Die Administrationsnetze sollen zur Koordinierung bestimmter Entscheidungen dienen, daher werden die Tokens $\oplus$ und $\ominus$ verwendet, um Zustimmung bzw. Ablehnung zu repräsentieren. Eine Enthaltung wird mit dem Token $\circ$ ausgedrückt. Wenn bei einem Koordinierungsprozess nicht rechtzeitig bis zu einem angegebenen Zeitpunkt eine Entscheidung (von einzelnen Beteiligten) getroffen wird, wird dies durch ein Token $\textcircled{T}$ verdeutlicht, das ein Time-Out repräsentiert. In besonderen Fällen kann auch ein Veto ausgesprochen werden, das durch $\oslash$ dargestellt wird. Dabei wird $O_S := \{\bullet, \oplus, \ominus, \circ, \textcircled{T}, \oslash\}$ auch als Menge der **Koordinierungstokens** bezeichnet.

- Um die verschiedenen beteiligten Benutzer in einem Koordinierungsprozess zu repräsentieren, werden sogenannte **Administratortokens** verwendet. Dabei werden die als Administratoren beteiligten Benutzer $a_1 \ldots, a_n$ durch die Tokens $\textcircled{a_1} + \ldots + \textcircled{a_n}$ im Admin–Netz repräsentiert. Die Menge $O_A$ enthält alle in einem Admin–Netz verwendeten Administratortokens.

- In komplexen Prozessen müssen, insbesondere wenn es um die Änderungen des Regelwerkes geht, die verschiedenen Vorschläge $p$, $p_1$, $p_2, \ldots, p_k$ auch im Admin–Netz repräsentiert werden. Dazu werden sogenannte **Proposaltokens** verwendet $\textcircled{p}\,\textcircled{p_1}\,\textcircled{p_2} \ldots \textcircled{p_k}$. Diese verweisen auf den entsprechenden Vorschlag, wobei Tokens mit unterschiedlichen Indizes niemals auf den gleichen Vorschlag verweisen[2] jedoch keine Angaben über den Inhalt machen. Die Menge $O_P$ enthält alle in einem Admin-Netz verwendeten Proposaltokens.

- alle bisher aufgezählten Tokens können als Paar oder Triple gekoppelt werden und bilden ein neues Token.

Alle oben beschriebenen Tokens sind in der Menge $O'$ enthalten, $O' = O_S \cup O_A \cup O_P$. Die Menge aller Tokens eine Administrationsnetzes $O$ enthält zudem noch die möglichen Koppelungen $O = O' \cup (O' \times O') \cup (O' \times O' \times O')$

Die Koppelung der Tokens wird durch eine Klammerung der entsprechenden Tokens dargestellt, wodurch implizit neue Tokens entstehen. Die Reihenfolge der so gekoppelten Tokens ändert sich nicht, sodass $(\bullet, \oplus)$ und $(\oplus, \bullet)$ zwei verschiedene Tokens sind. Ein Anwendungsbeispiel für die

[2]Falls ein inhaltlich identischer Vorschlag mehrfach eingereicht wurde, sei es vom gleichen oder verschiedenen Beteiligten, handelt es sich aus Sicht des Admin–Netzes um unterschiedliche Vorschläge.

im letzten Punkt erwähnte Kopplung wäre die Möglichkeit, das Abstimmungsverhalten der Benutzer zu einem Vorschlag zu dokumentieren. Falls der Administrator $a_3$ dem Vorschlag $p_5$ positiv gegenübersteht, kann dies durch die Kopplung $(a_3, p_5, \oplus)$ dargestellt werden.

Die Tokens werden auf den Stellen abgelegt und kodieren so verschiedene Informationen z.B. die Zustimmung zu einem Vorschlag. Die Belegung aller Stellen zu einem Zeitpunkt wird zu einer Markierung zusammengefasst.

**Definition 10.3 (Markierung)**
Eine **Markierung eines gerichteten Netzes N** mit Stellenmenge $S$ ist eine Abbildung $M : S \rightarrow (\mathbb{N}^{|O|})^n$, bei der jeder Stelle aus $S$ eine Multimenge über die Menge der Objekte $O$ zugeordnet wird.

Im Gegensatz zu einfachen Petri–Netzen (sogenannten Stellen-Transitionsnetzen vgl. [Pet62, Pet66]) kann das Schalten einer Transition bei Prädikat–Transitions–Netzen und damit auch bei Administrationsnetzen an komplexere Bedingungen geknüpft werden. Diese werden sowohl durch die Kantenanschriften, als auch durch die Inschriften in den Transitionen spezifiziert.

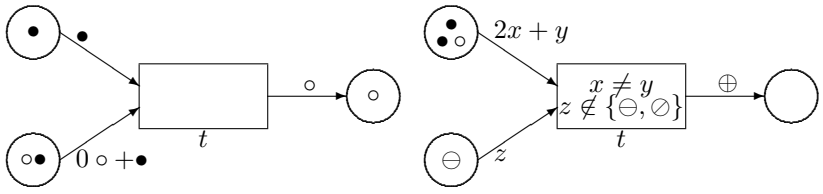
**Definition 10.4 (Kantenbeschriftung und Transitionsinschriften)**
Eine Kantenbeschriftung ist eine Abbildung $L : F \rightarrow \sum$, die jeder Kante eine formale Summe über die Menge der Token–Objekte $O$ und der Variablen über die Token–Objekte zuordnet, inklusive einer expliziten 0 Summe, die auch als Inhibitor bezeichnet wird. Eine Transitionsinschrift ist eine Abbildung $I : T \rightarrow FS$, die jeder Transition eine Formel aus $FS$ zuordnet.
Jede Variable, die in einer Kantenbeschriftung zu einer Ausgangstelle oder in einer Transitionsinschrift einer Tranistion $t \in T$ verwendet wird, muss auch in einer Kantenbeschriftung einer Eingangsstelle von $t$ verwendet werden.

Durch die Verwendung der Variablen in den Eingangsstellen soll sichergestellt werden, dass es stets eine klar definierte Bindung der Variablen in den Transitionsinschriften und den Kantenbeschriftungen zu den Ausgangstellen gibt. Diese Belegungen sind immer lokal zu einer Transition zu sehen.

Die Menge der Formeln $FS$ umfasst zunächst[3]

- den Test auf Gleichheit bzw. Ungleichheit zweier Variablen oder einer Variable und einem gegebenen Token,
- den Test auf Enthaltung bzw. Nichtenthaltung einer Variable in einer vorgegebenen Menge,
- die Temporaloperatoren *Until* und *After* (Def. 10.8),
- Regeln mit Feature Prädikaten (Def. 10.9).

Mehrere Ausdrücke innerhalb einer Transition werden logisch als und-verknüpft angesehen.




(a) Inhibitorkante (b) Transitionsinschriften

Abbildung 10.1: Beispiel für Kantenbeschriftungen und Transitionsinschriften

Durch eine Inhibitor–Beschriftung wird festgelegt, dass kein Token einer bestimmten Art auf der Stelle liegen soll. Diese Inhibitor–Beschriftung kann auch zusammen mit anderen "regulären"

[3]In den Abschnitten 11.2.1 und 11.3.2 werden noch weitere Formeln hinzugefügt.

Kantenbeschriftungen verwendet werden. In Abbildung 10.1(a) ist ein Beispiel für eine Kantenbeschriftung mit einem Inhibitor zu sehen. Während in Abbildung 10.1(b) ein Beispiel für eine Kombination aus Variablen als Kantenbeschriftung und Transitionsinschriften zur genaueren Bestimmung dieser Variablen zu sehen ist.

Im Folgenden wird die Aktivierung der Transitionen beschrieben und wie der Schaltvorgang genau aussieht. Die besonderen Transitionen mit Temporaloperatoren und Regeln mit Feature Prädikaten werden im nächsten Abschnitt genauer erläutert.

**Definition 10.5 (Aktivierung einer Transitionen)**
Eine Transition $t \in T$ ist bei einer Markierung $M$ genau dann aktiviert, wenn es für die vorhandenen Variablen in den Kantenbeschriftungen von den Eingangsstellen von $t$ eine gültige Bindung hinsichtlich der Transitionsinschrift gibt, sodass auf jeder Eingangsstelle genügend Tokens liegen, mit denen die Kantenbeschriftungen erfüllt sind. Wenn die Kanteninschrift keine Variablen enthält, kann dies als leere Bindung aufgefasst werden. Ein Inhibitor bei einer Kanteninschrift bewirkt, dass kein entsprechendes Token auf dieser Eingangsstelle liegen darf, damit eine Aktivierung der Transition erfolgt. Es ist möglich, dass es mehrere alternative Bindungen einer Variablen gibt, mit denen die Transition $t$ aktiviert ist. Wenn sich die Zahl der Bindungen oder die der Tokens auf den Eingangsstellen verändert, bleibt die Transition weiterhin aktiviert, solange es mindestens eine gültige Bindung gibt und alle Kantenbeschriftungen und die Transitionsinschrift erfüllt sind.

Die Einführung einer leeren Bindung erfolgt, um nicht stets eine Fallunterscheidung für Transitionen mit Variablen in den Kantenbeschriftungen und solchen ohne Variablen vornehmen zu müssen.

Die beiden Transitionen in Abbildung 10.1 sind nicht aktiviert, da zum einen die Inhibitor Bedingung nicht erfüllt ist, zum anderen die Variable $z$ nicht an das Token $\ominus$ gebunden werden darf.

**Definition 10.6 (Schalten einer Transition)**
Wenn eine Transition $t \in T$ mit einer bestimmten Bindung aktiviert ist, schaltet diese Transition, sofern keine Temporaloperatoren als Inschriften vorhanden sind, unmittelbar. Beim Schalten unter einer bestimmten Bindung werden gemäß der Kanteninschrift von den Eingangsstellen, die Tokens von den Eingangsstellen entfernt und Tokens gemäß der Kanteninschrift zu den Ausgangsstellen auf die Ausgangsstellen gelegt.

In Abbildung 10.2(a) ist die gleiche Transition wie in Abbildung 10.1(a) zu sehen, diesmal jedoch mit einer anderen Markierung, sodass die Transition aktiviert ist. Sobald die Transition schaltet, werden die Tokens gemäß der Kantenbeschriftung an den Eingangsstellen von diesen entfernt und entsprechend der Kantenbeschriftung zu den Ausgangsstellen Tokens auf diesen abgelegt.

Für die Markierung in Abbildung 10.2(b) bzw. 10.2(c) gibt es zwei mögliche Bindungen der Variablen $x$ und $y$. Bei der Abbildung 10.2(b) hat die Transition mit der Bindung $x = \bullet$,$y = \circ$ und $z = \oplus$ geschaltet, bei Abbildung 10.2(c) mit der Bindung $x = \circ, y = \bullet, z = \oplus$.

Die obigen Beispiele dienen zur Veranschaulichung von bestimmten Schaltverhalten. Ein Administrationsnetz ist wie folgt definiert.

**Definition 10.7 (Administrationsnetz (Admin–Netz))**
Ein Administrationsnetz (Admin–Netz) ist ein Quadrupel $(N, I, L, M_0)$, wobei $N = (S, T, F)$ ein gerichtetes Netz ist, $I$ die Inschriften der Transitionen $S$, $L$ eine Kantenbeschriftung für alle Kanten in $F$. $M_0$ ist die initiale Markierung des Admin–Netzes.

Auf die besonderen Transitionen, die zur Modellierung der Administrierungsprozesse benötigt werden, wird im Folgenden detailliert eingegangen.

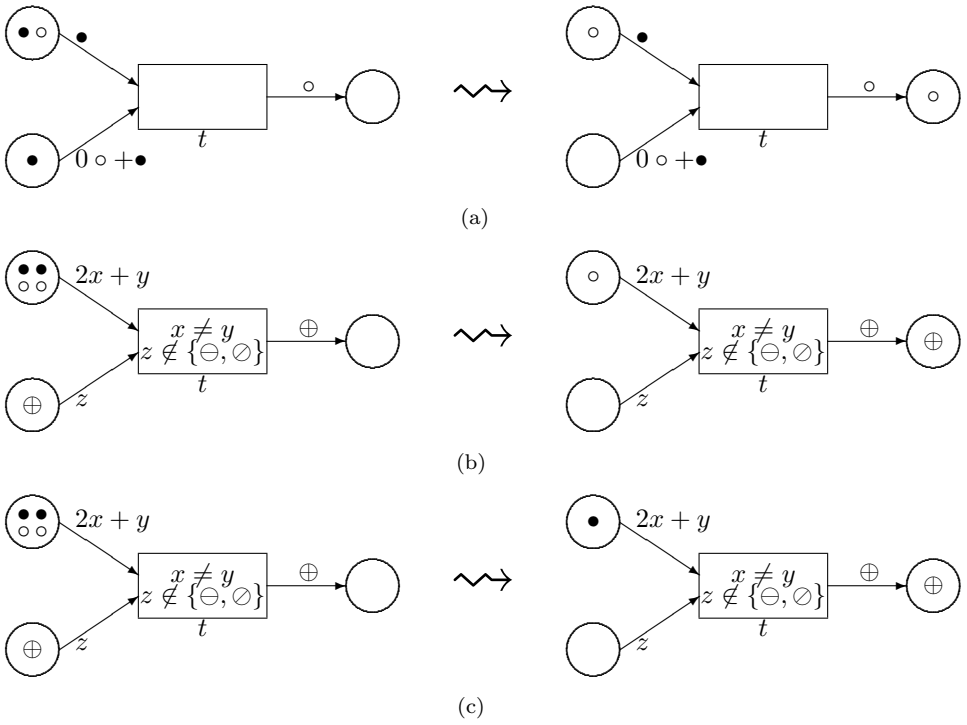
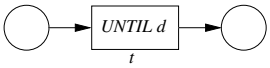


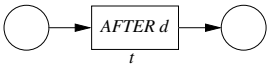

Abbildung 10.2: Beispiele für das Schalten von Transitionen

## 10.2 Besondere Transitionen in Admin–Netzen

Für die Spezifizierung von Koordinierungsprozessen wird ein bestimmtes Verhalten für eine Reihe von Transitionen angenommen. Daher wird dies nun detaillierter an Beispielen betrachtet, bevor im nächsten Abschnitt vollständige Admin–Netze modelliert werden. Neben der zeitlichen Einschränkung eines möglichen Zeitraumes, in dem eine Transition schalten kann, wird mit diesen Transitionen insbesondere die Einbindung der Administratoren in den Koordinierungsprozess modelliert.




(a) die Transition kann nach der Aktivierung **bis** zum Ablauf des Zeitraums $d$ schalten




(b) die Transition kann nach der Aktivierung erst **nach** Ablauf des Zeitraums $d$ schalten

Abbildung 10.3: Notationen für Timed Transition

Transitionen mit Temporaloperatoren, wie sie in Abbildung 10.3 zu sehen sind, verhalten sich nach der Aktivierung wie folgt:

**Definition 10.8 (Timed Transition)**
Falls die Transition $t$ den Temporaloperator *Until* $d$ enthält, kann die Transition nach der Aktivierung zum Zeitpunkt $\tau$ nur bis zum Zeitpunkt $\tau + d$ schalten. Falls die Transition $t$ den Temporaloperator *After* $d$ enthält, kann sie nach der Aktivierung zum Zeitpunkt $\tau$ erst nach dem Zeitpunkt $\tau + d$ schalten. Falls die Transition bei einer Markierung $M$ aktiviert ist und auch in der direkten Folgemarkierung $M'$ aktiviert ist, wird der Zeitpunkt nicht neu berechnet. Sobald jedoch eine Markierung $M''$ erreicht wird, bei der die Transition nicht aktiviert ist, gilt für alle Berechnungen der Zeitpunkt, bei dem eine

Markierung $M'''$ erreicht wird, in der die Transition erneut aktiviert ist. Sofern nicht anders vermerkt, werden alle Zeitangaben in Minuten vorgenommen.

Die Spezifizierung des Zeitraums $d$ für die Entscheidung der Administratoren in Minuten erscheint für die Koordinierungsprozesse adäquat[4].

Die in Abschnitt 9.4 vorgestellten Prädikate stellen einen Koordinierungsprozess dar, der mit einer beliebigen Besetzung von Benutzern in entsprechenden Rollen in einer Organisationseinheit durchgeführt werden kann. Wie in Abschnitt 9.2 werden diese Beteiligten durch die Auswertung der entsprechenden Feature Prädikate bestimmt. Diese Beteiligten werden gemäß Definition 10.2 durch sogenannte Administratortokens in den Administrationsnetzen repräsentiert. Eine Möglichkeit, diese Administratortokens in ein Administrationsnetz einzubringen, wäre durch eine Anfangsmarkierung $M_0$. Falls die Benutzer bei komplexeren Prozessen nur unter bestimmten Bedingungen beteiligt werden, wäre dies relativ komplex, da die Tokens auf mehrere Stellen gelegt werden müssen. Um diese zusätzlichen Randbedingungen bei der Definition von Administrationsnetzen zu vermeiden, werden sogenannte Feature Transitionen eingeführt, die beim Schalten eine Prädikatregel auswerten und die entsprechende Administratortokens auf ihren Ausgangsstellen erzeugen.

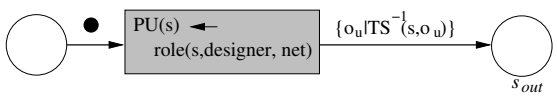
Um die Admin–Netze tatsächlich zur Steuerung der Koordinierungsprozesse einsetzen zu können, werden während des Ablaufs auch Informationen benötigt, welcher Benutzer/Subjekt durch ein bestimmtes Administratortoken repräsentiert wird. Daher wird eine Abbildung $TS : O_A \rightarrow \mathcal{S}$ von Administratortokens $O_A$ auf Subjekte $\mathcal{S}$ definiert. Die hierzu inverse Abbildung $TS^{-1} : \mathcal{S} \rightarrow O_A$ bildet jedes beteiligte Subjekt $s \in \mathcal{S}$ auf ein Token $o \in O_A$ ab. Die Einschränkung auf die beteiligten Subjekte erlaubt es, die Menge der Administratortokens relativ klein zu halten.

**Definition 10.9 (Feature Transition)**
Eine Feature Transition $t_F$ ist eine ausgezeichnete (grau ausgefüllte) Transition, die zusätzlich zu anderen Transitionsinschriften gemäß Definition 10.4, eine Regel der Form **PU(**$s$**)** $\leftarrow L_i$ enthält, wobei $s \in \mathcal{V}_\mathcal{S}$ die Subjekte bezeichnet, die am Prozess beteiligt werden sollen und die durch die Feature Prädikate $L_i \in \mathcal{F}$ charakterisiert werden. Bei der Schaltung einer Feature Transition $t_F$ werden alle möglichen Bindungen für $s \in \mathcal{V}_\mathcal{S}$ berechnet und alle entsprechenden Administratortokens gemäß der Abbildung $TS$ auf die Stellen $s_{out} \in S$ gelegt, deren Kanten von $t_F$ nach $s_{out}$ die Kantenbeschriftung $\{o_a | TS^{-1}(s, o_a)\}$ aufweisen.
Wenn es keine einzige gültige Bindung für Variable $s$ gibt, wird mit einer Fehlermeldung abgebrochen. Wenn die genaue Bestimmung der beteiligten Subjekte nicht von Bedeutung ist, kann die Feature Transition $t_F$ als Kurzform nur die Inschrift **PU**(s) enthalten und die entsprechenden Kanten zu den Ausgangsstellen erhalten die Beschriftung $a_1 + \cdots + a_n$. Falls es mehrere Feature Transitionen in einem Netz gibt, korrespondieren die Variablen in **PU** mit denen der Prädikate.

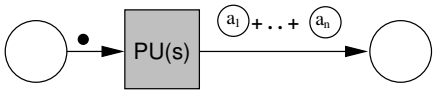
Die Abbildung 10.4(a) zeigt eine Feature Transition $t_F$, die beim Schalten Administratortokens für alle Subjekte, die die Rolle Developer in der Organisationseinheit Netzwerk haben, auf die Ausgangsstelle $s_{out}$ legt. Da bei der Entwicklung der Admin–Netze meistens nicht unbedingt die konkret beteiligten Subjekte von Interesse sind, sondern diese möglichst flexibel gehalten werden sollen, wird im folgenden Abschnitt 10.3 die vereinfachte Form von Feature Transitionen benutzt, wie sie in Abbildung 10.4(b) zu sehen ist.

Falls die Administratoren, die an einem Koordinierungsprozess teilnehmen sollen, bekannt sind und damit deren Administratortokens, die beim Durchlauf eines Admin–Netzes benutzt werden, kann eine Feature Transition wie in Abbildung 10.4(c) auch durch eine normale Transition ersetzt werden. Hierbei stehen dann als Kantenbeschriftung zur Ausgangsstelle $s_{out}$ die entsprechenden Administratortokens. In den meisten Fällen sollte es zur Verifizierung und Analyse der Administrationsnetze ausreichen, die Anzahl der Administratortokens zu kennen. Es macht keinen Sinn, ein Administratortoken als Kantenbeschriftung für eine Eingangskante einer Transition zu verwenden, da dann die Abbildung $TS$ im Vorfeld bekannt sein müsste.

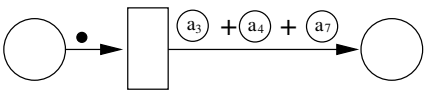
[4]Eine Einteilung in Sekunden erscheint für diese Vorgänge unnötig. Hingegen eine Dauer im Stundenbereich zwingend festzulegen, würde jede Dynamik aus komplexeren Koordinierungsprozessen, wie sie in Kaptiel 11 beschrieben werden, nehmen.




(a) vollständige Feature Transition




(b) vereinfachte Feature Transition




(c) Ersatz für Feature Transition

Abbildung 10.4: Feature Transition

Da Admin–Netze dazu dienen, Koordinierungsprozesse zu modellieren, müssen einzelne Entscheidungen eines beteiligten Administrators in der Netzstruktur dargestellt werden. Dies geschieht mit sogenannten *Interaktiven Transitionen*, bei denen der Administrator bestimmen kann, welche der aktivierten (interaktiven) Transitionen schaltet. Durch die Tokens, die dann beim Schalten auf die Ausgangsstellen gelegt werden, nimmt der Administrator Einfluss auf den Koordinierungsprozess.

**Definition 10.10 (Interaktive Transitionen)**
Interaktive Transitionen sind eine Menge von Transitionen, die alle über $n \geq 1$ gemeinsame Eingangstellen $s_{(in_1)}, \ldots, s_{(in_n)}$ verfügen und deren Zusammengehörigkeit durch ein hinterlegtes graues Feld visualisiert wird. Die entsprechende Kantenbeschriftung von dieser Eingangsstelle bzw. diesen Stellen zu den Interaktiven Transitionen, enthält genau eine Variable $a$ für ein Administratortoken sowie u.U. weitere Tokens bzw. deren entsprechenden Variablen.
Wenn bei einer Markierung $M$ eine Interaktive Transition mit einer Bindung aktiviert wird, stellt die Kantenbeschriftung sicher, dass auch alle anderen Tranistionen aus dieser Menge mit dieser Bindung aktiviert sind. Wenn die Interaktiven Transitionen aktiviert sind, werden alle Administratoren, deren Administratortoken auf der Eingangsstelle $s_{in}$ liegen und als eine mögliche Bindung der Variable $a$ an der Aktivierung der Interaktiven Transition beteiligt sind, benachrichtigt. Dabei werden gegebenenfalls weitere mögliche Bindungen von Variablen außer $a$ und temporale Bedingungen mitgeteilt. Jeder Administrator kann dann entscheiden, welche der Interaktiven Transitionen schalten soll. Beim Schalten wird die Bindung, die das Administratortoken enthält deren Entscheidung ausgeführt wird, verwendet. Falls zwischen den Interaktiven Transitionen ein wechselseitiger Ausschluss besteht und direkt nach dem Schalten einer Transition die Interaktiven Transitionen wieder aktiviert sind, eine Ausgangsstelle also gleichzeitig auch Eingangsstelle ist, werden die Administratoren nicht erneut benachrichtigt.

In Abbildung 10.5(a) ist ein Beispiel für Interaktive Transitionen zu sehen. Sobald die Interaktiven Transitionen *reject*, *neutral* und *accept* aktiviert sind, werden die drei Subjekte, die durch die auf der Stelle $s_{in}$ liegenden Administratortokens repräsentiert werden, benachrichtigt. Jedes Subjekt kann entscheiden, welche der drei Interaktiven Transitionen schalten soll, wobei beim Schalten das zum Subjekt gehörende Administratortoken von $s_{in}$ entfernt wird und abhängig von der schaltenden Transition eines der Tokens $\ominus, \circ$ oder $\oplus$ auf die Stelle $s_{out}$ gelegt wird. Letztlich werden also drei Tokens auf $s_{out}$ liegen, deren Zusammensetzung davon abhängig ist, welche Interaktive Transition gemäß der Entscheidung der beteiligten Subjekte geschaltet hat.

Eine Besonderheit bei Interaktiven Transitionen tritt ein, wenn zwischen den Interaktiven Transitionen ein wechselseitiger Ausschluss vorliegt (vgl. [GV03, S. 15ff] bzw. [Thi87]), wie es in Abbildung 10.5(b) bei einer Belegung der Stelle *proposal* mit nur einem Proposaltoken zu sehen ist. Da dort alle Transitionen nach dem Schalten einer Interaktiven Transition auch in der nachfolgenden Markierung aktiviert sind, wird dies nicht als erneute Aktivierung aufgefasst, bei der alle Administratoren benachrichtigt werden, sondern die Aktivierung bleibt bestehen. Dies hat dann auch die entsprechenden Konsequenzen, wenn die Transitionen zusätzliche Temporal Operatoren enthalten, die dann gemäß Definition 10.8 die Zeitpunkte nicht verändern.

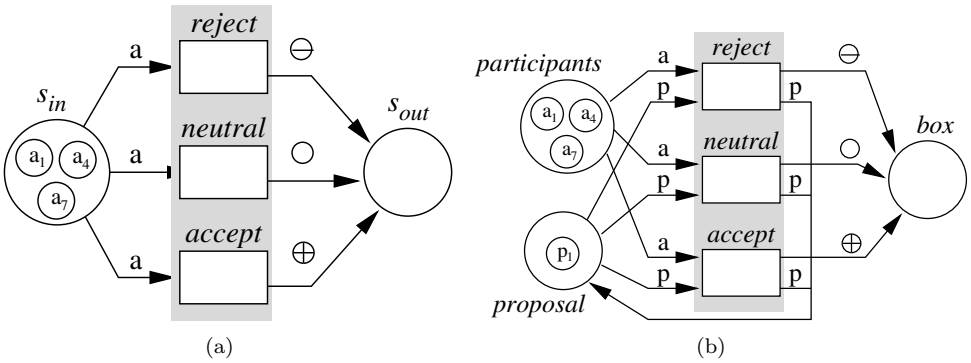



Abbildung 10.5: Interaktive Transitionen

## 10.3 Formale Spezifizierung von Partizipationsprädikate durch Administrationsnetze

Die Vorgehensweise bei den Partizipationsprädikaten wurde im Abschnitt 9.4 informell beschrieben, nun wird zu jedem Prädikat ein entsprechendes Admin–Netz präsentiert, dass insbesondere bei den komplexen Vorgehensweisen wesentlich exakter ist. Diese Admin–Netze verfügen über eine Stelle *in*, auf die zu Beginn der Auswertung ein Token • liegt, alle anderen Stellen sind leer. Als Ergebnis des Abstimmungsprozesses wird auf der Stelle *out* entweder das Token ⊕ liegen, was als Zustimmung zu verstehen ist. Damit gilt die Auswertung des dazugehörigen Prädikates als positiv. Bei allen anderen Tokens auf der Stelle *out* gilt die Auswertung als negativ, im einfachsten Fall zeigt das Token ⊖ an, dass die beteiligten Subjekte den Vorschlag abgelehnt haben. Mögliche weitere Tokens und die damit verbunden Gründe werden bei den jeweiligen Admin–Netzen vorgestellt.

### 10.3.1 Approve

Beim ersten Koordinierungsprozess wird die Zustimmung eines einzelnen Subjektes zu einem Vorschlag eingeholt.

Das Admin–Netz in Abbildung 10.6 soll dies formal spezifizieren. Bei der Anfangsmarkierung $M_0$ liegt genau ein Standardtoken auf *in*. Damit ist die Feature Transition *init* aktiviert. Beim Schalten werden die Feature Prädikate der Regel *PU* ausgewertet, für deren Belegung es gemäß der Definition 9.17 genau ein Subjekt gibt, dessen Administratortoken dann auf die Ausgangsstelle gelegt wird. Dieses Subjekt wird bei der Aktivierung der Interaktiven Transitionen *reject* und *accept* benachrichtigt und kann nun bestimmen, welche dieser Transitionen schaltet. Abhängig von dieser Entscheidung wird das Token ⊖ oder ⊕ auf die Stelle *out* gelegt.

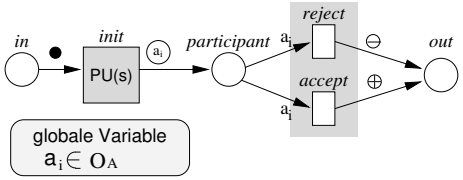



Abbildung 10.6: Approval

Problematisch ist bei diesem Admin–Netz lediglich, dass garantiert sein muss, dass nur ein Subjekt die Bedingungen der Feature Transitionen erfüllt. Ansonsten würden beim Schalten der Transition *init* mehrere Administratortokens auf die Stelle *participant* gelegt, da die Definition 10.9

fordert, dass alle möglichen Bindungen berechnet werden. Diese Definition zu ändern oder eine zweite Art von Feature Transition zu definieren, wäre eine Möglichkeit das vorliegende Problem zu lösen. Eine andere Möglichkeit ist es, durch entsprechende Regeln bei der Rollenzuweisung sicherzustellen, dass nur jeweils ein Subjekt bestimmte Rollen in den Organisationseinheiten innehat. Dieses würde aber in größeren Organisationen einen erhöhten Koordinierungsaufwand bedeuten, um dies sicherzustellen, auch wenn es durch entsprechende Ergänzungen bei den Regeln gewährleistet werden könnte.

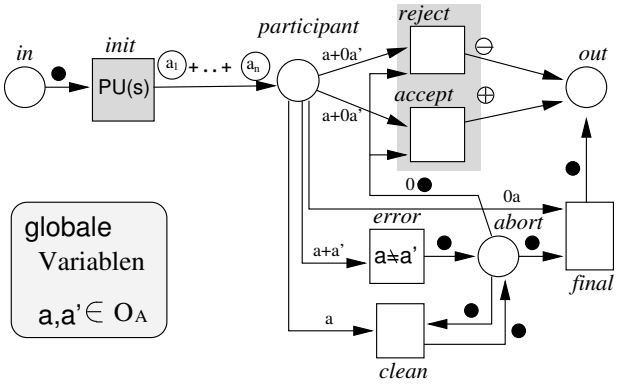



Abbildung 10.7: Approval (abgesichert)

An dieser Stelle erscheint es jedoch sinnvoll statt des Admin–Netzes in Abbildung 10.6 für das Prädikat *approve* das Admin–Netz aus Abbildung 10.7 zu verwenden. Dessen zusätzliche Transitionen gegenüber dem Vorgänger in Abbildung 10.6 bewirken, dass im Fall von $n > 1$ Administratoren den Prozess zu einem Ende zu führen, der diesen Fehler deutlich macht.

Bei der Markierung $M_0$ liegt ein Standardtoken auf *init* und alle anderen Stellen sind leer. Bei der Markierung $M_\oplus$ sind alle Stellen bis auf *out* leer, auf dem ein Token $\oplus$ liegt. Bei der Markierung $M_\ominus$ ist es entsprechend ein $\ominus$. Die Markierung $M_\bullet$ soll den Fehlerfall darstellen bei dem ein $\bullet$ auf *out* liegt und alle anderen Stellen leer sind.

**Satz 10.1**
Ausgehend von der Markierung $M_0$ erreicht das Admin–Netz in Abbildung 10.7 stets genau eine der drei Markierungen $M_\oplus$, $M_\ominus$ oder $M_\bullet$, unabhängig von der Anzahl $n \geq 1$ der erzeugten Administratortokens aber abhängig davon, welche Interaktive Transition schaltet.

**Beweis zu Satz 10.1**
Bei der Markierung $M_0$ ist nur die Feature Transition *init* aktiviert. Beim Schalten werden $n$ Administratortokens auf *participant* abgelegt.

Falls $n = 1$ sind die beiden Interaktiven Transitionen *reject* und *accept* aktiviert und schalten nach Vorgabe des Administrators. Dabei wird das Administratortoken von *participant* entfernt und entsprechend ein Token $\ominus$ bzw. $\oplus$ auf *out* platziert. Damit ist die Markierung $M_\ominus$ oder $M_\oplus$ erreicht.

Falls $n > 1$ ist nur die Transition *error* aktiviert und entfernt beim Schalten zwei Administratortokens von *participant* und platziert ein Standardtoken auf *abort*. Aufgrund der Inhibitorkante sind *reject* und *accept* auch dann nicht mehr aktiviert, wenn sich nur noch ein Administratortoken auf *participant* befindet.
Die Transition *clean* kann nun $n-2$ mal schalten und jeweils ein weiteres Administratortoken von *participant* entfernen, dabei wird das Standardtoken auf *abort* jedes Mal reproduziert.
Sobald *participant* leer ist, ist *final* aktiviert und verschiebt beim Schalten das Standardtoken von *abort* nach *out*. Damit liegt die Markierung $M_\bullet$ vor. □

Damit ist nicht nur der eingangs beschriebene Koordinierungsprozess, sondern auch das Verhalten des Partizipationsprädikates approve$(as, ip, s)$ aus Definition 9.17 formal spezifiziert. Dieses Prädikat ist genau dann gültig, wenn bei der Durchführung eines Koordinierungsprozesses, mit dem durch die Feature Prädikate festgelegtem Subjekt als Administrator, die Markierung $M_\oplus$ erreicht wird. Wenn die Markierung $M_\ominus$ erreicht wird ist der Vorschlag abgelehnt und das Prädikat damit ungültig. Bei der Markierung $M_\bullet$ liegt ein Fehler vor.

### 10.3.2 Multi Approve

Anstelle einer Zustimmung zu einem Administrierungsvorschlag durch einen einzelnen Administrator kann auch die Zustimmung einer Gruppe von Administratoren, die ein Administrationsteam bilden, erforderlich sein. Hierzu müssen zunächst alle beteiligten Administratoren bestimmt werden. Diese müssen dann entscheiden, ob sie dem Administrierungsvorschlag zustimmen oder ihn ablehnen. Die Ablehnung durch einen einzigen beteiligten Administrator soll hier ausreichen, damit der Vorschlag insgesamt als abgelehnt angesehen wird.

Das Admin–Netz in Abbildung 10.8 soll diesen Koordinierungsprozess formal spezifizieren.

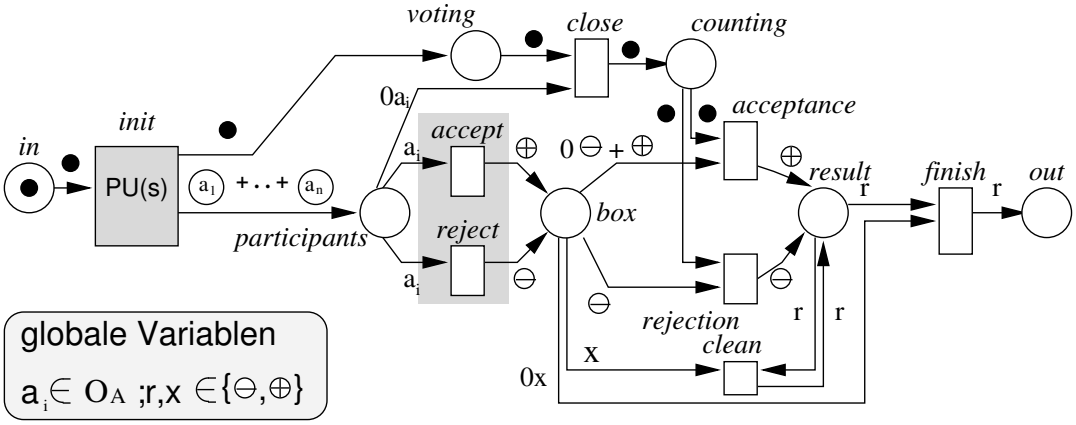



Abbildung 10.8: Multi Approve

Es seien $\textcircled{a_1}, \ldots, \textcircled{a_n}$ Administratorentokens, $n \in I\!N$. Es wird vorausgesetzt, dass die Administratortokens die $n$ am Prozess beteiligten Administratoren repräsentieren und die Variable $a$ mit einem beliebigen Administratortoken belegt werden kann. Für die Bindung der Variablen $x$ und $r$ gilt $x, r \in \{\oplus, \ominus\}$.

Darüber hinaus wird angenommen, dass die Administratoren bei der Aktivierung der Interaktiven Transitionen *accept* und *reject* über den Inhalt des Administrierungsvorschlages informiert worden sind. Der Beginn des Koordinierungsprozesses wird durch die Anfangsmarkierung $M_0$ repräsentiert, bei der genau ein Standardtoken auf der Stelle *in* liegt und alle anderen Stellen leer sind. Erreichbare Markierungen sind zum einen $M_\ominus$, bei der ein Token $\ominus$ auf der Stelle *out* liegt und alle anderen Stellen sind leer, zum anderen die Markierung $M_\oplus$, bei der ein Token $\oplus$ auf der Stelle *out* liegt und alle anderen Stellen sind leer. Es gilt:

**Satz 10.2**
Der Koordinierungsprozess, der durch das Admin–Netz in Abbildung 10.8 spezifiziert wird, erreicht ausgehend von dieser Anfangsmarkierung $M_0$ stets eine der beiden Markierungen $M_\ominus$ oder $M_\oplus$, die die beiden möglichen Abschlüsse des Prozesses darstellen.
Durch die Markierung $M_\ominus$ wird eine Ablehnung des Administrierungsvorschlages dargestellt, wenn mindestens einmal die Interaktive Transition *reject* geschaltet hat. Durch die Markierung $M_\oplus$ wird eine Annahme des Administrierungsvorschlages dargestellt, wenn von den Interaktiven Transitionen ausschließlich *accept* geschaltet hat.

Bei der Markierung $M_\oplus$ bzw. $M_\ominus$ ist keine Transition im Admin–Netz aus Abbildung 10.8 mehr aktiviert. Die Anzahl der erzeugten Administratortokens $n \geq 1$ kann beliebig sein.

**Beweis zu Satz 10.2**

1. Beim Schalten der Feature Transition *init* wird gemäß der Definition 10.9 für jeden Administrator ein Administratortoken auf die Stelle *participants* gelegt, insgesamt $n$. Gleichzeitig wird beim Schalten dieser Transition auch ein Standardtoken auf die Stelle *voting* gelegt. Aufgrund der Inhibitor–Kante von der Stelle *participants* zur Transition *close* wird diese erst aktiviert, wenn sich keine Tokens mehr auf dieser Stelle befinden. Die andere Eingangsstelle *voting* setzt voraus, dass zuvor die Transition *init* geschaltet haben muss und genau ein Standardtoken auf dieser Stelle liegt. Dadurch schaltet *close* genau einmal.

2. Jedes Token $\textcircled{a_1}, \ldots, \textcircled{a_n}$ wird beim Schalten von *accept* oder *reject* entfernt und ein $\oplus$ bzw. $\ominus$ auf *box* gelegt. (Jede Administratorentscheidung wird durch ein Token auf *box* repräsentiert.) Die Transitionen *accept* und *reject* schalten zusammen $n$-mal. Währenddessen wird keine andere Transition aktiviert. Danach liegen auf *box* $n$ Tokens $x$, $x \in \{\oplus, \ominus\}$, solange *close* noch nicht schaltet.

3. *Acceptance* und *recjection* können höchstens einmal schalten, nachdem *close* geschaltet hat. Wegen der Anschriften an den entsprechenden Kanten schaltet sogar gerade eine der beiden Transitionen. Wenn *reject* einmal schaltet, wird ein $\ominus$ auf *result* liegen. Wenn *reject* nicht geschaltet hat, wird das Token $\oplus$ auf *result* liegen. Auf *result* befindet sich entweder ein $\oplus$ oder ein $\ominus$. Auf *box* liegen dann noch $n - 1$ Token.

4. Die einzige dann aktivierte Transition ist *clean*, solange noch ein Token auf *box* liegt. *finish* schaltet erst, wenn alle Stellen bis auf *result* leer sind. Die Markierung auf *result* wird bei jedem Schalten reproduziert. Nachdem *box* von Token frei ist, ist nur noch *finish* aktiviert. Nach dem Schalten von *finish* liegt genau ein Token auf *out*. Die anderen Stellen sind unmarkiert. Da jede Transition mit einer Inhibitorkante auch eine reguläre Kante hat, ist keine Transition aktiviert. □

Gemäß der Definition der Feature Transitionen (Def. 10.9) wird beim Schalten für jeden beteiligten Administrator ein Administratortoken auf die entsprechende Stelle abgelegt. Bei der Aktivierung der Interaktiven Transitionen *accept* und *reject* kann nach Definition 10.10 jeder Administrator entscheiden, welche der beiden Transitionen mit "seinem" Administratortoken schalten soll. Entsprechend wird diese Entscheidung mit einem Token $\oplus$ bzw. $\ominus$ auf *box* repräsentiert.

Damit ist gezeigt, dass das Admin–Netz in Abbildung 10.8 den zu Beginn des Abschnitts beschriebenen Prozess formal spezifiziert. Dies beschreibt auch das Verhalten des Partizipationsprädikates multi-approve$(as, ip, s)$ aus Definition 9.18. Dieses Prädikat ist genau dann gültig, wenn bei der Durchführung eines Koordinierungsprozesses, mit dem durch die Feature Prädikate festgelegtem Subjekt als Administrator, die Markierung $M_\oplus$ erreicht wird. Wenn die Markierung $M_\ominus$ erreicht wird ist der Vorschlag abgelehnt und das Prädikat damit ungültig.

### 10.3.3 Quorum

Die im vorangegangenen Abschnitt vorgestellte Zustimmung durch eine Gruppe von Administratoren muss einstimmig erfolgen, damit der Administrierungsvorschlag angenommen wird. Darüber hinaus gibt es keinen zeitlichen Rahmen für die Entscheidung der Administratoren. Dies sind Einschränkungen, die für schnelle Entscheidungsprozesse nicht immer wünschenswert sind, oder im Falle der Einstimmigkeit, nicht unbedingt nötig sind.

Eine alternative Herangehensweise ist ein Koordinierungsprozess, der auf einen bestimmten Zeitraum zur Stimmabgabe beschränkt ist und zudem die Möglichkeit einer Enthaltung bietet. Der Administrierungsvorschlag wird dann angenommen, wenn nach Ablauf des Zeitraumes eine Zustimmung durch eine einfache Mehrheit der abgegebenen Stimmen vorliegt.

Aus dem Admin–Netz zur Zustimmung in Abbildung 10.8 wurde das Admin–Netz in Abbildung 10.9 inkrementell weiterentwickelt, um diesen Koordinierungsprozess formal zu spezifizieren.

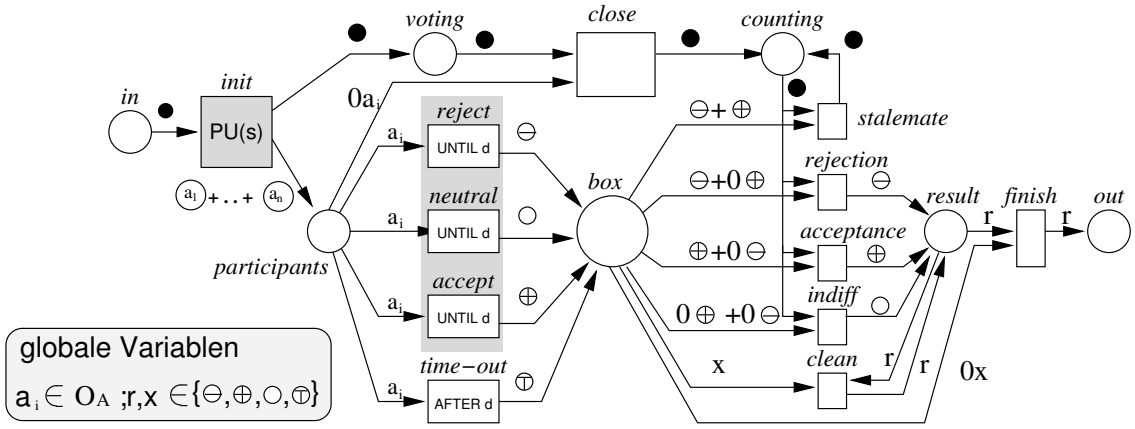



Abbildung 10.9: Quorum

Es wird vorausgesetzt, dass die Administratortokens $\textcircled{a_1}, \ldots, \textcircled{a_n}$ die $n$ am Prozess beteiligten Administratoren repräsentieren und die Variable $a$ mit einem beliebigen Administratortoken belegt werden kann. Für die Bindung der Variablen $x$ und $r$ gilt $x, r \in \{\oplus, \ominus, \circ, \textcircled{t}\}$.
Darüber hinaus wird angenommen, dass die Administratoren bei der Aktivierung der Interaktiven Transitionen *accept*, *reject* und *neutral* über den Inhalt des Administrierungsvorschlages informiert worden sind sowie über den Zeitraum $d$, der zur Abgabe der Entscheidung zur Verfügung steht (vgl. Def. 10.8). Der Beginn des Koordinierungsprozesses wird durch die Anfangsmarkierung $M_0$ repräsentiert, bei der genau ein Standardtoken auf der Stelle *in* liegt und alle anderen Stellen leer sind. Bei der Markierung $M_\ominus$ liegt ein Token $\ominus$ auf der Stelle *out* und alle anderen Stellen sind leer. Dagegen liegt bei der Markierung $M_\oplus$ ein Token $\oplus$ auf der Stelle *out* und alle anderen Stellen sind leer. Bei der Markierung $M_\circ$ liegt ein Token $\circ$ auf der Stelle *out* und alle anderen Stellen sind leer. Dann gilt:

**Satz 10.3**
Der Koordinierungsprozess, der durch das Admin–Netz in Abbildung 10.9 spezifiziert wird, erreicht ausgehend von dieser Anfangsmarkierung $M_0$ stets eine der drei folgenden Markierungen $M_\ominus$, $M_\oplus$ bzw. $M_\circ$, die die drei möglichen Abschlüsse des Prozesses darstellen.
Wenn die Interaktive Transition *reject* häufiger geschaltet hat als die Interaktive Transition *accept*, wird die Markierung $M_\ominus$ erreicht, wodurch eine Ablehnung des Administrierungsvorschlages dargestellt wird.
Wenn die Interaktive Transition *accept* häufiger geschaltet hat als die Interaktive Transition *reject*, wird die Markierung $M_\oplus$ erreicht, wodurch eine Annahme des Administrierungsvorschlages dargestellt wird.
Wenn die beiden Interaktiven Transitionen *accept* und *reject* gleich häufig geschaltet haben, wird die Markierung $M_\circ$ erreicht, wodurch eine Unentschieden dargestellt wird.
Bei der Markierung $M_\oplus$, $M_\ominus$ bzw. $M_\circ$ ist keine Transition im Admin–Netz aus Abbildung 10.9 mehr aktiviert. Die Anzahl der erzeugten Administratortokens $n$ kann beliebig sein.

**Beweis zu Satz 10.3**

1. *close* schaltet erst, wenn *participants* wieder leer ist. Die Argumentation entspricht der im Punkt 1 in Beweis 10.2.

2. Jedes Administratortoken $\textcircled{a_1}, \ldots, \textcircled{a_n}$ kann bis zum Zeitpunkt $d$ durch das Schalten von *accept*, *neutral* oder *reject* von *participants* entfernt werden. Jede Administratorentscheidung wird

durch ein Token $\oplus, \ominus$ bzw. $\circ$ auf *box* repräsentiert. Insgesamt können diese Transitionen zusammen bis zu $n$ mal schalten.

3. Nach Erreichen des Zeitpunktes $d$ kann *time-out* für jede der $k \geq 0$ auf *participants* verbliebenen Administratortokens einmal schalten. Danach liegen auf *box*, solange *close* noch nicht schaltet, $n$ Token $x$: $x \in \{\oplus, \ominus, \circ, \textcircled{T}\}$.

4. Die Anzahl der Tokens $\ominus$ und $\oplus$ geben die Anzahl der Schaltvorgänge der Interaktiven Transition *reject* bzw. *accept* wieder. Die Tokens vom Typ $\circ$ oder $\textcircled{T}$ die sich auf *box* befinden können, sind für das Ergebnis nicht von Bedeutung.
   Angenommen es liegen $k$ Token $\ominus$ und $l$ Token $\oplus$ auf der Stelle *box* $(k, l \geq 1)$, wenn *close* schaltet. Dann ist nach dem Schalten von *close* nur die Transition *stalement* aktiviert und kann bei $k \neq l$ insgesamt $|k - l|$ mal schalten, dabei werden die Token $\oplus$ und $\ominus$ von *box* entfernt, die Markierung von *counting* wird reproduziert.
   Wenn $k > l$ gilt, wird noch mindestens ein Token $\ominus$ auf *box* liegen, sodass nur die Transition *rejection* aktiviert ist und beim Schalten ein Token $\ominus$ von *box* entfernt und ein Token $\ominus$ auf *result* ablegt. Wenn dagegen $k < l$ gilt, wird noch mindestens ein Token $\oplus$ auf *box* liegen, sodass nur die Transition *rejection* aktiviert ist und beim Schalten ein Token $\oplus$ von *box* entfernt und ein Token $\oplus$ auf *result* ablegt.

   Für den Fall dass $k = l; k, l \geq 0$ kann die Transition *stalement* $k$ mal schalten, sodass kein Token $\oplus$ oder $\ominus$ auf der Stelle *box* übrig bleibt. Danach ist nur die Transition *indiff* aktiviert und legt beim Schalten das Token $\circ$ auf *result*.

5. Das Leeren von *box* und das Verschieben des Tokens von *result* auf *out* erfolgt analog zum Punkt 4 aus Beweis 10.2 und gilt entsprechend auch im Admin–Netz in Abbildung 10.9.

□

Der zu Beginn des Abschnitts beschriebene Koordinierungsprozess wird somit von dem Admin–Netz in Abbildung 10.9 formal spezifiziert. Außerdem liegt damit auch eine formale Spezifikation des Partizipationsprädikates quorum$(as, ip, s, d)$ aus Definition 9.19 vor. Wie schon beim Prädikat multi-approve() werden die beteiligten Administratoren mittels der Feature Prädikate $\mathcal{F}$ bestimmt. Wenn die Markierung $M_\oplus$ vorliegt, gilt der Vorschlag als angenommen und das Prädikat ist gültig. Ansonsten liegt die Markierung $M_\ominus$ oder $M_\circ$ vor und das Prädikat ist ungültig.

### 10.3.4 Vote

Beim Quorum sind alle Teilnehmer gleichberechtigt und jede Stimme hat das gleiche Gewicht. Eine andere Möglichkeit für das Vorgehen bei einem Koordinierungsprozess besteht darin, dass eine Teilmenge der beteiligten Administratoren das besondere Privileg hat, ein Veto abgeben zu können, welches auf jeden Fall die Ablehnung eines Vorschlages zur Folge hat.

In der Abbildung 10.10 ist ein Admin–Netz für einen solchen Koordinierungsprozess mit der Möglichkeit eines Veto dargestellt. Dieses Netz stellt eine inkrementelle Erweiterung des im Abschnitt 10.3.3 vorgestellten Quorums dar, bei dem die wesentlichen Strukturen für die Abstimmung nahezu identisch sind und bei der Auszählung lediglich eine Transition für das Veto sowie geeignete Kantenbeschriftungen an den anderen Transitionen ergänzt sind. Die Bestimmung der Beteiligten und die zeitliche Steuerung sowie das "Aufräumen" nach Erreichen des Zeitpunktes $d$, gestalten sich etwas komplexer als im vorangegangenen Admin–Netz.

Von den $n$ am Koordinierungsprozess beteiligten Administratoren seien $p$ mit dem Privileg ausgestattet ein Veto aussprechen zu dürfen. Es wird vorausgesetzt, dass die Administratortokens $\textcircled{a_1}, \ldots \textcircled{a_p}$ diese privilegierten Administratoren repräsentieren, während die Übrigen, durch die Administratortokens $\textcircled{a_{p+1}}, \ldots \textcircled{a_n}$ repräsentiert werden. Die Variablen $a$ und $a'$ können mit beliebigen Administratortokens belegt werden. Für die Bindung der Variablen $x$ und $r$ gilt in bei diesem Admin–Netz $x, r \in \{\oplus, \ominus, \circ, \oslash, \textcircled{T}\}$.

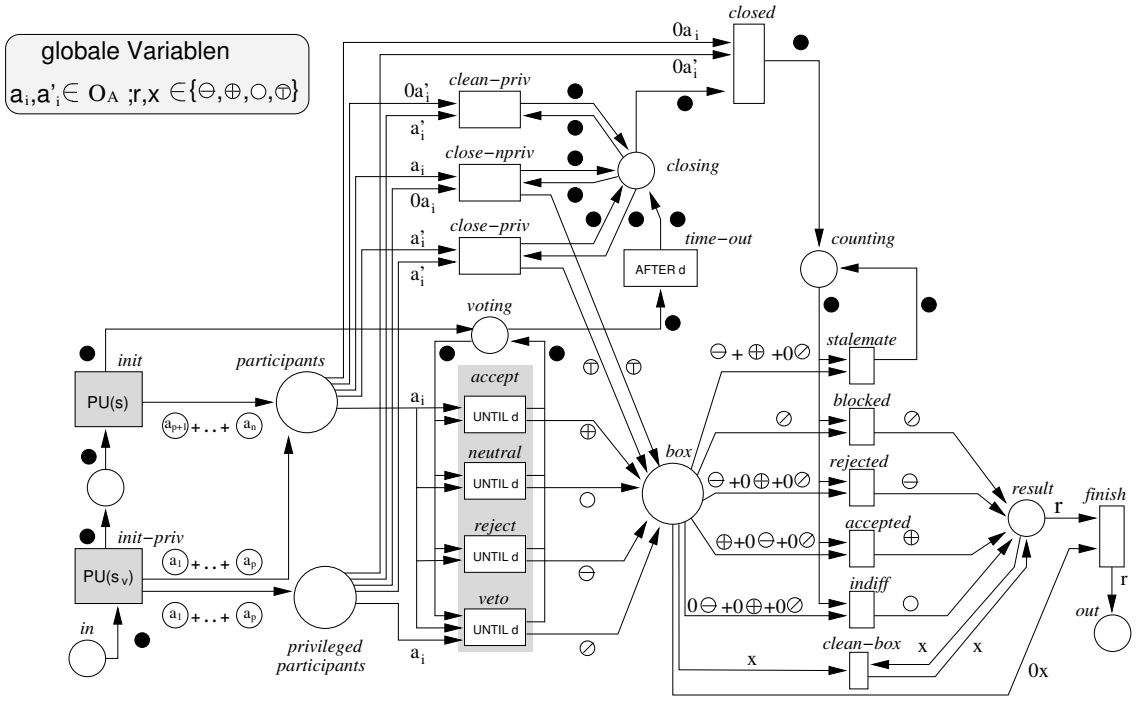



Abbildung 10.10: Vote

Darüber hinaus wird angenommen, dass die Administratoren bei der Aktivierung der Interaktiven Transitionen *accept*, *reject* und *neutral* über den Inhalt des Administrierungsvorschlages informiert worden sind sowie über den Zeitraum $d$, der zur Abgabe der Entscheidung zur Verfügung steht (vgl. Def. 10.8). Die privilegierten Administratoren werden zusätzlich über die Möglichkeit informiert, die Interaktive Transition *veto* schalten zu lassen. Der Beginn des Koordinierungsprozesses wird durch die Anfangsmarkierung $M_0$ repräsentiert, bei der genau ein Standardtoken auf der Stelle *in* liegt und alle anderen Stellen leer sind. Bei der Markierung $M_\ominus$ liegt ein Token $\ominus$ auf der Stelle *out* und alle anderen Stellen sind leer. Dagegen liegt bei der Markierung $M_\oplus$ ein Token $\oplus$ auf der Stelle *out* und alle anderen Stellen sind leer. Bei der Markierung $M_\circ$ liegt ein Token $\circ$ auf der Stelle *out* und alle anderen Stellen sind leer. Entsprechend liegt bei der Markierung $M_\oslash$ ein Token $\oslash$ auf *out* und alle anderen Stellen sind leer. Es gilt:

**Satz 10.4**
Der Koordinierungsprozess, der durch das Admin–Netz in Abbildung 10.10 spezifiziert wird, erreicht ausgehend von dieser Anfangsmarkierung $M_0$ stets genau eine der vier folgenden Markierungen $M_\ominus$, $M_\oplus$, $M_\circ$ bzw. $M_\oslash$, die die vier möglichen Abschlüsse des Koordinierungsprozesses darstellen.
Sobald mindestens einmal die Interaktive Transition *veto* geschaltet hat, wird die Markierung $M_\oslash$ erreicht. Falls diese Transition nicht geschaltet hat, gelten die Aussagen aus Satz 10.3 analog für das Netz in Abbildung 10.10 und den zugehörigen Markierungen.

Ausserdem gilt weiterhin, bei der Markierung $M_\oplus$, $M_\ominus$ bzw. $M_\circ$ und zusätzlich $M_\oslash$ ist keine Transition im Admin–Netz aus Abbildung 10.10 mehr aktiviert. Die Anzahl der erzeugten Administratortokens $n$ und der Anteil $p$ der Administratortokens kann beliebig sein.

**Beweis zu Satz 10.4**

1. Die Transition *closed* schaltet, wenn *participants* und *privileged participants* wieder leer sind. Dies wird durch die Inhibitor–Kanten von diesen Stellen erreicht. Zusätzlich kann diese Transition aufgrund des Standardtokens auf *closing* erst nach dem Schalten von *init-priv*, *init* und *time–out* aktiviert werden.

2. Für die Administratorentokens $\textcircled{a_{p_1}}, \ldots, \textcircled{a_n}$ gilt die Argumentation aus Punkt 2 aus Beweis 10.3, wobei das Standardtoken auf *voting* bei jedem Schalten einer beliebigen Interaktiven Transition reproduziert wird.
   Jedes Administratortoken, das sowohl auf *participants* als auch auf *privileged participants* liegt, kann bis zum Zeitpunkt $d$ durch das Schalten von *veto* von beiden Stellen entfernt werden, wodurch ein $\oslash$ auf *box* abgelegt wird.

3. Nach Erreichen des Zeitpunktes $d$ kann *time-out* schalten, entsprechend der verbliebenen Administratorentokens auf *participants* und *privileged participants* sind *close-priv*, *close-npriv* sowie *clean-priv* aktiviert.
   *close-priv* schaltet jeweils einmal für jeden privilegierten Administrator, der durch jeweils ein Administratortoken auf *participants* und *privileged participants* repräsentiert wird, und legt dabei ein $\textcircled{T}$ auf *box*, außerdem wird das Standardtoken auf *closing* reproduziert.
   Für jedes Administratortoken, das nur auf *participants* liegt und nicht auf *privileged participants*, schaltet *close-npriv* genau einmal und legt dabei ein $\textcircled{T}$ auf *box*. Außerdem wird das Standardtoken auf *closing* reproduziert.
   Falls ein Administratortoken, das beim Schalten von *init-priv* auf *participants* und *priviledged participants* gelegt wurde, durch das Schalten von *accept, reject* oder *neutral* lediglich auf *priviledged participants* verblieben ist, wird dieses beim Schalten von *clean-priv* entfernt. Dabei wird lediglich das Standardtoken auf *closing* reproduziert.

4. Beim Schalten von *close* liegen auf *box* $n$ Tokens $x$: $x \in \{\oplus, \ominus, \circ, \oslash, \textcircled{T}\}$. Wenn mindestens ein $\oslash$ auf *box* liegt, kann durch die um "$+0\oslash$" gegenüber Abbildung 10.9 erweiterten Kantenbeschriftungen von *stalemate, recjected accepted* und *indiff* nur die *blocked* schalten. Dabei wird auch das Standardtoken von *counting* entfernt und ein $\oslash$ auf *result* abgelegt.

5. Falls kein $\oslash$ auf *box* liegt, gilt der Punkte 4 aus Beweis 10.3 und es wird entsprechend der Mehrheit ein $\ominus, \oplus$ oder $\circ$ Token auf *result* abgelegt. Analog zu diesem Punkt kann nur eine der vier Transitionen *rejected*, *accepted*, *indiff* oder *blocked* schalten.

6. Das Leeren von *box* und das Verschieben des Tokens von *result* auf *out* erfolgt analog zum Punkt 4 aus Beweis 10.2 und gilt entsprechend auch im Admin–Netz in Abbildung 10.10.

□

Dadurch, dass beide Administratorentokens auf *participants* und *privileged participants* beim Schalten von *veto* entfernt werden, kann der dadurch repräsentierte Administrator keine Interaktive Transition mehr schalten lassen. Ebenso kann nicht erst eine Transition *accept, reject* oder *neutral* schalten und dann *veto*, da das gleiche Token auf beiden Stellen *participants* und *priviledged participants* vorausgesetzt wird.

Für jedes der $n-p$ Administratortokens, die nur auf *participants* liegen, schalten die Transitionen *accept*, *neutral*, *reject* sowie *close-npriv* genau einmal. Für jedes der $p$ Administratorentokens, die sowohl auf *participants* als auch auf *priviledged participants* liegen, schalten *accept*, *neutral*, *reject* und *veto* sowie *close-priv* genau einmal.

Der zu Beginn des Abschnitts beschriebene Koordinierungsprozess wird somit von dem Admin–Netz in Abbildung 10.10 umgesetzt. Außerdem liegt damit eine formale Spezifikation des Partizipationsprädikates $\mathsf{vote}(as, ip, s, s_v, d)$ aus Definition 9.19 vor. Wie zuvor werden die beteiligten Administratoren mittels der Feature Prädikate $\mathcal{F}$ bestimmt, dabei nehmen die Subjekte, die durch die Feature Prädikate festgelegten Bedingungen der Variable $s_v$ erfüllen als privilegierte Administratoren teil. Wenn die Markierung $M_\oplus$ vorliegt gilt der Vorschlag als angenommen und das Prädikat ist gültig. Ansonsten liegen die Markierungen $M_\oslash$, $M_\ominus$ oder $M_\circ$ vor und das Prädikat ist ungültig.

Somit wurde der Koordinierungsprozess für alle Partizipationsprädikate aus Abschnitt 9.4 durch Admin–Netze formal spezifiziert. Die inkrementelle Entwicklung hat dabei die Verifizierung der möglichen Schaltfolgen und damit der erreichbaren Markierungen erleichtert.

Die bisherigen Administrationsnetze spezifizieren nur, wie eine Entscheidung zu einem vorhandenen Vorschlag ermittelt wird. Es ist jedoch auch möglich nach einer Ablehnung weitere Vorgehensweisen zu modellieren, bei denen zum Beispiel ein abgelehnter Vorschlag überarbeitet wird und erneut über diesen abgestimmt wird.

# Kapitel 11

# Erweiterung von Admin–Netzen zu administrativen Workflows

Am Ende des vorangegangenen Kapitels wurden inkrementell immer komplexere Admin–Netze entwickelt, um so alle im Kapitel 9 diskutieren Bestätigungen formal zu spezifizieren. Diese Admin–Netze beschränkten sich jedoch auf unterschiedliche Vorgehensweisen zur Bestätigung eines Vorschlags. Bisher ist nicht modelliert, wie das Vorgehen nach der Ablehnung eines Vorschlags sein soll. Hier könnte es eine Möglichkeit sein, einen neuen – verbesserten – Vorschlag einzuholen, der dann vielleicht angenommen wird. Bei komplexeren Vorschlägen wie Änderungen am Regelwerk, die eine größere Anzahl von einzelnen Regeln umfassen, wäre es hingegen sinnvoll, diesen Vorschlag in mehrere Teile zu zerlegen, die dann einzeln behandelt werden können. Diese Erweiterungen wurden zum großen Teil bereits in [WL04a] vorgestellt.

Mit wachsender Komplexität wird auch der Umfang der Admin–Netze größer, sodass es schwierig wird, den Überblick zu behalten. Daher wird zunächst im folgenden Abschnitt 11.1 eine Strukturierungsmöglichkeit für Admin–Netze eingeführt. Diese erlaubt es, einzelne abgeschlossene Teile eines Vorgangs, wie die Bestätigung eines Vorschlags, zu kapseln.

Wenn die weitere Verfolgung eines Vorschlags modelliert werden soll, muss es eine Möglichkeit geben, Vorschläge in einem Admin–Netz neu zu erzeugen oder zu überarbeiten. Hierzu werden im Abschnitt 11.2 entsprechende Transitionen eingeführt. Diese Vorgehensweise wird im Abschnitt 11.3 noch um die Möglichkeit erweitert, komplexe Vorschläge in einzelne Teile zu zerlegen, über die dann einfacher eine Einigung erzielt werden kann. Im Abschnitt 11.4 wird all dies verwendet, um ein größeres Beispiel für einen komplexeren Workflow zu präsentieren. Eine weitere Anwendungsmöglichkeit, bei der verstärkt inhaltliche Aspekte von Bedeutung sind, wird im Abschnitt 11.5 präsentiert. Die Möglichkeiten der automatisierten formalen Analyse von Admin–Netzen werden im Abschnitt 11.6 erläutert.

## 11.1 Hierarchische Netzorganisation

Im Abschnitt 10.3 ist eine inkrementelle Entwicklung eines Administrationsnetze von einer einfachen Zustimmung über Quoren, bis hin zur Möglichkeit eines Vetos dargestellt worden. Dabei können Koordinierungsprozesse wie ein Quorum in Abbildung 10.9 oder die Abstimmung mit Vetorecht in Abbildung 10.10 auch Teil eines komplexen Prozesses sein. Aus diesem Grund erscheint es sinnvoll, bestimmte Details wie eine Abstimmung innerhalb komplexer Prozesse wie eine einzige Transition zu behandeln und dabei Details wie Deadlines oder Auszählungsverfahren auszublenden. Diese Technik ist bei Petri–Netzen als Teilnetz oder Subnetz bekannt. Zum einen können die Eigenschaften eines Subnetzes für sich genommen analysiert werden, sodass dies später nicht noch einmal geschehen muss. Zum anderen kann bei Bedarf jederzeit auf die formale Spezifizierung zurückgegriffen werden.

Im Folgenden wird zunächst die Struktur eines Subnetzes definiert, bevor einige Eigenschaften für ein wohlgeformtes Subnetz festgelegt werden, die es erlauben, ein Subnetz in anderen Admin–Netzen wie eine einzelne Transition zu behandeln.

**Definition 11.1 (Subnetz)**
Sei $AdN := (N, I, L, M_0)$ ein Administrationsnetz, dann ist $AdN' := (N', I', L', M'_0)$ ein Subnetz von $AdN$, wenn $N' = (S', T', F')$ und $S' \subseteq S$, $T' \subseteq T$, $F' = F \cap \{(S' \times T') \cup (T' \times S')\}$ gilt. Zudem gilt:

- $s \in S'$ ist genau dann eine **Eingangsstelle des Subnetzes** $AdN'$, wenn $\exists t' \in T' : (s, t') \in F'$ und $s$ ist eine Ausgangsstelle für eine Transition $t \in T - T'$.
  $S'_{in}$ ist die Menge aller einlaufenden Stellen von $AdN'$.

- $s \in S'$ ist genau dann eine **Ausgangsstelle des Subnetzes** $AdN'$, wenn $\exists t' \in T' : (t', s) \in F'$ und $s$ ist eine Eingangsstelle für eine Transition $t \in T - T'$.
  $S'_{out}$ ist die Menge alle auslaufenden Stellen von $AdN'$.

- $t \in T'$ ist genau dann eine **Eingangstransition von** $AdN'$, wenn $\exists s \in S'_{in} : (s, t) \in F'$. $T'_{in}$ ist die Menge aller Eingangstransitionen von $AdN'$.

- $t \in T'$ ist genau dann eine **Ausgangstransition von** $AdN'$, wenn $\exists s \in S'_{out} : (t, s) \in F'$. $T'_{out}$ ist die Menge aller Ausgangstransitionen von $AdN'$.

- $T'_{in} \cap T'_{out} = \emptyset$

Die Eingangs- und Ausgangsstellen eines Subnetzes bilden die Anknüpfungspunkte zum übergeordneten Admin–Netz. Durch die obige Definition ist es jedoch auch möglich, dass andere Transitionen, die nicht zum Subnetz gehören, Stellen aus der Menge $S'_{in}$ als Eingangsstellen haben. Da die Subnetze zunächst nur als Strukturierungsmaßnahmen genutzt werden, kann auf eine Definition von Unter- und Teilnetze, wie sie in [Sta90, Kapitel 16] oder auch [JV80] vorgestellt werden, verzichtet werden.

Ein Subnetz soll dann als wohlgeformt gelten, wenn es sich wie eine Transition verhält. Dies kann dadurch erreicht werden, dass beim Schalten zunächst alle Tokens, die zur Aktivierung beitragen von den Eingangsstellen entfernt werden, bevor die ersten Tokens auf den Ausgabestellen platziert werden. Diese dürfen dann im gleichen Schaltvorgang auf keinen Fall erneut eingelesen werden.

**Definition 11.2 (wohlgeformtes Subnetz)**
$AdN'$ wird genau dann als **wohlgeformtes Subnetz von** $AdN$ bezeichnet, wenn Folgendes gilt:

1. $M$ ist eine Markierung in $AdN$, die von $M_0$ erreichbar ist, und $M'$, die entsprechende (Teil-) Markierung von $AdN'$. Vor dem (ersten) Schalten einer Eingangstransition von $Adn'$ in $M$ bzw. $M'$ entspricht die Markierung auf allen **inneren Stellen** $s' \in S' - S'_{in} - S'_{out}$ der von $M'_0$, d.h. $M(s') = M'_0(s')$.

2. Eine Ausgangstransition von $AdN'$ wird erst dann aktiviert, wenn mindestens eine Eingangstransitionen von $AdN'$ geschaltet hat und alle Eingangstransitionen von $Adn'$ deaktiviert sind.

3. Nachdem eine Eingangstransition von $Adn'$ deaktiviert ist, kann sie erst wieder aktiviert werden, wenn mindestens eine Ausgangstransition von $Adn'$ geschaltet hat und alle Ausgangstransitionen von $Adn'$ wieder deaktiviert sind.

4. Nachdem die Ausgangstransitionen von $Adn'$ wieder deaktiviert sind und direkt vor dem nächsten Schalten einer Eingangstransition von $Adn'$ entspricht die Markierung auf allen inneren Stellen $s' \in S' - S'_{in} - S'_{out}$ wieder der von $M'_0$. Diese Markierung der inneren Stellen wird **Reset-Markierung** von $AdN'$ genannt.

Eine weitere mögliche Forderung könnte sein, dass nach der Aktivierung aller Eingangsstellen die Tokens auf den Eingangsstellen solange ausschließlich durch das Schalten der Eingangstransitionen entfernt werden, bis eine der Ausgangstransitionen geschaltet hat. Dies würde jedoch nicht das Subnetz betreffen, sondern dessen Einbettung in ein übergeordnetes Admin–Netz.

Solch eine Forderung ist nötig, wenn mehrere Eingangstransitionen nacheinander mit einer vorher unbekannten Menge von Tokens schalten sollen. Dies kann der Fall sein, wenn zum Beispiel auf einer Eingangsstelle die Administratortokens der Beteiligten an einer Bestätigung liegen und deren Anzahl nicht fest ist. Problematisch ist dabei, dass die Tokens von einer Transition außerhalb des Subnetzes abgezogen werden können, nachdem bereits eine oder mehrere Eingangstransitionen geschaltet haben. Dies kann, wie in Abbildung 11.1 zu sehen, zu Deadlocks führen.

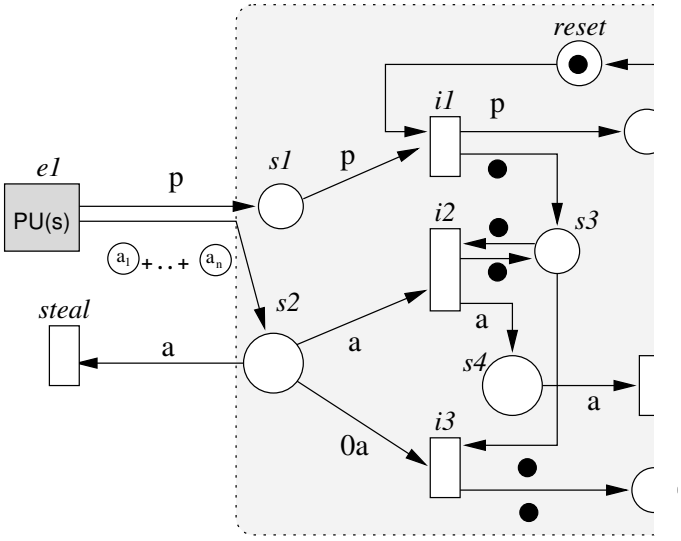



Abbildung 11.1: Subnetze

Mit dem Schalten der Transition *e1* wird ein Proposaltoken auf die Stelle *s1* und eine beliebige Menge Administratortokens auf die Stelle *s2* gelegt. Damit ist die Eingangstransition *i1* aktiviert und kann schalten. Das Standardtoken auf der Stelle *reset* wird stets beim Schalten der letzten Ausgangstransition wieder auf diese Stelle zurückgelegt, um so die 3. Eigenschaft eines wohlgeformten Netzes (Def. 11.2) zu erfüllen. Die Transition *i2* dient dazu alle Administratortokens von der Stelle *s2* auf die interne Stelle *s4* zu verschieben, bevor die eigentliche Bestätigung beginnt. Problematisch ist an dieser Stelle das Schaltverhalten der Transition *steal*, die nach dem Schalten von *e1* auch aktiviert ist. Da in Admin–Netzen keine Annahmen über Transitionen gemacht werden, die sich im wechselseitigem Ausschluss befinden, besteht die Möglichkeit, dass alle Administratortokens von *s2* durch *steal* zu entfernen, bevor Transition *i2* auch nur einmal geschaltet hat. Aufgrund der Stelle *reset*, die aber für ein wohlgeformtes Netz notwendig ist, kann dann die Eingangstransition *i1* des Subnetzes nicht mehr aktiviert werden.

Ein Lösungsansatz für dieses Problem ist in Abbildung 11.2 zu sehen. Hier wird durch das erste Schalten einer Eingangstransition bereits eine gewisse Anzahl von Administratortokens auf die Stelle *s4* gelegt, wobei gezeigt werden kann, dass diese Anzahl ausreichend ist, um zum Beispiel eine Bestätigung durchzuführen, und damit mindestens eine Ausgangstransition schaltet und die Reset–Markierung des Subnetzes wieder erreicht wird. Weitere Administratortokens können nach dem Schalten von *i1* über die Transition *i2* auf die Stelle *s4* verschoben werden. Erst wenn keine Administratortokens mehr auf *s2* liegen, kann die Transition *i3* schalten. Mit dem Verschieben des Standardtokens würde die eigentliche Bestätigung beginnen.
Diese Eigenschaft eines Subnetzes wird durch die folgende Definition gesondert herausgestellt.

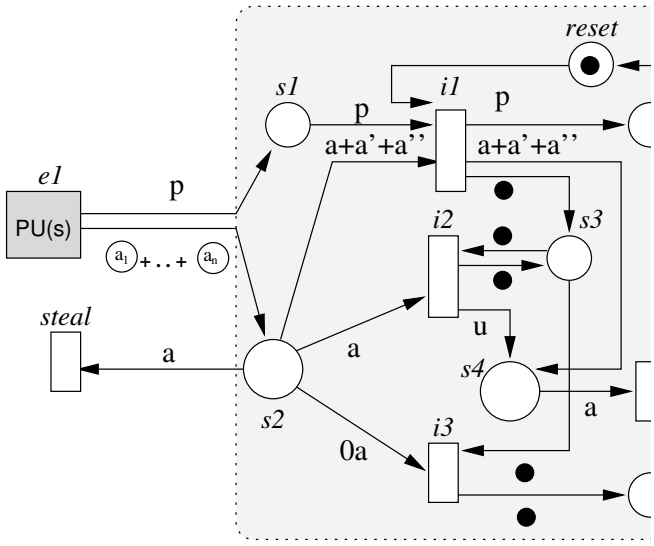
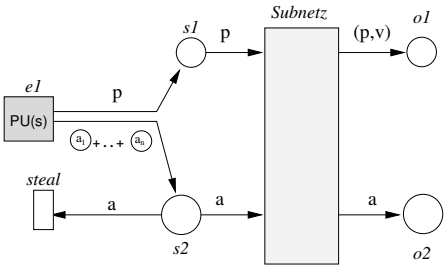


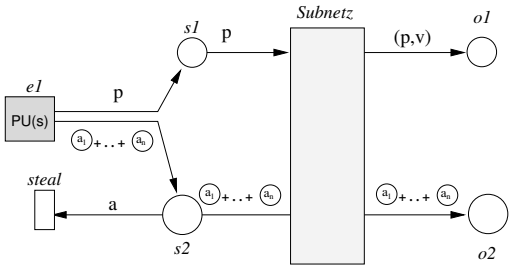

Abbildung 11.2: Subnetze atomar

**Definition 11.3 (Atomarität eines Subnetzes)**
Ein wohlgeformtes Admin–Netz verhält sich atomar, wenn es nach dem ersten Schalten einer Eingangstransition immer eine Schaltfolge gibt, sodass auf allen inneren Stellen die Belegung $M_0'$ erreicht wird.

Durch die Bedingungen 2 und 3 für ein wohlgeformtes Subnetz (Def. 11.2) wird gewährleistet, dass zunächst mindestens ein Token von den Eingangsstellen entfernt wird, bevor auf den Ausgangsstellen mindestens ein Token abgelegt wird. Diese Eigenschaft ermöglicht es, ein Subnetz zunächst wie eine Transition zu betrachten.




(a) Kantenbeschriftung mit Variablen




(b) Kantenbeschriftung mit möglichen Tokens

Abbildung 11.3: Einbindung von Subnetzen

Problematisch ist dabei, dass das Subnetz aus Abbildung 11.2 eigentlich eine beliebige Anzahl von Tokens von der Stelle *s2* einliest. Dieses Vorgehen würde bei einer Darstellung der Einbindung wie in Abbildung 11.3(a) nicht deutlich werden. Solange das Subnetz und dessen Einbindung nicht detailliert betrachtet wird, könnte angenommen werden, dass beim Schalten nur ein Token von *s2* entfernt wird[1]. Daher wird bei der Einbindung eines Subnetzes eine Darstellung, wie sie in Abbildung 11.3(b) zu sehen ist, bevorzugt. Diese soll verdeutlichen, dass eine variable Anzahl von Tokens beim Schalten von *s2* entfernt und auf *o2* erzeugt wird. Diese Darstellungsform ist an die Kantenbeschriftungen der Feature Prädikate (vgl. Def.10.9) angelehnt und muss noch formal definiert werden.

[1]Bei der automatisierten Analyse, wie sie im Abschnitt 11.6 noch vorgestellt wird, würde genau dies angenommen und führt dann unter Umständen zu falschen Ergebnissen.

**Definition 11.4 (Einbindung von atomaren Subnetzen)**
Beim Einbinden eines atomaren Subnetzes $AdN'$ in ein übergeordnetes Admin–Netz $AdN$ muss angegeben werden, welche Stelle $s \in S$ im übergeordneten Netz der Eingangs- bzw. Ausgangsstelle $s' \in S'_{in} \cup S'_{out}$ im Subnetz entspricht. Im übergeordneten Netz wird das Subnetz durch eine einzelne Transition dargestellt, die mit diesen Stellen entsprechend verbunden ist. Die Kantenbeschriftung gibt dabei die Art und Anzahl der beim Schalten entfernten und erzeugten Token wieder. Die Variablen auf den Kantenbeschriftungen der Ausgangsstellen *müssen nicht* an den Eingangsstellen verwendet werden. Die Kantenbeschriftungen können die Beschriftung $\textcircled{a_1} + \ldots \textcircled{a_p}$ enthalten, um eine beliebige aber feste Tokenmengen zu bezeichnen.

Die Bedingung aus Definition 10.4 bezüglich der Variablen in Ausgangskanten von Transitionen wird deshalb aufgehoben, da die Bedingungen, die zur Belegung der Variablen führen durch das Subnetz modelliert werden. Wenn sich diese Bedingungen auch als Transitions- und Kanteninschriften formulieren ließen, wäre das Subnetz überflüssig.

## 11.2 Erzeugen von Proposals

In den vorangegangenen beiden Kapiteln wurde ein Vorschlag in einen Koordinierungsprozess eingebracht und darüber entschieden, ohne dass es die Möglichkeit gibt, bei einer Ablehnung Änderungen einzubringen, die vielleicht zu einer Akzeptanz geführt hätten. In diesem Abschnitt wird eine neue Transition vorgestellt, durch die Admin–Netze um die Möglichkeit erweitert werden, neue Proposals in einen Koordinierungsprozess einzubringen.

Als Beispiel wird ein Koordinierungsprozess mit zwei Administratoren $a_1$ und $a_2$, herangezogen bei dem Beide kooperativ versuchen einen Konflikt oder Inkonsistenz durch eine neue Regel oder auch einen Regelsatz zu beseitigen[2]. Die Grundidee bei diesem Koordinierungsprozess ist, dass die beiden Beteiligten wechselseitig Vorschläge machen, die der andere begutachten kann und eine Entscheidung treffen soll. Falls ein Administrator den Vorschlag ablehnt, ist es nun an ihm einen neuen Vorschlag zu machen, der dem ersten Administrator wiederum zur Entscheidung vorgelegt wird. Wenn dieser den Vorschlag annimmt, wird dieser Vorschlag entsprechend als Ergebnis des Koordinierungsprozesses ausgegeben.
Dies impliziert natürlich, dass jeder Administrator nur das vorschlägt, was er selber auch akzeptiert. Darüber hinaus ist eine Grundannahme, dass sich die Vorschläge immer mehr annähern, bis schließlich ein Kompromiss gefunden wird, dem der jeweils andere auch zustimmen kann.

### 11.2.1 Propose Transition

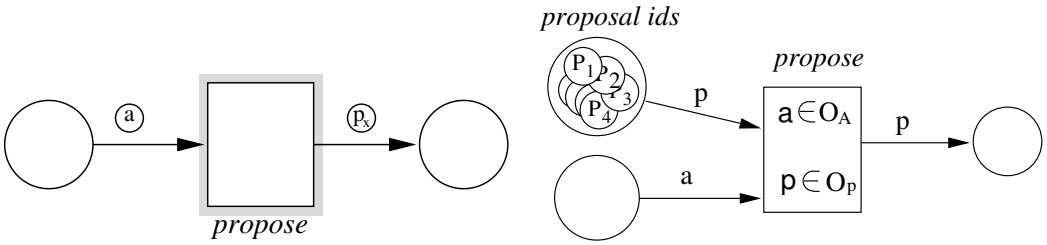
Um in einem Admin–Netz das Einbringen eines neuen Vorschlages modellieren zu können, wird eine neue Transition benötigt, die diese Interaktion mit einem Administrator darstellt und ein entsprechendes Proposaltoken generiert. Wie schon in 10.2 erwähnt, ist ein Proposaltoken nur ein Verweis auf einen Vorschlag, sodass der genaue Inhalt für den Ablauf in einem Admin–Netz keine Bedeutung hat.

**Definition 11.5 (Propose Transition)**
Eine Propose Transition $t_P$ ist eine ausgezeichnete (grau hinterlegte) Transition, die aufgrund der Kantenbeschriftung von den Eingangsstellen und den Transitionsinschriften zur Aktivierung genau ein Administratortoken und beliebige weitere Tokens anderen Typs benötigt.
Das Subjekt, dessen Administratortoken an der Aktivierung beteiligt ist, wird entsprechend benachrichtigt, wobei ihm auch die möglichen weiteren Tokens bzw. Belegungen der anderen Eingangsstellen mitgeteilt werden. Das Subjekt kann nun einen neuen Vorschlag erstellen, der durch ein Proposaltoken

[2]Dabei ist die genaue Form des Regelsatzes an dieser Stelle irrelevant. Er könnte neben neuen Regeln durchaus auch die Änderung von existierenden Regeln oder auch das Löschen von problematischen Regeln sowie das gleichzeitige Einfügen von neuen Regeln umfassen.

mit neuem Index repräsentiert wird. Die Bezeichnung für dieses Proposal wird gemäß der Kantenbeschriftung zu den Ausgangsstellen eindeutig generiert. Durch das Erstellen eines neuen Vorschlages schaltet die Propose Transition.
Falls die Propose Transition nach ihrer Aktivierung wieder deaktiviert wird, wird das Subjekt darüber benachrichtigt.

Ein einfaches Beispiel für eine Propose Transition ist in Abbildung 11.4(a) zu sehen, bei der ein bestimmtes Administratortoken zur Aktivierung benötigt wird und ein bereits festgelegtes Proposaltoken auf die Ausgangsstelle gelegt wird. Dies ist jedoch nur dann sinnvoll, wenn das Administratortoken, das diese Transition *propose* aktiviert, tatsächlich durch das restliche Netz auch eindeutig festgelegt ist. Gemäß der Definition 10.2 wird jedes Mal ein neues Proposaltoken Ⓟ generiert, das auf diesen Vorschlag verweist, wobei das $x$ für einen bisher nicht genutzten Index stehen soll.




(a) Einfache Propose Transition (b) Ersatz für Propose Transition

Abbildung 11.4: Generierung von Proposaltokens

Wenn im Vorfeld die maximale Anzahl von Vorschlägen, die während eines Koordinierungsprozesses gemacht werden, abgeschätzt werden kann, lässt sich die Propose Transition auch für weitere Analysezwecke durch die in Abbildung 11.4(b) dargestellte Transition ersetzen. Hierbei gibt es bei der Stelle *proposal ids*, außer der angezeigten Kante zur Transition *propose*, keine weiteren Kanten zu Nicht–Propose Transitionen. Bei der initialen Markierung $M_0$ werden ausreichend viele Proposaltokens auf diese Stelle gelegt, die als eine Art Vorratsbehälter dient. Jedes Mal, wenn die Transition schaltet, wird ein beliebiges Proposaltoken[3] aus diesem Vorratsbehälter entfernt und auf die Ausgangsstelle gelegt. Sollten mehrere Propose Transitionen im Admin–Netz enthalten sein, können entweder alle mit derselben Stelle als Vorratsbehälter verbunden werden, oder in der Markierung $M_0$ sind diese Stellen mit disjunkten Mengen von Proposaltokens belegt. Damit wird die in Definition 10.2 implizierte Bedingung, dass jeder Vorschlag durch genau ein bestimmtes Proposaltoken repräsentiert werden soll und somit auch neue Vorschläge niemals durch ein bereits verwendetes Proposaltoken dargestellt werden können[4], erfüllt.

Da das Administratortoken des Subjektes, das einen Vorschlag unterbreiten soll, bei der Modellierung des Admin–Netzes nicht zwingend feststeht, oder während des Prozesses auch variieren kann, ist es sinnvoll an dieser Stelle mit einer Variablen zu arbeiten, wie es bei dem Beispiel in Abbildung 11.5 geschieht. Bei Aktivierung der Transition *alternativ* wird dem Subjekt, dessen Administratortoken auf der Stelle *user* liegt, gemäß Definition 11.5 auch die Belegung der Variable $p$ mitgeteilt[5], sodass das Subjekt über dieses Proposaltoken auf den Inhalt der entsprechenden Vorschläge zurückgreifen kann und seinen neuen Vorschlag entsprechend anpassen kann. Die Transitionsinschrift $p' = new(u)$ zeigt an, dass gemäß Definition 10.2 ein Proposaltoken für einen (weiteren) Vorschlag des entsprechenden Subjektes, das durch das Administratortoken $a$ referenziert wird, erzeugt wird. Zwar kann an dieser Stelle der Forderung aus Definition 10.4, dass

[3]Bei der Aktivierung dieser Transition gibt es entsprechend der Anzahl der Tokens auf *proposal ids* viele Bindungen für die Variable $p$. Die Transition kann beim Schalten eine beliebige Bindung verwenden.
[4]Dies würde bedeuten, dass die Referenz des Proposaltokens geändert wird.
[5]Bei mehreren Proposaltokens auf der Stelle *previous* werden entsprechend alle (gültigen) Belegungen mitgeteilt.

jede Variable, die in einer Kantenbeschriftung zu einer Ausgangstelle enthalten ist, auch in einer Kanteninschrift von einer Eingangsstelle verwendet werden muss, nicht direkt genüge getan werden, aber es ist definiert, was eine mögliche Belegung für $p'$ sein kann[6].
Die Kantenbeschriftung $p'$ zur Ausgangsstelle *next* zeigt an, auf welche Stelle dieses Proposaltoken abgelegt wird.

Eine weitere Kante zu einer anderen Ausgangsstelle kann ebenfalls diese Kantenbeschriftung $p'$ enthalten, ohne dass dies zur Definition 11.5 im Widerspruch steht, da lediglich gefordert wird, dass der Vorschlag durch ein Proposaltoken mit neuem Index repräsentiert werden soll. Das gleiche Proposaltoken kann also durchaus auf mehrere Ausgangsstellen gelegt werden.

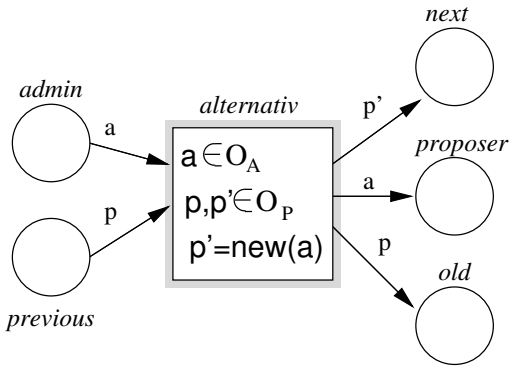



Abbildung 11.5: Komplexe Propose Transition

Wenn die Kantenbeschriftung einer Eingangsstelle *in* einer Propose Transition eine Variable $a$ für ein Administratortoken enthält, wird diese Transition bei mehreren Administratortokens auf dieser Stelle *in* mehrfach aktiviert, was nicht im Widerspruch zur Definition 11.5 steht. Jedes Subjekt, dessen Administratortoken in einer Belegung von $a$ bei der Aktivierung der Propose Transition enthalten ist, wird entsprechend benachrichtigt und nach seinem Vorschlag gefragt. Die Transition schaltet dann bei jedem eingehenden Vorschlag mit dem entsprechenden Administratortoken und legt ein Proposaltoken, welches auf den eingegangenen Vorschlag verweist, auf die zugehörige Ausgangsstelle. Die Vorschläge werden gemäß der Reihenfolge ihres Eingangs behandelt. Sollte durch das Schalten der Propose Transition diese deaktiviert werden, werden alle Subjekte gemäß Definition darüber benachrichtigt, ggf. auch darüber, dass ihr Vorschlag nicht mehr berücksichtigt wurde.
Bevor im Abschnitt 11.4 dieses genutzt wird, wird zunächst das erwähnte Beispiel eines wechselseitigen Vorschlages modelliert.

### 11.2.2 Wechselseitiger Vorschlag

Ein Ansatz, den zu Beginn dieses Abschnitts beschrieben Kooperationsprozess eines wechselseitigen Vorschlages durch ein Admin–Netz zu modellieren, ist in Abbildung 11.6 zu sehen. Sobald ein Standardtoken auf der Stelle *in* liegt, ist die Transition *init* aktiviert. Dabei sorgt das Standardtoken auf der Stelle *reset* dafür, dass diese Transition nach dem Schalten erst wieder aktiviert wird, wenn die Transition *success* geschaltet hat und damit ein Vorschlag als Ergebnis des vorangegangenen Koordinierungsprozesses auf der Stelle *result* liegt.
Doch zunächst sind nach dem Schalten von *init* die beiden Transitionen *participant 1* und *participant 2* aktiviert, die beim Schalten die Administratortokens $\textcircled{a_1}$ und $\textcircled{a_2}$ auf die Stellen *proposer* bzw. *decide* legen. Dabei muss ähnlich wie beim ersten Admin–Netz zum Prädikat approve im Abschnitt 10.3.1 bei der Auswertung jeweils genau ein Subjekt[7] die Regeln $P(s_1)$ und $P(s_2)$ erfüllen, und damit durch die Administratortokens $\textcircled{a_1}$ bzw. $\textcircled{a_2}$ repräsentiert werden.

[6]Im Abschnitt 11.3 wird eine erweiterte Version für Proposaltokens eingeführt, die es unter anderem auch ermöglicht, die verschiedenen Versionen eines Vorschlags mitzuverfolgen.
[7]Es macht wenig Sinn, wenn dies dasselbe Subjekt ist.

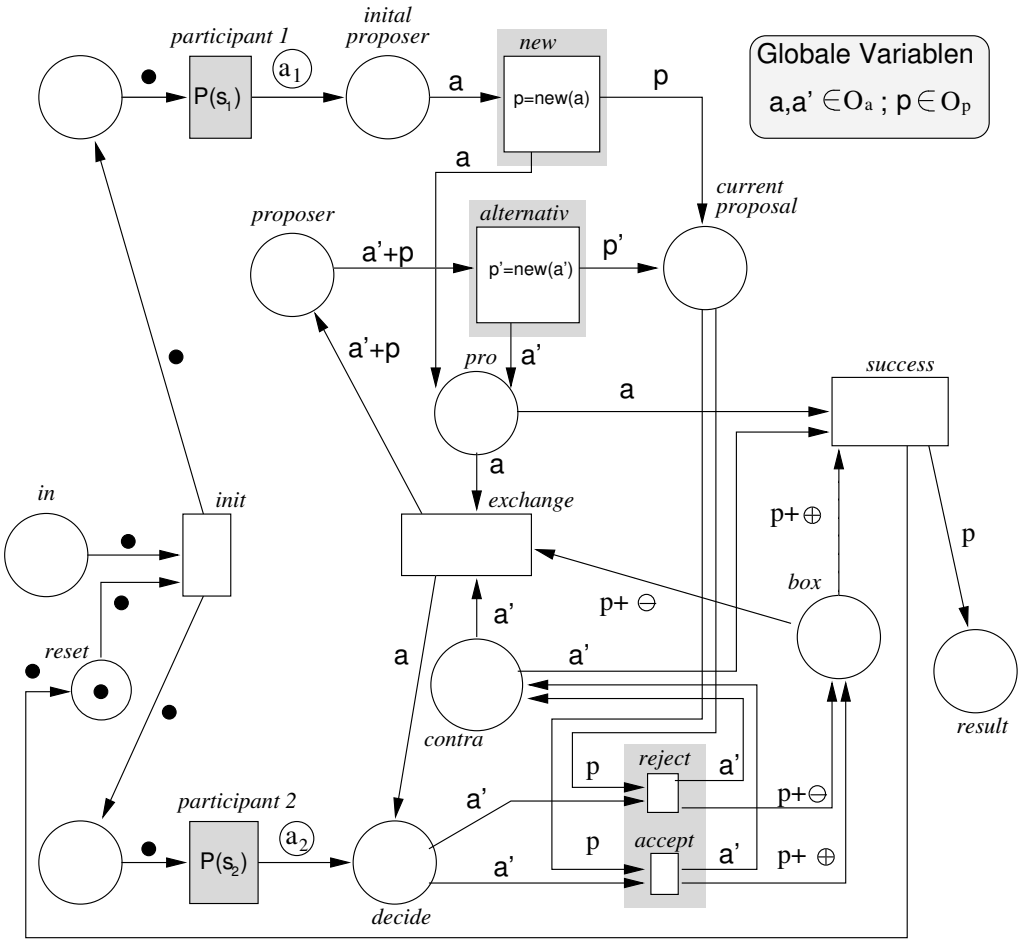



Abbildung 11.6: Wechselseitiger Vorschlag

Nachdem die Feature Transition *participant 1* geschaltet hat, aktiviert das Administratortoken des einen Administrators auf der Stelle *inital proposer* die Propose Transition *new*. Dieser Administrator wird benachrichtigt. Mit dem Eingang seines ersten Vorschlages schaltet die Transition *new* und legt dabei das Administratortoken auf die Stelle *pro* sowie das neu erzeugte Proposaltoken auf die Stelle *current proposal*. Wenn in der Zwischenzeit die Feature Transition *participant 2* geschaltet hat[8], sind nun die beiden Interaktiven Transitionen *accept* und *reject* aktiviert. Dadurch erhält der Administrator, dessen Administratortoken auf der Stelle *decide* liegt, eine Benachrichtigung. Diese enthält auch die einzige Belegung für die Variable $p$, nämlich das zuvor erzeugte Proposaltoken. Da dieses Token einen Verweis auf den tatsächlichen Inhalt des Vorschlags darstellt, kann der Administrator den Inhalt prüfen und entscheiden, ob die Transition *accept* oder *reject* schalten soll. In beiden Fällen wird "sein" Administratortoken von der Stelle *decide* entfernt und auf der Stelle *contra* abgelegt. Auf die Stelle *box* wird das Proposaltoken, das von der Stelle *current proposal* beim Schalten entfernt wird, gelegt. Abhängig welche Interaktive Transition – *accept* oder *reject* – schaltet, kommt zusätzlich noch das Token $\oplus$ bzw. $\ominus$ auf *box*.

Wenn zunächst davon ausgegangen wird, dass die Entscheidung negativ war und somit ein Proposaltoken und $\ominus$ auf der Stelle *box* liegen, ist damit die Transition *exchange* aktiviert. Die Stelle *contra* wurde mit dem Schalten der Interaktiven Transition belegt und die Stelle *pro* zum ersten Mal beim Schalten der Propose Transition *new*. Im Folgenden wird dies beim Schalten

[8] Ansonsten ist sie auf jeden Fall nach dem Schalten dieser Transition, die ja gleichzeitig mit der Feature Transition *participant 1* aktiviert wurde, aktiviert.

der Propose Transition *alternativ* geschehen. Beim Schalten sorgt die Transition *exchange* dafür, dass die Rollen der beiden Administratoren vertauscht werden, indem das Administratortoken, des Subjektes, das den letzten Vorschlag gemacht hat nun auf die Stelle *decide* gelegt wird und somit über den neu kommenden Vorschlag des Administrators entscheiden kann. Dessen Administratortoken landet zusammen mit dem Proposaltoken, das auf den alten Vorschlag verweist, auf der Stelle *proposer*.

Somit ist nun einzig die Propose Transition *alternativ* aktiviert, wodurch der Administrator, dessen Administratortoken auf *proposer* liegt und die einzig mögliche Belegung für die Variable $a'$ darstellt, benachrichtigt wird, dass er einen Vorschlag machen kann. Da die Variable $p$, die nur mit dem alten Proposaltoken belegt sein kann, auf keiner Kantenbeschriftung zu einer Ausgangsstelle auftaucht, wird dieses Token vernichtet. Das Administratortoken wird auf die Stelle *pro* gelegt und das neue Proposaltoken auf die Stelle *current proposal*. Damit sind nun allein wieder die beiden Interaktiven Transitionen *reject* und *accept* aktiviert und der erste Administrator, dessen Administratortoken $\textcircled{a_1}$ nun auf *decide* liegt, kann entscheiden, welche der beiden schalten soll. Falls der Administrator den neuen Vorschlag ablehnt, beginnt eine neue Runde, in der dieser Administrator dann seinen zweiten Vorschlag macht, über den der andere Administrator wiederum entscheidet.

Diese Runden werden solange fortgesetzt, bis einer der beiden Administratoren den Vorschlag des anderen akzeptiert und somit das Proposaltoken für den letzten und akzeptierten Vorschlag zusammen mit dem Token $\oplus$ auf der Stelle *box* liegt. Nun ist nicht die Transition *exchange*, sondern *success* aktiviert. Diese entfernt beim Schalten die beiden Administratortoken von den Stellen *pro* und *contra* und legt das Proposaltoken, das auf den akzeptierten Vorschlag verweist auf die Stelle *result*. Außerdem wird ein Standardtoken auf die Stelle *reset* gelegt, sodass nun die Transition *init* wieder aktiviert ist, wenn ein Standardtoken auch auf *in* gelegt wird.

## 11.3 Entwicklung eines Proposals

Im Abschnitt 11.2 wurde ein existierender Vorschlag stets komplett durch einen Neuen ersetzt. Dies ist vor allem bei komplexen Änderungen unter Umständen gar nicht nötig, sondern es reicht aus, einzelne Teilaspekte zu ändern. Um dies modellieren zu können, muss ein Vorschlag in verschiedene Teile zerlegt werden können. Später müssen alle, gegebenenfalls überarbeiteten, Teile, wieder zu einem Gesamtvorschlag zusammengeführt werden können.

Dazu werden zunächst im Abschnitt 11.3.1 die Referenzierungsmöglichkeiten bei Proposaltokens erweitert, wie bereits in der Definition 10.2 angekündigt. Anschließend wird im Abschnitt 11.3.2 eine neue Transition zum Aufteilen von Proposaltokens in mehrere Teilaspekte und eine zweite Transition zum Zusammenführen dieser Teilvorschläge zu einem neuen Gesamtproposal eingeführt. Die sich daraus ergebenden Möglichkeiten werden anhand eines Beispiels im Abschnitt 11.4.2 demonstriert.

### 11.3.1 Erweiterte Bezeichnung für Proposaltokens

Wenn ein Vorschlag im Laufe eines Prozesses verfeinert und abgeändert werden muss, ist es sinnvoll einen Überblick über die verschiedenen Versionen eines Vorschlags zu behalten. Hierzu wird die Schreibweise $[a, i]$ als Token genutzt. Dabei bezeichnet $a$ den Administrator, der den ursprünglichen Vorschlag gemacht hat und $i$ ist ein Zähler, der die Anzahl der neuen Vorschläge nachhält. Falls an einem Vorschlag noch Veränderungen, auch durch andere Beteiligte durchgeführt werden können, gibt das Tripel $[a, i, j]$ die Version $j$ des ursprünglich von Administrator $a$ eingereichten Vorschlags $i$ an.

Bei komplizierteren Vorschlägen, die dann zum Beispiel aus mehreren Zugriffsregeln bestehen, die zusammen gültig werden sollen, kann es bei einem Koordinierungsprozess immer wieder Uneinigkeit zu einzelnen Details geben. Daher ist es sinnvoll, wenn ein Vorschlag in mehrere disjunkte Teile zerlegt werden kann, die dann einzeln geklärt werden können. Um die einzelnen Teile zu

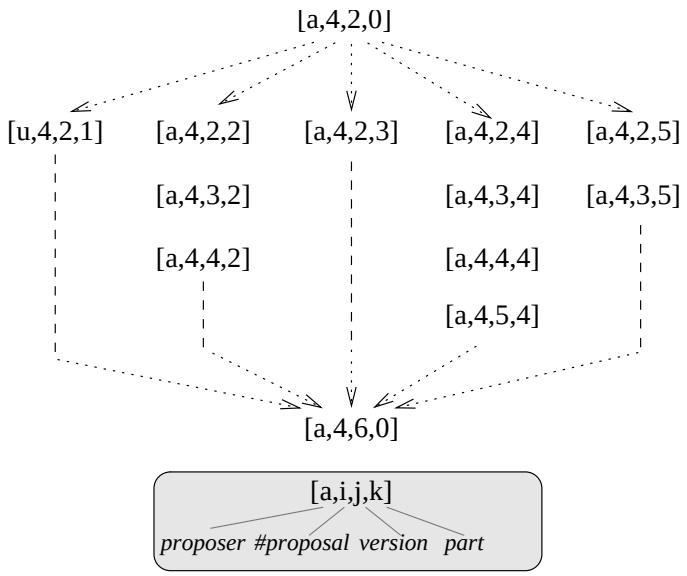



Abbildung 11.7: Versionsbaum

benennen, wird die Beschriftung eines Proposaltokens zu einem Quadrupel $[a, i, j, k]$ erweitert, bei dem an der vierten Stelle die einzelnen Teilvorschläge durchnummeriert werden. Es könnte irritierend sein, dass die Version nicht an letzter Stelle, sondern an vorletzter Stelle steht, jedoch verschieben sich so nicht die Bedeutungen der einzelnen Stellen, wenn einen Vorschlag in mehrere Teile zerlegt wird.
Wie zuvor wird vom genauen Inhalt des ursprünglichen Vorschlages und damit auch von dem der Teilvorschläge abstrahiert. Diese Aufteilung kann nicht ohne weiteres automatisch geschehen, sondern muss in der Verantwortung eines am Koordinierungsprozess beteiligten Administrators liegen. Im folgenden Abschnitt 11.3.2 wird gezeigt, wie dies in einem Admin–Netz modelliert werden kann.

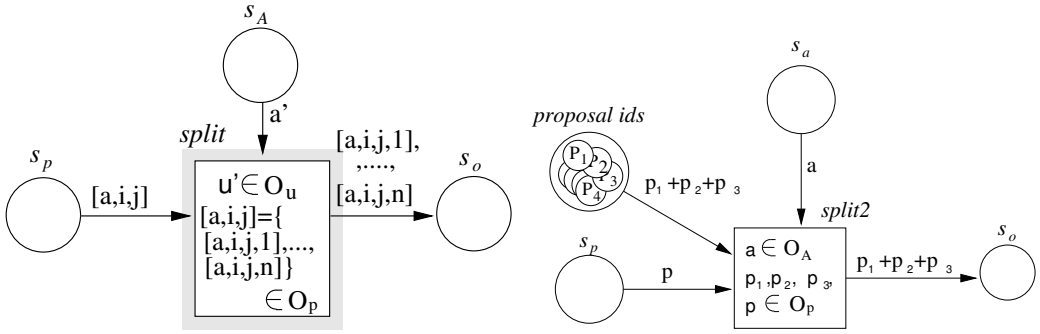
Das Vorgehen bei mehreren Teilvorschlägen ist in Abbildung 11.7 an einem Vorschlag, der mit $[a, 4, 2, 0]$ bezeichnet wird, exemplarisch skizziert. Zuvor kann dieser Vorschlag auch alternativ mit $[u, 4, 2]$ benannt worden sein, da die vierte Stelle noch nicht signifikant war. Es handelt sich also um den vierten, neuen Vorschlag den ein Administrator $a$ gemacht hat und der bereits zweimal überarbeitet wurde. Dieser Vorschlag wird in einem ersten Schritt in fünf Teilvorschläge zerlegt, die als $[a, 4, 2, 1]$, $[a, 4, 2, 2]$, $\ldots$, $[a, 4, 2, 5]$ bezeichnet werden. Dies ist nur eine Aufteilung des Gesamtvorschlages, daher wird bei diesem Schritt darauf verzichtet, die Versionsnummer zu erhöhen. Jeder einzelne dieser Teilvorschläge kann für sich genommen verändert werden, wobei jedes Mal die Versionsnummer dieses Teilvorschlages solange erhöht wird. Dies soll solange geschehen, bis alle Teilvorschläge akzeptiert sind. Im Beispiel bleiben der erste und dritte Teilvorschlag unverändert, während der Zweite einmal, der Fünfte zweimal und der vierte Teilvorschlag dreimal verändert wird. Somit hat der vierte Teilvorschlag die Versionsnummer 5.
Wenn diese Teilvorschläge wieder zu einem Gesamtvorschlag zusammengeführt werden, kann dies einfach geschehen, indem die Inhalte zusammengelegt werden. Bei diesem neuen Gesamtvorschlag muss die Versionsnummer um eins höher sein, als die maximale Versionsnummer aller Teilvorschläge. Die Bezeichnung nun $[a, 4, 6, 0]$ oder alternativ $[a, 4, 6]$. Der Grund für die Erhöhung der Versionsnummer beim Zusammenführen liegt darin, dass es theoretisch möglich ist, dass dieser Vorschlag erneut aufgeteilt wird. Wenn dies in vier oder mehr Teilvorschläge geschieht, gibt es wiederum einen Teilvorschlag mit der Bezeichnung $[a, 4, 5, 4]$. Da die Grenzen bei der Aufteilung jedes Mal beliebig gewählt werden können, würde dieser Teilvorschlag unter Umständen nicht mit dem Vorangegangenen übereinstimmen, was nicht mit der Definition 10.2 konform wäre[9].

[9] Alternativ könnte man auch die Versionsnummer jedes Mal beim Aufteilen erhöhen, diese Vorgehensweise wird

### 11.3.2 Transform Transitionen

Um das vorangegangene Beispiel in einem Admin–Netz zu modellieren, werden zwei besondere Transaktionen benötigt, eine Transition zum Aufteilen eines Vorschlags, die andere zum Zusammenführen der Teilvorschläge zu einem neuen Gesamtvorschlag.

**Definition 11.6 (Split Transition)**
Eine Split Transition $t_S$ ist eine ausgezeichnete Transition, die aufgrund der Kantenbeschriftung von den Eingangsstellen und den Transitionsinschriften zur Aktivierung genau einen Gesamtvorschlag, der Form $[a, i]$ bzw. $[a, i, j]$ und ein Administratortoken $a'$ sowie beliebige weitere Tokens anderen Typs benötigt. Das Subjekt, dessen Administratortoken an der Aktivierung beteiligt ist, wird entsprechend benachrichtigt, wobei ihm insbesondere das Proposaltoken bzw. die Belegung der entsprechenden Variable sowie die möglichen weiteren Tokens bzw. Belegungen der anderen Eingangsstellen mitgeteilt werden.
Das Subjekt kann nun den Vorschlag, auf den durch das Proposaltoken von der Eingangsstelle verwiesen wird, in eine beliebige Anzahl von $n$ Teilvorschlägen aufteilen. Durch das Festlegen dieser Aufteilung schaltet die Split Transition. Dabei wird insbesondere auf mindestens eine Ausgangsstelle, deren Kantenbeschriftung die Form $[a, i, j, 1], \ldots [a, i, j, n]$ hat, entsprechend der Aufteilung $n$ Proposaltokens gelegt, die auf die Teilvorschläge verweisen, deren Anzahl festgehalten wird.
Falls die Split Transition nach ihrer Aktivierung wieder deaktiviert wird, wird das Subjekt darüber benachrichtigt.




(a) Aufteilen eines Vorschlags in mehrere Teilvorschläge

(b) Alternative Darstellung für das Aufteilen

Abbildung 11.8: Split Transition

In Abbildung 11.8(a) ist ein Beispiel für eine Split Transition zu sehen. Das Subjekt dessen Administratortoken auf der Stelle $s_a$ liegt, wird bei einer Aktivierung benachrichtigt und aufgefordert, eine Aufteilung des Vorschlags vorzunehmen, dessen Proposaltoken auf der Stelle $s_p$ liegt. Sobald dies geschieht, wird für jeden Teilvorschlag ein Proposaltoken auf der Stelle $s_o$ platziert, die dann entsprechend des Koordinierungsprozesses weiter geleitet werden. Wenn im Vorfeld die Anzahl der Teilvorschläge bekannt ist[10], kann dieser Vorgang auch mit den in Abbildung 11.8(b) dargestellten Stellen und Transitionen modelliert werden. Die weitere Vorgehensweise ist analog zu der bei einer Propose Transition im Abschnitt 11.2.1. Auf einer Stelle *proposal id* werden zu Beginn ausreichend viele Proposaltokens platziert[11], bei jedem Schalten der Transition *split2* würden dann entsprechend viele Proposaltokens für jeden der drei Teilvorschläge auf die Stelle $s_o$ gelegt.

Falls bei der Eingangskante zu einer Split Transition durch die Beschriftung $[a, i]$ nicht auf die Versionsnummer eingegangen wird, erhält die Variable $j$ bei den Teilvorschlägen automatisch die Belegung 1. Darüber hinaus kann auch einfach verhindert werden, dass ein Teilvorschlag erneut

hier nicht genutzt.

[10] Im Beispiel wird von 3 Teilvorschlägen ausgegangen.

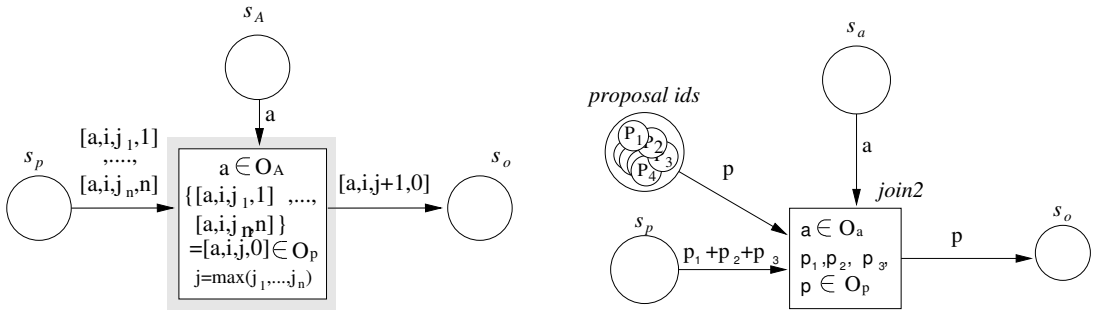
[11] Andere Split Transitionen und Proposal Transitionen, sowie die im folgenden dargestellte Join Transition wären dann ebenfalls mit dieser Stelle verbunden.

in mehrere Teile zerlegt wird, indem die Kantenbeschriftung an der Eingangsstelle dies explizit durch die Beschriftung $[a, i, j, 0]$ verlangt.
Das Festhalten der Anzahl der Teilvorschläge, zum Beispiel durch eine interne Speicherung, dass es sich um den Teilvorschlag $m$ von $n$ handelt, ist für das spätere Zusammenführen wichtig, da so festgestellt werden kann, ob alle Teilvorschläge vorliegen[12].

Um die überarbeiteten Teilvorschläge, wie im vorangegangenen Abschnitt 11.3.1 dargestellt, wieder zusammenführen zu können, wird eine weiter besondere Transition benötigt.

**Definition 11.7 (Join Transition)**
Eine Join Transition $t_J$ ist eine ausgezeichnete Transition, die aufgrund ihrer Kantenbeschriftung von einer Eingangsstelle alle Proposaltokens eines aufgeteilten Vorschlags $[a, i, j_1, 1], \ldots [a, i, j_n, n]$ und von einer weiteren Eingangsstelle das Administratortoken $a'$ eines verantwortlichen Subjektes zur Aktivierung benötigt. Eine Transitionsinschrift setzt eine Variable auf die maximale Versionsnummer eines Teilvorschlages ($j = max(j_1, \ldots, j_n)$). Bei der Aktivierung wird das Subjekt entsprechend benachrichtigt. Wenn dieses den aus den Teilvorschlägen zusammengesetzten Gesamtvorschlag zurückschickt, schaltet die Transition $t_J$ und ein neues Proposaltoken, das den zusammengesetzten Gesamtvorschlag repräsentiert, wird auf Ausgangsstelle $s_o$ gelegt, dessen Kante mit $[a, i, j + 1, 0]$ beschriftet ist.

Neben der Verantwortlichkeit für die Zusammenführung kann durch das Ablegen eines Administratortokens auf einer Eingangsstelle auch signalisiert werden, dass nun alle Proposaltokens der Teilvorschläge auf der anderen Eingangsstelle liegen[13]. Wie zuvor bereits erwähnt, wird intern die Anzahl der Teilvorschläge vermerkt, sodass die Bedingung, dass alle zugehörigen Proposaltokens auf einer Eingangsstelle liegen, leicht verifiziert werden kann. Die Abbildung 11.9(a) zeigt beispielhaft eine solche Join Transition.




(a) Zusammenführen der Teilvorschläge zu einem Gesamtvorschlag

(b) Alternative Darstellung für das Zusammenführen zu einem Gesamtvorschlag

Abbildung 11.9: Join Transitionen

Wenn die Anzahl der Teilvorschläge bekannt ist, ist die alternative Darstellung für eine Join Transition, wie sie in Abbildung 11.9(b) zu sehen ist, symmetrisch zur Split Transition. Wobei hier nun drei Proposaltokens von der Eingangsstelle $s_p$ die Transition *join2* aktivieren, beim Schalten abgezogen werden und ein einziges neues Proposaltoken für den Gesamtvorschlag auf die Ausgangsstelle $s_o$ gelegt wird. Bei dieser Darstellung geht leider verloren, dass zur Aktivierung auf den Eingangsstellen die Proposaltokens benötigt werden, deren Vorgänger oder die zuvor selbst beim Aufteilen eines gemeinsamen Gesamtvorschlages entstanden sind. Dies ließe sich aber relativ einfach durch zusätzliche Stellen nachhalten, auf denen entsprechende zusammengesetzte Tokens abgelegt werden.

[12] Diese Vorgehensweise ist der für die Partitionierung von Files im DRAGON SLAYER System angelehnt [WL99].
[13] Dies kann natürlich auch durch weitere Eingangsstellen erreicht werden, die aber nicht zwingend notwendig sind.

## 11.4 Workflow

Um die Möglichkeiten der zuvor beschriebenen Transitionen zu demonstrieren, wird nun ein umfangreiches Beispiel entwickelt. Dabei werden die Abstrahierungsmöglichkeiten durch Subnetze ausgenutzt, um in drei Teilschritten einen komplexen Prozess zu entwickeln. Ein wichtiger Aspekt ist dabei das Einbringen von Vorschlägen sowie deren Aufteilung in Teilvorschläge und deren Überarbeitung.

Für dieses Beispiel wird angenommen, dass ein Administrator, der als "Chairman" bezeichnet wird, zunächst einen Vorschlag einbringt, der von einer Gruppe von weiteren Administratoren, die alle zusammen nur als "Board" bezeichnet werden, bestätigt werden soll. Wenn der Vorschlag direkt akzeptiert wird, kann er ohne eine weitere Veränderung umgesetzt werden. Der interessantere Fall ist jedoch, wenn es keine Zustimmung zu dem ersten Vorschlag gibt, worauf weitere Administratoren herangezogen werden, die als "Experts" bezeichnet werden.
Nun wird der Vorschlag vom "Chairman" in Teilvorschläge aufgeteilt, die dann einzeln betrachtet werden sollen. Dabei fällt den Experten die Aufgabe zu, jeden Teilvorschlag solange zu überarbeiten, bis er vom "Board" bestätigt wird. Wenn alle Teilvorschläge so überarbeitet und akzeptiert sind, wird vom "Chairman" aus diesen Teilen wieder ein neuer Gesamtvorschlag erzeugt.
Dieser Gesamtvorschlag wird noch einmal abschließend dem Board vorgelegt und das Ergebnis entsprechend nach außen getragen. Wenn dieses Ergebnis eine Zustimmung war, gilt der Vorschlag als genehmigt, ansonsten als endgültig abgelehnt.

Dieser gesamte Workflow–Prozess wird in einzelne Subnetze aufgeteilt, die im Folgenden in einer Button–Up Reihenfolge besprochen werden. Zunächst wird im Abschnitt 11.4.1 die Bestätigung eines Vorschlages durch die Administratoren im "Board" in einem Subnetz spezifiziert. Dieses Subnetz wird sowohl bei Bestätigung der Gesamtvorschläge eingesetzt, als auch später bei Bestätigung der Teilvorschläge. Die Aufteilung und Bearbeitung der Teilvorschläge wird in einem Subnetz spezifiziert, das im Abschnitt 11.4.2 besprochen wird. Schließlich wird im Abschnitt 11.4.3 ein Admin–Netz vorgestellt, das den gerade beschriebenen Prozess spezifiziert. Diese Admin–Netze sind eine überarbeitete Version des in [WL04a] präsentierten Workflows.

### 11.4.1 Bestätigung

Sowohl die Bestätigung für einen Gesamtvorschlag, als auch für einzelne Teilvorschläge, soll mit einfacher Mehrheit getroffen werden. Die beteiligten Administratoren sowie der abzustimmende Vorschlag sollen extern vorgegeben werden. Zur weiteren Nutzung im Gesamtprozess sollen sowohl die beteiligten Administratoren, als auch der Vorschlag sowie das Ergebnis der Bestätigung nach Abschluss des Prozesses zur Verfügung stehen.

In Abbildung 11.10 ist ein Admin–Netz, das diesen Prozess spezifiziert, dargestellt. Dieses Admin–Netz ist eine inkrementelle Weiterentwicklung des als Quorum bezeichneten Admin-Netzes aus Abbildung 10.9. Dabei ist die Feature Transition *init* entfernt, da angenommen wird, dass die Administratortokens der Beteiligten, auf der Eingangsstelle *participants* liegen. Die neu eingefügte Konstruktion vor den Interaktiven Transitionen entspricht weitestgehend der aus Abbildung 11.2.

Im Admin–Netz in Abbildung 11.10 unterscheiden sich die Ausgangskanten der Interaktiven Transitionen sowie die Eingangskanten der Transitionen *stalement*, *acceptance*, *rejection*, *indiff* und *clean* von denen im Admin–Netz in Abbildung 10.9 dadurch, dass statt des einzelnen Tokens $x : x \in \{\ominus, \oplus, \circ, \textcircled{T}\}$ gekoppelte Tokens der Form $(a, p, x)$ stehen, mit $a \in O_A, p \in O_p, x \in \{\ominus, \oplus, \circ, \textcircled{T}\}$. Alle Administratortokens werden letztlich auf der Stelle *participants–out* abgelegt.

Der Beginn des Koordinierungsprozesses wird durch eine Anfangsmarkierung $M_0^Q$ repräsentiert, bei der eine beliebige Menge von Administratortokens auf der Stelle *participants-in*, sowie genau ein Proposaltoken auf der Stelle *proposal-in* und ein Standardtoken auf der Stelle *in* und eines auf der Stelle *reset* liegen. Die übrigen Stellen sind bei der Markierung $M_0^Q$ leer.
Bei der Markierung $M_\ominus^Q$ liegt auf der Stelle *participants-out* die gleiche Menge von Administratortokens, die bei der Markierung $M_0^Q$ auf *participants-in* liegt. Außerdem befindet sich ein Standardtoken jeweils auf *reset* und *out*. Auf der Stelle *result* befindet sich ein Tupel bestehend aus

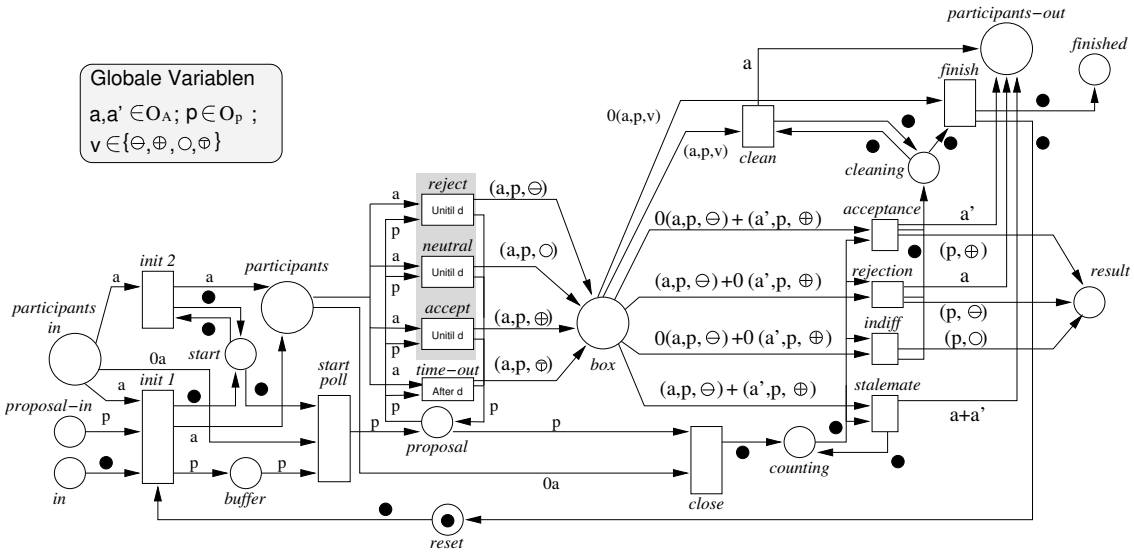



Abbildung 11.10: Quorum als Subnetz

dem Token $\ominus$ und einem Proposaltoken, dem Gleichen, das bei Markierung $M_0^Q$ auf *proposal-in* liegt.
Die Markierung $M_\oplus^Q$ unterscheidet sich von der Markierung $M_\ominus^Q$ nur insofern, dass auf der Stelle *result* ein Tupel bestehend aus einem Proposaltoken[14] liegt, das in diesem Fall mit dem Token $\oplus$ gekoppelt ist. Entsprechend unterscheidet sich die Markierung $M_\circ^Q$ von $M_\oplus^Q$ bzw. $M_\ominus^Q$ nur darin, dass das Tupel auf *result* aus dem Proposaltoken und dem Token $\circ$ besteht. Dann gilt:

**Lemma 11.1**
Der Koordinierungsprozess, der durch das Admin–Netz in Abbildung 11.10 spezifiziert wird, erreicht ausgehend von der Anfangsmarkierung $M_0^Q$ stets eine der drei folgenden Markierungen $M_\ominus^Q$, $M_\oplus^Q$ oder $M_\circ^Q$, welche drei mögliche Abschlüsse des Prozesses darstellen.
Wenn die Interaktive Transition *reject* häufiger geschaltet hat als die Interaktive Transition *accept*, wird die Markierung $M_\ominus^Q$ erreicht, was eine Ablehnung des Administrierungsvorschlages darstellt.
Wenn die Interaktive Transition *accept* häufiger geschaltet hat als die Interaktive Transition *reject*, wird die Markierung $M_\oplus^Q$ erreicht, wodurch eine Annahme des Administrierungsvorschlages dargestellt wird.
Wenn die beiden Interaktive Transition *accept* und *reject* gleich häufig geschaltet haben, wird die Markierung $M_\circ^Q$ erreicht, wodurch ein Unentschieden dargestellt wird.
Bei der Markierung $M_\oplus^Q$, $M_\ominus^Q$ bzw. $M_\circ^Q$ ist keine Transition im Admin–Netz aus Abbildung 11.10 mehr aktiviert. Die Anzahl $n$ der Administratortokens auf *particpants-in* kann beliebig sein.

**Beweis zu Lemma 11.1**

1. Beim Schalten von *init1* wird genau ein Proposaltoken von der Stelle *proposal-in* auf *buffer* verschoben, ein Standardtoken von *in* auf *start*, sowie ein Administratortoken von *participants-in* auf *participants*. Die übrigen $n-1$ Administratortokens auf *participants-in* werden beim Schalten von *init2* einzeln auf *participants* verschoben. Die Transition *start poll* schaltet erst, wenn die Stelle *participants-in* leer ist und verschiebt dabei das Proposaltoken von *buffer* auf *proposal*. Auf *participants* liegen dann $n$ Administratortokens.

2. Die Interaktiven Transitionen sowie *time-out* schalten zusammen insgesamt $n$ mal, analog zu den Punkten 2 und 3 im Beweis 10.3. Dadurch liegen $n$ gekoppelte Tokens der Form $(a, p, x)$ mit $a \in O_A, p \in O_p, x \in \{\ominus, \oplus, \circ, \overline{\oplus}\}$ auf *box*, wenn *close* aktiviert ist.

[14] Auch bei dieser Markierung ist das Proposaltoken das gleichen, das bei Markierung $M_0^Q$ auf *proposal-in* liegt.

3. Die Argumentation aus Punkt 4 im Beweis 10.3 gilt auch für dieses Admin–Netz mit dem Unterschied, dass anstelle eines einzelnen Tokens $x$ auf *result*, ein gekoppeltes Token der Form $(p, x)$ mit $p \in O_p, x \in \{\ominus, \oplus, \circ\}$ abgelegt wird.

4. Beim Schalten von *acceptance*, *reject* sowie *clean* wird jeweils ein gekoppeltes Token der Form $(a, p, x)$ von *box* entfernt und ein entsprechendes Administratortoken $a$ auf *participants-out* gelegt. Beim Schalten von *stalement* werden zwei gekoppelte Tokens $(a, p, \oplus)$ und $(a', p, \ominus)$ entfernt und entsprechend auch zwei Administratortokens $a$ und $a'$ abgelegt. Die Transition *finish* schaltet, wenn keine Tokens mehr auf *box* liegen. Dabei wird jeweils ein Standardtoken auf *out* und *reset* gelegt. Es befinden sich dann $n$ Administratortokens auf *participants-out*. Da die Administratortokens nur gekoppelt und entkoppelt werden, ist diese Menge identisch mit der, die auf *participants-in* lag.

□

Das Admin–Netz in Abbildung 11.10 soll als Subnetz verwendet werden. Dabei wird angenommen, dass *participants-in*, *proposal-in* sowie *in* als Eingangsstellen und *participants-out*, *result* sowie *out* als Ausgangsstellen verwendet werden. Damit sind *init1*, *init2* sowie *start poll* Eingangstransitionen und *stalemate*, *acceptance, rejection, indiff, clean* sowie *finish* Ausgangstransitionen. Bei der Anfangsmarkierung sind alle inneren Stellen dieses Subnetzes leer, mit Ausnahme von *reset*, auf der ein Standardtoken liegt.

**Satz 11.1**
Wenn das Admin–Netz aus Abbildung 11.10, wie angenommen, als Subnetz verwendet wird, ist es wohlgeformt und verhält sich atomar. Eine Eingangstransition schaltet nur bei einer Markierung $M_0^Q$. Danach wird stets genau eine der drei Markierungen $M_\ominus^Q$, $M_\oplus^Q$ oder $M_\circ^Q$ erreicht.

**Beweis zu Satz 11.1**
Vor dem Schalten von *init1* ist keine der Transitionen des Admin–Netzes aktiviert, sodass sich die Markierung der inneren Stellen nicht ändern kann.
Eine Ausgangstransition wird erst aktiviert, wenn *close* geschaltet hat. Zuvor haben mindestens die Eingangstransitionen *init1* und *start poll* geschaltet sowie *init2*, abhängig von der Anzahl der Administratortokens auf *participants-in*.
Aufgrund der Eingangskante von *reset* nach *init1* kann diese Eingangstransition erst schalten, wenn *finish* geschaltet hat. Alle inneren Stellen sind dabei leer, somit sind alle anderen Ausgangstransitionen dann auch deaktiviert.
Gemäß Lemma 11.1 sind die möglichen erreichbaren Markierungen von $M_0^Q$, die Markierungen $M_\ominus^Q$, $M_\oplus^Q$, $M_\circ^Q$. Bei allen vier Markierungen ist die Belegung der inneren Stellen identisch.
Wenn nach dem Schalten der ersten Eingangstransition *init1* Tokens durch Transitionen von der Eingangsstelle *participants-in* entfernt werden, die nicht zum Subnetz gehören, verhält sich das Subnetz so, als wäre dies vor dem Schalten der ersten Eingangstransition geschehen. Insbesondere gilt dies für die variable Anzahl der Administratortokens in den Markierung $M_0^Q$ und entsprechend für $M_\ominus^Q$, $M_\oplus^Q$ sowie $M_\circ^Q$.

□

Dieses Subnetz wird in den im Folgenden beschriebenen Admin–Netzen genutzt.

### 11.4.2 Bearbeitung eines Gesamtvorschlages in Teilvorschlägen

Das Admin–Netz in Abbildung 11.11 modelliert den Teil des Prozesses, in dem der (abgelehnte) Vorschlag in Teilvorschläge zerlegt wird, die einzeln modifiziert und bestätigt werden sollen, wobei hierzu das Subnetz aus Abbildung 11.10 verwendet wird. Dabei wird angenommen, dass nach einer endlichen Anzahl von $r_{max}$ Überarbeitungen, jedem Teilvorschlag schließlich zugestimmt wird. Diese überarbeiteten Teilvorschläge werden anschließend wieder zu einem Gesamtvorschlag zusammengeführt.

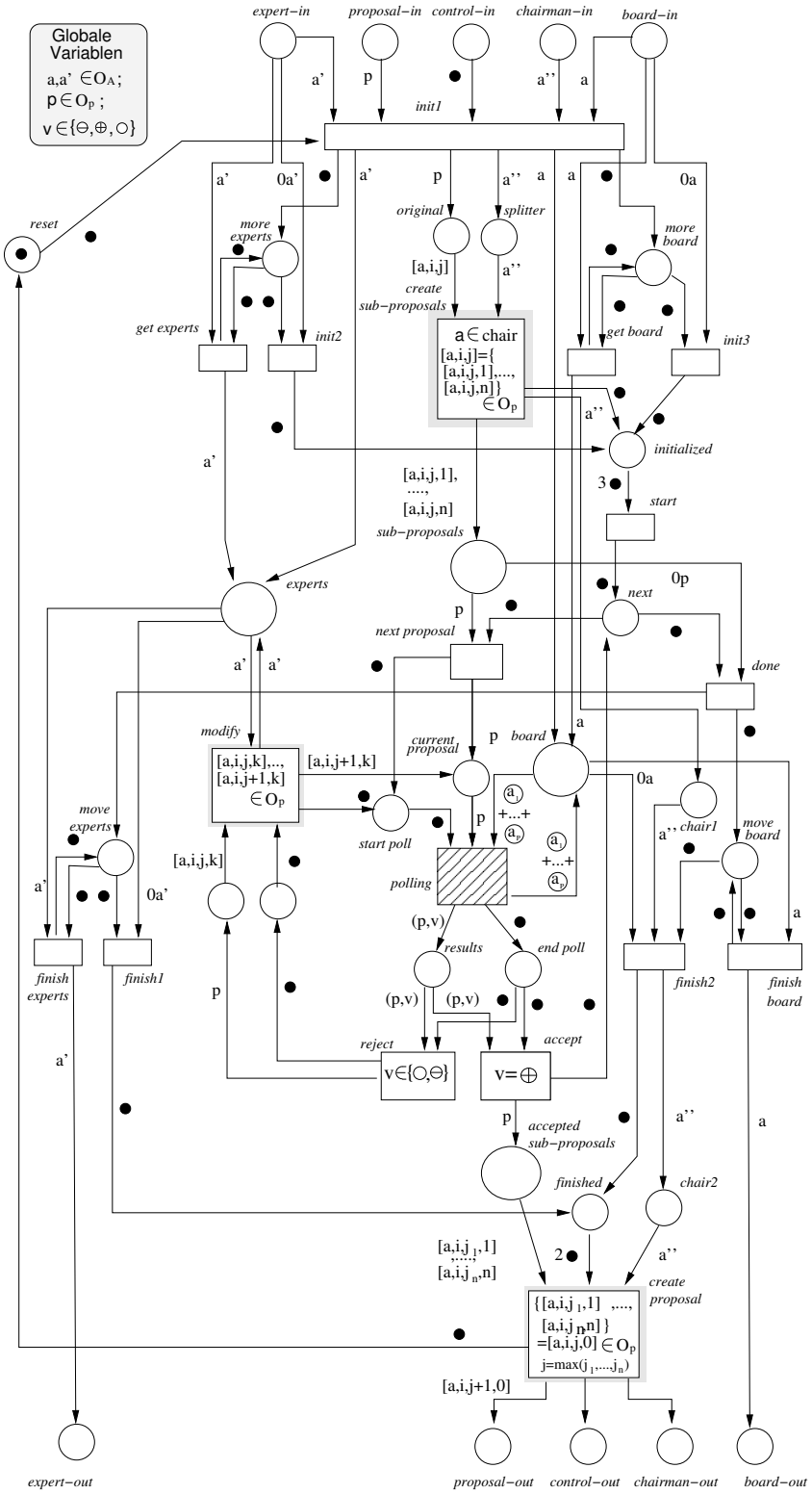



Abbildung 11.11: Bearbeitung von Teilvorschlägen

Bei den möglichen Anfangsmarkierungen $M_0^S$ für das Admin–Netz in Abbildung 11.11 liegt ein Proposaltoken auf *proposal-in*, ein Standardtoken jeweils auf *control-in* und *reset* sowie ein Administratortoken auf *chairman-in*, darüber hinaus liegt eine Menge von $l \geq 1$ Administratortokens auf *board-in* und eine weitere Menge von $l' \geq 1$ Administratortokens auf *expert-in*. Alle anderen Stellen sind bei der Markierung $M_0^S$ leer. Ausgehend von dieser Markierung $M_0^S$ wird letztlich eine Markierung $M_{finish}^S$ erreicht, bei der jeweils ein Standardtoken auf *control-out* und *reset* liegt. Auf der Stelle *chairman-out* liegt das gleiche Administratortoken, das bei Markierung $M_0^S$ auf *chairman-in* lag. Auf den Stellen *experts-out* und *board-out* die gleichen Mengen von Administratortokens, die bei $M_0^S$ auf *experts-in* bzw. *board-in* lagen. Abhängig von dem Proposaltoken das bei $M_0^S$ auf *proposal-in* lag und gemäß Abschnitt 11.3.1 auch als Token $[a, i, j]$ bezeichnet werden kann, wird bei der Markierung $M_{finish}^S$ das Proposaltoken $[a, i, j + r_{max} + 1]$ auf *proposal-out* liegen.

Obwohl die Abstimmung über die Teilvorschläge in ein Subnetz ausgelagert ist, ist der Umfang des Admin–Netzes so groß, dass die Wohlgeformtheit dieses Netzes besser in Teilschritten gezeigt werden kann.

### Initialisierung und Aufteilung

Ausgehend von einer Anfangsmarkierung $M_0^S$ mit $l'$ Administratortokens auf *experts-in* und $l$ Administratortokens auf *board-in* liegen bei der zugehörigen Markierung $M_{init}^S$ dieselben $l$ Administratortokens von *experts-in* auf *experts* sowie dieselben $l$ Administratortokens von *board-in* auf *board*. Das Administratortoken, welches bei $M_0^S$ auf *chairman-in* liegt, befindet sich nun auf *chair1*. Auf der Stelle *sub-proposals* befinden sich $n$ Proposaltokens und drei Standardtokens liegen auf der Stelle *initalized*. Alle anderen Stellen sind bei der Markierung $M_{init}^S$ leer.

**Lemma 11.2**
Der Koordinierungsprozess, der durch das Admin–Netz in Abbildung 11.11 spezifiziert wird, erreicht ausgehend von einer Anfangsmarkierung $M_0^S$ stets die entsprechende Markierung $M_{init}^S$. Die Zahl der Administratortokens $l \geq 1$ und $l' \geq 1$, sowie der Proposaltokens $n \geq 1$ kann beliebig sein.

**Beweis zu Lemma 11.2**
Bei einer Markierung $M_0^S$ ist ausschließlich die Transition *init1* aktiviert. Beim Schalten wird das Proposaltoken von *proposal-in* auf *original* verschoben, sowie das Administratortoken von *chairman-in* auf *splitter*. Das Standardtoken von *control-in* entfernt und jeweils ein Standardtoken auf *more experts* bzw. *more board* gelegt. Darüber hinaus wird jeweils ein Administratortoken von *expert-in* auf *experts* bzw. von *board-in* auf *board* verschoben. Nach dem Schalten von *init1* sind genau drei Transitionen aktiviert und können unabhängig voneinander schalten.
Die Transition *create sub-prosals* ist nach dem Schalten von *init1* aktiviert und legt gemäß Definition 11.6 beim Schalten $n$ Proposaltokens auf *sub-proposals*. Darüber hinaus wird das Administratortoken von *splitter* nach *chair1* verschoben. Außerdem wird ein Standardtoken auf *initialized* gelegt.
Wenn bei $M_0^S$ gilt $l' > 1$, ist *get board* aktiviert und kann noch $l' - 1$ mal schalten. Dabei wird jeweils ein Administratortoken von *board-in* auf *board* verschoben, wobei das Standardtoken auf *more experts* reproduziert wird. Danach oder wenn $l' = 1$ bei $M_0^S$ gilt, ist *init2* aktiviert und verschiebt das Standardtoken von *more experts* auf *initalized*.
Analog hierzu ist bei $l > 1$ *get board* aktiviert und verschiebt die Administratortokens von *board-in* auf *board*. Wenn *board-in* leer ist, schaltet *init3* und verschiebt das Standardtoken von *more board* nach *initalized*. □

Die Mindestanzahl der Administratortokens auf *expert-in* und *board-in* kann durch die Änderung der Kantenbeschriftungen sowohl von den entsprechenden Eingangs- als auch zu den Ausgangsstellen von *init1* erfolgen. Wenn beispielsweise mindestens 2 Administratoren als Experten und 3 Administratoren als Board teilnehmen sollen, kann dies durch die Änderung der Kantenbeschriftung von $a'$ zu $a'_1 + a'_2$ bzw. $u$ zu $a_1 + a_2 + a_3$ erreicht werden. Entsprechend muss dann

gelten $l' \geq 2$ und $l \geq 3$. Die Anzahl der Schaltvorgänge von *get experts* bzw. *get board* verändert sich entsprechend.

Der Administrator, auf den das Administratortoken auf *splitter* verweist, wird bei der Aktivierung der Transition *create sub-proposals* aufgefordert den Vorschlag, auf den das Proposaltoken verweist, in beliebig viele Teilvorschläge zu zerlegen. Sobald diese Aufteilung festgelegt wird, schaltet die Split Transition und legt für jeden der $n$ Teilvorschlag ein neues Proposaltoken auf die Stelle *sub-proposals*.

Bei der Markierung $M^S_{init}$ ist lediglich die Transition *start* aktiviert. Beim Schalten werden drei Standardtokens von *initalized* entfernt und ein Standardtoken auf *start* abgelegt. Diese direkte Folgemarkierung wird als $M^S_{start}$ bezeichnet.

### Bearbeitung der Teilvorschläge

Als Nächstes soll für jeden Teilvorschlag eine Bestätigung eingefordert werden. Falls diese nicht zustande kommt, wird der Teilvorschlag solange von den Experten verändert, bis dieser schließlich als positiv bestätigt wird. Erst danach wird der nächste Teilvorschlag bearbeitet.

Die Markierung $M^S_{done}$ ist eine entsprechende Folgemarkierung von $M^S_{start}$, bei der die Markierungen auf *board* und *experts* identisch zu $M^S_{start}$ sind. Für jedes der $n$ Proposaltokens der Form $[a, i, j, k]$ mit $1 \leq k \leq n$, das bei $M^S_{start}$ auf *sub-proposal* liegt, befindet sich nun ein Proposaltoken $[u, i, j + r_k, k]$ mit $0 \leq r_k \leq r_{max}$ auf *accepted sub-proposals*. Auf *move board* und *move experts* liegt jeweils ein Standardtoken. Alle anderen Stellen (insbesondere *sub-proposals*) sind leer.

Tabelle 11.1: Zuordnung der Stellen

| Bearbeitung von Teilvorschlägen (Abb. 11.11 ) | Quorum als Subnetz (Abb. 11.10) |
|---:|---|
| *board* | *participants-in* |
| *start poll* | *in* |
| *current proposal* | *proposal-in* |
| *board* | *participants-out* |
| *results* | *proposal-out* |
| *end poll* | *out* |

Das Admin–Netz aus Abbildung 11.10 ist als Subnetz mit der Bezeichnung *Quorum* in dieses Netz eingebunden. Die Zuordnung der Eingangs- und Ausgangsstellen ist der Tabelle 11.1 zu entnehmen. Es gilt:

**Lemma 11.3**
Ausgehend der Markierung $M^S_{start}$ erreicht das Admin–Netz in Abbildung 11.11 stets die Markierung $M^S_{done}$. Wobei eine maximale Anzahl von $r_{max}$ Überarbeitungen für jeden der $n$ Proposaltokens vorausgesetzt wird.

**Beweis zu Lemma 11.3**

1. Durch das Schalten der Transition *next proposal* wird ein beliebiges Proposaltoken $p \in O_P$ von der Stelle *sub-proposals* nach *current proposal* verschoben sowie das Standardtoken von *next* auf *start poll*. Nach dem Schalten ist das Subnetz aus Abbildung 11.10 mit der Markierung $M^Q_0$ aktiviert. Gemäß Satz 11.1 ist das Subnetz wohlgeformt und verhält sich atomar. Damit liegt nach dem Schalten des Subnetzes eine von dessen drei möglichen Endmarkierung $M^Q_\ominus, M^Q_\circ$ oder $M^Q_\oplus$ vor, sodass ein Standardtoken auf *end poll* und entweder das gekoppelte Token $(p, \ominus)$, $(p, \circ)$ oder $(p, \oplus)$ auf *results* liegt.

2. Falls eines der ersten beiden gekoppelten Token auf *results* liegt, schaltet *reject*. Damit ist die Transition *modify* aktiviert. Gemäß Definition 11.5 wird beim Schalten ein neues Proposaltoken auf *current proposal* gelegt. Außerdem wird das Standardtoken auf *start poll* verschoben. Damit ist das Subnetz erneut mit der Markierung $M^Q_0$ aktiviert und kann wie

im Punkt 1 schalten.
Es wird angenommen, dass nach spätestens $r_{max}$ Schaltungen des Subnetzes ein gekoppeltes Token der Form $(p, \oplus)$ auf *results* liegt. Damit ist *accept* aktiviert. Beim Schalten wird nur das Proposaltoken $p$ aus dem gekoppelten Token auf *accepted sub-proposals* geschoben, gleichzeitig wird das Standardtoken von *end poll* auf *next* verschoben.

3. Solange noch weitere Proposaltokens auf *sub-proposals* vorhanden sind, schaltet nun *next proposal* und die Punkte 1 und 2 werden mit dem neuen Proposaltoken durchlaufen.

4. Nachdem *next proposal* insgesamt $n$ mal geschaltet hat, ist *sub-proposals* leer und *done* aktiviert. Beim Schalten wird das Standardtoken von *next* entfernt und jeweils ein Standardtoken auf *move experts* und *move board* gelegt.

□

Wenn ein Unentschieden bei der Abstimmung auch zur Annahme des Teilvorschlags führen soll, könnte dies einfach durch eine Änderung der Inschriften in den Transitionen *accept* und *reject* erreicht werden[15]. Durch die Aktivierung der Transition *modify* wird der Teilvorschlag der Gruppe von Experten, deren Administratortokens auf der Stelle *experts* liegen, zur Überarbeitung vorgelegt. Dazu werden all diese Administratoren benachrichtigt. Die erste Reaktion eines Administrators, die eintrifft, wird als neuer Teilvorschlag wieder zur Bestätigung vorgelegt. Dabei wird die Versionsnummer des Teilvorschlages entsprechend erhöht, alle anderen Identifizierungsinformationen, wie der Hinweis auf das Subjekt, das den Vorschlag ursprünglich gemacht hat, sowie die Nummern des Teilvorschlages bleiben erhalten. Es wird nicht festgelegt, welcher Experte bei einer erneuten Ablehnung einen neuen Teilvorschlag erstellen kann bzw. soll.

Somit wurde gezeigt, dass ausgehend von der Markierung $M^S_{start}$ stets die entsprechende Markierung $M^S_{done}$ erreicht wird.

### Zusammensetzung der Teilvorschläge

Bevor bestimmte Eigenschaften für das gesamte Admin–Netz in Abbildung 11.11 nachgewiesen werden, wird gezeigt, wie die Markierung $M^S_{finish}$ erreicht wird.

**Lemma 11.4**
Ausgehend der Markierung $M^S_{done}$ erreicht das Admin–Netz in Abbildung 11.11 stets die Markierung $M^S_{finish}$. Wobei die Anzahl der Administratortokens $l, l' \geq 1$, die Anzahl $n \geq 1$ der Teilvorschläge und die maximale Anzahl von $r_{max}$ Überarbeitungen nicht von Bedeutung sind.

**Beweis zu Lemma 11.4**
Bei der Markierung $M^S_{done}$ sind sowohl *finish experts* als auch *finish board* aktiviert und können unabhängig voneinander schalten.

Beim Schalten von *finish experts* wird ein Administratortoken von *experts* nach *experts-out* verschoben. Insgesamt kann *finish experts* $l'$ mal schalten, wobei das Standardtoken auf *move experts* stets reproduziert wird. Danach ist *experts* leer und die Transition *finish1* aktiviert. Beim Schalten wird das Standardtoken von *move experts* nach *finished* verschoben.
Unabhängig davon kann *finish board* $l$ mal schalten und verschiebt dabei ein Administratortoken von *board* nach *board-out*, wobei das Token auf *move board* reproduziert wird. Danach ist *board* leer und *finish2* aktiviert. Beim Schalten wird sowohl das Standardtoken von *move board* nach *finished* verschoben, als auch das Administratortoken von *chair1* nach *chair2*.

Erst wenn *finish1* und *finish2* unabhängig voneinander geschaltet haben, ist *create proposal* aktiviert. Gemäß Definition 11.7 wird der Administrator, dessen Token sich auf *chair2* befindet, aufgefordert die Teilvorschläge auf *accepted proposals* zu einem Vorschlag zusammenzufügen. Wenn dies geschieht, wird ein entsprechendes Proposaltoken auf *proposal-out* abgelegt sowie ein Administratortoken auf *chairmain-out* und jeweils ein Standardtoken auf *control-out* und *reset*.

□

[15] Die Inschrift von *accept* müsste dann $v = \{\oplus, \circ\}$ und die von *reject* $v = \ominus$ lauten.

Das Admin–Netz in Abbildung 11.11 soll als Subnetz verwendet werden. Dabei wird angenommen, dass *expert-in*, *proposal-in*, *control-in*, *chairman-in* und *board-in* als Eingangsstellen und *expert-out*, *proposal-out*, *control-out*, *chairman-out* und *board-out* als Ausgangsstellen verwendet werden. Bei der Anfangsmarkierung liegt ein Standardtoken auf *reset*, alle anderen inneren Stellen dieses Subnetzes sind leer.

**Satz 11.2**
Wenn das Admin–Netz aus Abbildung 11.11 wie angenommen als Subnetz verwendet wird, ist es wohlgeformt und verhält sich atomar. Ausgehend von einer Markierung $M_0^S$ wird stets die Markierung $M_{finish}^S$ erreicht.

**Beweis zu Satz 11.2**
Durch die Lemmata 11.2, 11.3 und 11.4 wird gezeigt, dass ausgehend von einer Markierung $M_0^S$ genau die Folge von Markierung $M_0^S \rightsquigarrow M_{init}^S \rightarrow M_{start}^S \rightsquigarrow M_{done}^S \rightsquigarrow M_{finish}^S$ erreicht wird.
Vor dem Schalten der Eingangstransition *init1* ist keine andere Transition des Admin–Netzes aktiviert, sodass die Markierung $M_0^S$ für alle inneren Stellen vorliegen muss.
In $M_0^S$ ist nur die Eingangstransition *init1* aktiviert und schaltet. Die Ausgangstransitionen werden erst bei Markierung $M_{done}^S$ aktiviert.
Wenn die Markierung $M_{done}^S$ erreicht wird, ist keine der Eingangstransitionen aktiviert und *init1* wird erst wieder aktiviert, wenn die Ausgangstransition *create proposal* geschaltet hat. Damit wird $M_{finish}^S$ erreicht und alle Ausgangstransitionen sind deaktiviert.
Wenn die Markierung $M_{finish}^S$ vorliegt, liegt auf *reset* ein Standardtoken und alle anderen inneren Stellen sind leer. Dies stimmt mit der Markierung der inneren Stellen bei $M_0^S$ bzw. $M_0'$ überein.

Unter der Annahme, dass $l, l', n$ sowie $r_{max}$ beschränkt sind, wird ausgehend von $M_0^S$ stets die entsprechende Markierung $M_{finish}^S$ erreicht.
Wenn nach dem Schalten der ersten Eingangstransition *init1* Token durch Transitionen von den Eingangsstellen *experts-in* und *board-in* entfernt werden, die nicht zum Subnetz gehören, verhält sich das Subnetz so, als wäre dies vor dem Schalten der ersten Eingangstransition geschehen. Insbesondere gilt dies für die variable Anzahl der Administratortokens in den Markierung $M_0^S$ und entsprechend für $M_{finish}^S$.

□

Damit spezifiziert das Admin–Netz in Abbildung 11.11 die Überarbeitung der Teilvorschläge formal. Es kann als wohlgeformtes Subnetz, das atomar schaltet, eingesetzt werden.

### 11.4.3 Übergeordnetes Netz

Das in Abbildung 11.12 dargestellte Admin–Netz modelliert den zu Beginn des Abschnittes 11.4 skizzierten Workflow und nutzt dazu die beiden zuvor vorgestellten Subnetze.

Bei der Markierung $M_0^B$ liegt jeweils ein Standardtoken *start* sowie *reset* und alle anderen Stellen sind leer. Mögliche Endmarkierungen sind $M_\oplus^B$, $M_\ominus^B$ bzw. $M_\circ^B$, bei denen jeweils ein Standardtoken auf *reset* und *end* liegt sowie ein Proposaltoken gekoppelt mit $\oplus$, $\ominus$ bzw. $\circ$ auf *final result*.

Das Admin–Netz aus Abbildung 11.10 ist zweimal als Subnetz *first quorum* und *final quorum* eingebunden. Das Admin–Netz aus Abbildung 11.11 ist als *sub-proposal revision* eingebunden. Die Zuordnung der jeweiligen Stellen ist in der Tabelle 11.2 aufgeführt.

Beim Schalten der Transition *select board* wird eine beliebige aber feste Anzahl von $l \geq 1$ Administratortokens erzeugt. Beim Schalten von *create experts* wird entsprechend eine beliebige aber feste Anzahl von $l' \geq 1$ Administratortokens erzeugt. Dagegen sollte beim Schalten von *select chairman* genau ein Administratortoken auf *chair1* gelegt werden. Die Problematik ist ähnlich wie die beim Admin–Netz in Abschnitt 10.3.1. Nachdem ein Administrator, dessen Token auf *chair1* liegt, einen Vorschlag gemacht hat, werden alle anderen Administratortokens von *chair1* entfernt. Somit kann diese Anzahl $l'' \geq 1$ beliebig aber fest sein. Welche Administratortokens durch diese

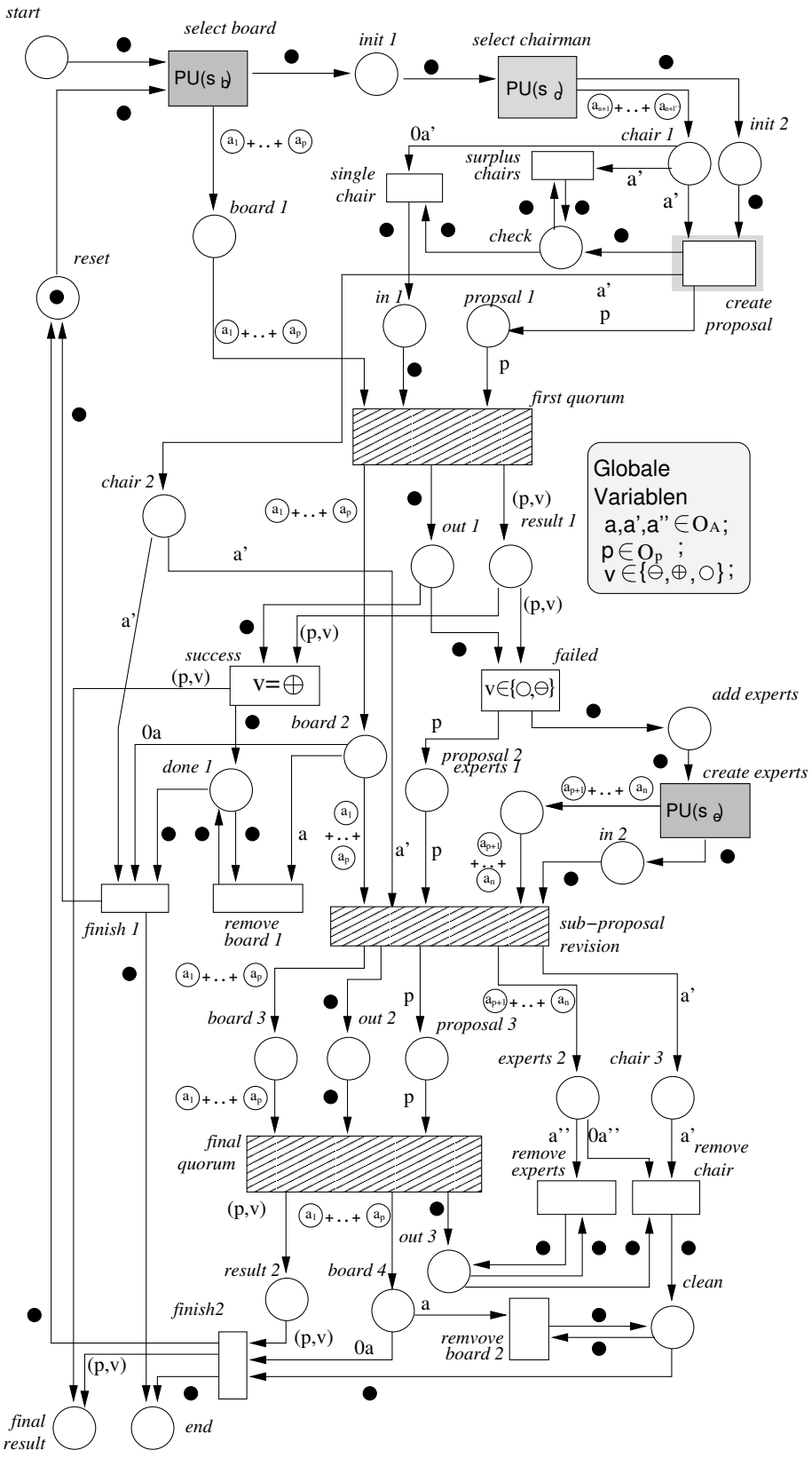



Abbildung 11.12: Bearbeitung eines Vorschlages

Transitionen erzeugt werden, spielt für die Analyse des Admin–Netzes keine Rolle. Zur eindeutigeren Darstellung wird in Abbildung 11.12 nicht nur an den Kanten der beiden Feature Transitionen *select board* und *create experts* die vereinfachte Darstellung von Mengen von Administratortokens verwendet "$a_1+\cdots+a_n$" (vgl. Definition 10.9), sondern auch an den Eingangskanten und Ausgangskanten zu den Subnetzen, da diese die gesamte Tokenmenge beim Schalten abziehen bzw. ablegen.

Tabelle 11.2: Zuordnung der Stellen

(a) first quorum

| Bearbeitung eines Vorschlages (Abb. 11.12 ) | Quorum als Subnetz (Abb. 11.10) |
|---|---|
| *board 1* | *participants-in* |
| *in 1* | *in* |
| *proposal 1* | *proposal-in* |
| *board 2* | *participants-out* |
| *results 1* | *proposal-out* |
| *out 1* | *out* |

(b) final quorum

| Bearbeitung eines Vorschlages (Abb. 11.12 ) | Quorum als Subnetz (Abb. 11.10) |
|---|---|
| *board 3* | *participants-in* |
| *out 2* | *in* |
| *proposal 3* | *proposal-in* |
| *board 4* | *participants-out* |
| *results 2* | *proposal-out* |
| *out 3* | *out* |

(c) sub-proposal revision

| Bearbeitung eines Vorschlages (Abb. 11.12 ) | Bearbeitung von Teilvorschlägen (Abb. 11.11) |
|---|---|
| *experts 1* | *expert-in* |
| *proposal 2* | *proposal-in* |
| *in 2* | *control-in* |
| *chair 2* | *chair-in* |
| *board 3* | *board-in* |
| *experts 2* | *expert-out* |
| *proposal 3* | *proposal-out* |
| *out 2* | *control-out* |
| *board 3* | *board-out* |

Bei der Markierung $M^B_{q1\oplus}$ liegen $l$ Administratortokens auf *board 2* sowie genau eines auf *chair 2* und ein Standardtoken auf *out 1*. Darüber hinaus liegt ein Proposaltoken gekoppelt mit $\oplus$ auf *result 1*. Alle anderen Stellen sind leer.
Die Markierungen $M^B_{q1\ominus}$ und $M^B_{q1\circ}$ sind mit $M^B_{q1\oplus}$ identisch, außer dass auf *result* das Proposaltoken mit $\ominus$ bzw. $\circ$ gekoppelt ist.

**Lemma 11.5**
Der Koordinierungsprozess, der durch das Admin–Netz in Abbildung 11.12 spezifiziert wird, erreicht ausgehend von einer Anfangsmarkierung $M^B_0$ stets genau eine der Markierungen $M^B_{q1\oplus}$, $M^B_{q1\ominus}$ oder $M^B_{q1\circ}$. Die Zahl der Administratortokens $l \geq 1$ und $l'' \geq 1$ kann beliebig sein.

**Beweis zu Lemma 11.5**
Bei der Markierung $M^B_0$ ist nur die Transition *select board* aktiviert. Beim Schalten werden die Standardtokens von *start* und *reset* entfernt und gemäß Annahme $l$ Administratortokens auf *board 1* abgelegt sowie ein Standardtoken auf *init*.
Nun kann *select chairman* schalten und dabei das Standardtoken auf *init 2* verschieben und $l''$ Administratortokens auf *chair 1* platzieren.
Die entsprechenden Administratoren können einen Vorschlag abgeben und beim Schalten wird ein Proposaltoken auf *proposal 1* platziert, sowie das betreffende Administratortoken auf *chair 2* verschoben. Das Standardtoken wird auf *check* verschoben.

Falls sich $l'' > 1$ Administratorentokens auf *chair 1* befinden, werden diese nun durch das $l'' - 1$ malige Schalten von *surplus chairs* entfernt. Wenn dies erfolgt ist oder $l'' = 1$ gilt, ist die Stelle *chair 1* leer und *single chair* aktiviert. Beim Schalten wird das Standardtoken auf die Stelle *in 1* gelegt.
Nun ist das Subnetz *first quorum* aufgrund der Zuordnung der Stellen in Tabelle 11.2(a) mit der Markierung $M_0^Q$ aktiviert und gemäß Satz 11.1 wird stets eine der Markierungen $M_\oplus^Q$, $M_\ominus^Q$ bzw. $M_\circ^Q$ erreicht. Aufgrund der Zuordnung der Stellen ist dann genau eine der Markierungen $M_{q1\oplus}^B$, $M_{q1\ominus}^B$ oder $M_{q1\circ}^B$ erreicht.

□

Abhängig davon, ob die Markierung $M_{q1\oplus}^B$ vorliegt, oder die Markierung $M_{q1\ominus}^B$ bzw. $M_{q1\circ}^B$, wird der Koordinierungsprozess auf unterschiedliche Arten fortgesetzt.

**Lemma 11.6**
Wenn die Markierung $M_{q1\oplus}^B$ vorliegt, erreicht das Admin–Netz in Abbildung 11.12 stets die Markierung $M_\oplus^B$, unabhängig von der Anzahl $l \geq 1$ der Administratortokens.

**Beweis zu Lemma 11.6**
Bei der Markierung $M_{q1\oplus}^B$ ist *success* als einzige Transition aktiviert. Beim Schalten wird das gekoppelte Token von *result 1* auf *final result* verschoben und das Standardtoken von *out 1* auf *done 1*.
Die Transition *remove board 1* ist nun aktiviert und kann nun $l$ mal schalten.
Danach ist die Stelle *board 2* leer und lediglich *finish 1* ist aktiviert. Beim Schalten wird das Administratortoken von *chair 2* entfernt und das Standardtoken von *done 1* entfernt und jeweils ein Token auf *end* und *reset* gelegt.
Damit ist die Markierung $M_\oplus^B$ erreicht. □

Da der erste Vorschlag des Administrators, der als "Chairman" fungiert, bereits erfolgreich war, kann der Koordinierungsprozess umgehend beendet werden. Falls dies nicht der Fall ist, wird der Vorschlag, wie im Abschnitt 11.4.2 besprochen, in Teilvorschläge zerlegt und einzeln beraten. Anschließend wird noch einmal über den neuen Gesamtvorschlag abgestimmt.

**Lemma 11.7**
Wenn die Markierung $M_{q1\ominus}^B$ oder $M_{q1\circ}^B$ vorliegt, erreicht das Admin–Netz in Abbildung 11.12 stets genau eine der drei Markierungen $M_\oplus^B$, $M_\ominus^B$ oder $M_\circ^B$ , unabhängig von der Anzahl $l \geq 1$ bzw. $l'' \geq 1$ der Administratortokens.

**Beweis zu Lemma 11.7**
Bei der Markierung $M_{q1\ominus}^B$ oder $M_{q1\circ}^B$ ist nur die Transition *failed* aktiviert. Beim Schalten wird das Proposaltoken von *result* entkoppelt und auf *proposal 2* abgelegt, sowie das Standardtoken von *out 1* auf *add experts* verschoben.
Die Transition *create experts*, die nun als Einzige schalten kann, erzeugt $l''$ Administratortokens auf *experts 1* und verschiebt das Standardtoken auf *in 2*.

Es liegt nun aufgrund der Zuordnung der Stellen in Tabelle 11.2(c) für das Subnetz *sub-proposal revision* die Markierung $M_0^S$ vor und diese ist somit aktiviert. Laut Satz 11.2 schaltet dieses Subnetz atomar und erreicht stets die Markierung $M_{finish}^S$.

Durch die daraus resultierende Belegung von *board 3, out 2* und *proposal 3* ist aufgrund der Zuordnung der Stellen in Tabelle 11.2(b) das Subnetz *final quorum* mit der Markierung $M_0^Q$ aktiviert. Dies schaltet atomar und erreicht gemäß Satz 11.1 stets genau eine der Markierungen $M_\ominus^Q$, $M_\circ^Q$ oder $M_\oplus^Q$.

Nun ist *remove experts* als einzige Transition aktiviert und kann insgesamt $l''$ mal schalten, wodurch jeweils ein Administratortoken von *experts 2* entfernt und das Standardtoken auf *out 3* reproduziert wird. Anschließend schaltet *remove chair* genau einmal. Danach ist *remove board 2* aktiviert und kann $l$ mal schalten.

Als letzte Transition ist nun *finish 2* aktiviert. Beim Schalten wird das Standardtoken von *clean* entfernt und jeweils ein Standardtoken auf *end* und *reset* platziert. Außerdem wird das gekoppelte Token von *result 2* nach *final result* verschoben.
Abhängig von diesem gekoppelten Token ist damit genau eine der drei Markierungen $M^B_{\oplus}$. $M^B_{\ominus}$ oder $M^B_{\circ}$ erreicht.

□

Das Admin–Netz in Abbildung 11.12 kann als Subnetz verwendet werden. Dabei wird angenommen, dass *start* als Eingangsstelle und als Ausgangsstellen *final result* sowie *end* genutzt werden.

**Satz 11.3**
Wenn das Admin–Netz aus Abbildung 11.12, wie angenommen, als Subnetz verwendet wird, ist es wohlgeformt und verhält sich atomar. Ausgehend von einer Markierung $M^B_0$ wird stets genau eine der drei Markierungen $M^B_{\oplus}$, $M^B_{\ominus}$ oder $M^B_{\circ}$ erreicht.

**Beweis zu Satz 11.3**
Vor dem Schalten der Eingangstransition *select board* ist keine Transition im Admin–Netz aktiviert.
Bei der Anfangsmarkierung $M^B_0$ schaltet als erstes die Eingangstransition genau einmal. Danach schaltet entweder die Ausgangstransition *finish 1*, wenn zuvor die Markierung $M^B_{q1\oplus}$ erreicht wird, oder die Ausgangstransition *finish 2* schaltet, wenn zuvor die Markierung $M^B_{q1\ominus}$ bzw. $M^B_{q1\circ}$ erreicht wurde.
Aufgrund der Eingangsstelle *reset* wird die Eingangstransition erst nach dem Schalten der Ausgangstransitionen *finish 1* oder *finish 2* wieder aktiviert.
Die Belegung der inneren Stellen ist bei der Anfangsmarkierung $M^B_0$ und den Endmarkierungen $M^B_{\oplus}$, $M^B_{\ominus}$ oder $M^B_{\circ}$ identisch.

Nach den Lemmata 11.5, 11.6 und 11.7 wird ausgehend von der Anfangsmarkierung $M^B_0$ stets genau eine der drei Markierungen $M^B_{\oplus}$, $M^B_{\ominus}$ oder $M^B_{\circ}$ erreicht.

□

An diesem komplexen Beispiel für einen administrativen Workflow wird deutlich, welche erweiterten Modellierungsmöglichkeiten durch Admin–Netz gegenüber anderen Ansätzen wie "Workflow Net" (vgl. [Aal98, ABtHK00]) gegeben sind. Dazu gehören neben der Darstellung der Beteiligten und der verschiedenen Vorschläge vor allem auch die Repräsentation der Ergebnisse. Je nach Ergebnis können dann auch alternative Vorgehensweisen formal spezifiziert werden. Weitere Inhalte der Regelsätze, wie zum Beispiel die betroffenen Rollen oder Organisationseinheiten etc., werden bisher nicht in den Admin–Netzen direkt durch Tokens modelliert. Mit dieser Arbeit sollen zunächst die grundlegenden Möglichkeiten der dezentralen Administrierung und der dazu nötigen Prozesse erarbeitet werden. Welche Möglichkeiten zur Darstellung von konkreten Inhalten noch gegeben sind, soll im folgenden Beispiel erläutert werden.

## 11.5 Automatisierte Verhandlung

Im Kapitel 4 sind Kontext Feature Prädikate eingeführt worden, die es unter anderem ermöglichen Zugriffsrechte abhängig von den gerade verwendeten Kommunikationskanälen zu gewähren. Die Sicherheit und damit die Vertrauenswürdigkeit dieser Kommunikationskanäle hängt von den verwendeten kryptographischen Verfahren ab, wobei eine mögliche Auswahl in Abbildung 4.5 dargestellt ist. Dabei können bei den beiden Hosts, die einen Kommunikationskanal aufbauen wollen, unterschiedlich Präferenzen für die entsprechenden kryptographischen Verfahren konfiguriert sein, insbesondere wenn sie zu unterschiedlichen Autorisierungssphären gehören. Beim Aufbau eines Kommunikationskanals müssen diese Hosts erst einmal eine Einigung darüber erzielen, welche kryptographischen Verfahren verwendet werden sollen. Um diese Einigung zu erreichen, kann durch Admin–Netze die automatisierte Verhandlung zwischen den beiden Hosts modelliert werden. Eine

mögliche Vorgehensweise ist der Austausch von verschiedenen Konfigurationen zwischen den beteiligten Hosts, bis eine gefunden wurde, die von beiden Seiten akzeptiert wird. Dabei ermöglichen die Admin–Netze, unabhängig von der existierenden Verhandlungsstrategie, die Modellierung eigener Vorgehensstrategien.

| | | | |
|---|---|---|---|
| EG1536 | ElGamal (keylength 1536bits) | DH | DiffieHellman |
| EG2048 | ElGamal (keylength 2048bits) | EKE | Encrypted Key Exchange |
| RSA1536 | RSA (keylengtth 1536bits) | IDEA | IDEA |
| RSA2048 | RSA (keylength 2048bits) | BF | Blowfish |

Tabelle 11.3: Tokens zur Repräsentierung kryptographischer Algorithmen und Verfahren

Um diese Verhandlungen zu modellieren, werden die in Kapitel 10 vorgestellten Admin–Netze[16] lediglich um eine Menge zusätzlicher Tokens erweitert, die die verschiedenen kryptographischen Verfahren und gegebenenfalls die benutzte Schlüssellänge repräsentieren. Eine mögliche Menge dieser Token ist in Tabelle 11.3 aufgeführt. Diese kann bei Bedarf entsprechend erweitert werden, jedoch muss bei den Verhandlungen die Semantik der einzelnen Token unter den beteiligten Partnern, vorzugsweise innerhalb einer Organisation, eindeutig sein.

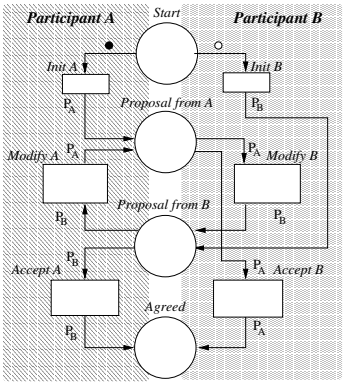



Abbildung 11.13: Ablauf für automatische Verhandlungen

Der grundlegende Verlauf einer solchen automatisierten Verhandlung durch ein Admin–Netz ist in Abbildung 11.13 skizziert. Dabei legt jede Seite durch die Transitionen die eigene Vorgehensweise fest. Die Stellen dienen zur Steuerung (*Start)*, dem Austausch der vorgeschlagenen Konfiguration (*Proposal from A, Proposal from B*) und zur Übermittlung der akzeptierten Konfiguration. Durch die Transitionen werden von jeder Seite die Vorgehensweisen spezifiziert. Dazu gehört im Einzelnen: Welche Konfiguration zu Beginn vorgeschlagen wird (*Init*), welche Konfiguration als Alternative für eine vorgeschlagene Konfiguration abgegeben wird (*Modify*), die Menge der Konfigurationen, die akzeptiert werden (*Accept*). Dabei wird davon ausgegangen, dass jede Partei die eigenen Vorschläge auch akzeptiert. Die Kantenbeschriftungen im Admin–Netz in Abbildung 11.13 sollen andeuten, wo ein neue Konfiguration vorgeschlagen wird.[17]. Bei der Anfangsmarkierung $M_0$ sind alle Stellen, außer *Init* leer. Abhängig davon, welche Belegung *Init* hat, beginnt dann bei •

[16]Die in den Abschnitten 11.2 und 11.3 vorgestellten Erweiterungen für Proposaltokens werden nicht benötigt, lediglich die in Abschnitt 11.1 vorgestellten Subnetze, könnten bei dieser Art der Verwendung von Admin–Netzen von Bedeutung sein.

[17]Als Variablen würden sie so nicht mit der Definition 10.4 übereinstimmen. Nach dieser Definition müssen Variablen, die als Kantenbeschriftung zu Ausgangsstellen verwendet werden auch auf den Kantenbeschriftungen von den Eingangsstellen auftauchen.

die Partei die als “Participant A” bezeichnet wird oder bei ∘ die Partei “Participant B”. Diese Belegung ist davon abhängig, welche Seite den Kommunikationskanal aufbauen will.

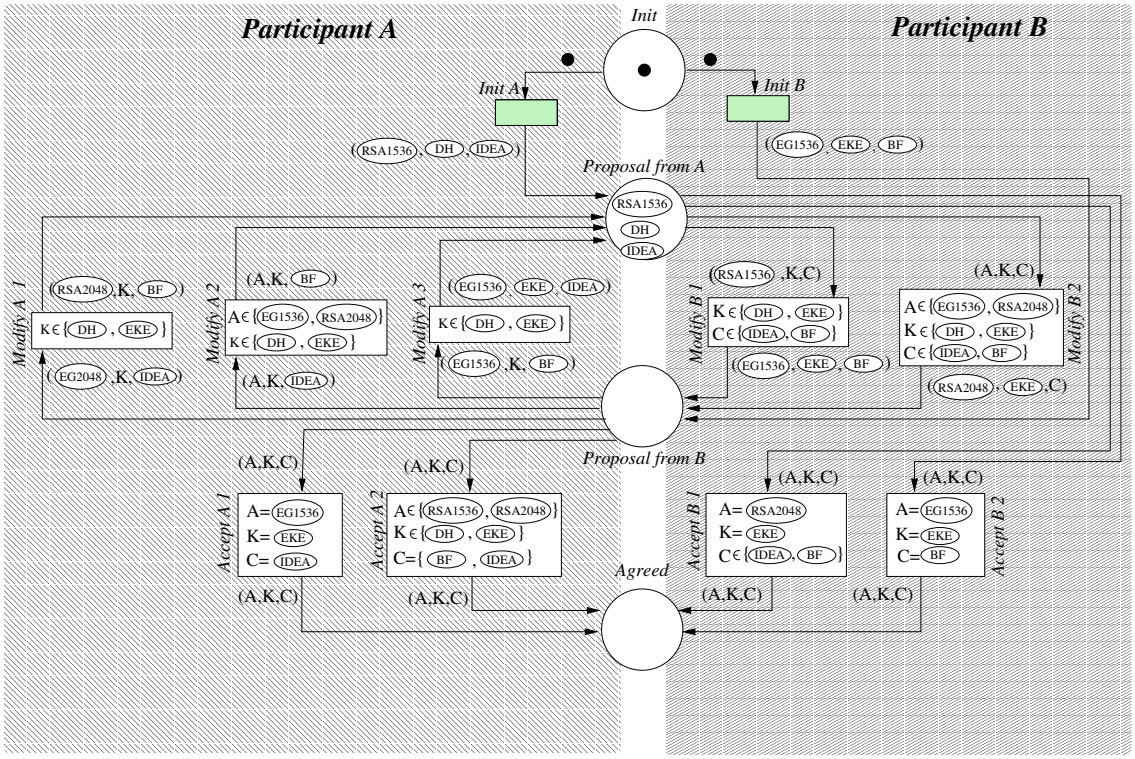



Abbildung 11.14: Beispiel einer automatisierten Verhandlung

Ein Beispiel, wie eine solche Verhandlung mit einem Admin–Netz konkret realisiert werden kann, ist in Abbildung 11.14 zu sehen. Hierbei ist zu beachten, dass die Transitionen der jeweils anderen Seite nicht bekannt sein müssen. Die Konfiguration, die ausgehandelt werden soll, besteht aus der Authentifizierungsmethode $A$, dem Protokoll für den Schlüsselaustausch (key exchange) $K$, sowie der Verschlüsselungsmethode (cyphermethod) $C$ und wird immer als Tripel $(A, K, C)$ ausgetauscht.

Die oberen Transitionen *Init A* und *Init B* legen fest, welche Konfiguration zu Beginn vorgeschlagen wird. Die folgenden mittleren Transitionen bestimmen, welche Konfiguration abgeändert werden soll, und wie diese Änderung aussieht. Wenn die Seite “Participant A” beginnt und entsprechend das Tripel $(RSA1536, DH, IDEA)$ auf der Stelle *Proposal from A* liegt, so ist die Transition *Modify B 1* aktiviert. Beim Schalten wird nun ein Tripel erzeugt, das die Konfiguration ElGamal mit einer Schlüssellänge von 1536 als Authentifizierungmethode, Encrypted Key Exchange zum Schlüsselaustausch und Blowfish als Verschlüsselungsverfahren vorschlägt.

Da die untere Transition *Accept*, die gleichen Variablen auf den Eingangskanten und den Ausgangskanten aufweisen, wird hier die Konfiguration nicht mehr abgeändert. Sobald eine Belegung der Variablen durch die Bedingungen der Transitionsinschriften erfüllt ist, können diese Transitionen schalten. Damit wird das Tripel auf die Stelle *Agreed* gelegt und die Verhandlungen sind beendet.

## 11.6 Möglichkeiten der Analyse von Admin–Netzen

Die bisherigen Analysen und Korrektheitsbeweise zu den Admin–Netzen in den Abschnitten 10.3 und 11.4 wurden “per Hand” entwickelt, wodurch neben einem relativ hohen Aufwand auch immer

die Gefahr gegeben ist, dass Einzelheiten übersehen werden. Daher ist es wichtig nach Möglichkeiten zu suchen, wie bestimmte Analysen der Admin–Netze, insbesondere die Erreichbarkeit von bestimmten Endmarkierungen ausgehend von einer vorgegebenen Anfangsmarkierung, automatisiert erfolgen können.

Die Berechnung von Erreichbarkeitsgraphen (vgl. [Sta90, Kap. 8]) ist eine Möglichkeit, allerdings sind diese Graphen nicht immer endlich. Alternativ könnten Stubborn Sets (vgl. [Val94]) verwendet werden, deren Berechnung jedoch ähnlich aufwendig ist. Bei der Analyse von High–Level Netzen mittels Überdeckungsgraphen [HJJJ85, Jen87] kommt es bei unbeschränkten Netzen zu überexponentiellen Laufzeiten (vgl. [Sta90]). Somit ließe sich ein Admin–Netz, wie in Abbildung 11.6 dargestellt, mit dieser Methode nicht analysieren[18].

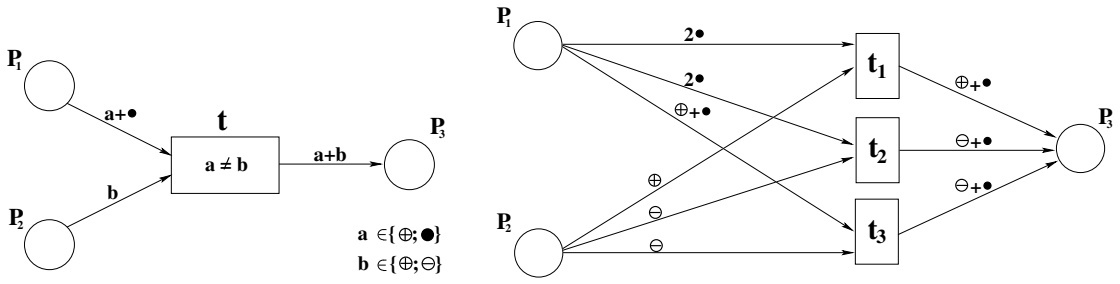
Die Invarianten Berechnung für PrT–Netze [GL81, GL83, Gen87] und Coloured Petri–Netzen [Jen81, Jen87] erfordert noch ein hohes Maß an netzspezifischem Wissen bzw. sind nicht allgemein übertragbar (vgl. [Gen87],[HG86]). Eine andere Möglichkeit ist, wie in [HG86] vorgestellt, die Einschränkung der High–Level Petri–Netze. Diese Einschränkungen sind jedoch für die Modellierung von Administrationsprozessen nicht vertretbar.
Auch in neueren Artikeln zu anderen Analysetools [LMP02] wird als Algorithmus zur Berechnung von Invarianten nur auf eine Arbeit [MV86] verwiesen, die unäre PrT–Netze berücksichtigt. Alternativ wird nur noch darauf hingewiesen, dass eine Invariante in einem Blacknetz Voraussetzung für eine Invariante im ursprünglichen Netz ist. Da bei diesen Netzen ein Großteil der Informationen über den Prozess verloren geht, ist es mit den dort gefundenen Invarianten jedoch nahezu unmöglich die tatsächlichen Schaltfolgen zu ermitteln.

Anstelle einer direkten Untersuchung der Admin–Netze wurde daher ein Analysetool entwickelt [Den05], das mit einem kombinierten Verfahren aus T–Invariantenanalyse und Erreichbarkeitsanalyse arbeitet. Die grundlegende Idee eines solchen kombinierten Verfahrens wurde in [KT98] allerdings "nur" für S/T Netze vorgestellt.
Da kein Verfahren zur automatischen Berechnung von Invarianten auf PrT–Netze und damit auch auf Admin–Netze existiert, wird das Admin–Netz in ein entfaltetes und schlingenfreies[19] S/T Netz überführt. Hierbei bleiben allerdings wichtige Informationen über die Tokens und Bindungen beim Schalten erhalten bzw. können später ohne weiteres rekonstruiert werden. In Abbildung 11.15 ist ein Beispiel zu sehen, wie ein Admin–Netz mit Variablen als Kantenbeschriftung und Transitionsinschriften (Abb. 11.15(a) in ein entsprechendes S/T Netz (Abb. 11.15(b)) überführt wird. Dabei werden nur die Bindungen, die aufgrund der Transitionsinschriften tatsächlich möglich sind, berücksichtigt. Zusätzlich schränken die Definitionsbereiche der Variablen und die Eingangskanten der Stellen die möglichen Belegungen weiter ein.
Bei der Entfaltung und Berechnung der Invarianten bleiben die Inhibitorinschriften (vgl. Def. 10.4) ebenso wie die Koppelung von Tokens zunächst unberücksichtigt. Die Interaktiven und Timed Transitionen (vgl. Def. 10.10 und 10.8) werden als normale Transitionen aufgefasst, die beliebig schalten können. Bei Feature Transitionen (Def. 10.9) sowie Split und Join Transitionen (Def. 11.6 und 11.7) kann vom Benutzer des Tools die Zahl der erzeugten Administrator- bzw. Proposaltokens festgelegt werden.
Auf Grundlage der berechneten T–Invarianten wird dann per Erreichbarkeitsanalyse geprüft, ob es eine Schaltfolge von der Anfangsmarkierung zur angegebenen Endmarkierung gibt. Dabei werden die Inhibitorbeschriftungen und Tokenkoppelungen berücksichtigt.

Im Detail werden bei einem entsprechenden Aufruf vom Analysetool die folgenden Schritte durchgeführt, wobei der Benutzer neben der Anfangs- und Endmarkierung auch über Parameter vor allem die Heuristik bei der Berechnung der Invarianten beeinflussen kann.

**Einfügen einer Zusatztransition:** Diese führt beim Schalten die Endmarkierung wieder in die Anfangsmarkierung über.

[18] Allerdings muss auch für das hier vorgestellte Admin–Tool [Den05] die Zahl der möglichen wechselseitigen Vorschläge eingeschränkt werden, um in vertretbarer Zeit zu einem Ergebnis zu gelangen.
[19] Schlingen werden durch das Einfügen einer zusätzlichen Stelle und Transition aufgelöst.




(a) Admin–Netz (b) entfaltetes Netz

Abbildung 11.15: Entfaltung von Admin–Netzen nach [Den05]

**Entfaltung des Admin–Netzes:** Wobei die Transitionsinschriften und Kantenbeschriftungen außer Inhibitor berücksichtigt werden.

**Berechnung einer minimalen semipositiven T-Invariante:** Dabei wird das Verfahren von [MS82] angewendet, das durch verschiedene Heuristiken zur Minimierung des Berechnungsaufwandes ergänzt wird.

**Prüfe ausgehend von der Anfangsmarkierung der Erreichbarkeit Endmarkierung:** Dabei werden nur die T-Invarianten berücksichtigt, bei denen die Zusatztransition einmalig schaltet. Hierzu wird eine erweiterte Version des Algorithmus von [KT98, Kos03] verwendet, der alle Erweiterungen der Admin–Netzes beim Schalten von Transitionen berücksichtigt. Sobald eine Schaltfolge gefunden ist, wird diese ausgegeben.

**Prüfen der ganzzahligen Linearkombinationen der Invarianten:** Auf allen ganzzahligen Linearkombinationen der T-Invarianten, bei denen die Zusatztransition genau einmal schaltet, wird nach einer gültigen Schaltfolge gesucht. Sobald eine Schaltfolge gefunden wird, wird diese ausgegeben. Wenn keine Schaltfolge gefunden ist, bricht der Algorithmus ohne Ergebnis ab.

**Berechnung der Subnetze:** Falls in der Schaltfolge Subnetze enthalten sind, werden diese rekursiv mit den entsprechenden Anfangs- und Endmarkierungen berechnet.

Die Anzahl der in einem Admin–Netz verwendeten Proposal– und Administratortokens kann eine große Auswirkung auf die Performance des Analysetools haben. Bei der Verwendung einer entsprechenden Variable $p \in O_p$ oder $a \in O_A$ entstehen für jedes Element weitere Möglichkeiten für die Bindungen und damit zusätzliche Transitionen im entfalteten Netz. Diese müssen dann auch bei der Berechnung der Invarianten berücksichtigt werden. Daher kann die Anzahl der Administratortokens, die bei einer Feature Transition erzeugt werden, gezielt festgelegt werden. Diese Anzahl kann auch bei der Einbindung von Subnetzen berücksichtigt werden (vgl. Def. 11.4). Außerdem kann die Anzahl der verwendeten Proposaltokens eingeschränkt und die Zahl der Teilvorschläge, in die ein Vorschlag aufgeteilt werden kann, festgelegt werden. Hierdurch wird besonders die Zahl der möglichen Bindungen reduziert, was zu kleineren Matrizen bei der Entfaltung führt.
Die obigen Einschränkungen sollen dazu dienen, schneller einfache Fehler im Admin–Netz zu entdecken. Bei den anschließenden Untersuchungen können diese Einschränkungen in der Anzahl der verwendeten Tokens immer weiter erhöht werden.

Bei der Berechnung der T-Invarianten können als Optimierung zum einen minimale Invarianten mit gleichem Träger/Support frühzeitig entfernt werden (vgl. [Sta90, S. 119ff]). Dabei wird für durch Berechnung des größten gemeinsamen Teilers (ggT) überprüft, ob Zeilen gekürzt und durch andere ersetzt werden können. Zum Anderen kann als weitere Optimierungsmaßnahme die Analyse auf solche Invarianten beschränkt werden, bei denen die eingefügte Zusatztransition (für den Übergang von Anfangs- zur Endmarkierung) genau einmal schaltet. Bei den meisten Linearkombinationen muss dieser Wert lediglich erhöht werden. Durch Wechselwirkung mit der vor-

angegangenen Optimierungsmaßnahme kann es jedoch zu Fehlern kommen, sodass keine Lösung gefunden wird. Daher lässt sich diese Optimierungsmaßnahme auch abschalten.
Darüber hinaus können solche Restriktionen für die maximale Anzahl von Schaltungen auch bei allen anderen Transitionen angegeben werden.

Mit diesem Analysetool wurden exemplarisch das Administrationsnetz aus Abschnitt 10.3.2 sowie die Administrationsnetze aus Abschnitt 11.4 überprüft. Bereits bei der Analyse des Admin–Netzes "multi approve" aus Abbildung 10.8 zeigt sich, dass abhängig von der Größe der Variablenmenge, die Zahl der berechneten Invarianten überproportional steigen kann. In dem entsprechenden Admin–Netz ist dies die Anzahl der Administratortokens, die beim Schalten von *init* erzeugt werden. Da die Invarianten alle möglichen Schalthäufigkeiten beschreiben, liegt die Zahl bei drei Administratortokens schon bei 28. Die Zahl der berechneten Invarianten steigt bei vier Administratortokens auf 60. Trotzdem werden mögliche Schaltfolgen schnell gefunden. Allerdings lässt sich hieraus die Empfehlung ableiten, zunächst mit einer kleinen Tokenmenge zu beginnen, um einfache Fehler im Admin–Netz schnell zu finden. Für die angegebenen Anfangs- und Endmarkierungen werden durch das Analysetool entsprechende Schaltfolgen gefunden.

Bei der ersten Untersuchung des Admin–Netzes aus Abbildung 11.12 wird die Version aus [WL04a, Abb.15] verwendet, bei der nur die Eingangskante der Stelle *board 1* mit $\textcircled{a_1} + \ldots + \textcircled{a_p}$ gekennzeichnet und die Ausgangskante mit $u$ bzw.[20] $a$ beschriftet ist. Dadurch geht das Analystool davon aus, dass das Subnetz *first quorum* beim Schalten nur ein beliebiges Token von dieser Stelle entfernt und die anderen zurückbleiben. Dementsprechend kann keine gültige Schaltfolge zu der Endmarkierung gefunden werden, bei denen nur auf den Stellen *final result, end* sowie *reset* Tokens liegen dürfen.
Die Problematik ergibt sich aus der dynamischen Anzahl von Administratortokens, die durch Feature Prädikate erzeugt werden können. Die angegebenen Subnetze können damit entsprechend umgehen, jedoch werden die internen Details (wie beabsichtigt) in diesem Admin–Netz verborgen. Dadurch geht das Analysetool bei einer Kantenbeschriftung mit der Variable $a$ von einem Administratortoken aus, das beim Schalten des Subnetzes *first quorum* von *board1* entfernt wird und auf *board 2* erzeugt wird. Entsprechende Annahmen wurden auch bei den Subnetzen *sub-proposal revision* und *final quorum* vom Analysetool getroffen. Aus diesem Grund wird die entsprechenden Eingangs- und Ausgangskanten aller Subnetze mit der Beschriftung $\textcircled{a_1}+\ldots+\textcircled{a_p}$ gekennzeichnet, die das Schalten mit einer beliebigen aber festen Menge von Tokens anzeigt (vgl. Def. 11.4). Dies muss auch bei der Einbindung des Subnetzes *polling* im Admin–Netz zur Bearbeitung der Teilvorschläge in Abbildung 11.11 geschehen, obwohl hierdurch dem Analysetool keine Fehler angezeigt würden, da die Administratortokens nach der positiven Abstimmung über den letzten Teilvorschlag alle entfernt werden. Die Tatsache, dass das Analysetool davon ausgeht, dass beim Schalten des Subnetzes nicht alle Administratortokens von *board* entfernt werden, wird somit nicht bemerkt. Es wäre zu überlegen, ob generell auf solche Schleifen bei Subnetzen verzichtet werden sollte.

Insgesamt wird mit der Realisierung des Analysetools in [Den05] jedoch gezeigt, dass eine automatisierte Analyse von Admin–Netzen möglich ist. Die Nutzung eines kombinierten Verfahrens aus T–Invarianten und Erreichbarkeitsanalyse ermöglicht zum einen das schnelle Auffinden von Flüchtigkeitsfehlern, die sich bei der Eingabe der Admin–Netze ergeben und später auch bei der Modellierung von neuen Administrationsprozessen auftreten können. Zum anderen können durch die Festlegung von Schalthäufigkeiten auch Analysen bei unterschiedlichem Verhalten der Administratoren durchgeführt werden.

[20] In [WL04a] werden die Administratortokens noch als Usertokens bezeichnet.

# Kapitel 12

# Fazit und Ausblick

Mit dieser Arbeit wird ein dezentraler Ansatz zur Spezifikation einer rollenbasierten Autorisierung unter der Berücksichtigung von hybriden Zugriffsprivilegien präsentiert. Einzelne Aspekte der Arbeit sind bereits in [WL01, WL03b, WL04a, WL04b] veröffentlicht. Kernpunkt des Konzeptes zur Autorisierung ist die Einführung sogenannter **Autorisierungssphären**, die für die Spezifizierung der Zugriffsstrategien in einem bestimmten Teil einer Organisation verantwortlich sind. Dies ermöglicht eine völlig dezentrale Spezifizierung der Zugriffsstrategien, bei denen sich die verantwortlichen Autorisierungsteams nur auf die Zugriffsprivilegien konzentrieren können, die für die Arbeitsabläufe in den zugeordneten Organisationseinheiten und/oder zum Schutz der Objekte notwendig sind. Der Umfang dieser dezentral spezifizierten Regelsätze ist damit wesentlich kleiner als ein zentraler Regelsatz, der die Autorisierung für die gesamte Organisation enthält. Somit wird nicht nur die Auswertung der entsprechenden Zugriffsanfragen schneller sein. Ein wesentlicher Vorteil ist, dass diese Zugriffsstrategien damit auch übersichtlicher und besser durch die Mitglieder des Autorisierungsteams verifiziert werden können.

Eine dezentrale Spezifizierung, die auch **hybride Zugriffsprivilegien** berücksichtigt, wurde bisher noch nicht vorgestellt. Andere Ansätze [BVS00, BVS02] berücksichtigen nur geschlossene Systeme. Die hier vorgestellten Zugriffsstrategien können im Gegensatz zum Standard RBAC [FSG+01] auch die Bedingungen einer verteilten Umgebung bei der Autorisierung berücksichtigen. Darüber hinaus wird dargestellt, wie die Besonderheiten beim Zugriff auf ein Dateisystem besser in den Zugriffsregeln modelliert werden können.

Mit der Definition einer Hierarchie über die Autorisierungssphären wird die Möglichkeit einer Vererbung von Zugriffsstrategien entlang der Organisationsstruktur geschaffen. Hierdurch kann die von [PDF99, S.223] befürchtete Auflösung einer Organisation bei einer völligen Dezentralisierung verhindert werden. Über diese Hierarchie der Autorisierungssphären ist es möglich, bestimmte Vorgaben durchzusetzen und allgemeine Vorgehensweisen vorzugeben.

Die Vererbung von Zugriffsstrategien wird erstmalig in [WL01] vorgestellt. In der vorliegenden Arbeit wird das **Vererbungsprinzip** konkretisiert und demonstriert, wie dieses mittels entsprechender Datalog–Regeln umgesetzt werden kann. Dabei wird sichergestellt, dass das Vererbungsprinzip nicht von den Autorisierungsteams unterlaufen werden kann. Die Korrektheit der Umsetzung durch die vorgegebenen Regelsätze ist dabei nachgewiesen worden.

Für das auftretende Problem der **Vererbungskonflikte** bei einer Mehrlinien Organisation werden verschiedene Lösungsstrategien präsentiert. Dazu können insbesondere bei der Veränderung des Scopes, die umfangreichen Feature Prädikate, die in Kapitel 3 vorgestellt werden, herangezogen werden, um die Zugriffsregeln zu präzisieren.

Da in modernen Organisationen ein Zugriff auf Daten einer anderen Abteilung unerlässlich ist, wird dieser erstmalig in der vorliegenden Arbeit auch für eine dezentrale Autorisierung vorgestellt. Dabei müssen sowohl die Zugriffsstrategien beachtet werden, die den Arbeitsablauf für die Rolle festlegen, als auch die Zugriffsstrategien, die zur Sicherung der Daten spezifiziert werden. Dies

hat zur Konsequenz, dass zwei Autorisierungssphären bei einer entsprechenden Zugriffsanfrage involviert sind. Wenn diese nicht zum selben Ergebnis kommen, liegt ein Koordinierungskonflikt vor. Diese Koordinierungskonflikte werden nicht durch eine einzige übergeordnete Stelle aufgelöst. Statt dessen wird abhängig von der jeweiligen Autorisierungssphäre der Rolle und der Autorisierungssphäre des Objektes die sogenannte koordinierende Autorisierungssphäre ermittelt. Diese ist dafür verantwortlich, mittels Entscheidungsregeln den Koordinierungskonflikt aufzulösen. Hierzu werden verschiedene Lösungsschemata präsentiert, die auch jeweils abhängig von bestimmten Rollen, Objekten etc. oder Bedingungen der verteilten Umgebung mittels Feature Prädikaten sowie Kontext Feature Prädikaten und Kontext Trust Prädikaten spezifiziert werden können.

Genauso wie die Umsetzung des Vererbungsprinzips wird auch die Bestimmung der verantwortlichen Autorisierungssphäre und damit die gesamte Auswertung der Zugriffsanfrage durch Datalog–Regeln spezifiziert. Durch die Beweise im Abschnitt 6.4.3 für die Vererbung sowie im Abschnitt 7.2 für die Koordinierung wird die korrekte Umsetzung durch die entsprechenden Datalog–Regeln nachgewiesen.

Die Auswertung dieser Zugriffsanfrage auch zwischen zwei Autorisierungssphären erfolgt völlig dezentral. Damit hat der gesamte Ansatz das Potenzial, auch in großen Organisationen effizient eingesetzt zu werden. Die bisherigen Zugriffsstrategien beziehen sich auf eine kleine Beispielorganisation. Die Grenzen der Modellierungsmöglichkeiten können nur aufgedeckt werden, wenn die Abläufe einer existierenden Organisation detailliert nachgebildet werden. Dies kann nur in enger Kooperation geschehen, sollte aber bei entsprechender Gelegenheit eine reizvolle Aufgabe in der Zukunft sein.

Diese Zugriffsstrategien sowie die Spezifizierungen zur Vererbung und Koordinierung werden von den gemeinschaftlich verantwortlichen Autorisierungsteams vorgenommen. Die detaillierten Möglichkeiten der Administrierung können ebenfalls durch entsprechende Regeln definiert werden. Bei der Spezifizierung der Administrationsprädikate sind beispielhaft diejenigen definiert, die zur Verwaltung der Rollen und der entsprechenden Hierarchie $<_{\mathcal{R}}$ benötigt werden. Da die Administrationsprädikate für die anderen Elemente, die in Tabelle 3.1 und 4.1 auf den Seiten 42 und 61 aufgelistet sind, in gleicher Weise administriert werden können, wird aus Platzgründen auf eine explizite Definition in dieser Arbeit verzichtet.

Eine Besonderheit dieses Ansatzes ist es, dass nicht ein einzelner Benutzer für die Administrierung verantwortlich, sondern gemeinschaftlich ein Autorisierungsteam. Dabei werden verschiedene Möglichkeiten der Partizipierung vorgestellt. Weitere lassen sich bei Bedarf relativ einfach modellieren. Hierzu müssen zum einen weitere Partizipierungsprädikate definiert werden. Wichtig ist jedoch zum anderen die Modellierung der Abläufe. Diese Details der Administrierungsprozesse können mit Hilfe einer neu eingeführten Klasse von Petri–Netzen detailliert spezifiziert werden. Die korrekte Umsetzung der bisherigen Administrierungsprozesse durch diese Admin–Netze ist bewiesen. Die Korrektheit kann auch mittels eines eigens entwickelten Analysetools automatisiert[1] nachgewiesen werden. Dabei kommt ein kombiniertes Verfahren aus Berechnung der T–Invarianten und Erreichbarkeitsgraphen zum Einsatz. Dieses Analyseverfahren lässt sich in zukünftigen Arbeiten auch auf andere PrT–Netze übertragen.

Zur Einbindung des Autorisierungssystems in ein beliebiges Dateisystem muss der existierende Interpreter an dieses angepasst werden, um eine effiziente Auswertung der Feature und Kontext Feature Prädikate zu ermöglichen. Darüber hinaus muss die Auswertung der Prädikate zur Prüfung der Konsistenz gewährleistet sein. Die Regelsätze müssen mit geeigneten Mechanismen an die für die Auswertung zuständigen Hosts repliziert werden.
Für die Durchführung der Administrierungsprozesse wird ein Tool benötigt, dass auf Grundlage der angegebenen Admin–Netze die Entscheidungsprozesse durchführt. Hierzu zählt insbesondere die Ermittlung der beteiligten Administratoren und deren Benachrichtigung sowie die Entgegen-

[1] Eine Spezifizierung der erlaubten und verboten Folgemarkierung wird dabei immer noch “von Hand” nötig sein.

nahme der entsprechenden Reaktionen. Für die Besonderheiten der anderen Transitionen müssen von diesem Tool ebenfalls entsprechende Interfaces für die beteiligten Administratoren bereitgestellt werden.

Neben dieser Einbindung in ein reales System und den damit verbundenen Erfahrungen in der Modellierung der Zugriffsstrategien gibt es weitere Möglichkeiten die Konzepte aus dieser Arbeit an mehreren Stellen fortzuführen. Dazu gehört zunächst die in [WL03a] vorgeschlagenen Zugriffe zwischen verschiedenen Organisationen auf die aktuellen Gegebenheiten anzupassen. Wobei auch untersucht werden muss, welche Informationen zwischen den Organisationen ausgetauscht werden.

Im vorgestellten Ansatz kann ein Subjekt stets nur eine Rolle aktivieren, wobei ein Wechsel bei jeder Zugriffsanfrage möglich ist. Diese Vorgehensweise sollte keine Einschränkung darstellen. Bei einer Erweiterung zu mehreren aktiven Rollen können theoretisch auf Grund der hybriden Zugriffsstrategien weitere Konflikte auftreten, indem der Zugriff für eine aktive Rolle erlaubt ist, während er für eine andere aktive Rolle verboten ist. Dies kann vom Benutzer entweder durch eine Deaktivierung der entsprechenden Rolle gelöst werden. Alternativ kann hierzu auch ein ähnlich Verfahren wie bei der Auflösung der Koordinierungskonflikte verwendet werden. Unter Umständen muss auch eine Möglichkeit geschaffen werden, die Zugriffsregeln gezielt mit einer bestimmten aktiven Rolle auszuwerten.

Bei der Administrierung der Zugriffsregeln wird bisher der Bezugsrahmen, der durch die Feature Prädikate spezifiziert wird, nicht mitberücksichtigt. Das bedeutet, es wird nicht überprüft, für welche Subjekte in welchen aktiven Rollen und für welche Objekte, die einen bestimmten Objekttyp oder anderen Eigenschaften aufweisen, die neuen Regeln modelliert werden. Entsprechende Einschränkungen können mittels Feature Prädikaten problemlos modelliert werden. Jedoch liegt die Schwierigkeit darin, bei einer neuen Zugriffsregel zu prüfen, ob der darin durch Feature Prädikate definierten Scope, mit dem durch die Feature Prädikate der Administrierungsregeln definierte Scope übereinstimmt oder zumindest in diesem vollständig enthalten ist.
Um dies zu prüfen, müssten alle möglichen Belegungen der Variablen für die Zugriffsregel berechnet werden bzw. alternative Methoden für diese Überprüfung entwickelt werden.

Aufbauend auf der bisherigen Administrierungssprache, mit deren Hilfe sowohl die Zugriffsrechte, als auch die verschiedenen Mengen wie Rollen und Organisationseinheiten, sowie die Relationen in diesen Mengen verwaltet werden können, müsste hierzu ein Konzept für Administrierung der Administrierungssprache entwickelt werden. Dies würde bedeuten, dass modelliert werden kann, wie das Recht zur Administrierung innerhalb einer Organisation delegiert werden kann. Bei der Definition dieser Prädikate sollen auch die Inhalte der neuen Regeln berücksichtigt werden, wie es im vorangegangen Abschnitt bereits für die Inhalte der Zugriffsregeln geschehen ist. Nur so kann eine gezielte Delegierung erfolgen.
Die Schwierigkeit liegt darin, die Zahl der hierzu benötigten Prädikate möglichst klein zu halten. Die vorliegende Arbeit hat gezeigt, dass stets sehr ähnliche Prädikate für die verschiedenen Elemente und deren Hierarchien benötigt werden. Durch die strenge Typisierung sind die jeweils eigenen Prädikate jedoch nötig.

# Anhang A

# Literaturverzeichnis


[Aal98] Aalst, W.M.P. van der: *The Application of Petri Nets to Workflow Management*. The Journal of Circuits, Systems and Computers, 8(1):21–66, 1998.

[ABtHK00] Aalst, Wil M. P. van der, Alistair P. Barros, Arthur H. M. ter Hofstede und Bartek Kiepuszewski: *Advanced Workflow Patterns*. In: *Conference on Cooperative Information Systems*, Seiten 18–29, 2000.

[ABW88] Apt, K. R., H. A. Blair und A. Walker: *Foundations of Deductive Databases and Logic Programming*, Kapitel Towards a theory of declarative knowledge, Seiten 89–148. In: Minker, Jack [Min88], 1988.

[ANS04] *Information Technology - Role Based Access Control*. InterNational Committee for Information Technology Standards (formerly NCITS), Februar 2004. ANSI/INCITS 359-2004.

[AS00] Ahn, Gail-Joon und Ravi Sandhu: *Role-based authorization constraints specification*. TISSEC, 3(4):207–226, 2000.

[Bau96] Baumgarten, Bernd: *Petri-Netze - Grundlagen und Anwendungen*. Spektrum Akademischer Verlag, 2 Auflage, 1996.

[BBF00] Bertino, Elisa, Piero A. Bonatti und Eiena Ferrari: *TRBAC: A Temporal Role-based Access Control Model*. In: *Proceedings of the 5th ACM Workshop on Role-Based Access Control (RBAC-00)*, Seiten 21–30, N.Y., Juli 26–27 2000. ACM Press.

[BBF01] Bertino, E., P. Bonatti und E. Ferrari: *TRBAC: A Temporal Role-based Access Control Model*. ACM Transactions on Information and System Security, 4(3), August 2001.

[BBK94] Beth, Thomas, Malte Borcherding und Birgit Klein: *Valuation of Trust in Open Networks*. In: Gollmann, Dieter (Herausgeber): *ESORICS 94, Third European Symposium on Research in Computer Security*, Band 875 der Reihe *Lecture Notes in Computer Science*, Seiten 3–18, Brighton, UK, November 7 – 9 1994. Springer Verlag.

[BDS03] *Bundesdatenschutzgesetz*, Januar 2003.

[BF93] Borenstein, N. und N. Freed: *MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, September 1993. Obsoletes: RFC 1341, Updated by RFC1590.

[BRR87a] Brauer, Wilfried, Wolfgang Reisig und Grzegorz Rozenberg (Herausgeber): *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986*, Band 254 der Reihe *Lecture Notes in Computer Science*. Springer, 1987.

[BRR87b] Brauer, Wilfried, Wolfgang Reisig und Grzegorz Rozenberg (Herausgeber): *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986*, Band 254 der Reihe *Lecture Notes in Computer Science*. Springer Verlag, 1987.

[BVS00] Bonatti, Piero, Sabrina de Capitani di Vimercati und Pierangela Samarati: *A modular approach to composing access control policies*. In: Jajodia, Sushil und Pierangela Samarati (Herausgeber): *Proceedings of the 7th Conference on Computer and Communications Security (CCS-7)*, Seiten 164 – 173, Athens, Greece, November 1-4 2000. ACM Press.

[BVS02] Bonatti, Piero, Sabrina de Capitani di Vimercati und Pierangela Samarati: *An algebra for composing access control policies*. ACM Transactions on Information and System Security, 5(1):1–35, 2002.

[CDK02] Coulouris, George, Jean Dollimore und Tim Kindberg: *Verteilte Systeme - Konzepte und Design*. Pearson Studium, 3. überarbeitete Auflage, 2002.

[CGT89] Ceri, Stefano, Georg Gottlob und Letizia Tanca: *What You Always Wanted to Know About Datalog (And Never Dared to Ask)*. IEEE Transactions on Knowledge and Data Enineering, 1(1):146–166, März 1989.

[CGT90] Ceri, Stefano, Georg Gottlob und Letizia Tanca: *Logic Programming and Databases*. Surveys in Computer Science. Springer Verlag, Berlin – Heidelberg – New York, 1990.

[CH93] Christensen, Søren und Niels Damgaard Hansen: *Coloured Petri Nets Extended with Place Capacities, Test Arcs and Inhibitor Arcs.* In: Rozenberg, Grzegorz (Herausgeber): *Advances in Petri Nets 1993*, Band 674 der Reihe *Lecture Notes in Computer Science*, Seiten 186–205, Berlin – Heidelberg – New York, 1993. Springer Verlag.

[CW87] Clark, D. D. und D. R. Wilson: *A Comparison of Commercial and Military Computer Security Policies.* In: *IEEE Symposium on Security and Privacy*, Seiten 184–195, 1987.

[DBS+01] Ductal, K., M. Bogdanowisz, F. Scapolo, J. Leijten und J.-C. Burgelman (Herausgeber): *Scenarios for Ambient Intelligence in 2010*, IPTS Seville, Februar 2001. European Commission - Information Society Directorate-General. ISBN 92-894-0735-2.

[Den05] Denno, Torsten: *Entwicklung eines Analyzers für Administrationsnetze*. Diplomarbeit, Fachbereich Informatik, Universität Dortmund, Dezember 2005.

[Des98] Desel, Jörg: *Petrinetze, lineare Algebra und lineare Programmierung: Analyse, Verifikation und Korrektheitsbeweise von Systemmodellen.*, Band 26. B.G.Teubner Verlagsgesellschaft Teubner-Texte zur Informatik, Leipzig, 1998.

[DJ01] Desel, Jörg und Gabreil Juhás: *"What Is a Petri Net?´´ – Informal Answers for the Informed Reader*. In: Ehrig, Hartmut, Gabriel Juhas, Julia Padberg und Grzegorz Rozenberg (Herausgeber): *Unifying Petri Nets: Advances in Petri Nets*, Band 2128 der Reihe *Lecture Notes in Computer Science*, Seiten 1–25. Springer Verlag, Berlin – Heidelberg – New York, 2001.

[DR00] Daemen, Joan und Vincent Rijmen: *Rijndael for AES*. In: NIST (Herausgeber): *The Third Advanced Encryption Standard Candidate Conference, April 13–14, 2000, New York, NY, USA*, Seiten 343–347, Gaithersburg, MD, USA, 2000. National Institute for Standards and Technology.

[Eck04] Eckert, Claudia: *IT-Sicherheit, Konzepte – Verfahren – Protokolle*. Oldenbourg Verlag, München, Wien, 3. überarbeitete und erweiterte Auflage, 2004.

[ElG85] ElGamal, T.: *A Public-Key Cryptosystem and a Signature Scheme Bases on Discrete Logarithms*. IEEE Transactions on Information Theory, IT 31(4):469–472, 1985.

[FA05] Ferrari, Elena und Gail-Joon Ahn (Herausgeber): *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT)*, Stockholm, Sweden, Juni 2005. ACM SIGSAC, ACM Press.

[Far02] Farkas, Julius: *Theorie der einfachen Ungleichung*. Journal für reine und angewandte Mathematik, 124:1–27, 1902.

[Fay16] Fayol, Henri: *Administration Industrielle et Générale: Prévoyance, Organisation, Commandement, Coordination, Controle*. Dunot et E. Pinat, Paris, 1916.

[Fay29] Fayol, Henri: *Allgemeine und industrielle Verwaltung*. Oldenbourg, München, 1929. Übersetzung von [Fay16].

[FB96] Freed, N. und N. Borenstein: *RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, November 1996. Obsoletes RFC1521, RFC1522, RFC1590 Status: DRAFT STANDARD.

[FBK99] Ferraiolo, David F., John F. Barkley und D. Richard Kuhn: *A role-based access control model and reference implementation within a corporate intranet*. ACM Transactions on Information and System Security, 2(1):34–64, Februar 1999.

[FCK95] Ferraiolo, David, Janet Cugini und Rick Kuhn: *Role Based Access Control (RBAC): Features and Motivations*. In: *Annual Computer Security Applications Conference*. IEEE Computer Society Press, 1995.

[FF03] Ferrari, Elena und David Ferraiolo (Herausgeber): *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, Como, Italy, Juni 2003. ACM SIGSAC, ACM Press.

[FK92] Ferraiolo, D. und R. Kuhn: *Role-Based Access Controls*. In: *15th NIST-NCSC National Computer Security Conference*, Seiten 554–563, Oktober 1992.

[FKC03] Ferraiolo, David F., D. Richard Kuhn und Ramaswamy Chandramouli: *Role-Based Access Control*. Artech House computer security series. Artech House, Norwood, MA, 2003.

[FSG+01] Ferraiolo, David F., Ravi Sandhu, Serban Gavrila, D. Richard Kuhn und Ramaswamy Chandramouli: *A Proposed Standard for Role-Based Access Control*. ACM Transactions on Information and System Security, 4(3):224–274, August 2001.

[Gen87] Genrich, Hartmann J.: *Predicate/Transition Nets*. In: Brauer, Wilfried et al. [BRR87a], Seiten 207–247.

[GL79] Genrich, H. J. und K. Lautenbach: *The analysis of distributed systems by means of predicate/transition-nets*. In: Kahn, G. (Herausgeber): *Semantic of Concurrent Computation*, Band 70 der Reihe *Lecture Notes in Computer Science*, Seiten 123–146, Berlin – Heidelberg – New York, 1979. Springer Verlag.

[GL81] Genrich, H. J. und K. Lautenbach: *System modelling with high-level Petri nets.* Theoretical Computer Science, 13:109–136, Jan 1981. Predicate Transistion Petri Nets.

[GL83] Genrich, Hartmann J. und Kurt Lautenbach: *S-Invariance in Predicate/Transition Nets.* In: Pagnoni, Anastasia und Grzegorz Rozenberg (Herausgeber): *European Workshop on Applications and Theory of Petri Nets*, Band 66 der Reihe *Informatik-Fachberichte*, Seiten 98–111. Springer Verlag, 1983.

[GL88] Gelfond, Michael und Vladimir Lifschitz: *The Stable Model Semantics for Logic Programming.* In: Kowalski, Robert A. und Kenneth Bowen (Herausgeber): *Proceedings of the Fifth International Conference on Logic Programming*, Seiten 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

[GMMP91] Ghezzi, C., D. Mandrioli, S. Morasca und Mauro Pezzè: *A Unified High-Level Petri Net Model For Time-Critical Systems.* IEEE Transactions on Software Engineering, 17(2):160–172, feb 1991.

[GV03] Girault, Claude und Rüdiger Valk (Herausgeber): *Petri Nets for Systems Engineering – A Guide to Modeling, Verification, and Applications.* Springer Verlag, Berlin – Heidelberg – New York, 2003.

[Hal05] Halfter, Uwe: *Entwicklung eines Interpreters für eine modulare Autorisierungssprache.* Diplomarbeit, Fachbereich Informatik, Universität Dortmund, March 2005.

[Her03] Herrmann, Peter: *Trust-Based Protection of Software Component Users and Designers.* In: Nixon, Paddy und Sotirios Terzis (Herausgeber): *iTrust*, Band 2692 der Reihe *Lecture Notes in Computer Science*, Seiten 75–90, Crete, Greece, Mai 2003. Springer.

[HG86] Haddad, Serge und Claude Girault: *Algebraic structure of flows of a regular coloured net.* In: Brauer, Wilfried et al. [BRR87b], Seiten 73–88.

[HJJJ85] Huber, Peter, Arne M. Jensen, Leif O. Jepsen und Kurt Jensen: *Towards reachability trees for high-level Petri nets.* In: Rozenberg, Grzegorz, Hartmann J. Genrich und Gérard Roucairol (Herausgeber): *Advances in Petri Nets 1984, European Workshop on Applications and Theory in Petri Nets*, Band 188 der Reihe *Lecture Notes in Computer Science*, Seiten 215–233. Springer Verlag, 1985.

[ISO00a] International Organization for Standardization: *Qualitätsmanagment - Anforderungen*, Genf, 2000. ISO.

[ISO00b] International Organization for Standardization: *Qualitätsmanagment - Grundlagen und Begriffe*, Genf, 2000. ISO.

[Jen81] Jensen, Kurt: *Coloured Petri Nets and the Invariant-Method.* Theoretical Computer Science, 14:317–336, 1981.

[Jen87] Jensen, Kurt: *Coloured Petri Nets.* In: Brauer, Wilfried et al. [BRR87a], Seiten 248–299.

[Jen96a] Jensen, Kurt: *Coloured petri nets: Basis Concepts, analysis methods and practical use.* Monographs in theretical computer science. Springer Verlag, Berlin – Heidelberg – New York, 2. Auflage, 1996. volume 1.

[Jen96b] Jensen, Kurt: *Coloured petri nets: Basis Concepts, analysis methods and practical use.* Monographs in theretical computer science. Springer Verlag, Berlin – Heidelberg – New York, 2. Auflage, 1996. volume 2.

[Jen96c] Jensen, Kurt: *Coloured petri nets: Basis Concepts, analysis methods and practical use*. Monographs in theretical computer science. Springer Verlag, Berlin – Heidelberg – New York, 2. Auflage, 1996. volume 3.

[JF04] Jaeger, Trent und Elena Ferrari (Herausgeber): *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies (SACMAT)*, Yorktown Heights, NY, Juni 2004. ACM SIGSAC, ACM Press.

[Jøs99] Jøsang, Audun: *An Algebra for Assessing Trust in Certification Chains*. In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 1999*, San Diego, CA, 1999. The Internet Society.

[JSS97] Jajodia, Sushil, Pierangela Samarati und V. S. Subrahmanian: *A logical language for expressing authorizations*. In: *Proceedings of the IEEE Symp. on Research in Security and Privacy*, Seiten 31–42, Oakland, CA, Mai 1997. IEEE Computer Society Press.

[JSSB97] Jajodia, Sushil, Pierangela Samarati, V. S. Subrahmanian und Eliza Bertino: *A Unified Framework for Enforcing Multiple Access Control Policies Security*. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Band 26,2 der Reihe *SIGMOD Record*, Seiten 474–485, New York, Mai 13–15 1997. ACM Press.

[JSSS01] Jajodia, Sushil, Pierangela Samarati, Maria Luisa Sapino und V. S. Subrahmanian: *Flexible Support for Multiple Access Control Policies*. ACM Transactions on Database Systems, 26(2):214–260, June 2001.

[JV80] Jantzen, Matthias und Rüdiger Valk: *Formal Properties of Place/Transition Nets*. In: Brauer, Wilfried (Herausgeber): *Advanced Course: Net Theory and Applications*, Band 84 der Reihe *Lecture Notes in Computer Science*, Seiten 165–212. Springer, 1980.

[Kie99] Kieser, Alfred (Herausgeber): *Organisationstheorien*. Kohlhammer, Stuttgart, Berlin, Köln, 3. überarbeitete und erweiterte Auflage, 1999.

[KMPP04] Koch, M., L. V. Mancini und F. Parisi-Presicce: *Administrative scope in the graph-based framework*. In: Jaeger, Trent und Elena Ferrari [JF04], Seiten 97–104.

[Kos03] Kostin, Alexander E.: *Reachability Analysis in T-Invariant-Less Petri Nets*. IEEE Transactions on Automatic Control, 48(6):1019–1024, 2003.

[KSM03] Kern, Axel, Andreas Schaad und Jonathan Moffett: *An administration concept for the enterprise role-based access control model*. In: Ferrari, Elena und David Ferraiolo [FF03], Seiten 3–11.

[KSS03] Kuhlmann, Martin, Dalia Shohat und Gerhard Schimpf: *Role mining - revealing business roles for security administration using data mining technology*. In: *Proceedings of the eighth ACM symposium on Access control models and technologies*, Seiten 179–186, Como, Italy, 2003. ACM Press.

[KT98] Kostin, Alexander E. und Svetlana A. Tchoudaikina: *Yet another reachability algorithm for Petri nets*. SIGACT News, 29(4):98–110, 1998.

[LMP02] Lautenbach, Kurt, Jörg Müller und Stephan Philippi: *Modellierung, Simulation und Analyse mit dem Petri-Netz-Tool POSEIDON*. In: Desel, Jörg und Mathias Weske (Herausgeber): *Promise*, Band 21 der Reihe *LNI*, Seiten 163–174. GI, 2002.

[LR94] Lautenbach, Kurt und Hanno Ridder: *Liveness in Bounded Petri Nets Which Are Covered by T-Invariants.* In: Valette, Robert (Herausgeber): *Application and Theory of Petri Nets*, Band 815 der Reihe *Lecture Notes in Computer Science*, Seiten 358–375. Springer, 1994.

[Min88] Minker, Jack (Herausgeber): *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann Publishers Inc., Los Altos, CA, 1988.

[MS82] Martinez, J. und Manuel Silva: *A Simple and Fast Algorithm to Obtain All Invariants of a Generalized Petri Net.* In: Girault, Claude und Wolfgang Reisig (Herausgeber): *Selected Papers from the First and the Second European Workshop on Application and Theory of Petri Nets*, Band 52 der Reihe *Informatik-Fachberichte*, Seiten 301–310. Springer Verlag, 1982.

[MV86] Memmi, Gérard und Jacques Vautherin: *Analysing Nets by the Invariant Method.* In: Brauer, Wilfried et al. [BRR87b], Seiten 300–336.

[NO94] Nyanchama, Matunda und Sylvia L. Osborn: *Access Rights Administration in Role-Based Security Systems.* In: *IFIP Workshop on Database Security*, Seiten 37–56, 1994.

[NO96] Nyanchama, Matunda und Sylvia Osborn: *The Role Graph Model Collaborative Systems.* In: Youman, Charles E. et al. [YSC96], Seiten 25 – 31.

[NO99] Nyanchama, Matunda und Sylvia Osborn: *The role graph model and conflict of interest.* ACM Transactions on Information and System Security, 2(1):3–33, Februar 1999.

[OG00] Osborn, Sylvia und Yuxia Guo: *Modeling Users in Role-Based Access Control.* In: Rebensburg, Klaus, Charles Youman und Vijay Atluri (Herausgeber): *Proceedings of the fifth ACM Workshop on Role-Based Access Control*, Seiten 31–38, Berlin, Germany, Juli 26–27 2000. ACM SIGSAC, ACM Press.

[OHL03] Osborn, Sylvia L., Yan Han und Jun Liu: *A methodology for managing roles in legacy systems.* In: Ferrari, Elena und David Ferraiolo [FF03], Seiten 33–40.

[Olb01] Olbertz, Dirk: *Verteilte Suche auf verschiedenen Metadatenbankformaten.* Diplomarbeit, Informatik III, Universität Dortmund, April 2001.

[OS02] Oh, Sejong und Ravi Sandhu: *A Model for Role Administration Using Organization Structure.* In: Sandhu, Ravi und Elisa Bertino [SB02], Seiten 155–162.

[OSM00] Osborn, Sylvia, Ravi Sandhu und Qamar Munawer: *Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies.* ACM Transactions on Information and System Security, 3(2):85 – 106, Mai 2000.

[PDF99] Picot, Arnold, Helmut Dietl und Egon Franck: *Organisation – Eine ökonomische Perspektive.* Schäffer-Poeschel Verlag, Stuttgart, 2. überarbeitete und erweiterte Auflage, 1999.

[Pet62] Petri, Carl Adam: *Kommunikation mit Automaten.* Schriften des IIM 2, Institut für Instrumentelle Mathematik, Bonn, Germany, 1962.

[Pet66] Petri, Carl Adam: *Kommunikation mit Automaten.* Technischer Bericht RADC-TR-65–377, Vol. 1, Griffiss Air Force Base, New York, 1966. englische Übersetzung von [Pet62].

[Rob01] Robbins, Stephen R.: *Organisation der Unternehmung.* Pearson Studium, München, 9. Auflage, 2001.

[RSA78] Rivest, Ronald L., A. Shamir und L.M Adleman: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM, 21(2):120–126, Feb. 1978.

[RYA00] Rebensburg, Klaus, Charles Youman und Vijay Atluri (Herausgeber): *Proceedings of the fifth ACM Workshop on Role-Based Access Control*, Berlin, Germany, Juli 26–27 2000. ACM SIGSAC, ACM Press.

[SA98a] Sandhu, Ravi und Gail-J. Ahn: *Decentralised group hierachies in UNIX: An experiment and lessons learned.* In: *Proceedings of 21st NIST-NCSC National Computer Security Conference*, Arlington, VA, Oktober 1998.

[SA98b] Sandhu, Ravi und Gail-J. Ahn: *Group hierarchies with decentralized user assignment in Windows NT.* In: *Proceedings International Association of Science and Technology for Development (IASTED) Conference on Software Engineering*, Las Vegas, Nevada, Oktober 1998.

[San96] Sandhu, Ravi: *Rationale for the RBAC96 family of access control models.* In: Youman, Charles E. et al. [YSC96]. Article No. 9.

[San98] Sandhu, Ravi S.: *Role Activation Hierarchies.* In: Youman, Charles und Trent Jaeger [YJ98], Seiten 33–40.

[SB97] Sandhu, Ravi S. und Venkata Bhamidipati: *The URA97 Model for Role-Based User-Role Assignment.* In: *Proceedings of the IFIP WG 11.3 Workshop on Database Security*, Seiten 262–275, Lake Tahoe, CA, August 1997.

[SB02] Sandhu, Ravi und Elisa Bertino (Herausgeber): *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT)*, Monterey, CA, USA, Juni 2002. ACM SIGSAC, ACM Press.

[SBC+97] Sandhu, Ravi, Venkata Bhamidipati, Edward Coyne, Srinivas Canta und Charles Youman: *The ARBAC97 Model for Role-Based Administration of Roles: Preliminary Description and Outline.* In: Youman, Charles et al. [YCJ97], Seiten 41–54.

[SBM99] Sandhu, Ravi, Venkata Bhamidipati und Qamar Munawer: *The ARBAC97 model for role-based administration of roles.* ACM Transactions on Information and System Security, 2(1):105–135, Februar 1999.

[SCFY96] Sandhu, R., E. Coyne, H. Feinstein und C. Youman: *Role Based Access Control Models.* IEEE Computer, 29(2):38 – 47, Feb 1996.

[Sch94] Schneier, Bruce: *Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish).* In: Anderson, Ross J. (Herausgeber): *Proceedings of the Fast Software Encryption, Cambridge Security Workshop*, Band 809 der Reihe *Lecture Notes in Computer Science*, Seiten 191–204, Cambridge,UK, December 9-11 1994. Springer.

[Sch96a] Schneier, Bruce: *Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C.* Addison–Wesley, Bonn [u.a.], 1. Auflage, 1996.

[Sch96b] Schneier, Bruce: *Applied Cryptography: protocols, algorithms, and source code in C.* J. Wiley & Sons, Inc, 2. Auflage, 1996.

[Seg03] Seghezzi, Hans Dieter: *Integriertes Qualitätsmanagement – Das St. Galler Konzept.* Carl Hanser Verlag, München, Wien, 2. vollständig überarbeitete und erweiterte Auflage, 2003.

[SFK00] Sandhu, Ravi, David Ferraiolo und Richard Kuhn: *The NIST Model for Role-based Access Control: Towards a Unified Standard*. In: Rebensburg, Klaus, Charles Youman und Vijay Atluri (Herausgeber): *Proceedings of the fifth ACM Workshop on Role-Based Access Control*, Seiten 47–64, Berlin, Germany, Juli 26–27 2000. ACM SIGSAC, ACM Press.

[SGG00] Silberschatz, Avi, Peter Galvin und Greg Gagne: *Applied Operating System Concepts*. John Wiley & Sons, Inc, 1. Auflage, 2000.

[She88] Shepherdson, J. C.: *Foundations of Deductive Databases and Logic Programming*, Kapitel Negation in logic programming, Seiten 19–88. In: Minker, Jack [Min88], 1988.

[SJ01] Sandhu, Ravi und Trent Jaeger (Herausgeber): *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT)*, Chantily, VA, USA, Mai 2001. ACM SIGSAC, ACM Press.

[SM99] Sandhu, Ravi und Qamar Munawer: *The ARBAC99 Model for Administration of Roles*. In: *Proceedings of the 15th Annual Computer Security Applications Conference*, Phoenix, Arizona, Dec 1999.

[SP98] Sandhu, Ravi und Joon S. Park: *Decentralized user-role assignment for Web-based intranets*. In: Youman, Charles und Trent Jaeger [YJ98], Seiten 1 – 12.

[Sta90] Starke, Peter H.: *Analyse von Petri–Netz–Modellen*. B.G. Teubner, Stuttgart, 1990.

[SV01] Samarati, Pierangela und Sabrina de Capitani di Vimercati: *Access Control: Policies, Models, and Mechanisms*. In: Focardi, R. und R. Gorrieri (Herausgeber): *Foundations of Security Analysis and Design : Tutorial Lectures*, Band 2171 der Reihe *Lecture Notes in Computer Science*, Seiten 137–196, 2001.

[SW78] Schiffers, Michael und H.F Wedde: *Analyzing Program Solutions of Coordination Problems by CP-Nets*. In: Mazurkiewicz, A. (Herausgeber): *Proceedings of the 7th Symposium on Mathematical Foundations of Computer Science*, Band 64 der Reihe *Lecture Notes in Computer Science*, Seiten 416–422. Springer Verlag, Zakopane, Poland, 1978.

[SZ99] Schulte-Zurhausen, Manfred: *Organisation*. Verlag Vahlen, München, 2. völlig überarbeitete und erweiterte Auflage, 1999.

[Tat01] Tatah, Veye: *The Implementation of a file Service for the distributed file system Dragon Slayer III*. Diplomarbeit, Fachbereich Informatik, Universität Dortmund, Oktober 2001.

[Tay11] Taylor, F. W. (Herausgeber): *The Principles of Scientific Management*. Harper & Row, London, 1911.

[TCG03a] Trusted Computing Group: *TPM Main Part 1 Design Principles*, Specification Version 1.2 Auflage, Oktober 2003.

[TCG03b] Trusted Computing Group: *TPM Main Part 2 TPM Structures*, Specification Version 1.2 Auflage, Oktober 2003.

[TCG03c] Trusted Computing Group: *TPM Main Part 3 Commands*, Specification Version 1.2 Auflage, Oktober 2003.

[Thi87] Thiagarajan, P. S.: *Elementary Net Systems*. In: Brauer, Wilfried et al. [BRR87b], Seiten 26–59.

[Tho97] Thomas, Rshan K.: *Team-based Access Control (TMAC): A Primitve for Applying Role-based Access Controls in Collaborative Enviroments.* In: *Proceedings of the 2nd ACM workshop on Role-based access control*, Seiten 13 – 19, Fairfax, VA USA, November 6-7 1997. ACM SIGSAC, ACM Press.

[Val83] Valk, R.: *Infinitive Behaviour of Petri Nets.* Theoretical Computer Science, 25:311–341, 1983.

[Val94] Valmari, A.: *State of the Art Report: Stubborn Sets.* Petri Net Newsletter, (46):531–547, 1994.

[vG88] Gelder, Allen van: *Foundations of Deductive Databases and Logic Programming*, Kapitel Negation as failure using tight derivations for general logic programs, Seiten 149–176. In: Minker, Jack [Min88], 1988.

[vG89] Gelder, Allen van: *The alternating fixpoint of logic programs with negation.* In: *Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, Seiten 1–10. ACM Press, 1989.

[vGRS91] Gelder, Allen van, Kenneth A. Ross und John S. Schlipf: *The well-founded semantics for general logic programs.* Journal of the ACM, 38(3):619–649, Jul 1991.

[WJ01] Wijesekera, Duminda und Sushil Jajodia: *Policy algebras for access control: the propositional case.* In: Reiter, Mike und Pierangela Samarati (Herausgeber): *Proceedings of the 8th Conference on Computer and Communications Security (CCS-8)*, Seiten 38–47, Philadelphia, PA, November 5-8 2001. ACM Press.

[WJ02] Wijesekera, Duminda und Sushil Jajodia: *Policy algebras for access control – the predicate case.* In: Atluri, Vijayalakshmi (Herausgeber): *Proceedings of the 8th Conference on Computer and Communications Security (CCS-8)*, Seiten 171–180, Washingtion, DC, Nov 18-22 2002. ACM Press.

[WJ03] Wijesekera, Duminda und Sushil Jajodia: *A propositional policy algebra for access control.* ACM Transactions on Information and System Security, 6(2):286–325, 2003.

[WL99] Wedde, Horst F. und Mario Lischka: *New Dimensions in Distributed Journalism Through Dragon Slayer III.* In: *Proc. of the 7th Euromicro Workshop on Parallel and Distributed Processing*, Seiten 112–118, Madeira, Portugal, Feb 1999. Euromicro, IEEE Computer Society Press.

[WL01] Wedde, Horst F. und Mario Lischka: *Modular Authorization.* In: Sandhu, Ravi und Trent Jaeger (Herausgeber): *Proceedings of the sixth ACM Symposium on Access Control Models and Technologies (SACMAT)*, Seiten 97–105, Chantilly, Virginia, May 3-4 2001. ACM SIGSAC, ACM Press.

[WL03a] Wedde, Horst F. und Mario Lischka: *Composing Heterogenous Access Policies between Organizations.* In: *Proceedings of the IADIS International Conference e-Society 2003*, Seiten 477–484, Lisbon/ Portuagal, June, 3-6 2003. International Association for Development of the Information Society, IADIS Press.

[WL03b] Wedde, Horst F. und Mario Lischka: *Cooperative Role-Based Administration.* In: Ferrari, Elena und David Ferraiolo (Herausgeber): *Proceedings of the eigth ACM Symposium on Access Control Models and Technologies (SACMAT)*, Seiten 21–32, Como, Italy, June, 2–3 2003. ACM SIGSAC, ACM Press.

[WL04a] Wedde, Horst F. und Mario Lischka: *Modular Authorization and Administration.* ACM Transactions on Information and System Security, 7(3):363–391, August 2004.

[WL04b] Wedde, Horst F. und Mario Lischka: *Role-Based Access Control in Ambient and Remote Space*. In: Jaeger, Trent und Elena Ferrari (Herausgeber): *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies (SACMAT)*, Seiten 21–30, Yorktown Heights, NY, June, 2–4 2004. ACM SIGSAC, ACM Press.

[WS00] Wedde, Horst F. und Jens-Oliver Siepmann: *A Universal Framework for Managing Metadata in the Distributed Dragon Slayer System*. In: *Euromicro Workshop on Multimedia and Telecommunications*, Seiten 2096–2102. Euromicro, IEEE Computer Society Press, September 2000.

[YCJ97] Youman, Charles, Edward Coyne und Trent Jaeger (Herausgeber): *Proceedings of the 2nd ACM Workshop on Role-Based Access Control (RBAC-97)*, Fairfax, VA, USA, November 1997. ACM SIGSAC, ACM Press.

[YJ98] Youman, Charles und Trent Jaeger (Herausgeber): *Proceedings of the third ACM Workshop on Role-Based Access Control (RBAC-98)*, Fairfax, VA, USA, Oktober 22–23 1998. ACM SIGSAC, ACM Press.

[YKB93] Yahalom, R., B. Klein und T. Beth: *Trust Relationships in Secure Systems–A Distributed Authentication Perspective*. In: *IEEE Computer Society Symposium on Research in Security and Privacy*, Seiten 150–164, Oakland, CA, Mai 1993.

[YO99] Youman, Charles und Sylvia Osborn (Herausgeber): *Proceedings of the fourth ACM Workshop on Role-Based Access Control (RBAC-99)*, Fairfax, VA, USA, November 6–7 1999. ACM SIGSAC, ACM Press.

[YSC96] Youman, Charles E., Ravi S. Sandhu und Edward J. Coyne (Herausgeber): *Proceedings of the first ACM Workshop on Role-based access control*, Gaithersburg, Maryland, US, 1996. ACM SIGSAC, ACM Press.

# Anhang B

# Index