Universität Karlsruhe (TH)

DIPLOMARBEIT

Semantische Zugriffskontrolle - Rechtemanagement mit Ontologien und Regeln

von

cand. inform. Mike Sibler

eingereicht am 27.07.2005 am
Institut für Angewandte Informatik
und Formale Beschreibungsverfahren (AIFB)
der Universität Karlsruhe

Referent: Prof. Dr. Studer
Coreferent: Prof. Dr. Stucky
Betreuer: Dipl. Inform. Max Völkel

Erklärung:

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, 27. Juli 2005

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Probleme heutiger Zugriffskontrollsysteme:

Seit Anbeginn der Entwicklung von Mehrbenutzer-Computersystemen galt es die Daten der einzelnen Benutzer vor Missbrauch zu schützen. Durch die zunehmende Vernetzung von Computersystemen wird immer mehr Nutzern der Zugriff auf verteilte Informationsbestände ermöglicht. Aufgrund dieser Entwicklungen wurde viel Aufwand zum Schutz der zu verwaltenden Informationen investiert. Im Mittelpunkt dieser Bemühungen steht bis heute das Rechtemanagement. Ziel des Rechtemanagements ist es, den Zugriff auf Ressourcen nur berechtigten Benutzern zu gestatten. Um dies zu gewährleisten, umfasst das Rechtemanagement sowohl die Ressourcen- und Benutzerverwaltung als auch die Kontrolle der Zugriffe von Benutzern auf Ressourcen. Diese Zugriffskontrollfunktion kann wiederum in Authentifizierung und Autorisierung unterteilt werden. Die Authentifizierung bezeichnet den Vorgang der Identifikation eines Benutzers gegenüber einem System durch ein eindeutiges Merkmal, z.B. Benutzername und Passwort. Die Autorisierung erfolgt nach einer erfolgreichen Authentifizierung und überprüft, ob ein Benutzer die zur Ausführung der gewünschten Aufgabe notwendigen Zugriffsrechte im System besitzt.

Das Hauptaugenmerk dieser Arbeit liegt auf der Autorisierung. Über die Jahrzehnte haben sich hierzu eine Reihe von Verfahren entwickelt, die ihre Nützlichkeit in vielfältigen Anwendungen unter Beweis gestellt haben. Viele dieser Mechanismen haben jedoch eine mangelnde Flexibilität gemeinsam, da sie sehr spezifisch für die jeweiligen Anwendungen entworfen wurden und im Allgemeinen eng mit diesen verwoben sind. Dies hat zur Folge, dass Anpassungen an Strategien zur Rechtevergabe meist nur unter großem Aufwand möglich sind [Loc03]. Solche Anpassungen treten jedoch aufgrund organisationsspezifischer Änderungen, beispielsweise aufgrund von Firmenfusionen oder Neuausrichtung von Bereichsstrukturen, immer wieder auf.

Grundidee zur Überwindung dieser Probleme:

Ein erster Ansatz zur Behebung der mangelnden Flexibilität existierender Zugriffskontrollsysteme ist die Ausgliederung der Zugriffskontrolle in eine wiederverwendbare Komponen-

te. Diese Auslagerung der Zugriffskontrolle erlaubt die zentrale Beschreibung des Rechtemanagements einer Organisation über Anwendungs- und Systemgrenzen hinweg. Die in der Komponente definierten Sicherheitsstrategien werden durch Anbindung konkreter Anwendungen auf diese übertragen. Anfallende Änderungen der Organisationsstruktur eines Unternehmens lassen sich auf einfache Art und Weise global durch Anpassung der in der Komponente beschriebenen Sicherheitsstrategien durchführen.

Um eine vielseitige Verwendbarkeit der Zugriffskontrollkomponente zu gewährleisten, muss diese in der Lage sein, die wichtigsten Eigenschaften gängiger Zugriffskontrollsysteme umsetzen zu können. Individuelle Anpassungen des Rechtemanagements können durch Erweiterung der Komponente vorgenommen werden. Die Umsetzung von Erweiterungen ermöglicht die Anpassungen der Komponente an Eigenheiten organisationsspezifischer Sicherheitsstrategien und trägt wesentlich zur Flexibilität und damit zur vielseitigen Verwendbarkeit des Zugriffskontrollsystems bei.

Wichtig ist zudem, dass die Zugriffkontrollkomponente trotz aller zuvor beschriebenen Eigenschaften einfach zu verwenden sein soll. Dies bedeutet insbesondere, dass die Beantwortung von Zugriffsanfragen ohne großen Aufwand möglich ist.

Um die formulierten Ideen umzusetzen, wird der Kern des Zugriffskontrollsystems durch ein Zugriffskontrollmodell beschrieben. Dieses Modell beschreibt existierende Ansätze des Rechtemanagements, bietet Möglichkeiten der Anpassung an individuelle Gegebenheiten und enthält alle Informationen, die zur Beantwortung von Autorisierungsanfragen erforderlich sind. Wissensbasierte Systeme in Form von Ontologien und Regeln bieten die idealen Voraussetzungen zur Umsetzung eines solchen Modells. Obwohl es sich bei Wissensmanagement um eine relativ junge Disziplin handelt, wurden in den letzten Jahren - angetrieben durch die Vision des Semantic Web - erhebliche technische Fortschritte in der Beschreibung von komplexen Modellen und deren Interpretation erzielt. Ontologien modellieren hierbei komplexe Sachverhalte in Form von Wissensbasen. Regeln werden eingesetzt, um Schlussfolgerungen auf den durch Ontologien beschriebenen Modellen abzuleiten. Mittels Regeln kann somit Logik zur Interpretation von Anfragen, die in althergebrachten Systemen in Quellcode formuliert ist, bereits zur Entwurfszeit in ein Modell eingebracht werden. Übertragen auf das Zugriffskontrollmodell können Regeln zur Beantwortung von Autorisierungsanfragen eingesetzt werden, ohne dass dafür Programmcode notwendig ist.

Somit bietet die Beschreibung der Eigenschaften von Zugriffskontrollsystemen mittels Ontologien und Regeln folgende Vorteile:

Wiederverwendbarkeit: Da das Rechtemanagement unabhängig von einem konkreten Anwendungsfall modelliert ist, kann es über Applikationsgrenzen hinweg eingesetzt werden. Die zuvor bereits erwähnten Anpassungen existierender Systeme können somit unabhängig von einer speziellen Anwendung durchgeführt werden, was die Einführung und Administration einer Zugriffskontrolle wesentlich vereinfacht.

Die Zugriffskontrollkomponente kann aber auch unternehmensweit eingesetzt werden und dabei applikationsspezifische Anpassungen definieren. Somit können trotz systemweiter einheitlicher Verwaltung des Rechtemanagements anwendungsspezifische Besonderheiten durch Definition spezifischer Regeln beschrieben werden.

Flexibilität und Erweiterbarkeit: Das durch die Ontologie bereitgestellte Modell kann als eine Art modularer Baukasten betrachtet werden. Abhängig von den gewünsch-

ten Anforderungen kann die Zugriffskontrolle durch Instantiierung geeigneter, in der Ontologie beschriebener Konzepte, individuell an die Bedürfnisse konkreter Systeme angepasst werden. Reichen die in dem Zugriffskontrollmodell vorgesehenen Konzepte für die konkreten Anforderungen nicht aus, können mit vergleichsweise geringem Aufwand konzeptionelle Erweiterungen vorgenommen werden.

Zudem lassen sich Anpassungen von bereits existierenden Sicherheitsstrategien durch einfache Änderungen der zugrunde liegenden Ontologien und Regeln durchführen, ohne Anpassungen am Programmcode vornehmen zu müssen. Die durch diesen ontologiebasierten Ansatz erreichte Flexibilität und die damit einhergehende leichte Anpassungsfähigkeit ist der zentrale Vorteil des in dieser Arbeit vorgestellten Rechtemanagementsytems.

Mächtigkeit der Zugriffsanfragen: Ein weiteres wichtiger Vorteil, der durch den Einsatz von Ontologien und Regeln erreicht werden soll, ist die einfache Benutzbarkeit des Systems. Dazu gehört, dass allgemeine Zugriffsanfragen auf einfache Art und Weise an das System gerichtet werden können. Eine Anfrage kann hierbei auf unterschiedlichen Abstraktionsebenen formuliert werden. Spezielle Zugriffsanfragen beantworten, ob ein gegebener Benutzer auf ein gegebenes Objekt zugreifen darf oder nicht. Auch allgemeinere Anfragen sind formulierbar. Diese werden im Folgenden beispielhaft dargestellt:

1. Welche Rechte besitzt Benutzer `Joe` im System? Auch die Umkehrung, also die Anfrage welche Rechte ein Benutzer des Systems nicht hat, soll durch die Zugriffskontrollkomponente beantwortet werden können.
2. Welche Benutzer dürfen lesend auf die Datei `build.xml` zugreifen? Diese Anfrage wertet sämtliche Rechte aus, die auf einem Objekt definiert sind.
3. Was darf `Alice`, was `Bob` nicht darf?

Wie an diesen Beispielen sichtbar wird, sind eine Vielzahl unterschiedlicher Anfragen möglich. Das semantische Modell, auf dem die Beschreibung der Zugriffskontrollkomponente basiert, sollte in der Lage sein, sowohl spezielle als auch allgemeine Zugriffsanfragen möglichst einfach beantworten zu können.

1.2 Ziele der Arbeit

Ziel dieser Arbeit ist die Umsetzung eines Zugriffskontrollsystems in Form einer anwendungsunabhängigen, flexiblen und erweiterbaren Zugriffskontrollkomponente. Diese Komponente beschreibt Zugriffskontrollsysteme allgemein und erlaubt durch Erweiterungen eine Umsetzung spezifischer Sicherheitsstrategien eines Unternehmens. Somit versucht diese Arbeit Rechtemanagement in eine zentrale Komponente auszugliedern, die Gemeinsamkeiten gängiger Zugriffskontrollsysteme vereint und individuelle Anpassungen ermöglicht. Diese Vorgehensweise kann mit den in den letzten Jahren forcierten Bemühungen der Prozessmodellierung verglichen werden. Während Prozessmodelle eine Abfolge von Ereignissen – typischerweise Geschäftsvorgänge eines Unternehmens – beschreiben, modelliert das

in dieser Arbeit entworfene Zugriffskontrollsystem Sicherheitsstrategien eines Unternehmens. Ähnlich wie bei Geschäftsprozessen erlaubt diese Vorgehensweise die Formulierung und damit einhergehend die Austauschbarkeit bzw. Wiederverwendbarkeit organisationsspezifischer Rechtemanagementlösungen.

Die Ausgliederung der Zugriffskontrolle in eine anwendungsunabhängige Komponente, die sich durch Erweiterbarkeit und Anpassbarkeit zur Beschreibung organisationsspezifischer Sicherheitsstrategien eignet, erfolgt durch die Beschreibung eines semantischen Modells, welches die wichtigsten Eigenschaften eines Zugriffskontrollsystems beschreibt. Die Erstellung dieses Modells ist zentraler Aspekt dieser Arbeit.

Zur Umsetzung des semantischen Modells der Zugriffskontrolle werden Ontologien und Regeln verwendet. Das Modell soll trotz einer großen Ausdruckmächtigkeit eine einfache Verwendbarkeit und einen flexiblen Einsatz gewährleisten. Die zu erstellenden Ontologien sollen in der Lage sein, Eigenschaften gängiger Zugriffskontrollsysteme beschreiben zu können. Regeln interpretieren daraufhin die Informationen der Wissensbasis und dienen der Entscheidungsfindung bei der Beantwortung von Zugriffsanfragen. Durch den Einsatz von Regeln soll die Flexibilität des Modells erzielt werden. Erweiterungen des Modells zur Beschreibung anwendungsspezifischen Verhaltens sollen allein durch konzeptionelle Erweiterung der Ontologien und Definition neuer Regeln und ohne zusätzlichen Implementierungsaufwand vollzogen werden können.

Die Eignung dieses Modells wird in einer prototypischen Implementierung eines Zugriffskontrollsystems verifiziert und durch Nachbildung bestehender Zugriffskontrollsysteme evaluiert.

1.3 Gliederung der Arbeit

In Kapitel 2 werden zunächst Grundlagen für die Beschreibung eines Zugriffskontrollmodells durch semantische Technologien gelegt. Bevor mit OWL eine spezielle Ontologiesprache betrachtet wird, werden Ontologien und Regeln allgemein beschrieben. Weiterhin werden Eigenschaften von Zugriffskontrollmodellen, basierend auf existierenden Systemen und einschlägiger Literatur, analysiert. Zum Abschluss wird der Stand der Technik, in Bezug auf die semantische Beschreibung von Zugriffskontrollsystemen, erarbeitet.

Basierend auf diesen Grundlagen wird in Kapitel 3 das in dieser Arbeit verfolgte allgemeine Konzept zur Modellierung des Zugriffskontrollsystems vorgestellt. Zunächst wird die kombinierte Verwendung von Ontologien und Regeln besprochen. Das Zugriffskontrollmodell kann als eigenständiges Modul über Applikationsgrenzen hinweg eingesetzt werden. Das Zusammenspiel mit externen Anwendungen, die abstrakte Architektur des Modells sowie Besonderheiten für den Einsatz eines semantischen Autorisierungssystems werden zum Abschluss des Kapitels vorgestellt.

Kapitel 4 und Kapitel 5 beschreiben den Kern dieser Arbeit – das semantische Modell der Zugriffskontrolle. In Kapitel 4 werden durch ontologiebasierte Modellierung die Elemente eines Autorisierungsdienstes und deren Eigenschaften beschrieben. Zur einfacheren Verständlichkeit wird hierbei ein schichtenbasierter Ansatz verfolgt. Kapitel 5 beschreibt Regeln, die die zuvor vorgestellten Ontologien interpretieren. Die Regeln sollen dem Modell die Mächtigkeit verleihen, Autorisierungsanfragen ohne jeglichen Programmcode beantworten zu können. Ein besonderer Schwerpunkt liegt hier auf der Auflösung von Konflikten bei

widersprüchlicher Vergabe von Verboten und Genehmigungen.

Um dieses semantische Modell durch externe Anwendungen zugreifbar zu machen, wird in Kapitel 6 die notwendige Softwareinfrastruktur beschrieben. Diese erleichtert den Einsatz des Modells, in dem es Schnittstellen für die wichtigsten Aufgaben wie das initiale Setup, die Administration und die Beantwortung von Anfragen zur Verfügung stellt. Zudem entstanden im Verlaufe dieser Arbeit einige Hilfswerkzeuge für die Bearbeitung von Ontologien, die kurz vorgestellt werden.

In Kapitel 7 wird das in dieser Arbeit entwickelte Zugriffskontrollsystem basierend auf den initialen Zielsetzungen evaluiert. Positive Aspekte und Probleme, die sich durch die Verwendung semantischer Technologien ergeben, werden aufgezeigt und deren Bedeutung für die praktische Verwendbarkeit der entwickelten Lösung beschrieben.

Das abschließende Kapitel 8 fasst die im Verlauf dieser Arbeit gewonnen Erkenntnisse zusammen und bewertet die Verwendbarkeit des entstandenen Zugriffskontrollsystems in realen Anwendungen. Im Verlauf dieser Arbeit gewonnene Erfahrungen bei der Modellierung komplexer Systeme durch semantische Technologien in Form von Ontologien und Regeln werden zusammenfassend dargestellt. Weiterhin wird ein Ausblick auf weitere interessante Aspekte des semantischen Rechtemanagements gegeben, deren Realisierung jedoch den Rahmen dieser Arbeit sprengen würde.

Im Anhang wird die Erweiterbarkeit des Zugriffskontrollsystems durch exemplarische Umsetzung der UNIX Zugriffskontrolle unter Beweis gestellt. Abschnitt B des Anhangs erklärt theoretische Grundlagen zur Stratifikation von nichtmonotonen Programmen.

Kapitel 2

Grundlagen

Dieses Kapitel vermittelt die grundlegenden Kenntnisse, die zum Verstehen dieser Arbeit notwendig sind. In Abschnitt 2.1 werden die verwendeten semantischen Technologien in Form von Ontologien und Regeln, die zur Modellierung des Zugriffskontrollsystems verwendet werden, vorgestellt. Abschnitt 2.3 befasst sich mit der Beschreibung allgemeiner Eigenschaften der Zugriffskontrolle und beschreibt gängige Zugriffskontrollsysteme. Abschließend werden in Abschnitt 2.4 Arbeiten, die Ansätze der semantischen Modellierung von Zugriffskontrollsystemen verfolgen, beispielhaft vorgestellt.

2.1 Ontologien

Der Begriff „Ontologie“ hat seinen Ursprung in der Philosophie und bezeichnet dort die *„Lehre vom Sein, von den Ordnungs-, Begriffs- und Wesensbestimmungen des Seienden“* [Dud94].

Eine Ontologie in der Informatik ist nach Gruber [Gru93] „*eine explizite Spezifikation einer gemeinsamen Konzeptualisierung*“. Diese Definition wurde später durch Borst [Bor97] ergänzt zu: *„Ontologien werden als formale Spezifikation einer gemeinsamen Konzeptualisierung definiert.*“

Dass Ontologien ein gängiges Mittel zur Modellierung von Wissensbasen im Wissensmanagement sind, ergibt sich aus der Definition von Swartout et al. [SPKR96]: „*Eine Ontologie ist eine hierarchisch strukturierte Menge von Ausdrücken zur Beschreibung einer Domäne, die als Grundgerüst für eine Wissensbasis dienen kann.*“

Uschold und Grüninger [UnGr96] nennen drei verschiedene Haupteinsatzgebiete von Ontologien:

- Kommunikation zwischen Anwendungen
- Automatisches Schließen (Reasoning)
- Repräsentation und Wiederverwendung von Wissen

2.1.1 Bestandteile von Ontologien [GLC04]

Es gibt verschiedene Beschreibungssprachen für Ontologien, die sich in ihrer Ausdrucksmächtigkeit unterscheiden. Für weitere Details sei an dieser Stelle auf [Str03] verwiesen.

Unabhängig von einer konkreten Beschreibungssprache besteht eine Ontologie nach Gruber [Gru93] aus einer Menge von Bausteinen, die zur Begriffsbestimmung nun kurz erläutert werden. Da die Bezeichnungen nicht eindeutig sind, werden mögliche Synonyme in Klammern hinzugefügt.

Klassen (Konzepte, Entitäten, Typen): Eine Klasse beschreibt eine Menge von Objekten die gemeinsame Eigenschaften aufweisen.

Instanzen (Objekte): Instanzen sind konkrete Ausprägungen von Klassen. So wäre beispielsweise `Herr Müller` eine Instanz der Klasse `Person`.

Relationen (Beziehungen): Relationen modellieren Beziehungen zwischen Klassen einer Domäne. Ontologien bestehen typischerweise aus binären Relationen, wobei das erste Argument den Definitionsbereich (engl.: domain) und das zweite Argument den Bildbereich (engl.: range) bestimmt. Die binäre Relation `subClassOf` definiert beispielsweise Vererbungsbeziehungen zwischen Klassen und wird für den Aufbau von Taxonomien verwendet.

Funktionen: Dies sind spezielle Relationen, deren Zielbereich nur genau ein Element enthalten kann. Übertragen auf Konzeptinstanzen bedeutet dies, dass eine funktionale Relation höchstens einmal referenziert werden kann. Der Vater einer Person könnte beispielsweise als funktionale Relation definiert werden.

Eigenschaften (Attribute): Attribute sind spezielle binäre Relationen, deren Bildbereich auf einfache Datentypen wie z.B. Zeichenketten, Ganzzahlen oder Wahrheitswerte eingeschränkt ist. Einfache Eigenschaften von Konzepten, wie z.B. der `Name` einer `Person`, lassen sich mittels Attributen modellieren.

Axiome: Nach Gruber [Gru93] werden Axiome verwendet, um Ausdrücke zu modellieren die stets wahr sind. Axiome beschreiben Wissen, die nicht formal mit den übrigen Elementen einer Ontologie beschrieben werden können. Auch zur Konsistenzprüfung von Ontologien werden Axiome verwendet. Ein Beispiel für die Verwendung von Axiomen wäre die Definition von Einschränkungen auf Relationen und Attributen, mittels derer sich beispielsweise die maximale Anzahl von Referenzen eines Objekts festlegen lassen. Auch die in Abschnitt 2.2 vorgestellten Regeln lassen sich den Axiomen zuordnen.

In dieser Arbeit wird zur Modellierung des Modells der Zugriffskontrolle die Web Ontology Language (OWL) verwendet. Die wichtigsten Eigenschaften von OWL werden im nächsten Abschnitt vorgestellt.

2.1.2 Web Ontology Language (OWL)

Wie der Name *Web Ontology Language* (OWL) [OWL04] bereits vermuten lässt, ist OWL eine konkrete Ontologiesprache mit besonderem Augenmerk auf der Verwendung im World Wide Web, beziehungsweise dem Semantic Web. Seit Februar 2004 ist OWL eine Empfehlung des World Wide Web Konsortiums (W3C). OWL ging historisch betrachtet aus der Ontologiesprache DAML+OIL hervor. Technisch gesehen ist OWL eine Anwendung von

RDF (Ressource Description Framework) und RDFS (RDF Schema). Unter anderem erweitert OWL die Ausdrucksmächtigkeit von RDFS um Beziehungen zwischen Klassen (z.B. Disjunktion), Kardinalitäten, Gleichheitsbeziehungen, zusätzliche Datentypen und Eigenschaften von Relationen wie z.B. Symmetrie oder Transitivität.

OWL unterscheidet drei zunehmend ausdrucksmächtigere Teilsprachen. Ziel dieser Unterteilung ist es die Entwicklung von Werkzeugen, die beispielsweise automatisches Schließen auf Wissensbasen ermöglichen, zu vereinfachen.

OWL Lite: OWL Lite stellt eine einfach zu implementierende Untermenge von OWL dar und kann zur Beschreibung einfacher Taxonomien mit einfachen Beschränkungen eingesetzt werden.

OWL DL: Die in OWL DL definierten Einschränkungen garantieren die Entscheidbarkeit von Schlussfolgerungen auf der durch die Ontologie beschriebenen Wissensbasis. DL steht hierbei für Description Logic (Beschreibungslogik). Die Beschreibungslogik ist eine entscheidbare Untermenge der Prädikatenlogik erster Stufe.

OWL Full: Dies ist die mächtigste Sprachebene von OWL. Durch die erweiterte Ausdrucksmächtigkeit ist die Entscheidbarkeit jedoch nicht mehr garantiert.

Obwohl OWL eine sehr neue Sprache ist, findet sie bereits breite Anwendung und wird von manchen [Fen04] als Ontologiesprache der Zukunft angesehen.

2.2 Regeln

Ontologiesprachen wie OWL weisen verschiedene Einschränkungen auf. Die Ausdrucksmächtigkeit von OWL reicht beispielsweise nicht aus um Verknüpfungen von Fakten zur Herleitung neuer Fakten beschreiben zu können. Ontologien repräsentieren statisches Wissen und können Abläufen, wie diese in Programmiersprachen dargestellt werden, nicht beschreiben. Die Anwendung von Regeln auf Ontologiesprachen erlaubt es, Konzepte über die explizit vorhandenen Relationen von Ontologiesprachen hinaus miteinander in Beziehung zu setzen. Logik zur Interpretation des Modells kann durch Regeln in Ontologien eingebracht werden.

Regeln bestehen typischerweise aus einer Schlussfolgerung zwischen einem Rumpf (engl.: body) und einem Kopf (engl.: head). Gelten die Angaben im Rumpf, können die im Regelkopf spezifizierten Information abgeleitet werden. Durch diesen einfachen Formalismus erlauben Regeln automatisches Schließen (engl. Reasoning bzw. Inferencing) auf Ontologien.

$\forall$ `x,y,z: hasGrandfather(?x,?z)` $\leftarrow$ `hasFather(?x,?y)` $\wedge$ `hasFather(?y,?z)`

Regel 2.1: Einfache Regel zur Beschreibung von Verwandschaftsverhältnissen.

Ein anschauliches Beispiel für den Einsatz von Regeln stellt die Formalisierung von Familienbeziehungen dar. Enthält die Ontologie beispielsweise die Information über den Vater einer Person, kann durch eine Regel, die den Vater des Vaters dieser Person sucht, der Großvater bestimmt werden. Dies wird durch Regel 2.1 formalisiert. Somit kann durch Regeln auf einfache Art und Weise neues Wissen durch Verknüpfung bereits existierender Informationen gewonnen werden.

Abbildung 2.1 veranschaulicht die Einordnung von OWL und Regeln im „Semantic Web Stack“ [Her04].

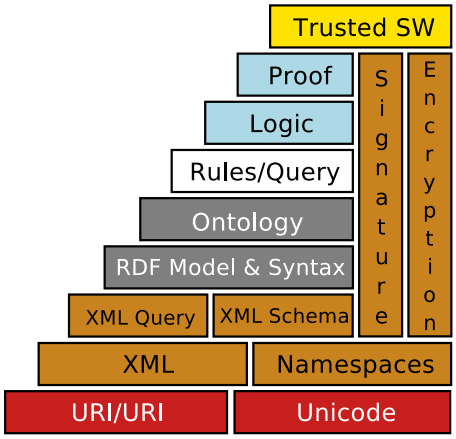

Abbildung 2.1: Einordnung von Ontologien und Regeln im „Semantic Web Stack“.

Regeln erweitern die formale Spezifikation einer Ontologie um Wissen, die für die „intelligente“ Auswertung der Daten in einer Ontologie notwendig sind. Ontologiesprachen definieren bereits eine Menge von Beziehungen, wie z.B. `subClassOf` Beziehungen, mit denen sich Auswertungen von Taxonomien durchführen lassen. Regeln basierend auf Ontologien erlauben es, beliebige Ontologieelemente miteinander in Beziehung zu setzen und erweitern durch diese kombinierte Auswertung die Mächtigkeit des automatischen Schließens. Regeln sind somit ein bedeutender Schritt hin zur Vision des für Maschinen verständlichen Semantic Web. Vermutlich wird die effiziente und skalierbare Auswertung von Regeln für den Erfolg des Semantic Web ausschlaggebend sein.

2.2.1 Semantic Web Rule Language (SWRL)

Mit der Semantic Web Rule Language [SWRL04] arbeitet der W3C derzeit an einer Standardisierung einer Regelsprache. SWRL basiert auf einer Kombination der OWL-DL / OWL-Lite Sprachen und der Rule Markup Language. SWRL erweitert OWL um eingeschränkte Regeln der Prädikatenlogik erster Stufe und sogenannte Built-Ins. Rumpf und Kopf einer SWRL Regel können Konjunktionen folgender Ausdrücke enthalten:

OWL Klassenatome: `C(x)` wobei `C` ein OWL Konzept beschreibt.

OWL Relationen: `P(x,y)` wobei `P` eine OWL Relation beschreibt.

Built-in Prädikate: `sameAs(x,y)`, `differentFrom(x,y)`,...

x und y können hierbei Variablen, OWL-Instanzen oder Elemente einer konkreten OWL-Domäne sein. Built-Ins existieren beispielsweise für Zeichenketten und Zahlen.

Sowohl der Rumpf als auch der Kopf einer SWRL-Regel können leer sein. Konjunktionen im Kopf und Disjunktionen im Rumpf einer Regel sind erlaubt. Aufgrund der Struktur der SWRL-Regeln lassen sich diese in Hornklauseln umformen. Hornklauseln besitzen die folgende Struktur:

$$H \leftarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n$$

Da für die Auswertung von Hornklauseln effiziente Algorithmen existieren, bringt die Überführbarkeit von SWRL-Regeln in Hornklauseln große Vorteile. SWRL zeichnet sich durch eine einfache und klare Semantik aus und unterstützt die nahtlose Integration von Regeln in OWL. Die Erweiterung von OWL-DL um SWRL-Regeln ist jedoch nicht entscheidbar.

2.2.2 DL-safe Rules

Die Grundidee von *DL-safe Rules* ist es, SWRL so zu beschränken, dass die Entscheidbarkeit gewährleistet ist. Im Allgemeinen bestehen *DL-safe Rules* aus beliebigen Hornklauseln, deren Literale sogenannten *DL-Atomen* entsprechen. Im Kontext von OWL beschreiben *DL-Atome* Konzepte und Relationen. Diese Hornklauseln müssen jedoch *DL-safe* sein. Dies bedeutet, dass jede Variable eines Regelkopfs in einem positiven *Nicht-DL-Ausdruck* im Rumpf enthalten sein muss. Ein *Nicht-DL-Ausdruck* beschreibt ein Prädikat beliebiger Stelligkeit. Diese Einschränkung ermöglicht, dass jede Variable durch Konstanten gebunden wird die explizit in der Wissensbasis vorhanden sind und garantiert somit die Entscheidbarkeit des logischen Programms.

Um die Benutzung von *DL-safe Rules* zu vereinfachen wird *DL-Safeness* durch automatische Ergänzungen erzwungen. Regel 2.1 ist nicht DL-safe, da die Variabeln x, y und z nicht in *Nicht-DL-Ausdrücken* im Rumpf der Regel enthalten sind. Um *DL-Safeness* zu erzwingen wird diese Regel entsprechend Regel 2.2 erweitert. Das Faktum O(a) wird für alle Konzeptinstanzen a zur Ontologie hinzugefügt. Somit werden die Regeln auf bekannte Individuen eingeschränkt. Für weitere Details sei der interessierte Leser an dieser Stelle auf [MSS04] verwiesen.

```
∀ x,y,z: hasGrandfather(?x,?z) ← hasFather(?x,?y) ∧
 hasFather(?y,?z) ∧ O(?x) ∧ O(?y) ∧ O(?z)
```

Regel 2.2: DL-safe Erweiterung von Regel 2.1.

2.2.3 Regeln in KAON2

KAON2 (KArlsruhe ONtology and semantic web tool suite 2) [KAON05] ist ein Softwarepaket zur Bearbeitung und Auswertung von Ontologien. KAON2 unterstützt unter anderem

eine API zur Verwaltung von OWL-DL und SWRL Ontologien und einen Reasoner zur Beantwortung von Anfragen.

Das Reasoning von KAON2 unterstützt die $\mathcal{SHIQ}(\mathbf{D})$ Untermenge von OWL-DL. Diese umfasst alle Eigenschaften von OWL-DL, außer Nominale (auch Enumerated Classes genannt).

Im Gegensatz zu den verfügbaren DL Reasonern wie FaCT[1], RACER[2] oder Pellet[3], die auf dem Tableaukalkül basieren, verwendet KAON2 einen neuartigen Algorithmus, der eine $\mathcal{SHIQ}(\mathbf{D})$ Wissensbasis in ein disjunktives Datalog Programm überführt. Dies bedeutet, dass eine OWL Wissensbasis vor dem automatischen Schließen in ein Datalog Programm transformiert wird. Aufgrund dieser Transformation können Regeln in Form von DL-safe Rules der Wissensbasis direkt hinzugefügt werden.

Darüber hinaus erweitern spezielle Eigenschaften von KAON2 die Möglichkeiten des automatischen Schließens auf OWL Ontologien:

Nicht-OWL-Prädikate: Die zuvor vorgestellten *Nicht-DL-Ausdrücke* werden in KAON2 als sogenannte *Nicht-OWL-Prädikate* (engl. Non-OWL-Predicates) bezeichnet. Diese Prädikate zeichnen sich durch eine beliebige Stelligkeit aus und können somit zur Beschreibung von Beziehungen zwischen mehreren Konzeptinstanzen verwendet werden. Ein konkretes Beispiel der Verwendung eines solchen Prädikats im Zugriffskontrollmodell stellt das Prädikat `accessGranted(?subject,?object,?operation, ?result)` dar. Dieses Prädikat vereint alle Informationen, die zur Beantwortung einer Zugriffsanfrage notwendig sind. Die Beantwortung von Zugriffsanfragen erfolgt durch einfache Deduktion des Prädikats `accessGranted`. Nach Ende der Berechnungen kann dem in der Variable `?result` gespeicherten Wahrheitswert entnommen werden, ob der Zugriff gestattet ist oder nicht.

Nichtmonotone Negation: Zudem unterstützt KAON2 auch nichtmonotones Schließen (engl.: Nonmonotonic Reasoning). Im Gegensatz zu monotonen Logiken, ist es beim nichtmonotonen Schließen möglich, Schlussfolgerungen bei Auftreten neuer Information zu revidieren. Eine Konklusion wird somit akzeptiert, sofern keine Information vorliegt, die diesem Schluss widerspricht. Nichtmonotones Schließen in KAON2 ist jedoch nur für DLP (Description Logic Programs) Fragmente möglich. DLP beschränkt OWL-Axiome auf Hornklauseln und verbietet somit insbesonders die Beschreibungen von Disjunktionen im Kopf einer Regel. Eine der wesentlichen Erweiterung ist die Beschreibung der nichtmonotonen Negation, die auch als *Negation as failure* bezeichnet wird. Hierbei handelt es sich um einen Mechanismus, der beispielsweise in Prolog verwendet wird. Zur Berechnung einer Negation not(p) versucht man zu zeigen, das p gilt. Ist dies erfüllt, wird not(p) zu falsch ausgewertet. Umgekehrt gilt not(p), falls die Ausführung von p fehlschlägt. Bei *Negation as Failure* schlägt eine Ausführung fehl, wenn keine Information über p in der Wissensbasis enthalten ist. Voraussetzung zur Berechnung der nichtmonotonen Negation ist, dass das zugrunde liegende logische Programm stratifiziert ist (vgl. Abschnitt B). Die Umsetzung der

[1] http://www.cs.man.ac.uk/~horrocks/FaCT
[2] http://www.racer-systems.com
[3] http://www.mindswap.org/2003/pellet/index.shtml

Nichtmonotonen Logik in KAON2 ist noch in einem experimentellen Stadium und sollte mit Bedacht eingesetzt werden.

Im Zugriffskontrollmodell wird die nichtmonotone Negation zur Beantwortung von Zugriffsanfragen verwendet. Durch nichtmonotone Negation lässt sich prüfen, ob eine gewünschte Eigenschaft nicht gilt. Liegt beispielsweise für eine Anfrage keine positive Berechtigung vor, d.h. es kann keine Information in der Wissensbasis dazu gefunden werden, wird der Zugriff verboten. Ohne nichtmonotone Negation könnte ein solcher Sachverhalt nicht formuliert werden.

Beide Eigenschaften erhöhen die Mächtigkeit des automatischen Schließens und sind für die Beantwortung von Zugriffsanfragen im Zugriffskontrollsystem von entscheidender Bedeutung. Mit KAON2 als Reasoner lässt sich die Logik zur Beantwortung von Berechtigungsanfragen ohne zusätzlichen Programmcode direkt durch Regeln beschreiben.

2.3 Zugriffskontrollsysteme

Seit den Anfängen von Mehrbenutzer-Computersystemen, in denen mehrere Benutzer Zugriff auf eine Menge von Ressourcen haben, ist Zugriffskontrolle Gegenstand der Forschung. Zugriffskontrolle wird in [DoD88] sehr allgemein definiert als:

The process of limiting access to the resources of a system only to authorized programs, processes or other systems (in a network).

Nach [San94] ist der Zweck der Zugriffskontrolle die Operationen, die ein berechtigter Benutzer eines Computersystems ausführen darf, einzuschränken. Diese Beschränkungen zielen sowohl auf Aktionen, die der Benutzer direkt ausführen kann, als auch auf Aktionen, die Programme im Auftrag des Benutzers durchführen. Allgemein versucht Zugriffskontrolle unautorisierte Aktivitäten zu unterbinden.

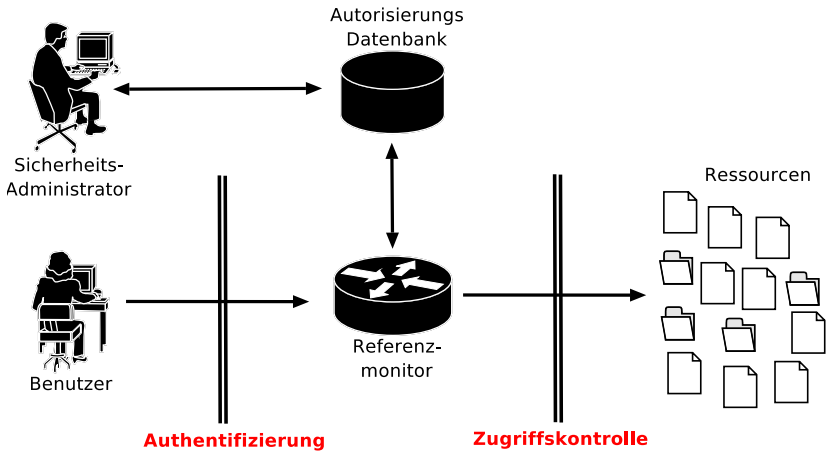

Abbildung 2.2: Interaktion von Zugriffskontrolle und anderen Sicherheitsdiensten in einem Computersystem.

Zugriffskontrolle alleine kann diese Aufgaben nicht bewerkstelligen, sondern arbeitet zusammen mit anderen Sicherheitsdiensten eines Computersystems. Abbildung 2.2 verdeutlicht dieses Zusammenspiel. Die Authentifizierung überprüft, ob der Benutzer tatsächlich derjenige ist, der er vorgibt zu sein. Zugriffskontrolle wird durch einen Referenzmonitor erzwungen, der jede Aktion des Benutzers überwacht. Der Referenzmonitor befragt hierzu den Autorisierungsdienst, um festzustellen ob der Benutzer die von ihm gewünschte Aktion ausführen darf. Der Autorisierungsdienst wird von einem Sicherheitsadministrator administriert, der Benutzerrechte entsprechend der Sicherheitsstrategien einer Organisation vergibt. Auch der Benutzer kann auf den Autorisierungsdienst Einfluss nehmen, um z.B. Berechtigungen auf eigenen Dateien weiterzugeben. Nur wenn die Autorisierung erfolgreich war, erlaubt der Referenzmonitor dem Benutzer den Zugriff.

Es ist wichtig zu verstehen, dass Zugriffskontrolle und Authentifizierung voneinander unabhängige Vorgänge sind. Der Referenzmonitor geht davon aus, dass eine Überprüfung der tatsächlichen Benutzeridentität vor der Autorisierungsanfrage stattfand.

Im Folgenden werden, aufbauend auf den Definitionen des „Orange Book“ des amerikanischen Verteidigungsministeriums, die grundlegenden Konzepte eines Zugriffskontrollsystems erklärt [Nus98]. Im Mittelpunkt der Zugriffskontrolle steht die Anfrage von *Subjekten*, die eine *Operation* auf einem bestimmen *Objekt* durchführen wollen. Dies ist in Abbildung 2.3 schematisch dargestellt.

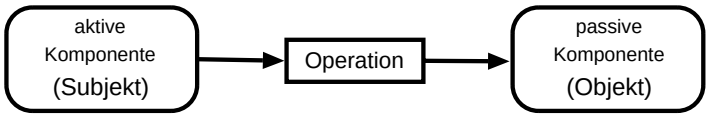

Abbildung 2.3: Zentrale Elemente der Modellierung einer Zugriffskontrolle.

Bei einem Subjekt handelt es sich hierbei um einen Benutzer, einen Prozess oder einen Agenten eines Benutzers. Unter einem Objekt versteht man im Kontext heutiger Zugriffskontrollsysteme ein beliebiges Datenobjekt oder eine Systemressource. Beispiel für Operationen wären das Schreiben einer Datei oder das Lesen eines Datensatzes aus einer Datenbank.

Bei der Realisierung eines Zugriffskontrollsystems müssen zwei Aspekte betrachtet werden:

1. **Autorisierung:** Die Autorisierung beschreibt die eigentliche Nutzung des Zugriffskontrollsystems. Bevor ein Nutzer auf eine Ressource zugreifen darf, muss überprüft werden, ob dieser die gewünschte Operation auf dem Zielobjekt ausführen darf. Dies wird anhand von Zugriffskontrollinformationen des Subjekts bzw. Objekts entschieden.

2. **Administration:** Alle Aktionen, die Zugriffskontrollinformationen in ein System einbringen, werden unter dem Begriff der Administration zusammengefasst. Beispiele für administrative Vorgänge wären das Hinzufügen neuer Berechtigungen oder neuer Subjekte.

2.3.1 Zugriffskontrollinformationen

Zugriffskontrollinformationen beschreiben, ob ein Subjekt zur Ausführung einer bestimmten Operation auf einem Objekt befugt ist. Diese Information kann in Form einer Matrix dargestellt werden, deren Zeilen die Subjekte und Spalten die Objekte enthalten. Jedes Matrixelement enthält die Operationen, die das Subjekt auf dem jeweiligen Objekt ausführen darf. Diese Matrix wird als **Zugriffskontrollmatrix** bezeichnet. Tabelle 2.1 stellt eine einfache Zugriffskontrollmatrix dar.

	Datei 1	**Datei 2**
Bob	Own Read Write	
Alice	Read	Own Read Write

Tabelle 2.1: Beispiel einer Zugriffskontrollmatrix.

Da jedes Subjekt i.A. nur auf eine geringe Menge der verwalteten Objekte zugreifen darf, sind Zugriffskontrollmatrizen sehr groß und nur spärlich besetzt. Dadurch werden diese Matrizen in praktischen Anwendungen in Form von Zugriffskontrolllisten bzw. Fähigkeitslisten komprimiert.

Zugriffskontrolllisten (engl.: Access Control List (ACL)) repräsentieren die Spalten einer Zugriffskontrollmatrix. Jedes Objekt ist verbunden mit einer Liste von Paaren, wobei jedes Paar für ein spezifisches Subjekt alle Rechte auf dem jeweiligen Objekt definiert. Abbildung 2.4 veranschaulicht die Abbildung der Zugriffskontrollmatrix 2.1 auf eine Zugriffskontrollliste.

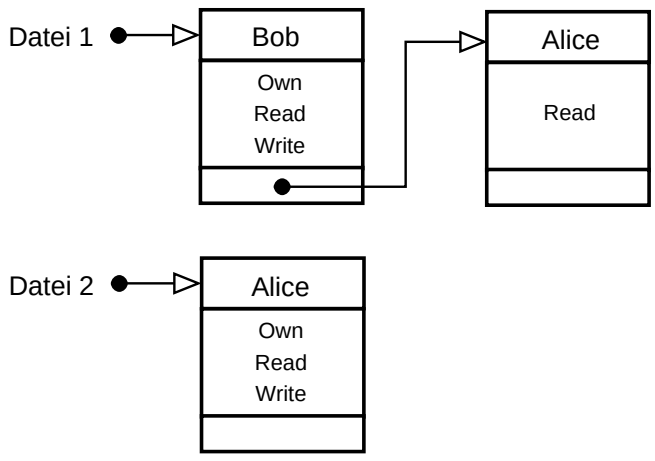

Abbildung 2.4: Verwendung von Zugriffskontrolllisten als Zugriffskontrollinformation.

Bei **Fähigkeitslisten** (engl.: Capabilities) werden Zeilen der Zugriffskontrollmatrix abgebildet. Jedes Subjekt ist mit einer Liste verbunden, die für jedes Objekt auf dem der Benutzer eine Operation ausführen darf diese Operation spezifiziert. Abbildung 2.5 zeigt exemplarisch die Verwendung von Fähigkeitslisten.

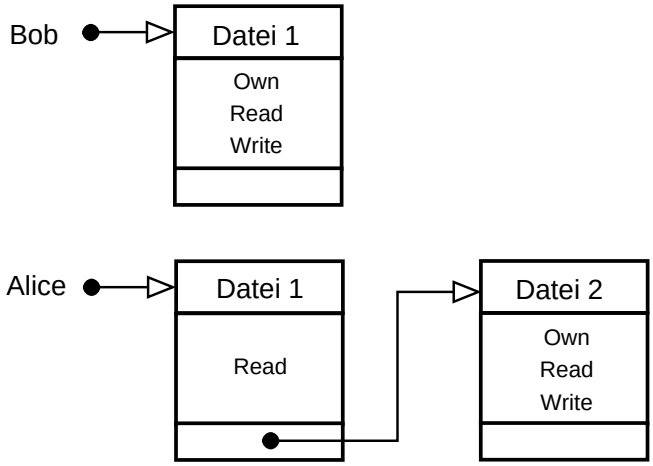

Abbildung 2.5: Verwendung von Fähigkeitslisten als Zugriffskontrollinformationen.

Obwohl in den 70er Jahren einige Computersysteme mit Fähigkeitslisten implementiert wurden, konnten sich diese nicht durchsetzen. Heute überwiegen ACL Implementierungen.

Sowohl Zugriffskontrolllisten als auch Fähigkeitslisten haben einen großen Nachteil bei der Suche nach Elementen, die nicht direkt referenziert werden. So ist es relativ einfach in Zugriffskontrolllisten für jedes Objekt die Menge aller Subjekte mit entsprechendem Zugriff zu erhalten. Viel komplizierter ist jedoch die Suche nach allen Objekten auf die ein Subjekt zugreifen darf, wie dies z.B. beim Löschen eines Subjektes erforderlich werden kann. Dies erfordert ein vollständiges Durchsuchen der Zugriffskontrolllisten aller Objekte und ist dementsprechend ineffizient. Ähnliches gilt für Fähigkeitslisten.

Darüber hinaus gibt es noch andere Arten der Zugriffskontrollinformationen. Ein Beispiel hierfür sind **Markierungen** (engl.: Labels), die jedem Subjekt und Objekt Sicherheitsstufen zuweisen. Durch Vergleich dieser Stufen kann dann ein Zugriff erlaubt werden oder nicht. Auch der **Kontext** eines Zugriffs kann für eine Anfrage entscheidend sein, wenn z.B. der Zugriff nur zu einer gewissen Tageszeit gestattet ist.

Das allgemeine Ziel unterschiedlicher Repräsentationen von Zugriffskontrollinformationen liegt in der Erleichterung der Administration und der Reduktion der zu verwaltenden Daten. Dieses Ziel wird auch mit der Definition von Generalisierungsmechanismen unterstützt. Beispielsweise können häufig Subjekte zu Gruppen oder Rollen aggregiert werden, so dass nicht mehr jedem Subjekt einzeln sondern einer Gruppe von Subjekten das Recht zur Ausführung einer entsprechende Operationen zugewiesen werden kann. Auch auf Objektseite kann eine solche Aggregation z.B. in Form von Verzeichnissen stattfinden.

2.3.2 Modelle der Zugriffskontrolle in der Literatur [San94, Nus98]

In der Literatur werden im Wesentlichen drei Referenzmodelle unterschieden, die in praktischen Systemen auch Anwendung finden. Man unterscheidet zwischen wahlfreier (engl.: discretionary), regelbasierter (engl.: mandatory) und rollenbasierter (engl.: role based) Zugriffskontrolle. Die Anforderungen an die wahlfreie und regelbasierte Zugriffskontrolle wurden vom amerikanischen Verteidigungsministerium in den „Trusted Computer System Evaluation Criteria (TCSEC)“ definiert. Rollenbasierte Zugriffskontrollmodelle haben hingegen

in den letzten Jahren nicht nur in der wissenschaftlichen Literatur, sondern auch in praktischen Anwendungen große Beachtung gefunden.

Es ist wichtig anzumerken, dass die hier vorgestellten Referenzmodelle nicht als disjunkte Modelle zu betrachten sind, da sich Eigenschaften dieser Modelle durchaus kombinieren lassen. Zudem sind diese Modelle nicht erschöpfend, was Abbildung 2.6 vor Augen führt.

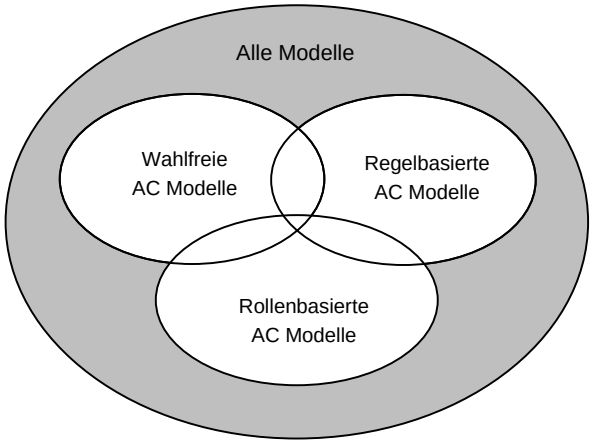

Abbildung 2.6: Inklusionsbeziehungen der Referenzmodelle zur Zugriffskontrolle.

Wahlfreie (Discretionary) Zugriffskontrolle: Discretionary bedeutet übersetzt „beliebig“. Bezogen auf die Zugriffskontrolle bedeutet dies, dass für jedes Subjekt und jedes Objekt im System die zulässigen Zugriffsarten vom Eigentümer des Objekts nach eigenem Ermessen vergeben werden. Dieses Modell beruht somit auf den folgenden Grundannahmen:

- Jedes Objekt hat einen Eigentümer. Hierbei handelt es sich typischerweise um das Subjekt, welches das Objekt erzeugt. Der Eigentümer eines Objekts kann jedoch durch entsprechende Weitergaben geändert werden.
- Der Eigentümer eines Objekts kann nach freiem Ermessen anderen Subjekten Zugriff auf dieses geben.

Bei jedem Zugriff wird überprüft, ob das Subjekt eine Berechtigung zur Ausführung dieser Operation auf dem entsprechenden Objekt hat. Ist dies der Fall, wird der Zugriff gestattet, sonst abgelehnt.

Die Kontrolle des Informationsflusses zwischen Subjekten ist durch diese Art der Zugriffskontrolle nicht gewährleistet. So kann die Zugriffskontrolle umgangen werden, in dem ein Subjekt von einem Objekt, auf dem er Leserechte besitzt, eine lokale Kopie erstellt. Da er nun Eigentümer dieser Kopie ist, kann er das Leserecht an andere Subjekte weitergeben. Somit können Leserechte ohne Wissen des Objektbesitzers weitergegeben werden.

In der Praxis wird dieses eigentumsbasierte Modell, erweitert um die Möglichkeit der Aggregation von Subjekten zu Subjektgruppen, in Unix Systemen eingesetzt. Zugriffsrechte werden in Unix mittels Zugriffskontrolllisten für jedes Objekt definiert, wie dies in Abbildung 2.7 dargestellt ist.

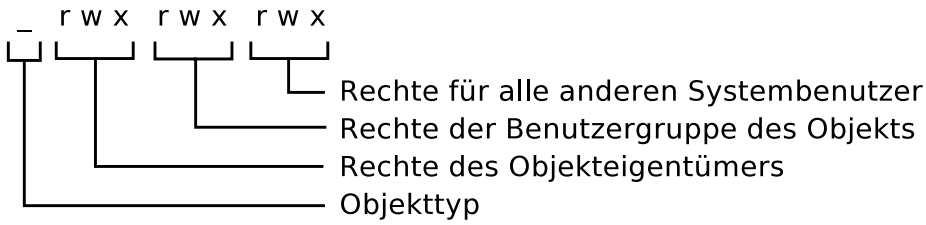

Abbildung 2.7: Zugriffskontrollliste in UNIX Systemen.

Unix unterscheidet nur die Rechte lesen (r), schreiben (w) und ausführen (x). Zugriffsrechte auf einem Objekt können für den Besitzer, die Benutzergruppe des Objekts und für alle anderen Benutzer des Systems vergeben werden. Durch den Befehl `chmod` kann der Besitzer Rechte an Objekten weitergeben und mit `chown` den Eigentümer des Objekts wechseln.

Regelbasierte (Mandatory) Zugriffskontrolle: Die Kontrolle von Informationsflüssen ist hingegen das erklärte Ziel regelbasierter Zugriffsmodelle. Der Ansatz stammt aus dem militärischen Bereich. Alle Subjekte und Objekte werden mittels einer Sicherheitsstufe klassifiziert. Diese Sicherheitsstufen legen den Grad der Vertrauenswürdigkeit eines Subjekts bzw. die Vertraulichkeit eines Objekts fest und sind hierarchisch aufgebaut. Beispiel für Sicherheitsstufen wären *TopSecret* > *Secret* > *Confidential*. Steht eine Sicherheitsstufe auf der linken Seite des „>“ Symbols, dominiert diese alle weiter rechts stehenden Stufen.

In regelbasierten Systemen werden Berechtigungen anhand von Beziehungen zwischen den Sicherheitsstufen von Objekten und Subjekten vergeben. Die Art der Beziehung hängt hierbei von der konkreten Operation ab, die auf einem Objekt ausgeführt werden soll. Typische Beispiel sind „No read up“ und „No write down“, wobei „No read up“ bedeutet, dass der Lesezugriff auf ein Objekt nur dann erlaubt ist, wenn die Sicherheitsstufe des Subjekts die des Objekts dominiert. „No write down“ erlaubt hingegen einen Zugriff nur dann, wenn die Klassifikation des Subjekts von der des Objekts dominiert wird. Die Einhaltung dieser Eigenschaften verhindert, dass Informationen von höheren Stufen auf niederere Stufen weitergeben werden können. Nach den Entwicklern der hier beschriebenen Zugriffskontrollfunktionen nennt man dieses Modell auch das *Modell von Bell und LaPadule*.

Ein Problem solcher Systeme ist nach [San94], dass ein Subjekt, welches als *Secret* spezifiziert ist, *TopSecret* Dokumente schreiben darf, obwohl es darauf keinen Lesezugriff hat. Hierdurch können Daten auf höheren Ebenen durch Überschreiben zerstört werden. Viele Systeme beschränken daher den Schreibzugriff ausschließlich auf Dokumente der Sicherheitsstufe des Subjekts. Die Regel „No write down“ bedeutet zudem, dass eine als *Secret* klassifizierte Person keine Dokumente der Ebene *Confidential* bearbeiten darf, ohne dass die Sicherheitsstufe des Dokument auf *Secret* angehoben wird. Um dies zu verhindern, kann ein Benutzer sich in unterschiedlichen Sicherheitsstufen am System anmelden, um die Berechtigungen zu erhalten, die abhängig von seiner Aufgabe gerade benötigt werden.

Rollenbasierte (Role Based) Zugriffskontrolle: Ravo Samji et al. entwickelten in den neunziger Jahren ein mehrstufiges rollenbasiertes Zugriffskontrollmodell, das unter dem Titel RBAC96 [RBAC96] veröffentlicht wurde. Zentrales Konzept dieses Modells ist eine **Rolle**. Eine Rolle repräsentiert einen typischen Aufgabenbereich innerhalb einer Organisation mit den zugehörigen Zugriffsrechten. Nach [San96] ist eine Rolle *„eine benannte Ansammlung von Benutzern und Berechtigungen“*.

Die direkte Zuweisung von Berechtigungen an Rollen erleichtert die Administration, da gerade in Unternehmen sich die Zuordnung eines Subjekts zu einem Verantwortungsbereich häufiger ändert als die Verantwortungsbereiche und die damit verbundenen Zugriffsrechte selbst.

Typische hierarchische Organisationsstrukturen, wie sie in vielen Organisationen heute zu finden sind, können durch hierarchische Rollenbeziehungen repräsentiert werden. Hierbei werden Zugriffsberechtigungen von den allgemeinen an spezialisiertere Rollen vererbt. Abbildung 2.8 stellt ein typisches Beispiel einer Rollenhierarchie dar.

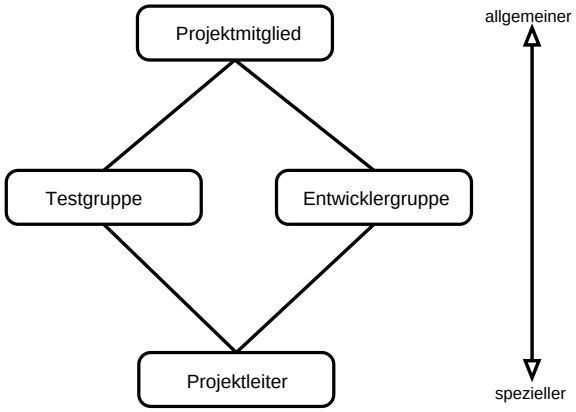

Abbildung 2.8: Einfaches Beispiel eines hierarchischen Rollenmodells.

Es besteht die Möglichkeit auf Rollen Beschränkungen (engl.: Constraints) zu definieren. Diese Beschränkungen erlauben beispielsweise Rollen als disjunkt zueinander zu beschreiben, so dass ein Subjekt das Mitglied einer Rolle ist nicht einer anderen Rolle zugewiesen werden kann. Somit kann man eine gewünschte Zuständigkeitstrennung erreichen. Zudem können Rollen als voneinander abhängig definiert werden. Einem Subjekt kann eine abhängige Rolle nur dann zugewiesen werden, wenn es bereits Mitglied der als abhängig definierten Rolle ist.

Aufgrund der Eigenschaften rollenbasierter Systeme werden diese vor allem in Systemen großer Organisationen eingesetzt. SAP verwendet ein erweitertes rollenbasiertes Modell zur Implementierung der Zugriffskontrolle in den Systemen SAP R/3 und SAP Enterprise Portal [SAP03].

Eine detaillierte Betrachtung der individuellen Eigenschaften und Gemeinsamkeiten dieser Referenzmodelle findet in Kapitel 4 im Zuge der Modellierung eines allgemeinen Modells für ein Zugriffskontrollsystem statt.

2.4 Stand der Technik

In diesem Abschnitt werden Ansätze aus dem Gebiet der semantischen Zugriffskontrollmodellierung exemplarisch dargestellt. Die beiden hier vorgestellten Arbeiten beschreiben Teilaspekte von Zugriffskontrollsystemen.

2.4.1 eXtensible Access Control Markup Language (XACML)

Die *eXtensible Access Control Markup Language* (XACML) [OAS05] ist eine XML basierte Sprache zur Beschreibung und zum Austausch von Zugriffskontrollstrategien zwischen Anwendungen. Diese Strategien werden in sogenannten „Policies“ spezifiziert, deren Auswertung den Zugriff von Subjekten auf Ressourcen gestattet oder verbietet. XACML besitzt seit 2003 den Status eines OASIS[1] Standards. Mehrere Firmen, darunter Entrust Inc., IBM, Sun und BEA Systems Inc. sind an der Entwicklung von XACML beteiligt.

Eine der großen Neuerungen von XACML ist die anwendungsunabhängige Definition von Zugriffskontrollinformationen, so dass eine Sicherheitsstrategie nach einmaliger Spezifizierung für unterschiedliche Applikationen eingesetzt werden kann. Somit können Rechtemanagementstrategien an zentrale Stelle für das gesamte Unternehmen verwaltet werden.

Der Standard umfasst eine Sprache zur Spezifizierung von Zugriffskontrollstrategien (engl.: access control policy) und eine Sprache zur Beschreibung von Zugriffskontrollanfragen und -antworten.

Um die Unterschiede zwischen XACML und dem in dieser Arbeit entworfenen Modell zu verdeutlichen, wird der syntaktische Aufbau von Strategien und Anfragen kurz dargestellt.

```
<Policy RuleCombiningAlgId="permit-overrides">
    <Target/>
    <Rule Effect="Permit">
        <Target>
            <Subjects/>
        </Target>
    </Rule>
</Policy>
```

Abbildung 2.9: Syntax einer XACML Strategie.

Abbildung 2.9 stellt die vereinfachte Syntax einer XACML *Strategie* (engl.: Policy) dar. Mittels einer Strategie wird beschrieben, unter welchen Bedingungen ein Zugriff zu gestatten ist. Gegebenenfalls können für eine Anfrage mehrere *Strategien* zutreffend sein. Um die Ergebnisse mehrerer auszuführender *Strategien* zu kombinieren, wird mit `RuleCombiningAlgId` ein Algorithmus referenziert, der die Ergebnisse miteinander kombiniert. In `<Target/>` wird festgelegt, welche Eigenschaften eine *Subjekt*, eine *Ressource* oder ein *Objekt* einer *Anfrage* zu erfüllen hat, damit eine *Strategie* angewendet werden kann. `<Rule Effect="Permit">` beschreibt die Auswirkungen einer Regel. Im obigen Beispiel handelt es sich dabei

[1] Organization for the Advancement of Structured Information Standards (http://www.oasis-open.org)

um die Gestattung eines Zugriffs. Jede *Strategie* kann mehrere *Regeln* umfassen. Das Element `<Target>` in `<Rule>` bestimmt, welche Eigenschaften die Elemente einer *Regel* haben müssen, damit diese ausgeführt wird. `<Subjects/>` beschreibt notwendige Eigenschaften von *Subjekten* einer *Regel*. Ob ein Element einer *Anfrage* die entsprechenden Eigenschaften erfüllt, wird durch Ausführung einer referenzierten Funktion überprüft, die die in der *Strategie* definierten Eigenschaften mit den Elementen der Anfrage vergleicht.

Eine Autorisierungsanfrage zur Zugriffskontrolle referenziert Subjekte, Objekte und Aktionen und wird in XACML, wie in Abbildung 2.10 dargestellt, formuliert.

```
<Request>
    <Subject> <Attribute/> </Subject>
    <Resource> <Attribute/> </Resource>
    <Action> <Attribute/> </Action>
    <Environment/>
<Request>
```

Abbildung 2.10: Syntax einer XACML Anfrage.

Mittels der im System definierten *Strategien* wird über eine Berechtigung entschieden. Eigenschaften der Elemente werden in `<Attribut/>` der Anfrage mitübergeben und bei der Ausführung entsprechender *Regeln* mit den unter `Target` abgelegten Werten verglichen.

Durch Definition neuer Strategien ist XACML individuell anpassbar. Die Unterstützung zusätzlicher Kombinationsalgorithmen erfordert jedoch Programmieraufwand und auch neue Funktionen zum Vergleich von Attributen zwischen Anfragen und Strategien müssen individuell implementiert werden.

Seit kurzem ist der XACML Standard in der Version 2.0 verabschiedet. Unterschiedliche Implementierungen des XACML Standards sind verfügbar, unter anderem von Sun Microsystems [1]. Durch die Standardisierung und die vorliegenden umfassenden Implementierungen wird erwartet, dass XACML in der Praxis weite Verbreitung findet.

Der Schwerpunkt von XACML liegt in der Formulierung einer XML basierten Sprache zur Beschreibung von Zugriffskontrollstrategien und zugehörigen Anfrage- und Antwortsprachen. Die Interpretation dieser Strategien, Anfragen und Antworten erfolgt explizit durch Programmcode, der durch entsprechende Elemente der Sprache aufgerufen werden kann. Der in dieser Arbeit verfolgte wissensbasierte Ansatz versucht gerade diese Notwendigkeit von zusätzlichem Code zur Interpretation des Modells durch Regeln zu ersetzen und somit eine größere Flexibilität und einfachere Anpassbarkeit zu gewährleisten.

Bezüglich der Bereitstellung der Informationen zur Beantwortung einer Anfrage unterscheidet sich XACML von der in dieser Arbeit verwendeten Vorgehensweise. Bei XACML werden Attribute von Benutzern, Ressourcen und Aktionen zur Anfragezeit mitübergeben und durch entsprechende Auswertung von Funktionen mit den in der zugehörigen Strategie angegebenen Werten verglichen. Bevor eine Zugriffsanfrage hingegen in einem semantischen Modell durch Regeln beantwortet werden kann, muss das Wissen über die Eigenschaften der interessierenden Elemente explizit in der Ontologie abgelegt werden.

[1] http://sunxacml.sourceforge.net

2.4.2 Ein regelbasiertes XML Zugriffskontrollmodell

Ziel der in [Anu03] vorgestellten Arbeit ist die Beschreibung und Umsetzung eines Zugriffskontrollmodells für XML Dokumente. Das Modell erlaubt die Vergabe von Lese- oder Schreibberechtigungen für einzelne Elemente des Dokuments. Die Arbeit besteht im Wesentlichen aus der Beschreibung eines XML basierten Zugriffskontrollmodells und der Definition einer XML basierten Regelsprache namens *XML Declarative Description* (XDD).

Das Zugriffskontrollmodell ist sehr einfach aufgebaut. Das zentrale Element ist die Autorisierung, die als Quintupel

```
<subject, object, privilege, type, sign>
```

dargestellt wird. `subject` kann ein Benutzer, eine Benutzergruppe, eine Rolle oder andere Berechtigungsnachweise (engl.: Credentials), wie das Alter oder Geschlecht eines Benutzers sein. `object` beschreibt ein Element des XML-Dokuments, wobei hier XPath Ausdrücke zur Adressierung verwendet werden. Während in `privilege` die Operationen `read` oder `write` eingesetzt werden, erlaubt `type` die Definition von Weitergabestrategien. Diese legen fest, ob Autorisierungen auf der XML-Dokumentstruktur ausgehend von einem speziellen Element auf dessen Kindelemente übertragen werden. Berechtigungen können positiv oder negativ definiert werden, was in `sign` festgelegt wird. Durch eine Autorisierung wird somit festgelegt, ob ein Subjekt auf ein entsprechendes Element des XML Dokuments lesend oder schreibend zugreifen darf oder nicht.

Zudem werden Strategien definiert, die im Falle von Konflikten durch positive und negative Autorisierungen bzw. fehlende Informationen eine Zugriffsentscheidung herbeiführen.

Um dieses Zugriffskontrollmodell interpretieren zu können, wird *XDD* – eine XML basierte Regelsprache – definiert. Diese erweitert gewöhnliche XML Elemente um Verbindungen von XML Elementen, die zur Formalisierung sogenannter *XDD Beschreibungen* verwendet werden. Hierbei handelt es sich um eine Menge von XML Klauseln der Form:

$$H \leftarrow B_0, \ldots, B_n$$

H und B_i sind hierbei XML Ausdrücke, wobei H den Kopf und die B_i den Rumpf der Regel darstellen. Eine solche Klausel bedeutet, dass der Ausdruck H abgeleitet werden kann, wenn alle Ausdrücke im Rumpf der Regel erfüllt sind.

Abbildung 2.11 zeigt ein einfaches Beispiel einer solchen Regel. Diese gibt jedem Benutzer der Gruppe `Doctor` ein Schreibrecht auf einem `patient` Element des Dokuments `patients.xml`.

XDD ist eine reine Beschreibungssprache und bringt selbst keine Mechanismen zur Auswertung der beschriebenen Regeln mit. Die Auswertung von XDD erfolgt durch Anwendung des *Equivalent Transformation (ET)* Paradigmas. Dieses basiert auf semantikbewahrenden Transformationen, die einen formalisierten Ausdruck P solange in Ausdrücke $P_1, P_2, \ldots$ umformen, bis die gewünschte Aussage P_n abgeleitet ist. Basierend auf der XDD Formalisierung und dem ET Paradigma wurde die *XML Equivalent Transformation (XET)* Engine entwickelt. Wird eine Anfrage in Form einer XDD Beschreibung übergeben, interpretiert XET die Daten eines XML Dokuments anhand von XDD Regeln und leitet so eine Antwort ab.

Der Schwerpunkt der vorgestellten Arbeit liegt auf der Definition einer Regelsprache zur Beantwortung von Autorisierungsanfragen. Das eigentliche Zugriffskontrollmodell ist sehr schlicht und ausschließlich für die Verwendung auf XML-Dokumenten vorgesehen.

```
<Authorization>
    <privilege rdf:resource="#write"/>
    <sign rdf:resource="#plus"/>
    <subject rdf:resource="$S:user"/>
    <object>
        <Object>
            <level rdf:resource="#instance"/>
            <target rdf:resource="patients.xml"/>
            <path>
                <patient $P:patient/>
            </path>
        </Object>
    </object>
</Authorization>
←   <User rdf:about=$S:user>
        <group rdf:resource="#Doctor"/>
        $E:userProperties
    </User>
```

Abbildung 2.11: Beispiel einer XDD Regel zur Beschreibung einer Autorisierung.

Das in dieser Diplomarbeit erstellte Modell ist in der Lage, dieses speziell auf die XML-Dokumentstruktur ausgerichtete Zugriffskontrollmodell zu beschreiben. Die regelbasierte XML Zugriffskontrolle zeigt jedoch einige interessante Aspekte der Modellierung eines Autorisierungssystems auf und diente als wichtige Grundlage des zu erstellenden semantischen Modells. Aufgrund des Einsatzes semantischer Technologien ist der Einsatz einer speziellen Regelsprache wie *XD* nicht notwendig, da *DL-Safe Rules* Bestandteil des in dieser Arbeit verwendeten Reasoners sind.

Kapitel 3

Konzept

In diesem Kapitel wird das konzeptionelle Vorgehen der Realisierung einer Zugriffskontrollkomponente vorgestellt. Die Beschreibung eines Zugriffskontrollsystems in Form einer eigenständigen Komponente ermöglicht eine einfache und vielseitige Verwendbarkeit. Hauptvorteil einer komponentenbasierten Zugriffskontrolle ist die Beschreibung von unternehmensweiten Sicherheitsstrategien an zentraler Stelle, über Applikations- und Systemgrenzen hinweg. Änderungen des Rechtemanagements, z.B aufgrund von Anpassungen der Organisationsstruktur eines Unternehmens, können durch Modifikation der Zugriffskontrollkomponente zentral durchgeführt werden.

Damit diese Komponente vielseitig einsetzbar ist, muss sie in der Lage sein gängige Zugriffskontrollsysteme auf einfache Art und Weise umsetzen zu können. Organisationsspezifische Besonderheiten eines Rechtemanagementsystems sollten durch einfache Erweiterbarkeit des Systems beschreibbar sein. Ziel dieser Arbeit ist es, die Umsetzung solcher Erweiterungen ohne zusätzlichen Programmcode zu ermöglichen. Aufbauend auf semantischen Technologien wird hierzu ein Zugriffskontrollmodell entwickelt, dass in der Lage ist, Antworten auf Autorisierungsanfragen durch Auswertung der im Modell enthaltenen Fakten zu beantworten. Dieses Zugriffskontrollmodell beschreibt den Kern des eigentlichen Zugriffskontrollsystems. Um dieses semantische Modell in Form einer eigenständigen Komponente für externe Anwendungen verwendbar zu machen, werden entsprechende Schnittstellen in Form von APIs[1] zur Verfügung gestellt. Die Kombination aus Modell und programmatischen Schnittstellen realisiert das eigentliche Zugriffskontrollsystem.

Abschnitt 3.1 beschreibt allgemeine Grundlagen des Zugriffskontrollmodells basierend auf den zu Beginn formulierten Zielsetzungen. Die Beantwortung von Zugriffsanfragen erfolgt durch Regeln, die das ontologiebasierte Modell auswerten. Das allgemeine Zusammenspiel von Ontologien und Regeln wird in Abschnitt 3.2 vorgestellt. Die Architektur des entwickelten Zugriffskontrollsystems und die Interaktion der Zugriffskontrollkomponente mit externen Anwendungen beschreibt Abschnitt 3.3.

3.1 Das Zugriffskontrollmodell

Das Hauptaugenmerk dieser Arbeit liegt auf der Beschreibung von Zugriffskontrollsystemen durch ein ontologiebasiertes Modell. Wie in Abschnitt 2.3 erläutert, werden in der

[1] Application Programming Interface

Literatur mehrere Arten von Zugriffkontrollsystemen unterschieden. Das ontologiebasierte Modell muss eine ausreichende Mächtigkeit zur Beschreibung gängiger Zugriffskontrollsysteme zur Verfügung stellen. Dies bedeutet, dass Eigenschaften konkreter Modelle auf ein allgemeineres Abstraktionsniveau gehoben werden. Die Herausforderung dabei ist es, Gemeinsamkeiten zu erkennen und notwendige Abgrenzungen zwischen unterschiedlichen Systemen herauszuarbeiten.

Ein weiterer zentraler Punkt ist die Flexibilität des entworfenen Zugriffskontrollsystems. Durch Erweiterungen sollen Benutzer in der Lage sein, das System auf individuelle Besonderheiten des eingesetzten Rechtemanagements anpassen zu können. Individuelle Anpassungen können mittels konzeptioneller Erweiterung der Ontologie und Spezifikation von Regeln zur Interpretation dieser zusätzlichen Konzepte umgesetzt werden. Hierzu muss sichergestellt werden, dass die zusätzlichen Regeln nicht mit den bereits vorhandenen in Konflikt stehen. Diese individuellen Erweiterungen und die Abstraktion von konkreten Systemen gewährleistet die gewünschte Flexibilität.

Der zentrale Aspekt der Modellierung liegt auf einer intuitiven Benutzbarkeit des Systems. Dazu gehört, dass bekannte Elemente aus existierenden Zugriffskontrollsystemen, wie z.B. das Konzept einer Rolle, im Zugriffskontrollmodell beschrieben werden und zur internen Verarbeitung von Zugriffsanfragen auf vorhandene Konzepte abgebildet werden. Der Benutzer des Modells muss sich somit nicht mit internen Details der Modellierung beschäftigen, sondern kann in gewohnter Art und Weise seine Daten in das Zugriffskontrollsystem einbringen.

Wie in Abschnitt 2.3 beschrieben, ist die zentrale Aufgabe eines Rechteverwaltungssystems zu entscheiden, ob ein Subjekt eine Operation auf einem Objekt durchführen darf. Ausgehend von dieser Anfrage muss das Modell in der Lage sein anhand vorliegender Informationen einen Zugriff zu gestatten oder zu verbieten. Formal entspricht eine Autorisierungsanfrage somit der Abbildung

```
(subject, object, operation) → {accessGranted, accessDenied}.
```

Im Falle mehrstelliger Operationen werden Objekte einer Anfrage in Form von Mengen oder Listen übergeben.

Grundsätzlich können in Zugriffskontrollsytemen zwei Aufgabentypen unterschieden werden:

1. **Operative Aufgaben:** Hierbei handelt es sich um die Beantwortung von einfachen Zugriffsanfragen. Für einen gegebenen Benutzer soll entschieden werden, ob dieser eine Operation auf einem festgelegten Objekt ausführen darf. Im Allgemeinen wird bei operativen Aufgaben der Zustand des Modells nicht verändert.

2. **Administrative Aufgaben:** Administrative Aufgaben bewirken generell eine Änderung des Zustands der Wissensbasis. Die Administration des Systems kann unterteilt werden in die initiale Konfiguration des Modells und administrative Anpassungen zur Laufzeit des Systems. Bei der Initialisierung des Systems, die einmalig bzw. bei notwendigen Anpassungen des Rechtemanagements aufgrund organisatorischer Änderungen durchgeführt wird, werden organisationsspezifische Sicherheitsstrategien durch konzeptionelle Erweiterungen der Wissensbasis eingeführt. Dies geschieht durch die Beschreibung neuer Konzepte und deren Eigenschaften und der Definition

zusätzlicher Regeln. Beispiele administrativer Tätigkeiten zur Laufzeit eines Systems sind das Anlegen neuer Benutzer und Dateien, oder die Weitergabe von Rechten durch den Eigentümer eines Objekts.

Da administrative Aufgaben das eigentliche Modellverhalten beeinflussen, muss sichergestellt werden, dass diese nur von berechtigten Personen ausgeführt werden können. Ziel ist es diese Kontrolle durch das Modell selbst durchzuführen, d.h. dass Modell auf sich selbst anzuwenden. Es soll somit möglich sein, Rechte zur Vergabe von Rechten zu spezifizieren. Durch die Anwendung des Modells auf sich selbst, stehen alle Möglichkeiten die für die Vergabe von Berechtigungen an gewöhnliche Nutzer des Systems verwendet werden, auch für die Beschreibung von Administrationsberechtigungen zur Verfügung. Im Vergleich zu gängigen Systemen, in denen üblicherweise eine Gruppe von Administratoren zur Durchführung sämtlicher administrativer Tätigkeiten berechtigt ist, stellt die Möglichkeit der feingranularen Vergabe von Administrationsrechten einen großen Vorteil dar.

3.2 Modellierung mit Ontologien und Regeln

Das semantische Modell, das in den folgenden Kapiteln vorgestellt wird, beruht auf Ontologien und Regeln. Ontologien beschreiben hierbei die Elemente, die für die Modellierung eines Zugriffkontrollsystems benötigt werden. Es werden also beispielsweise die Konzepte einer Operation oder eines Subjekts mit deren typischen Eigenschaften modelliert. Für konkrete Anwendungen werden diese Konzepte dann geeignet instantiiert. Eine Ontologie stellt somit in Form einer Wissensbasis die eigentliche Datenschicht dar. Diese Daten werden zum Zeitpunkt einer Anfrage durch Regeln ausgewertet. Mittels Regeln kann man Logik zur Interpretation des Modells, welche üblicherweise in einer konkreten Anwendung in Programmcode explizit ausformuliert werden muss, bereits zur Entwurfszeit in das Modell einbringen. Gerade für die geforderte dynamische Erweiterbarkeit des Systems ist diese Eigenschaft ausschlaggebend, da somit ohne Anpassung von Quellcode nur durch Definition zusätzlicher Regeln das Modellverhalten erweitert oder angepasst werden kann. Ein einfache Regel könnte z.B. eine Anfrage positiv beantworten, falls in der Wissensbasis eine Berechtigungsinstanz existiert, die dem Subjekt die entsprechende Operation auf dem angefragten Objekt genehmigt.

In dieser Arbeit wird KAON2 [KAON05] als Infrastruktur zur Bearbeitung von Ontologien und zum automatischen Schließen verwendet. KAON2 erweitert binäre OWL Relationen um zusätzliche logische Konstrukte beliebiger Stelligkeit. Diese sogenannten *Nicht-OWL-Prädikate* erlauben die logische Verknüpfung beliebig vieler OWL-Instanzen und ermöglichen gemeinsam mit der Verwendung nichtmonoter Negation die komplette Interpretation des Modells durch Regeln. Hierzu gehört auch die Auflösung von Konflikten bei etwaigen positiven und negativen Berechtigungen. Das Modell ist somit in der Lage Zugriffsanfragen ohne zusätzlichen Programmcode durch Auswertung von Regeln zu beantworten. Dies gewährleistet eine maximale Flexibilität des Modells, da Änderungen ohne Anpassung der Implementierung durchgeführt werden können.

Aufgrund der modellgestützten Interpretation der Ontologien durch Regeln, kann diese Vorgehensweise den Bemühungen der modellgestützten Architekturen (engl.: Model Driven Architectures (MDA)) zugeordnet werden. Im Gegensatz zu MDA wird aus dem formalen

Code jedoch kein ausführbarer Programmcode generiert, sondern das Modell wird durch einen Reasoner interpretiert.

Um das Modell zu administrieren und es für andere Systeme zugreifbar zu machen, ist Programmcode jedoch unerlässlich. Änderungen der Ontologien, z.B. durch Hinzufügen neuer Objekte oder Beschreibung neuer Konzepte, können nur programmatisch durchgeführt werden. Dadurch muss den oben genannten administrativen Aufgaben stets entsprechender Programmcode gegenüberstehen, der die beabsichtigen Zustandsänderungen in die Ontologien einbringt. Damit andere Systeme die Zugriffskontrolle verwenden können, sind ebenfalls einfache Schnittstellen zur Verfügung zu stellen. Die dazu notwendige Software Infrastruktur wird in der nachfolgenden Architekturbeschreibung durch entsprechende APIs dargestellt.

3.3 Architektur des Zugriffskontrollsystems

Wie in Abbildung 2.2 auf Seite 13 beschrieben, kann ein Zugriffskontrollsystem nie als unabhängige Anwendung betrachtet werden, sondern arbeitet stets mit anderen Anwendungen zusammen. Während in vielen heutigen Implementierungen die Zugriffskontrolle eng mit der eigentlichen Anwendung verwoben ist, wird in dieser Arbeit ein komponentenbasierter Ansatz verfolgt. Das Zugriffskontrollsystem ist als eigenständiges Modul realisiert, das von der externen Anwendung über festgelegte Schnittstellen angesprochen werden kann. Dieser modulare Aufbau entspricht dem Prinzip eines Baukastens, in dem komplette Anwendungen durch Auswahl und Kombination einzelner Bausteine realisiert werden. Die Hauptvorteile derartiger modularer Systeme sind die einfache Austauschbarkeit und die Wiederverwendbarkeit von Bausteinen.

Der Schwerpunkt dieser Arbeit liegt auf der Modellierung eines Autorisierungssystems, das als unabhängiges Modul realisiert wird. Die modulinterne Verwendung von Wissensbasen und Regeln werden für den Anwender völlig transparent hinter Modulschnittstellen verborgen. Somit können auch Anwendungen, die nicht auf semantischen Technologien basieren, dieses Zugriffskontrollmodul verwenden.

3.3.1 Interaktion mit externen Anwendungen

Abbildung 3.1 gibt eine Übersicht über eine mögliche Interaktion zwischen externer Anwendung und der in dieser Arbeit entwickelten Autorisierungskomponente.

Externe Anwendung: Die externe Anwendung wird in Abbildung 3.1 vereinfacht dargestellt. Es wird von der konkreten Aufgabe der Anwendung und deren Aufbau abstrahiert. Um ein vollständiges Zugriffskontrollsystem zu realisieren, muss die externe Anwendung einen Authentifizierungsdienst zur Überprüfung der Identität des Anwenders zur Verfügung stellen. Nur wenn diese Authentifizierung erfolgreich war, wird eine Zugriffsanfrage an den Referenzmonitor weitergeleitet. Der Referenzmonitor überprüft vor jedem Zugriff auf eine Ressource, ob eine Berechtigung für diesen Zugriff vorliegt. Dies erfolgt durch eine entsprechende Anfrage an das Autorisierungssystem. Nur wenn der Autorisierungsdienst den Zugriff gestattet, leitet der Referenzmonitor der Zugriff auf eine Ressource weiter. Der Referenzmonitor kann nur durch die externe

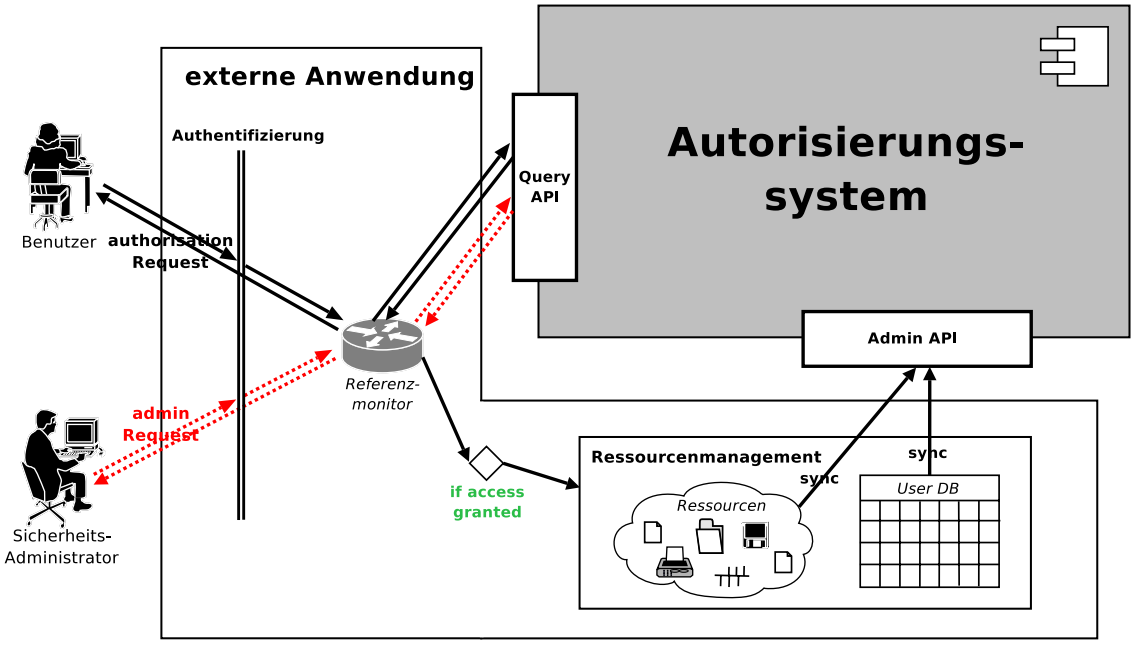

Abbildung 3.1: Interaktion zwischen externer Anwendung und Autorisierungssystem.

Anwendung implementiert werden, da nur sie den Zugriff auf verwaltete Ressourcen durchführen kann.

Ressourcenmanagement: Das Ressourcenmanagement ist Teil der externen Anwendung und ist für die Verwaltung von Ressourcen verantwortlich. Ressourcen umfassen hierbei alle Arten von passiven Objekten wie etwa Dateien, Verzeichnisse oder Drucker, aber auch aktive Elemente, wie z.B. die Benutzer einer Anwendung. Bei real existierenden Anwendungen werden diese Ressourcen typischerweise in Datenbanken verwaltet, eine ontologiebasierte Verwaltung wäre jedoch ebenfalls denkbar.

Autorisierungssystem: Das Autorisierungssystem stellt den eigentlichen Kern dieser Arbeit dar. Wie bereits erwähnt, handelt es sich hierbei um einen modularen Baustein. Abbildung 3.2 beschreibt den internen Aufbau des Autorisierungssystems. Zentrales Element ist hierbei das Zugriffskontrollmodell, das durch Ontologien und Regeln beschrieben wird.

Die nach außen sichtbaren Schnittstellen sind unterteilt in eine **Setup API** und eine **Query API**. Die Setup API wird verwendet, um das Zugriffskontrollmodell auf anwendungsspezifische Eigenheiten anzupassen. Dies kann in einem initialen Schritt bei der Einführung des Zugriffskontrollmodells oder aber bei Anpassung des existierenden Rechtemanagements notwendig sein. Dabei kann es sich um konzeptionelle Erweiterungen der Ontologien und Regeln oder aber Konfiguration vorhandener Konzepte handeln.

Konkrete Berechtigungsanfragen werden an die Query API gestellt. Der Zugriff auf diese Schnittstelle sollte nur dem Referenzmonitor erlaubt sein. Man unterscheidet zwei Arten von Anfragen: Autorisierungs- und Administrationsanfragen. Eine einfache Autorisierungsanfrage (*AuthorisationRequest*) prüft, ob das gegebene Subjekt

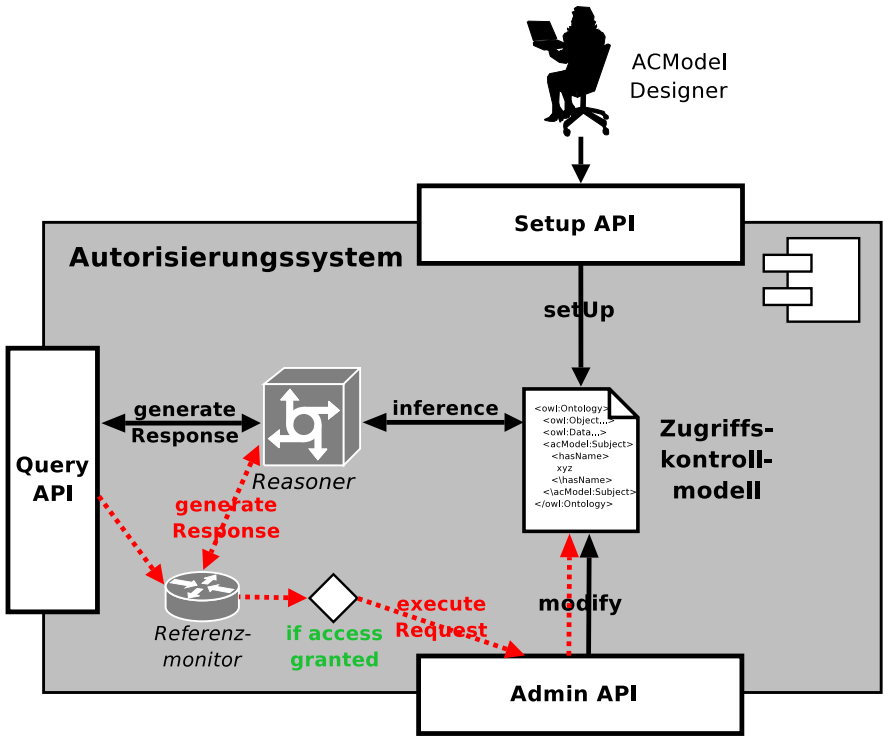

Abbildung 3.2: Interner Aufbau des Autorisierungssystems.

eine Operation auf einem Objekt ausführen darf. Mittels der in Regeln und Ontologien definierten Informationen wird überprüft, ob eine passende Berechtigung vorliegt und eine entsprechende Antwort generiert. Diese Antwort wird an den Referenzmonitor zurückgegeben. Administrative Anfragen (*AdminRequest*) verändern den Zustand der Wissensbasis, durch Änderung von Ontologieinstanzen. Bevor ein administrative Anfrage ausgeführt wird, muss ebenfalls überprüft werden, ob eine geeignete Berechtigung vorliegt. Ist dies der Fall, wird diese Operation durch Aufruf einer geeigneten Methode der **Admin API** ausgeführt. Die Admin API greift hierbei direkt auf die Ontologie zu und führt entsprechende Änderungen aus.

3.3.2 Herausforderungen für die praktische Verwendbarkeit

Bei einem semantischen Modell müssen alle Informationen, die für die Berechnung der Berechtigung eines Zugriffs notwendig sind, in der Ontologie enthalten sein. Der Datenabgleich zwischen dem Ressourcemanagement der externen Anwendung und dem semantischen Autorisierungssystem ist eine große Herausforderung. Wurde beispielsweise die Rollenzugehörigkeit eines Mitarbeiters aufgrund einer Beförderung geändert, oder ein Objekt gelöscht, so müssen diese Änderungen auf die Ontologien übertragen werden. Dieser Vorgang wird in Abbildung 3.1 durch die `sync()` Beziehungen dargestellt. Dieser Datenabgleich kann auf unterschiedliche Art und Weise erfolgen, beispielsweise durch direkten Zugriff auf die Ontologie mittels eines Texteditors oder eines Ontologiewerkzeugs wie Protégé [Pro00]. Um die Kapselung des Autorisierungssystems und die Konsistenz der enthaltenen Ontologien nicht zu verletzen, ist die Ausführung administrativer Tätigkeiten unter Verwen-

dung der Admin API, wie in Abbildung 3.1 dargestellt, zu bevorzugen. Möglich ist auch die Durchführung des Datenabgleichs unter Verwendung des Referenzmonitors. Diese Vorgehensweise stellt sicher, dass nur autorisierte Personen auf die Admin API zugreifen dürfen. Durch die notwendige Überprüfung der Autorisierung sind diese indirekten Zugriffe auf die Admin API natürlich weniger effizient.

Viele Alternativen sind auch bei der Festlegung der Häufigkeit eines Datenabgleichs möglich. Denkbar sind beispielsweise Push- oder Pullstrategien. Bei einer Pushstrategie löst eine Änderung im Ressourcenmanagement einen Abgleich mit der Ontologie aus. Bei einer Pullstrategie hingegen kümmert sich das Autorisierungssystem selbst um die Konsistenz seiner Daten, z.B. durch Abgleich vor jeder Anfrage oder aber in regelmäßigen Zeitabständen. Die konkrete Vorgehensweise hängt stark von der konkreten Anwendung ab. Handelt es sich um ein sehr statisches System, in dem Ressourcen nur selten geändert werden, wie z.B. einem Webserver, könnte eine Pushstrategie eingesetzt werden. Allgemein muss eine Abwägung zwischen dem hohen Ressourcenverbrauch durch häufigen Datenabgleich und fehlerhafter Entscheidungen aufgrund von Dateninkonsistenzen stattfinden.

Eine weitere Herausforderung ist die Performanz des Autorisierungsdienstes. Die Auswertung von Regeln, das sogenannte automatische Schließen, basiert auf komplexen Algorithmen, die sich in der Berechnungsdauer einer Zugriffsanfrage bemerkbar machen. Berechtigungsanfragen in einem Zugriffskontrollsystem sind im Allgemeinen nicht echtzeitkritisch. Die Geduld eines Anwenders ist jedoch sicherlich begrenzt.

Kapitel 4

Modellierung I – Ontologien

Aufbauend auf dem zuvor vorgestellten Konzept beschreibt dieser Abschnitt den Entwurf einer Ontologie, die die konzeptionellen Elemente eines Modells der Zugriffskontrolle formal spezifiziert. Zuerst wird in Abschnitt 4.1 eine abstrakte Übersicht über den strukturellen Aufbau des Ontologiemodells gegeben, bevor dieses in den nachfolgenden Abschnitten schichtenweise analysiert wird.

4.1 Struktur des Zugriffskontrollmodells

Um eine gute Verständlichkeit der Wissensbasis und eine einfachere Verwendbarkeit zu gewährleisten, ist die ontologische Beschreibung des Zugriffskontrollmodells in Schichten unterteilt. Jede dieser Schichten repräsentiert einen eigenständigen Abstraktionsgrad und ist für sich betrachtet abgeschlossen. Die Ontologien, die die einzelnen Schichten repräsentieren, importieren hierbei die übergeordneten Schichten, so dass allgemeine Informationen höherer Schichten stets verfügbar sind.

Dem Modell zur Beschreibung eines Rechtemanagementsystems liegt eine konzeptionelle Unterteilung auf fünf Ebenen zugrunde. Abbildung 4.1 beschreibt diesen Aufbau grafisch. Nicht durchgezogene Umrandungen kennzeichnen hierbei optionale Schichten.

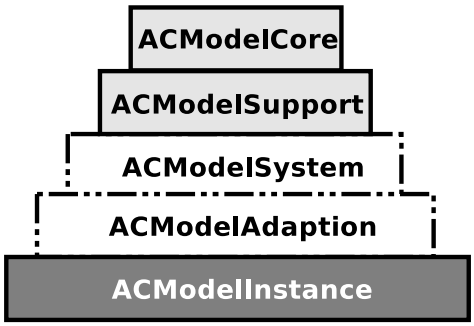

Abbildung 4.1: Struktur des Zugriffskontrollmodells.

Das **Kernmodell** (engl.: ACModelCore) beschreibt die wesentlichen Konzepte, die zur Modellierung eines Zugriffskontrollsystems notwendig sind auf sehr abstraktem Niveau. Hierzu gehören beispielsweise Berechtigungen, Operationen, Subjekte und Objekte. Trotz

ausreichender Mächtigkeit ist eine Beschreibung eines konkreten Zugriffskontrollsystems mit den Konzepten des Kernmodells nur in wenigen Fällen sinnvoll.

Eine Hilfestellung zur Erstellung konkreter Zugriffskontrollinstanzen bietet die zweite Ebene, die sogenannte **Ergänzungsschicht** (engl.: ACModelSupport). Diese verfeinert die im Kernmodell eingeführten Konzepte, um diese einfacher auf reale Anforderungen anwenden zu können. So werden beispielsweise Subjekte um Subjektgruppen bzw. Subjekthierarchien erweitert. Basierend auf diesen Verfeinerungen lassen sich Regeln definieren, welche die Konzepte miteinander in Beziehung setzen, so dass Zugriffsanfragen beantwortet werden können.

Um die Erstellung konkreter Systeme zu erleichtern, beschreibt die **Systemebene** (engl.: ACModelSystems) aus der Literatur bekannte und in heutigen Standardsystemen eingesetzte Zugriffskontrollmodelle. Die Verwendung dieser Ebene ermöglicht beispielsweise die Umsetzung eines rollenbasierten Systems durch einfache Instantiierung vorhandener Konzepte.

Reicht die Mächtigkeit dieser Ebenen nicht zur Beschreibung individueller Sicherheitsstrategien aus, kann auf der **Adaptionsschicht** (engl.: ACModelAdaption) eine Anpassung erfolgen. Diese Anpassungen umfassen i.A. eine konzeptionelle Erweiterung vorhandener Eigenschaften und die Beschreibung neuer Regeln zur Interpretation der zusätzlichen Konzepte.

Auf der untersten Ebene, der **Instanzschicht** (engl.: ACModelInstance), wird ein konkretes Zugriffskontrollsystem durch Instantiierung der in den übergeordneten Schichten eingeführten Konzepte realisiert. Aufgrund dieser logischen Trennung von Konzepten und Instanzen kann ein Zugriffskontrollsystem in verschiedene Systemen oder Anwendungen durch einfache Übertragung der Instanzschicht wiederverwendet werden.

Ein kurzes Beispiel, welches in Abbildung 4.2 veranschaulicht wird, soll diese Unterteilung verdeutlichen. Herr Maier bekommt in seiner Firma die *Rolle* eines „SalesManagers“ zugewiesen. Im Kernmodell, der höchsten Abstraktionsebene, wird Herr Maier als eine Instanz der allgemeinen *Subjekt*klasse dargestellt. Dies beschreibt, dass Herr Maier im System eine aktive Rolle besitzt. Da Herr Maier eine Person ist, die nicht weiter in Untereinheiten zerlegt werden kann, wird die allgemeine Subjektklasse in der Ergänzungsschicht zu einem *atomaren Subjekt* (engl.: AtomicSubject) verfeinert. Die *Rolle* „SalesManager“ ist ein systemspezifisches Konzept rollenbasierter Zugriffskontrollsysteme und ist in der Systemschicht enthalten. Da eine Rolle aus einer Menge von Personen oder aus anderen Rollen bestehen kann, wird diese als Subkonzept *zusammengesetzter Subjekte* (engl.: CompositeSubject) beschrieben. Die Organisation in der Herr Maier arbeitet, erweiterte das Konzept der atomaren Subjekte aufgrund individueller Benutzereigenschaften zur Klasse *MyUser*. Diese Anpassung wird in der Adaptionsschicht vorgenommen. Auf der Instanzebene werden Herr *Maier* und die Rolle *SalesManager* durch Instantiierung der entsprechenden Konzepte eingeführt.

In den nachfolgenden Abschnitten werden das Kernmodell, die Ergänzungs- und Systemschicht genauer erläutert.

4.2 Das Kernmodell (ACModelCore)

Das Kernmodell enthält die wichtigsten Komponenten, die zur Modellierung eines Zugriffskontrollsystems benötigt werden. Kursive Schreibweise von Begriffen kennzeichnen im fol-

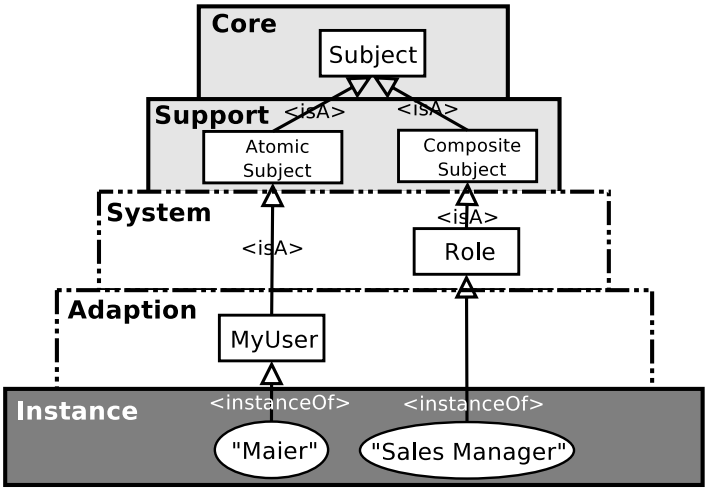

Abbildung 4.2: Einordnung des Beispiels in die Schichten des Modells

genden Kapitel deutsche Bezeichnungen von Konzepten der Wissensbasis.

Zugriffskontrollelement (ACEntity): Das allgemeine Konzept *ACEntity* umfasst die wichtigsten Elemente, die der direkten Beschreibung von Zugriffskontrollinformationen dienen. Hierzu gehören *Subjekte*, *Objekte*, *Zugriffsrechte* und *Berechtigungen*.

Zusätzlich wird das abstrakte Konzept *GeneralACEntity* eingeführt. Es dient der Abgrenzung von systemspezifischen Konzepten anderer Ontologieschichten, wie z.B. Sicherheitsstufen regelbasierter Systeme. Alle Elemente des Kernmodells sind diesem Konzept untergeordnet und somit eindeutig klassifizierbar. Abbildung 4.3 stellt die Unterklassenbeziehungen der Konzepte des Kernmodells grafisch dar.

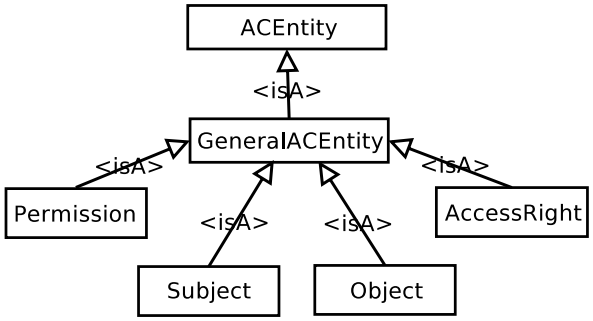

Abbildung 4.3: Vererbungsbeziehungen der allgemeinsten Konzepte des Kernmodells

Die Klasse *Subjekt* (Subject) beschreibt die aktiven Elemente eines Zugriffskontrollsystems. Konkrete Ausprägungen eines *Subjekts* können hierbei Benutzer, Gruppen, Rollen oder aber auch Prozesse und Prozessgruppen sein.

Objekte (Object) sind passive Elemente eines Zugriffskontrollsystems, die der Verwaltung des Systems unterliegen. Beliebige Datenobjekte, wie Dateien oder Verzeichnisse, und Systemressourcen wie Drucker oder Prozessoren sind Beispiele solcher *Objekte*.

Das Konzept *Operation* modelliert Aktionen auf *Objekten*. Es gibt durchzuführenden Aktionen einen Namen und definiert die Anzahl von *Objekten*, die für die Durchführung der *Operation* notwendig sind. Die einstellige Operation „read“ gehört dieser Klasse an.

Ein *Zugriffsrecht* (AccessRight) verknüpft eine Operation mit einer Menge von Objekten. Es beschreibt, dass die referenzierte *Operation* auf den enthaltenen *Objekten* ausgeführt werden darf. Soll modelliert werden, dass auf eine Objektinstanz „o1“ lesend zugegriffen werden darf, referenziert das zugehörige *Zugriffsrecht* eine *Objekt*instanz „o1“ und eine *Operations*instanz namens „read“. Wie der Abbildung 4.4 entnommen werden kann, enthält ein Zugriffsrecht hierzu Referenzen zu einer Menge von *Operationen* und *Objekten*[1].

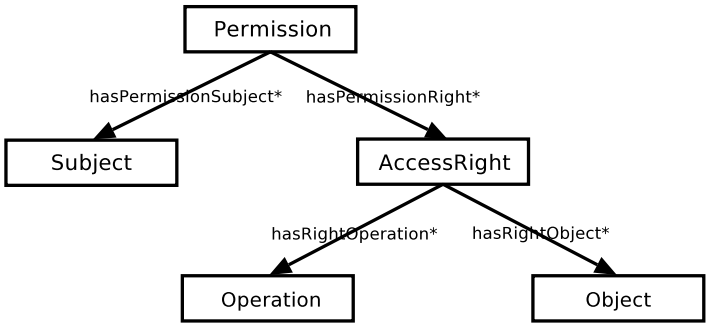

Abbildung 4.4: Beziehungen von Berechtigungen und Zugriffsrechten.

Das zentrale Konzept des Modells ist jedoch die *Berechtigung* (engl.: Permission). Eine *Berechtigung* weist einem *Subjekt Zugriffsrechte* zu. Durch eine *Berechtigung* wird formuliert, dass das referenzierte *Subjekt* auf die im *Zugriffsrecht* definierten *Objekte* mit den dort referenzierten *Operationen* zugreifen darf. Wird eine *Autorisierungsanfrage* gestellt, so wird überprüft, ob eine entsprechende *Berechtigung* vorhanden ist und ggf. der Zugriff gestattet. Abbildung 4.4 verdeutlicht den Zusammenhang zwischen *Berechtigungen* und *Zugriffsrechten* grafisch. Alle in der Abbildung dargestellten Relationen haben entsprechende inverse Relationen. Aus Gründen der Übersichtlichkeit wurde auf deren Darstellung jedoch verzichtet.

Die konzeptionelle Trennung von *Berechtigungen* und *Zugriffsrechten* ermöglicht eine sehr flexible Administration der Zugriffskontrollsteuerung. Wie zuvor erwähnt, beschreibt ein *Zugriffsrecht* die *Objekte* auf denen eine *Operation* durchgeführt werden darf. *Berechtigungen* weisen eines oder mehrere dieser *Zugriffsrechte* einem oder mehreren *Subjekten* zu. Durch die Möglichkeit der Mehrfachreferenzierung von *Subjekten* und *Zugriffsrechten* kann ein Administrator unterschiedliche Strategien zur Abbildung organisationsspezifischer Sicherheitsstandards anwenden:

Objektzentrierte Administration von Berechtigungen: Dieser Ansatz entspricht weitestgehend den in Abschnitt 2.3.1 vorgestellten Zugriffskontrolllisten. Aus-

[1] In den grafischen Abbildungen der folgenden Abschnitte bedeutet das * Symbol hinter einer Relation, dass eine Instanz diese Relation mehrmals referenzieren kann. Fehlt dieses Symbol, so muss eine Instanz die Relation genau einmal aufweisen.

gehend von einem Objekt werden alle Benutzer aufgelistet, die auf dieses zugreifen dürfen. Um dies mit dem zuvor vorgestellten Konzept der *Berechtigung* abzubilden, darf jede Berechtigungsinstanz nur je ein *Zugriffsrecht* aber beliebig viele *Subjekte* referenzieren. Abbildung 4.5 verdeutlicht dies beispielhaft.

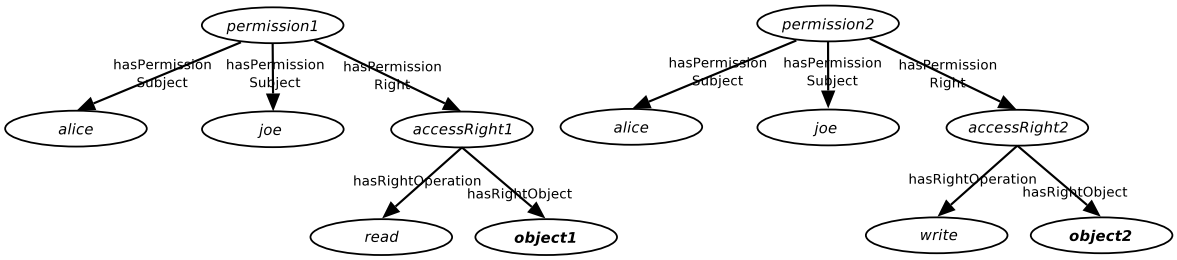

Abbildung 4.5: Objektzentrierte Administration von Berechtigungen.

Subjektzentrierte Administration von Berechtigungen: Die in Abschnitt 2.3.1 vorgestellten Fähigkeitslisten verfolgen einen subjektbasierten Ansatz. Für jeden Benutzer werden alle Objekte aufgelistet, auf die dieser zugreifen darf. Im Gegensatz zur objektzentrierten Abbildung von *Berechtigungen*, darf hier eine Berechtigungsinstanz nur ein *Subjekt*, aber beliebig viele *Zugriffsrechte* referenzieren. Dieses Vorgehen wird in Abbildung 4.6 anhand der in Abbildung 4.5 beschriebenen Zugriffsinformationen dargestellt.

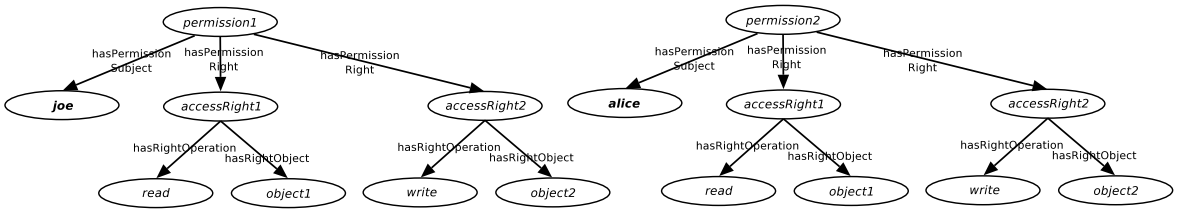

Abbildung 4.6: Subjektzentrierte Administration von Berechtigungen.

Mischversion: Das Konzept der *Berechtigung* lässt jedoch auch Kombinationen obiger Strategien zu, so dass in einer *Berechtigung* mehrere *Subjekte* und *Zugriffsrechte* miteinander in Beziehung gesetzt werden können. Dies ermöglicht eine zusätzliche Vereinfachung der Administration, wie in Abbildung 4.7 ersichtlich wird. Durch Kombination obiger Ansätze reicht eine Berechtigungsinstanz zur Beschreibung der in Abbildung 4.5 und 4.6 beschriebenen Informationen aus. Verständlicherweise erfordert dieses Vorgehen eine bedachte Vorgehensweise des Administrators, da eine Zuweisung eines neuen *Zugriffsrechts* zu einer vorhandenen *Berechtigung* für alle in der *Berechtigung* referenzierten *Subjekte* gilt. Dies bedeutet, dass ein Administrator nur dann einer *Berechtigung* ein *Zugriffsrecht* zuweisen darf, wenn jedem *Subjekt* der Berechtigung der Zugriff gewährt werden soll. Ist dies nicht der Fall, legt der Administrator eine neue Berechtigungsinstanz an.

Zugriffskontrollstrategien (ACPolicy): In bestimmten Situationen muss die Ausgabe eines Zugriffskontrollsystems explizit vorgegeben werden, beispielsweise wenn keine

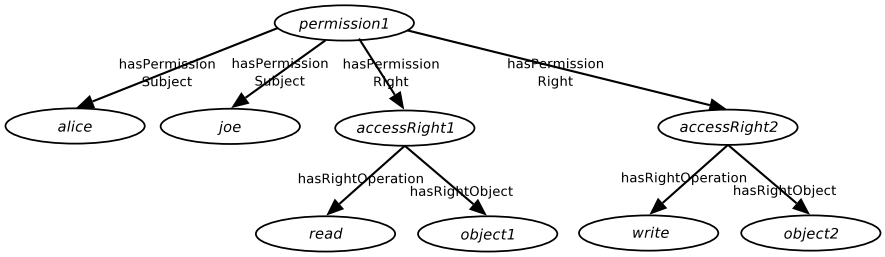

Abbildung 4.7: Kombination von subjekt- und objektzentrierter Administration.

Informationen zur Beantwortung einer Anfrage vorhanden sind oder diese sich widersprechen. Strategien zur Auflösung solcher Situationen werden durch das Konzept der *Zugriffskontrollstrategien* beschrieben. Die Umsetzung dieser *Strategien* erfolgt wie in Abschnitt 5.3 beschrieben durch Regeln.

Das Kernmodell spezifiziert *Standardstrategien* (DefaultRulePolicies) für den Fall, dass keine Informationen für eine Anfrage in der Wissensbasis vorhanden sind. Ein Zugriff kann in einem solchen Fall entweder verboten oder gestattet werden. In diesem Zusammenhang spricht man auch von offenen und geschlossenen Systemen. Abbildung 4.8 stellt die möglichen Ausprägungen einer solchen Standardstrategie dar. Referenziert ein *Zugriffskontrollmodell* eine *openWorldAssumption* Instanz, ist alles was nicht explizit verboten wurde, erlaubt. Dies bedeutet dass im Falle fehlender Informationen ein Zugriff gestattet wird. Bei *closedWorldAssumption* hingegen gilt die Regel, dass was nicht explizit genehmigt ist, verboten wird. Dies wird als Standardverhalten eines Zugriffskontrollmodells angenommen.

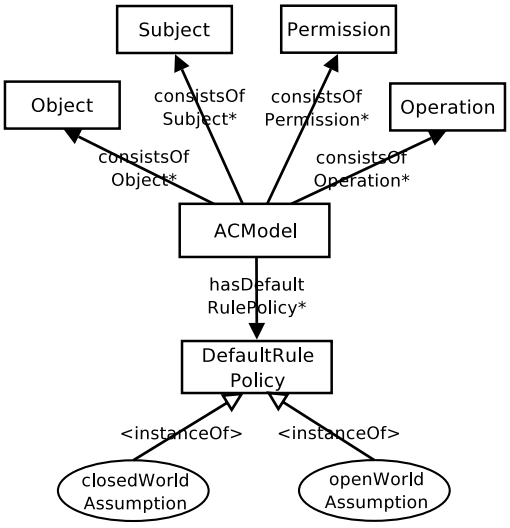

Abbildung 4.8: Das Zugriffskontrollmodell und zugehörige Standardstrategien.

Zugriffskontrollmodell (ACModel): Das Konzept des *Zugriffskontrollmodells* beschreibt ein Zugriffskontrollsystem als Ganzes. Alle zuvor beschriebenen *Zugriffskontrollelemente* können nur innerhalb eines solchen Modells existieren. Dies wird durch die in Abbildung 4.8 dargestellten Relationen modelliert. Das Zugriffskontrollmodell legt

durch Referenzen auf geeignete Strategien Regeln zur Interpretation der im Modell enthaltenen Informationen fest und bestimmt dadurch die grundlegenden Eigenschaften des Systems.

4.3 Die Ergänzungsebene (ACModelSupport)

Die Ergänzungsebene erweitert das Kernmodell um viel benutzte allgemeine Zugriffskontrollelemente. Ziel hierbei ist es, ausdrucksstarke Konzepte zur Verfügung zu stellen, die die Erstellung eines konkreten Zugriffskontrollsystems erleichtern. Um dies zu erreichen werden abstrakte Konzepte vorgestellt, die die Aggregation von Elementen erlauben. Diese abstrakten Elemente werden durch Vererbungsbeziehungen auf die Klassen der Kernschicht angewendet, um beispielsweise Gruppen, Hierarchien oder Listen von Subjekten beschreiben zu können.

4.3.1 Abstrakte aggregierte Elemente

Um die Administration großer Systeme zu erleichtern, werden in Zugriffskontrollsystemen im Allgemeinen nicht einzelne *Subjekte* oder *Objekte* verwaltet, sondern aggregierte Elemente wie zum Beispiel Subjektgruppen oder Objekthierarchien. Im Folgenden werden diese zusammengesetzten Elemente in Form von Gruppen, Hierarchien und Listen abstrakt modelliert, um dann in den nächsten Abschnitten auf konkrete Zugriffskontrollkonzepte abgebildet zu werden.

Abbildung 4.9 gibt eine grafische Übersicht der abstrakten aggregierten Elemente, bevor deren Eigenschaften in den nächsten Absätzen ausführlich betrachtet werden.

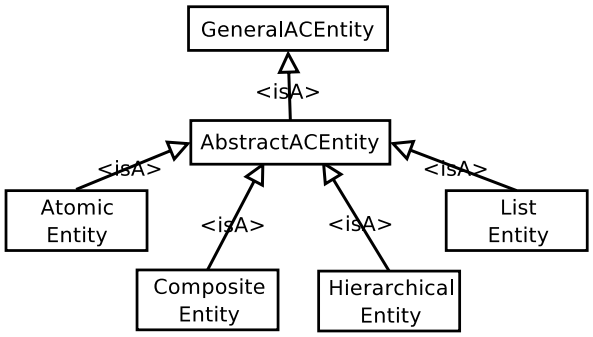

Abbildung 4.9: Abstrakte Konzepte zur Aggregation von Elementen.

Atomare Elemente (AtomicEntity): Ein *atomares Element* beschreibt eine Einheit, die nicht weiter zerteilt werden kann. Ein Beispiel für ein atomares Subjekt wäre eine Person.

Elementgruppen (CompositeEntity): Dieses Konzept beschreibt Elemente, die aus *atomaren Elementen* oder anderen *Elementgruppen* zusammengesetzt sind. Dadurch lassen sich beispielsweise Benutzergruppen modellieren. Wie in Abbildung 4.10 zu sehen ist, referenziert eine *Gruppe* ihre Elemente durch die Relation `#consistsOf`. Die

inverse Relation `#isMemberOfCompositeEntity` beschreibt die Mitgliedschaft eines Elements in einer *Gruppe*. Da eine *Elementgruppe* Mitglied einer anderen *Gruppe* sein kann, sind obige Relationen transitiv definiert.

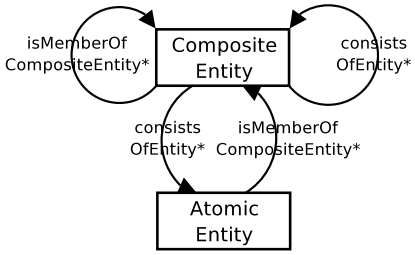

Abbildung 4.10: Relationen von Elementgruppen.

Hierarchische Elemente (HierarchicalEntity): Will man die Struktur von Dateisystemen in Computersystemen modellieren, so sind diese im Allgemeinen baumartig strukturiert. Um Beziehungen modellieren zu können, in denen ein Element mehreren anderen Elementen unter- bzw. übergeordnet ist, wird das Konzept der *Hierarchie* (Hierarchy) eingeführt. *Hierarchien* bestehen aus sogenannten *Hierarchieknoten* (HierarchyNodes), denen andere Knoten über- bzw. untergeordnet sind. Bei der Vergabe von *Berechtigungen* auf hierarchisch modellierten Elementen ist durch diese Hierarchiebeziehungen die automatische Weitergabe von *Berechtigungen* auf über- bzw. untergeordnete Ebenen möglich. Ein solcher Weitergabemechanismus verringert den Administrationsaufwand von komplexen Strukturen. Alle Berechtigungen die beispielsweise auf einem Verzeichnis definiert sind, können durch Verweis einer geeigneten Weitergabestrategie unter geringem Aufwand auf alle Unterverzeichnisse propagiert werden.

Abbildung 4.11 stellt die notwendigen Konzepte und deren Beziehungen zur Beschreibung *hierarchischer Elemente* dar.

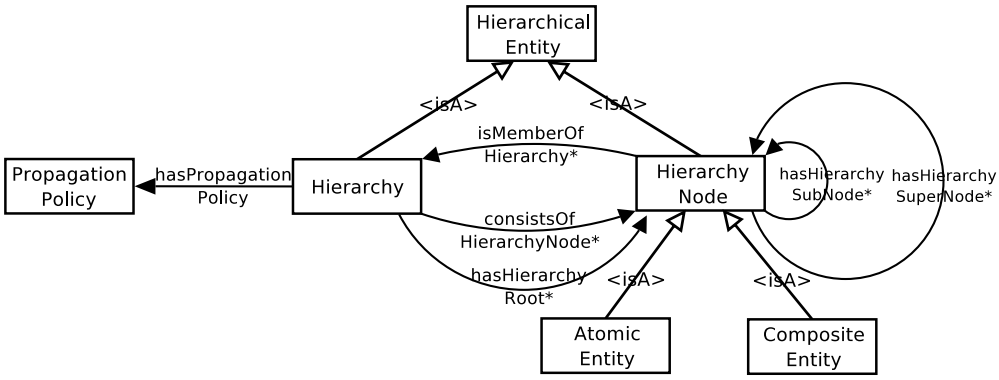

Abbildung 4.11: Relationen zur Beschreibung hierarchische Elementstrukturen.

Ein *Hierarchieknoten* modelliert die Elemente, die in einer *Hierarchie* enthalten sind. Hierbei kann es sich entweder um *atomare* oder *zusammengesetzte Elemente* handeln. Ein *Hierarchieknoten* hat Referenzen zu den ihm unter- und übergeordneten Elemente.

Diese Referenzen sind hierbei transitiv definiert, so dass eine Auswertung stets alle über- und untergeordneten Elementen liefert. Die Relation `#isMemberOfHierarchy` ordnet einen *Knoten* einer *Hierarchie* zu.

Das Konzept der *Hierarchie* beschreibt ein hierarchisch aufgebautes Element. Ein solches besteht aus einer Menge von *Hierarchieknoten* und weist mindestens einen *Hierarchieknoten* als Wurzel aus. Die Wurzel ist für die Verarbeitung von *Hierarchien* unverzichtbar, da diese den Einstiegspunkt der Bearbeitung einer Hierarchiestruktur festlegt. Beim Aufbau von *Hierarchien* ist es wichtig Zyklen zu verhindern, da ansonsten Endlosschleifen bei der Bearbeitung nicht ausgeschlossen werden können. Die Erkennung von Zyklen auf *Hierarchien* ist durch Regeln jedoch leicht beschreibbar.

Zwischen den Ebenen einer *Hierarchie* können Eigenschaften weitergegeben werden. Ein Beispiel hierfür wäre die Weitergabe von *Berechtigungen* auf Objekthierarchien, die ausgehend vom einem Verzeichnis auf alle Unterverzeichnisse übertragen werden. Um festzulegen, ob Eigenschaften an die unter- oder die übergeordneten Elemente weitergegeben werden sollen, referenziert jede Hierarchieinstanz eine geeignete *Weitergabestrategie* (vgl. Abschnitt 4.3.3).

Listenelemente (ListEntity): *Listen* beschreiben eine Verzeichnisstruktur, in der die Elemente linear verkettet sind. Sie werden in einem Zugriffskontrollsystem dort eingesetzt, wo die Reihenfolge von Elementen bedeutend ist. Dies ist z.B. bei listenbasierten *Zugriffsrechten* der Fall, die im folgenden Abschnitt eingeführt werden.

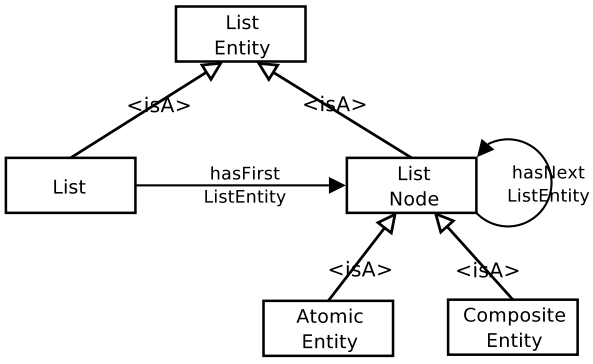

Abbildung 4.12: Relationen zur Beschreibung von einfach verketteten Listen.

Eine Liste wird durch die Konzepte *Listenknoten* (ListNode) und *Liste* (List) modelliert. Abbildung 4.12 skizziert die Beziehungen zwischen diesen. Ein *Listenknoten* beschreibt ein Element einer *Liste*. Jeder Knoten hat einen Verweis auf das nächste Element der *Liste*. Der eindeutige Einstiegspunkt einer *Liste* wird durch eine Referenz auf den ersten *Listenknoten* definiert. Ausgehend von diesem Einstiegsknoten können alle Elemente der *Liste* durch wiederholten Zugriff auf das nächste Element extrahiert werden. Diese Modellierung entspricht der direkten Umsetzung einer einfach verketteten Liste.

4.3.2 Erweiterungen der Kernschichtkonzepte

Die im letzten Abschnitt vorgestellten abstrakten Elemente werden nun auf die Konzepte des Kernmodells angewendet. Dies bedeutet, dass die Konzepte des Kernmodells, wie z.B. *Subjekte* oder *Objekte*, um die Eigenschaften von Gruppen, Hierarchien oder Listen erweitert werden. Technisch gesehen entspricht diese Erweiterung der Bildung von Unterklassen, die sowohl von konkreten Elementen des Kernmodells als auch von aggregierten abstrakten Konzepten erben. Zudem werden in diesem Abschnitt eine Reihe von Erweiterungen eingeführt, die die Beschreibung von Elementen konkreter Zugriffskontrollsysteme erleichtern.

Aggregierte Subjekte: Für die Zugriffskontrolle können *Subjekte* in Form von Personen, Gruppen oder auch Hierarchien verwendet werden. Die Möglichkeit der Aggregation von *Subjekten* erlaubt eine vereinfachte Administration des Rechtemanagements, da Berechtigungen nicht nur auf einzelnen Benutzern, sondern auf Benutzergruppen oder Benutzerhierarchien vergeben werden können.

Atomare Subjekte (AtomicSubject) sind *Subjekte*, die nicht weiter zerlegt werden können. Dabei könnte es sich beispielsweise um eine Person oder einen Prozess handeln. Modelliert wird ein atomares Subjekt, wie in Abbildung 4.13 dargestellt, als Subkonzept von *Subject* und *AtomicEntity*. Atomare Subjekte erben dadurch Relationen zur Beschreibung von Mitgliedschaften in Gruppen und Hierarchien.

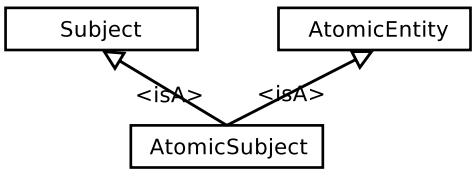

Abbildung 4.13: Vererbungsstruktur zur Beschreibung atomarer Subjekte.

Ein *zusammengesetztes Subjekt* (CompositeSubject) beschreibt eine Gruppe von Systembenutzern. Eine *Subjektgruppe* besteht aus *atomaren Subjekten* oder anderen *Subjektgruppen* und erbt die Eigenschaften der Konzepte *Subject* und *CompositeEntity*.

Das Konzept der *Subjekthierarchien* (SubjectHierarchy) dient der Beschreibung weitverbreiterter hierarchischer Organisationsstrukturen. Abbildung 4.14 zeigt die Vererbungsbeziehungen hierarchischer Subjekte und die notwendigen Beschränkungen der Relationsdomänen auf Subkonzepte abstrakt definierter Elemente. Beispielsweise kann in einer Subjekthierarchie das Wurzelelement nur ein atomares oder zusammengesetztes Subjekt sein. Um dies zu beschreiben, muss die entsprechende Relation auf Instanzen des jeweiligen Subkonzepts eingeschränkt werden. Dies könnte durch Einschränkung der Domäne der jeweiligen OWL-Relation unter Verwendung der in OWL spezifizierten `allValuesFrom` Beschränkungen beschrieben werden. Diese Einschränkungen sind im derzeitigen Modell jedoch nicht enthalten, da die entstehenden Disjunktionen beim automatischen Schließen in der aktuellen Version von KAON2 nicht korrekt behandelt werden.

Aggregierte Objekte: Alle Erweiterungen, die an *Subjekten* exemplarisch vorgestellt wurden, werden auch auf *Objekten* modelliert. *Objekte* werden untergliedert in *atomare*

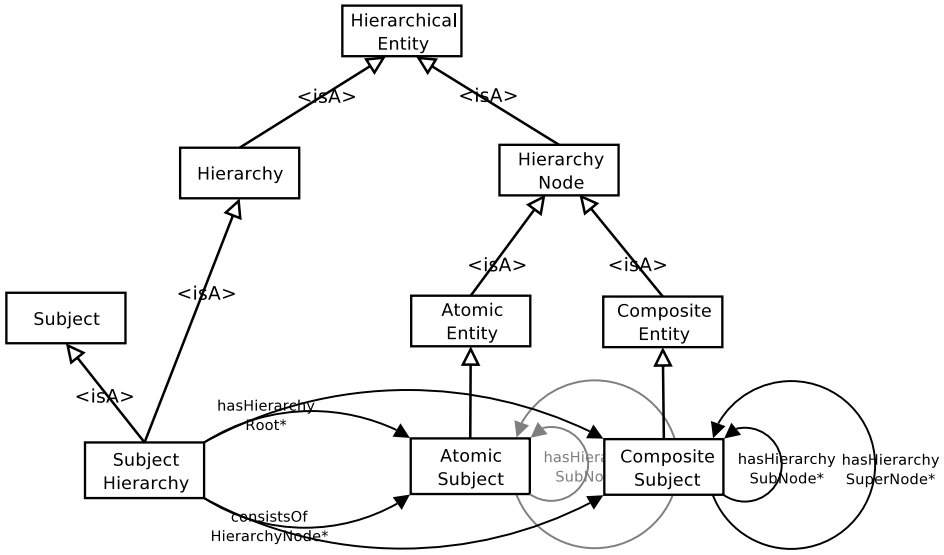

Abbildung 4.14: Beschreibung hierarchischer Subjekte.

(z.B. eine Datei), *zusammengesetzte* (z.B. ein Dateiordner) und *hierarchische Objekte* (z.B. ein Verzeichnisbaum). Zusätzlich können *Objekte* auch in Form einer *Objektliste* (ObjectList) organisiert werden. Diese wird beispielsweise eingesetzt um *Zugriffsrechte* für *Operationen* mit genau festgelegter Reihenfolge der Operanden zu definieren. Abbildung 4.15 zeigt den Aufbau von Objektlisten schematisch.

Aggregierte Operationen: Um die Vergabe von *Berechtigungen* zu vereinfachen, lassen sich auch *Operationen* in Gruppen zusammenfassen. Die Bündelung einer Schreib- und Leseoperation in einer *Operationsgruppe* und deren Zuweisung zu einem *Zugriffsrecht* würde beispielsweise beschreiben, dass eine *Berechtigung* zum Schreiben eines Objekts stets auch lesenden Zugriff gestattet. Im Gegensatz zu den zuvor genannten aggregierten Elementen ist eine hierarchische Strukturierung bei Operationen nicht vorgesehen.

Mengen- und listenbasierte Zugriffsrechte: *Zugriffsrechte* geben an auf welchen *Objekten* eine *Operation* ausgeführt werden kann. Abhängig von der Bedeutung der Reihenfolge der *Objekte* in einem *Zugriffsrecht* werden *mengen- und listenbasierte Zugriffsrechte* unterschieden. Bei den *mengenbasierten Zugriffsrechten* besitzt die Reihenfolge der *Objekte* einer *Operation* keine Bedeutung. Eine mengenbasierte *Operation* `o`, die auf den Objektinstanzen `a` und `b` ausgeführt wird, könnte folglich als `o(a,b)` oder `o(b,a)` formuliert werden ohne deren Bedeutung zu ändern. Dies entspricht der allgemeinen Form des Zugriffsrechtes, wie sie im Kernmodell beschrieben wurde.

Der Reihenfolge der Operanden kommt bei *listenbasierte Zugriffsrechten* (ListBasedAccessRight) hingegen eine bedeutende Rolle zu. Ein *Zugriffsrecht* `o(a,b)` wäre hier nicht mit `o(b,a)` vergleichbar. Ein anschauliches Beispiel eines listenbasierten Zugriffsrechts stellt die Delegation von Rechten dar. Besitzt ein Subjekt das Recht zur Ausführung einer Rechtedelegation, so darf er Rechte eines Subjektes auf ein anderes Subjekt übertragen. Für die Autorisierungskontrolle ist es hierbei von Bedeutung, ob Subjekt `Joe` das Recht hat Berechtigungen von `Bob` an `Alice` zu übertragen, oder

aber von `Alice` an `Bob`. Daher müssen die Operanden des zugehörigen Zugriffsrechts als Listen organisiert sein. Um die Reihenfolge der *Objekte* zu modellieren, wurde die Objektreferenz eines *listenbasierten Zugriffsrechts* auf Instanzen von *Objektlisten* beschränkt. Die Struktur eines *listenbasierten Zugriffsrechts* wird in Abbildung 4.15 beschrieben.

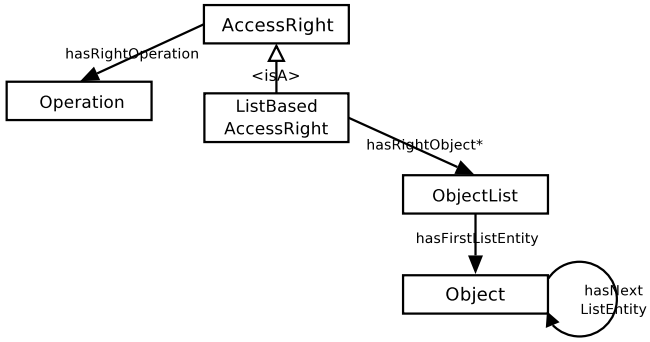

Abbildung 4.15: Beschreibung listenbasierter Zugriffsrechte

Verglichen mit *listenbasierten Zugriffsrechten* ist die Verwendung von *mengenbasierten Zugriffsrechten* einfacher. Während *Objekte listenbasierter Rechte* vor der Zuweisung zu einem *Zugriffsrecht* stets zuerst in einer *Objektliste* strukturiert werden müssen, können *mengenbasierte Zugriffsrechte* die *Objekte* direkt referenzieren. Somit ist für die Erstellung listenbasierter Rechte zusätzlicher Administrationsaufwand notwendig. Auch Zugriffsanfragen nach listenbasierten Zugriffsrechten erfordern zusätzlichen Aufwand, da in einem ersten Schritt zu einer gegebenen Objektliste ein passendes Gegenstück in der Wissensbasis gefunden werden muss. Dies erfordert aufgrund möglicher Weitergaben auf Objektgruppen bzw. Objekthierarchien eine Vorverarbeitung durch Regeln (vgl. Abschnitt 5.2.3).

Positive und negative Berechtigungen: Berechtigungen des Kernmodells werden intuitiv als Genehmigungen verstanden, die einem Benutzer den Zugriff auf eine Ressource gestatten. Auf der Ergänzungsschicht können *Berechtigungen* jedoch auch negativ formuliert werden und verbieten in einem solchen Fall dem *Subjekt* der *Berechtigung* den Zugriff. So könnte beispielsweise der Zugriff auf ein Verzeichnis mittels einer positiven *Berechtigung* genehmigt werden. Sollen jedoch einzelne enthaltene Dateien nicht lesbar sein, so werden hierfür negative *Berechtigungen* ausgesprochen. *Positive* und *negative Berechtigungen* werden, wie in Abbildung 4.16 dargestellt, als Subkonzepte der allgemeinen *Berechtigung*sklasse modelliert.

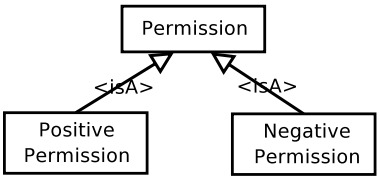

Abbildung 4.16: Vererbungsbeziehungen positiver und negativer Berechtigungen.

Erweiterte Zugriffskontrollmodelle: Wie oben dargestellt, können *Berechtigungen* der Ergänzungsschicht positiv oder negativ formuliert werden. Dies kann zu Konflikten bei der Beantwortung von Zugriffsanfragen führen, die durch entsprechende Konfliktauflösungsstrategien behoben werden. Um potentielle Konfliktsituationen leichter erkennen zu können, werden Zugriffskontrollmodelle anhand der Art ihrer Berechtigungen weiter untergliedert. Abbildung 4.17 stellt die konzeptionellen Erweiterungen dar.

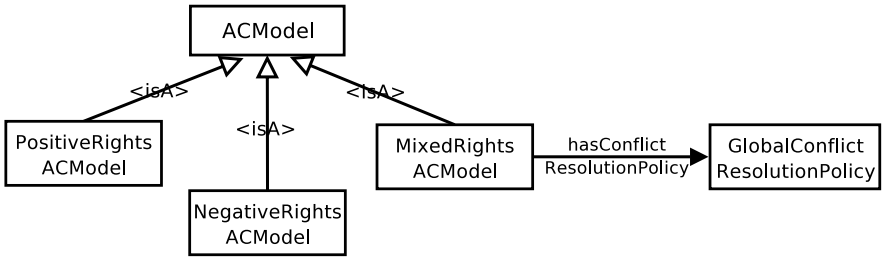

Abbildung 4.17: Erweiterung des Konzepts der Zugriffskontrollmodelle.

Das Konzept *PositiveRightsACModel* kennzeichnet hierbei Zugriffskontrollmodelle in denen ausschließlich positive Berechtigungen vergeben werden können. Dies entspricht der Mehrzahl heute üblicher Systeme. *NegativeRightsACModel* erlaubt hingegen ausschließlich die Vergabe von Verboten. Konflikte können nur bei *gemischten Modellen* (*MixedRightsACModel*) Instanzen auftreten, da diese positive und negative Berechtigungen zulassen. Zur Auflösung dieser Konflikte referenziert ein *gemischtes Zugriffskontrollmodell* eine *globale Konfliktauflösungsstrategie* (GlobalConflictPolicy). Ausprägungen von *Konfliktauflösungsstrategien* werden in Abschnitt 4.3.3 vorgestellt.

Administrative Operationen: Administrative Operationen dienen der Beschreibung von administrativen Vorgängen, deren Ausführung den Zustand des Zugriffskontrollmodells ändern. Wie bei allgemeinen Zugriffen, muss die ausführende Person zur Durchführung administrativer Aufgaben eine passende Berechtigung besitzen. Die Überprüfung dieser Berechtigung führt dazu, dass das Zugriffskontrollmodell bei administrativen Tätigkeiten auf sich selbst angewandt wird. Um Tätigkeiten zu beschreiben, die den Zustand des Modells ändern, wird das Konzept der *administrativen Operation* (AdministrativeOperation) eingeführt. Instanzen dieses Konzepts sind beispielsweise Operationen zur Erstellung von Subjekt- oder Berechtigungsinstanzen. Jeder Instanz einer administrativen Operation der Wissensbasis steht eine entsprechende Methode der in Abschnitt 6.2.2 vorgestellten Admin API gegenüber, deren Ausführung die erwünschte Zustandsänderung des semantischen Modells bewirkt.

Auch für administrative Operationen soll es möglich sein einfache Autorisierungsanfragen stellen zu können, ohne dass es durch Ausführung der Operation zu einer Änderung des Modellzustandes kommt. Hierzu unterscheidet die in Abschnitt 6.2.3 untersuchte Query API einfache Autorisierungsanfragen und administrative Autorisierungsanfragen. Nur bei administrativen Autorisierungsanfragen wird nach erfolgreicher Autorisierungskontrolle die zugehörige administrative Operation auf dem Modell ausgeführt.

4.3.3 Modellierung zusätzlicher Strategien

Wie bereits erwähnt, bestimmen Strategien das Verhalten bei Anfragen, die aufgrund fehlender oder mehrdeutiger Informationen nicht beantwortet werden können. In einem solchen Fall werden Kontextbetrachtungen zur Entscheidungsfindung herangezogen. So könnte beispielsweise bei einem hierarchischen Objekt nicht nur die Erlaubnis auf dem aktuellen Objekt, sondern auch auf übergeordneten Ebenen für eine Anfrage von Interesse sein. Abbildung 4.18 gibt einen grafischen Überblick über die Konzepte zusätzlicher Strategien, bevor diese im Folgenden detailliert besprochen werden.

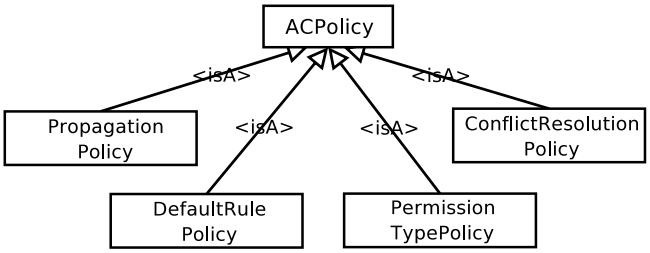

Abbildung 4.18: Klassen von Zugriffskontrollstrategien.

Weitergabestrategien auf Hierarchien: Auf Hierarchien können Weitergaben von Eigenschaften zwischen den Hierarchieebenen definiert werden. Ob diese Weitergaben von untergeordneten an übergeordnete Ebenen oder umgekehrt stattfindet, legt eine *Weitergabestrategie* (PropagationPolicy) fest. Jede Hierarchieinstanz muss eine *Weitergabestrategie* referenzieren. Die folgenden Beispiele verdeutlichen die unterschiedlichen *Weitergabestrategien*: Auf Objektverzeichnissen wird beispielsweise definiert, dass alle *Berechtigungen*, die ein Benutzer auf einem Verzeichnis hat, auch auf allen darin enthaltenen Ordnern oder Dateien gültig sind. *Berechtigungen* werden also von höheren Hierarchieebenen auf niederere übertragen. Bei Subjekthierarchien können Weitergaben aber genau umgekehrt definiert werden, da beispielsweise jeder Manager alle Rechte seiner untergebenen Angestellten besitzt.

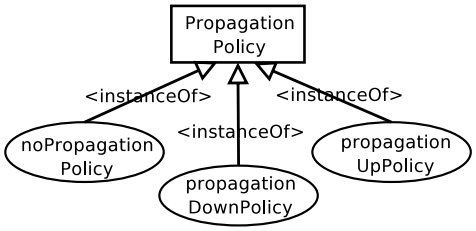

Abbildung 4.19: Strategien zur Weitergabe von Eigenschaften auf Hierarchien

Wie in Abbildung 4.19 dargestellt, lassen sich drei *Strategien* unterscheiden. *NoPropagationPolicy* legt fest, dass *Berechtigungen* nicht an andere Ebenen einer *Hierarchie* weitergegeben werden. Eine *Berechtigung* ist somit nur auf der Ebene gültig, auf der sie definiert wurde. Werden *Berechtigungen* von höheren an niedere Schichten weitergegeben, so wird dies durch die *Strategie propagationDownPolicy* modelliert. Eine

Weitergabe von niederen an höhere Schichten beschreibt die Instanz *propagationUpPolicy*. Alle diese *Strategien* werden als Instanzen des Konzepts *PropagationPolicy* definiert.

Konfliktauflösungsstrategien: Erlaubt ein Zugriffskontrollmodell positive und negative Berechtigungen, kann es zu Konflikten kommen. Diese können durch die in Abbildung 4.20 dargestellten Konfliktauflösungsstrategien (ConflictResolutionPolicies) beseitigt werden. Ziel der Beschreibung von Strategien ist es, Konflikte möglichst am Ort ihrer Entstehung zu behandeln. Daher unterscheidet man zwischen *hierarchischen* und *globalen Konfliktauflösungsstrategien*.

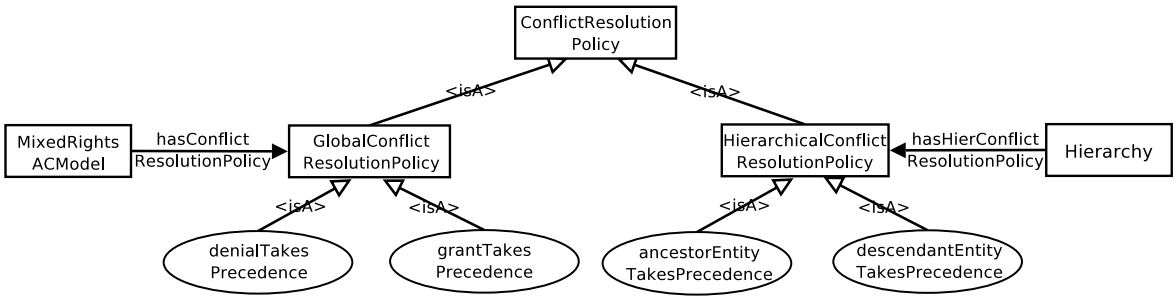

Abbildung 4.20: Konfliktauflösungsstrategien im Überblick

In Hierarchien kann es durch widersprüchliche Berechtigungen auf unterschiedlichen Ebenen und entsprechenden Weitergabestrategien zu Konflikten kommen. *Hierarchische Konfliktauflösungsstrategien* (HierarchicalConflictResolutionPolicy) versuchen einen Konflikt lokal, durch Betrachtung von Hierarchiebeziehungen der in Widerspruch stehenden Berechtigungselemente, aufzulösen. Die Instanz *ancestorEntityTakesPrecedence* beschreibt, dass *Berechtigungen* auf höheren Ebenen einer Elementhierarchie die auf niederen Stufen dominiert. Oder einfacher formuliert: Ist der Zugriff auf ein Element erlaubt, dann auch auf alle in der *Hierarchie* untergeordneten Elemente. Bei *descendantEntityTakesPrecedence* muss die ausschlaggebende Berechtigung in der Hierarchie möglichst nahe beim angefragten Element liegen. Der Zugriff auf eine Datei mit negativer *Berechtigung* wird hierbei unabhängig von den auf höheren Ebenen definierten *Berechtigungen* verboten.

Globale Konfliktauflösungsstrategien (GlobalConflictResolutionPolicy) entsprechen der höchsten Ebene von Konfliktauflösungsstrategien. Konnte ein Konflikt auf *Hierarchien* nicht aufgelöst werden oder aber ein Konflikt tritt aufgrund von Mitgliedschaften widersprüchlicher Gruppen auf, werden Zugriffsentscheidungen durch einfache Vorrangregeln herbeigeführt. Die Instanz *denialTakesPrecedence* verweigert einen Zugriff, falls in der Menge der widersprüchlichen *Berechtigungen* einer Anfrage mindestens eine negative *Berechtigung* enthalten ist. *GrantTakesPrecedence* gestattet einen Zugriff im Falle mindestens einer positiven *Berechtigung*.

4.3.4 Vereinfachung der Administration durch Konzeptinstantiierungen

Ein Ziel dieser Arbeit ist es, die Administration des Modells so einfach wie möglich zu gestalten. Um dies zu gewährleisten, wurden in vorherigen Abschnitten Konzepte aggregierter Zugriffskontrollelemente eingeführt. Wie jedes herkömmliche System hat auch das Zugriffskontrollmodell einen ausgezeichneten Benutzer, der Berechtigungen zur Ausführung sämtlicher Aktionen im System hat. In Anlehnung an UNIX Systeme wird dieser Benutzer als „*Root*“ bezeichnet. Die in Abbildung 4.21 dargestellten Konzeptinstanzen sind notwendig um diesem ausgezeichneten Benutzer alle möglichen Rechte auf einem System zuzuweisen.

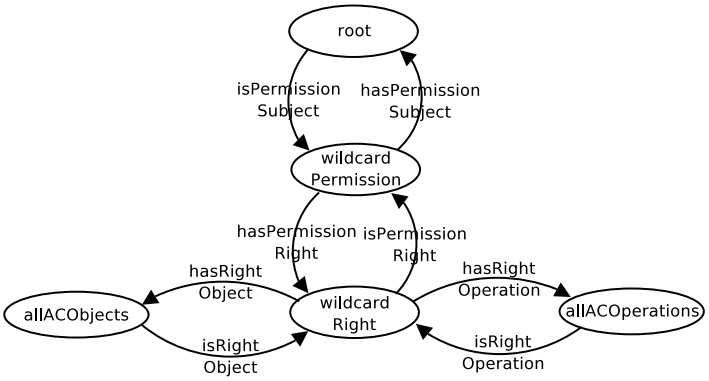

Abbildung 4.21: Konzeptinstantiierungen zur Vereinfachung der Administration.

Das atomare Subjekt *Root* kann alle im System definierten Operationen auf alle enthaltenen Objekte anwenden. Somit benötigt man Operations- und Objektgruppen, die alle Operationen (*allACOperations*) bzw. Objekte (*allACObjects*) enthalten. Die einzelnen Elemente des Zugriffskontrollmodells werden diesen Gruppen durch die in Abschnitt 5.2.4 vorgestellten Regeln zugeordnet. Somit bringt diese Zuordnung keinen zusätzlichen Administrationsaufwand mit sich, sondern erfolgt automatisch zur Berechnungszeit einer Anfrage. Diese Operations- und Objektgruppen werden, wie in Abbildung 4.21 dargestellt, daraufhin einem Zugriffsrecht namens *wildcardRight* zugewiesen. Durch die Berechtigungsinstanz *wildcardPermission* wird *Root* das entsprechende Recht zur Ausführung aller im System definierten Operationen vergeben. Der Begriff „wildcard“ im Namen der Instanzen deutet daraufhin, dass es sich um Platzhalter für konkrete Instanzen handelt.

4.4 Die Systemebene (ACModelSystems)

Wie in Abschnitt 2.3.2 auf Seite 16 beschrieben, unterscheidet die einschlägige Literatur regelbasierte, wahlfreie und rollenbasierte Zugriffskontrollmodelle. Da durch die semantische Modellierung von Benutzern und Objekten zusätzliches Wissen über diese existiert, wird in dieser Arbeit der Begriff der **kontextbasierten Zugriffskontrollmodelle** eingeführt. Bei einem kontextbasierten Modell werden bei einer Zugriffsanfrage stets zusätzliche Subjekt- oder Objektinformationen, wie beispielsweise das Alter eines Benutzers zur Entscheidungsfindung herangezogen.

Zur Modellierung der aus der Literatur bekannten Zugriffkontrollmodelle, sind Anpassungen und Erweiterungen der bisher besprochenen Konzepte notwendig. Ziel der Modellierung spezifischer Zugriffskontrollsysteme ist es, möglichst viele Regeln und Konzepte des allgemeinen Modells wiederzuverwenden. Ist dies nicht möglich, werden systemspezifische Konzepte entworfen, die mittels Regeln auf die allgemeinen Konzepte abgebildet werden. Dies gewährleistet, dass die Verwendung und die Berechnung von Zugriffsanfragen stets gleich vonstatten geht. Jedes der im Folgenden behandelten Modelle besitzt Eigenheiten, die durch Erweiterung vorhandener oder Modellierung neuer Konzepte beschrieben werden müssen.

Abbildung 4.22 zeigt die Konzepte zur Beschreibung unterschiedlicher Arten von Zugriffskontrollmodellen. Zur Unterscheidung von systemspezifischen Konzepten und allgemeinen Klassen der Kern- und Ergänzungsschicht werden alle in diesem Abschnitt eingeführten Konzepte als Subklassen des Konzepts *SpecificACEntity* modelliert.

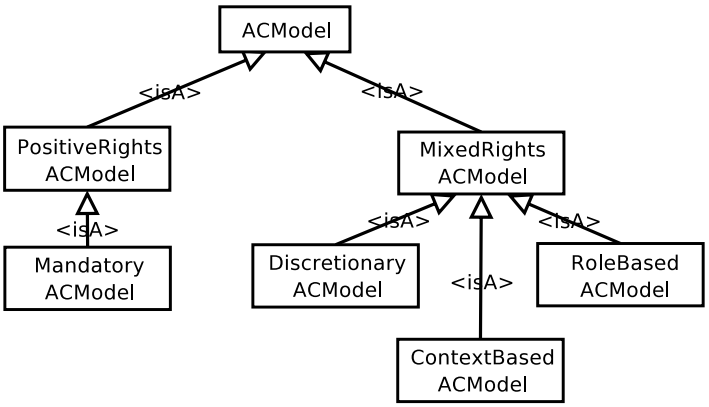

Abbildung 4.22: Erweiterung der Zugriffskontrollmodelle auf der Systemebene.

4.4.1 Regelbasierte Zugriffskontrollmodelle (MandatoryACModel)

Zentrale Aspekte der regelbasierten Zugriffskontrolle sind Klassifikationen von Subjekten und Objekten anhand von Sicherheitsstufen. Während beim allgemeinen Zugriffskontrollmodell Berechtigungen explizit durch Zuordnung eines Subjekts zu einem Zugriffsrecht vergeben werden, geschieht dies bei regelbasierten Systemen durch Vergleich der Sicherheitsstufen von Objekten und Subjekten. So ist eine Leseoperation typischerweise nur durchführbar, wenn die Klassifikation des Subjekts die des Objekts dominiert, um den Informationsfluss von niederen Vertrauensebenen auf höhere zu vermeiden.

In regelbasierten Systemen werden Berechtigungen nicht durch einen Administrator vergeben, sondern müssen zum Zeitpunkt einer Anfrage durch Vergleich der Subjekt- und Objektklassifikationen berechnet werden. Dieser Vergleich erfolgt durch Regeln, die in Abschnitt 5.4.1 auf Seite 80 besprochen werden. Eine besondere Herausforderung der Modellierung regelbasierter Konzepte ist es, Klassifikationen von Objekten und Subjekten so zu beschreiben, dass diese mittels Regeln einfach zu vergleichen sind.

In einem *regelbasierten Zugriffskontrollmodell* können ausschließlich positive Berechtigungen vergeben werden. Daher ist in Abbildung 4.22 das *regelbasierte Modell* als Subklasse des Konzepts des *positiven Zugriffskontrollmodells* dargestellt. Durch die explizite

Berechnung von Zugriffsberechtigungen zur Laufzeit bedeutet das Nichtvorhandensein einer passenden Berechtigung, dass dem Subjekt der Zugriff nicht gestattet ist. Somit handelt es sich bei einer regelbasierten Zugriffskontrolle stets um ein geschlossenes Zugriffskontrollsystem, d.h. dass alle Zugriffe die nicht explizit erlaubt sind, verboten werden. Dies wird durch geeignete Einschränkungen der `#hasDefaultRulePolicy` Relation des *regelbasierten Zugriffskontrollmodells* erreicht.

Abbildung 4.23 zeigt eine grafische Übersicht der wichtigsten zusätzlichen Konzepte eines regelbasierten Zugriffskontrollmodells, bevor diese in den darauf folgenden Abschnitten genauer betrachtet werden.

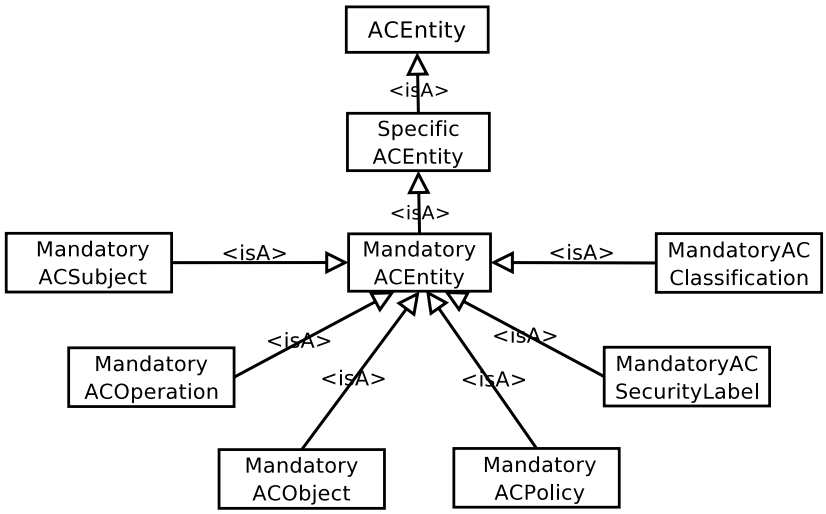

Abbildung 4.23: Überblick der wichtigsten Konzepte der regelbasierten Zugriffskontrolle.

Regelbasierte Sicherheitsstufen (MandatoryACSecurityLabel): *Sicherheitsstufen* bestimmen die Klassifikation von *Objekten* und *Subjekten* und beschreiben deren Vertraulichkeit bzw. Integrität. *Sicherheitsstufen* stellen Totalordnungen dar, d.h. eine Stufe dominiert alle ihr untergeordneten Stufen. Modelliert werden diese Inklusionsbeziehungen in Form einer *Hierarchie*, wobei jeder *Hierarchieknoten* genau einen übergeordneten und einen untergeordneten Knoten referenzieren kann. Abbildung 4.24 stellt den Aufbau von *Sicherheitsstufen* grafisch dar.

Wie Hierarchien von Subjekten und Objekten, bestehen auch hierarchische *Sicherheitsstufen* aus *Hierarchieknoten* (SecurityLabelNode) und der eigentlichen *Hierarchie* (SecurityLabelHierarchy). Da jedes Element einer Sicherheitsstufe auch Mitglied aller untergeordneten Stufen ist, wurde die Weitergabestrategie auf Hierarchien von Sicherheitsstufen auf die Instanz *propagationUpPolicy* beschränkt.

Regelbasierte Klassifikationen (MandatoryACClassification): Das Konzept der *Klassifikation* erlaubt eine einfache Vergleichbarkeit der Beziehungen von Subjekt- und Objektsicherheitsstufen durch Regeln. Die zuvor vorgestellten *Sicherheitsstufen* werden i.A. nicht systemweit sondern bezogen auf spezielle Anwendungen, auch *Kategorien* genannt, vergeben. Eine *Kategorie* wird hierbei als Objektgruppe modelliert, die alle Objekte umfasst, die einem Projekt zugeordnet sind.

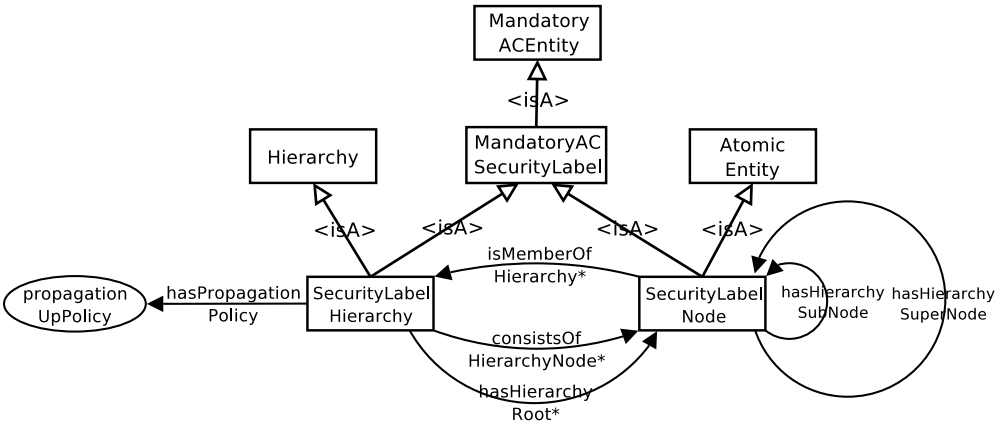

Abbildung 4.24: Struktur der Sicherheitsstufen regelbasierter Systeme.

Die Grundidee einer *Klassifikation* ist es, alle Subjekte und Objekte einer Kategorie anhand ihrer *Sicherheitsstufen* in einem gemeinsamen Konzept zu vereinen. Hierzu wurde eine *Klassifikationshierarchie* (ClassificationHierarchy) eingeführt, die analog zu den im System definierten Sicherheitsstufen aufgebaut ist. Das bedeutet, dass jeder Sicherheitsstufe und jeder Kategorie ein entsprechender *Klassifikationsknoten* (ClassificationNode) gegenübersteht. Subjekte und Objekte werden anhand ihrer Sicherheitsstufen den jeweiligen Ebenen einer *Klassifikation* zugewiesen. *Klassifkationshierarchien* werden automatisch generiert und ausschließlich als modellinternes Konzept verwendet, das für den Benutzer nicht sichtbar ist.

Das folgende Beispiel verdeutlicht die zugrunde liegende Idee: Eine Organisation beschäftigt sich mit zwei Projekten namens Apollo und Sirius. Diese Projekte werden im Folgenden als *Kategorien* bezeichnet. Objekten und Subjekten können in diesen Projekten die Sicherheitsstufen *Top Secret*, *Secret* und *Unclassified* zugewiesen werden. John wurde in Projekt Sirius als *Top Secret* klassifiziert. Benutzer Tom wird im Projekt Apollo die Sicherheitsstufe *Top Secret* und im Projekt Sirius *Unclassified* zugewiesen. Beispielhaft werden nun drei Objekte A, B und C eingeführt. A und B werden der Vertraulichkeitsstufe *Secret* im Projekt Apollo zugewiesen. C ist ein *Top Secret* Dokument des Projektes Sirius.

Anhand dieser Informationen werden nun die notwendigen Klassifikationshierarchien für die Kategorien Apollo und Sirius erzeugt und die Subjekte und Objekte passend eingeordnet. Wird ein neues Projekt dem Modell hinzugefügt, erfolgt dies durch Aufruf einer entsprechenden Methode der Admin API. Diese erzeugt anhand der im System definierten Sicherheitsstufen die notwendigen Instanzen zur Beschreibung einer Klassifikationshierarchie programmatisch. Die Einordnung der Objekte und Subjekte in die Klassifikation erfolgt durch Referenzierung des jeweiligen Klassifikationsknotens. Abbildung 4.25 zeigt das Ergebnis dieses Vorgangs exemplarisch. Beziehungen zwischen Sicherheitsstufen von Objekten und Subjekten eines Projekts können in Klassifikationen sehr einfach analysiert werden.

Klassifikationshierarchien sind ein Hilfskonzept, das die Beantwortung von Anfragen regelbasierter Systeme durch Kombination der Sicherheitsstufen von Objekten und

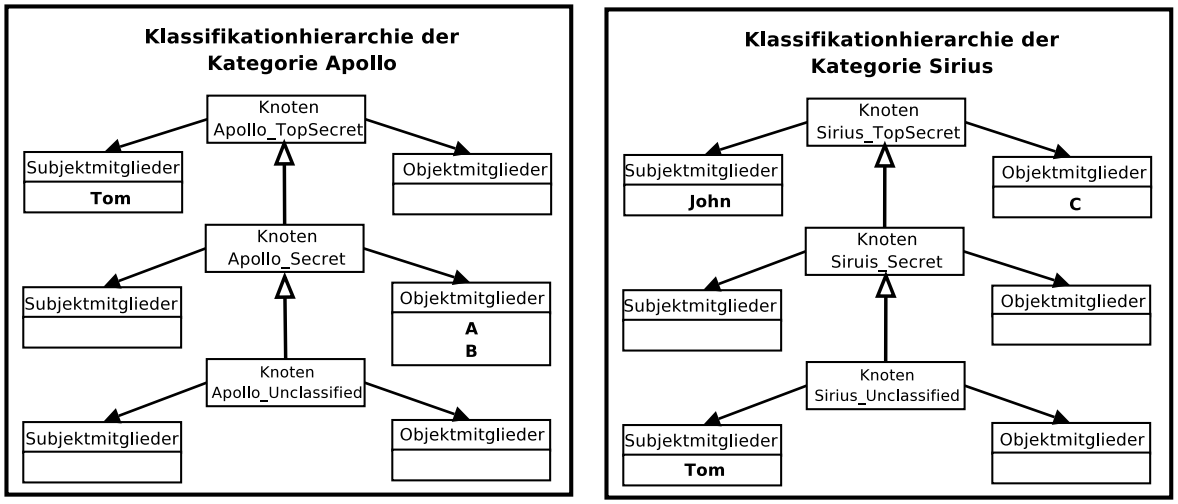

Abbildung 4.25: Beispiel einer generierten Klassifikation.

Subjekten in einem zentralen Konzept vereinfachen. Generell können die darin enthaltenen Informationen vollständig durch Regeln abgeleitet werden. Durch den notwendigen Vergleich der Kategorien und Sicherheitsstufen des Objekts und Subjekts einer Zugriffsanfrage unter Berücksichtigung der verschiedenen auf den Operationen definierten Weitergabestrategien, wären diese Regeln jedoch sehr komplex. Das zentrale Konzept der Klassifikationshierarchie verbessert zudem die Übersichtlichkeit des regelbasierten Zugriffskontrollmodells.

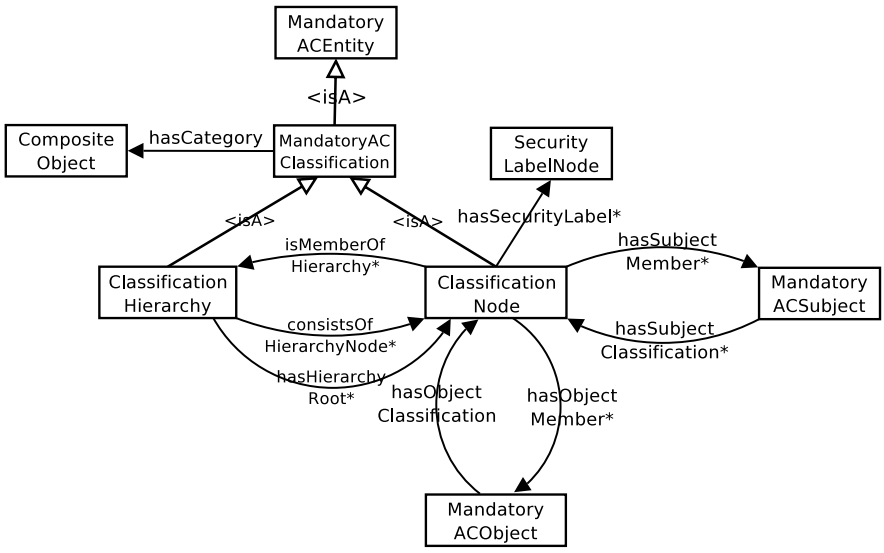

Abbildung 4.26: Struktur regelbasierter Klassifikationen.

Zur Modellierung einer Klassifikation werden die Konzepte *Klassifikationsknoten* (ClassificationNode) und *Klassifikationshierarchien* (ClassificationHierarchy) eingeführt. Wie aus Abbildung 4.25 ersichtlich wird, repräsentieren die *Klassifikationsknoten* hierbei die einzelnen Ebenen einer *Klassifikationshierarchie*. Die Anzahl der Ebenen und deren Anordnung entsprechen denen der Sicherheitsstufen. Um Objekte

und Subjekte referenzieren zu können, besitzt jeder Knoten entsprechende Relationen. Die *Klassifikationshierarchie* beschreibt die Beziehungen zwischen den einzelnen Stufen. Es ist wichtig zu verstehen, dass für jedes Projekt innerhalb eines regelbasierten Systems eine solche *Klassifikationshierarchie* angelegt werden muss. Abbildung 4.26 veranschaulicht die Konzepte und Relationen zur Modellierung einer Klassifikation grafisch.

Anpassung von Konzepten der Kernschicht: Um die speziellen Eigenschaften regelbasierter Elemente beschreiben zu können, sind eine Reihe von konzeptionellen Erweiterungen der allgemeinen Zugriffskontrollelemente notwendig.

Da jedes *regelbasierte Subjekt* (MandatoryACSubject) in einem regelbasierten System klassifiziert werden muss, weist dieses eine zusätzliche Referenz zu einem *Klassifikationsknoten* auf. Die Zuweisung zu diesem Knoten erfolgt für den Anwender transparent, da diese Referenz nach Auswahl einer Kategorie und einer Sicherheitsstufe vom System automatisch generiert wird. Auch *regelbasierte Subjekte* werden um *atomare*, *zusammengesetzte* und *hierarchische regelbasierte Subjekte* erweitert. Abbildung 4.27 stellt diese Erweiterungen dar.

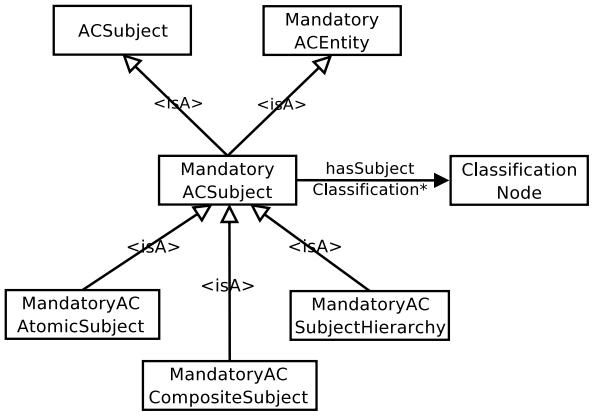

Abbildung 4.27: Struktur regelbasierter Subjekte.

Wie regelbasierte Subjekte werden auch *regelbasierte Objekte* (MandatoryACObject) durch Zuweisung einer Sicherheitsstufe klassifiziert. Durch Mitgliedschaft in einer *Objektgruppe*, die die Kategorie des Objektes beschreibt, ist jedes Objekt eindeutig einem Klassifikationsknoten zugeordnet.

Bei regelbasierten Systemen wird für jede *regelbasierte Operation* (MandatoryACOperation) einzeln festgelegt, wie die Sicherheitsstufen von Objekten und Subjekten zueinander liegen müssen, damit ein Zugriff gestattet wird. Dies wird wie in Abbildung 4.28 dargestellt durch Zuweisung einer geeigneten *Weitergabestrategie* (ClassificationPropagationPolicy) beschrieben.

Strategien zur Interpretation von Klassifikationsbeziehungen (MandatoryACPolicy): Wie bereits erwähnt, werden Berechtigungen in regelbasierten Systemen basierend auf Klassifikationsbeziehungen zwischen Subjekten und Objekten vergeben. Eine *regelbasierte Zugriffskontrollstrategie* (MandatoryACPolicy) beschreibt die Art der mög-

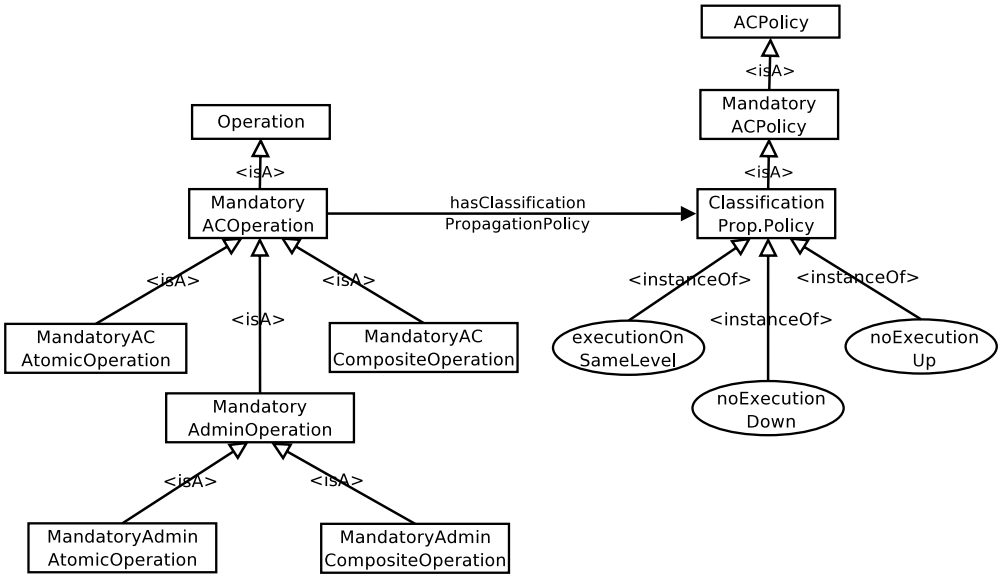

Abbildung 4.28: Regelbasierte Operation und Strategien zur Weitergabe von Berechtigungen zwischen Klassifikationsstufen.

lichen Beziehungen einer *regelbasierten Operation*. Folgende Strategien, die auch in Abbildung 4.28 dargestellt sind, werden unterschieden:

ExcecutionOnSameLevel: Diese Richtlinie erlaubt den Zugriff auf ein Objekt nur dann, wenn die Sicherheitsstufe des Subjekts mit der des Objekts übereinstimmt. Sind somit Subjekt und Objekt dem gleichen Klassifikationsknoten zugewiesen, ist der Zugriff erlaubt.

NoExcecutionDown: Mit dieser Strategie darf eine Operation nur dann ausgeführt werden, wenn die Klassifikation des Objekts die des Subjekts dominiert. Das bedeutet, dass das Objekt in der Klassifikationshierarchie der gleichen Ebene bzw. einem Superknoten der Subjektklassifikation zugeordnet sein muss. *NoExecutionDown* wird verwendet, um die für einen Schreibzugriff geltende Regel „No write down“ zu modellieren.

NoExcecutionUp: Dieses Konzept beschreibt, dass ein Subjekt nur dann ein auf ein Objekt zugreifen darf, wenn es die Klassifikation des Objektes dominiert. Es handelt sich somit um eine Verallgemeinerung der Regel „No read up“.

Wie aus diesen zusätzlichen Konzepten ersichtlich wird, werden Zugriffsanfragen in einem regelbasierten System durch die Zuordnung von Subjekten und Objekten zu Klassifikationsknoten und dem Vergleich dieser Knoten beantwortet.

Wird eine Zugriffsanfrage gestellt, wird in einem ersten Schritt die Kategorie des angefragten Objekts extrahiert, d.h. festgestellt in welchem Projekt es Verwendung findet. Daraufhin wird die Klassifikation des Subjekts in der Kategorie des Objekts festgestellt. Anhand der in der Operation referenzierten Strategie wird festgestellt, welche Objekt- und Subjektbeziehungen für die Vergabe von Berechtigungen notwendig sind. Durch Vergleich der Klassifikationsknoten des Objekts und Subjekts wird daraufhin ein Zugriff gestattet oder nicht. Die Regeln, die dies formal beschreiben, werden in Abschnitt 5.4.1 vorgestellt.

4.4.2 Wahlfreie Zugriffskontrollmodelle (DiscretionaryACModel)

Der zentrale Aspekt wahlfreier Zugriffskontrollsysteme ist die Definition von Eigentum an Objekten. Jedes Objekt besitzt einen eindeutigen Eigentümer, der zur Ausführung sämtlicher Operationen auf diesem Objekt berechtigt ist. Der Besitzer kann zudem Berechtigungen an diesem Objekt an andere Subjekte weitergeben oder diese auch wieder entziehen. Im Gegensatz zu anderen Systemen in denen nur ausgezeichnete Subjekte Administrationsaufgaben wahrnehmen können, ist dies hier prinzipiell jedem Subjekt erlaubt. Operationen zum Wechsel eines Eigentümers oder zur Delegation von Rechten dürfen nur dann ausgeführt werden, wenn das anfragende Subjekt Besitzer eines Objekts ist, oder aber wenn er eine Berechtigung auf einem Objekt durch Delegation erhalten hat.

Während bei allgemeinen Systemen Berechtigungen durch explizite Zuweisung von Subjekten zu Zugriffsrechten vergeben werden, geschieht dies in wahlfreien Systemen zum Zeitpunkt der Erzeugung von Objekten bzw. der Weitergabe oder Rücknahme von Berechtigungen. Wird ein Objekt erzeugt, so erhält der Eigentümer durch Regel 5.35 Zugriff auf alle Operationen auf diesem Objekt. Bei der Weitergabe von Zugriffsrechten werden für den Empfänger der Weitergabe neue Berechtigungsinstanzen in die Wissensbasis eingebracht. Bei der Rücknahme einer Delegation wird die entsprechende Berechtigung wieder gelöscht.

Die notwendigen Anpassungen und Erweiterungen zur Beschreibung eines wahlfreien Zugriffskontrollmodells werden in den folgenden Abschnitten untersucht.

Anpassung von Konzepten der Kernschicht: *Wahlfreie Subjekte* (DiscretionaryACSubject) können durch eine zusätzliche Relation als Besitzer von *wahlfreien Objekten* (DiscretionaryACObbject) definiert werden. Die Objekte ihrerseits enthalten eine inverse Relation, die den jeweiligen Eigentümer referenziert. Diese zusätzlichen Relationen werden in Abbildung 4.29 dargestellt.

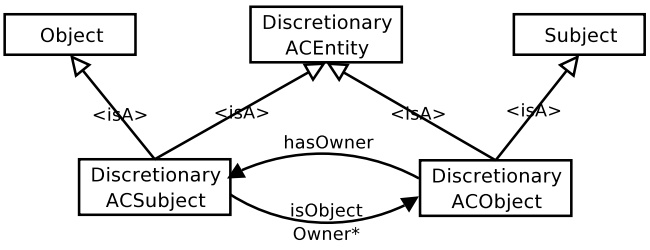

Abbildung 4.29: Zusätzliche Relationen von wahlfreien Subjekten und Objekten.

Zur Modellierung der Weitergabe von Zugriffsrechten, der sogenannten Delegation, weist ein wahlfreies Zugriffsrecht (DiscretionaryACAccessRight) eine zusätzliches Attribut `#isDelegable` auf. Der darin beschriebene Wahrheitswert legt fest, ob ein Zugriffsrecht weitergegeben werden darf oder nicht. So könnte man beispielsweise die Weitergabe eines Löschrechtes an andere Subjekte unterbinden.

Strategien für Weitergabe und Rücknahme von Rechten (DiscretionaryACPolicy): In einem wahlfreien System können unterschiedliche Strategien für die Weitergabe und Rücknahme von Rechten definiert werden. Diese Strategien werden durch Regeln umgesetzt und definieren Beschränkungen für die Weitergabe und Rücknahme von Rechten. Eine Übersicht der verwendeten Konzepte gibt Abbildung 4.30.

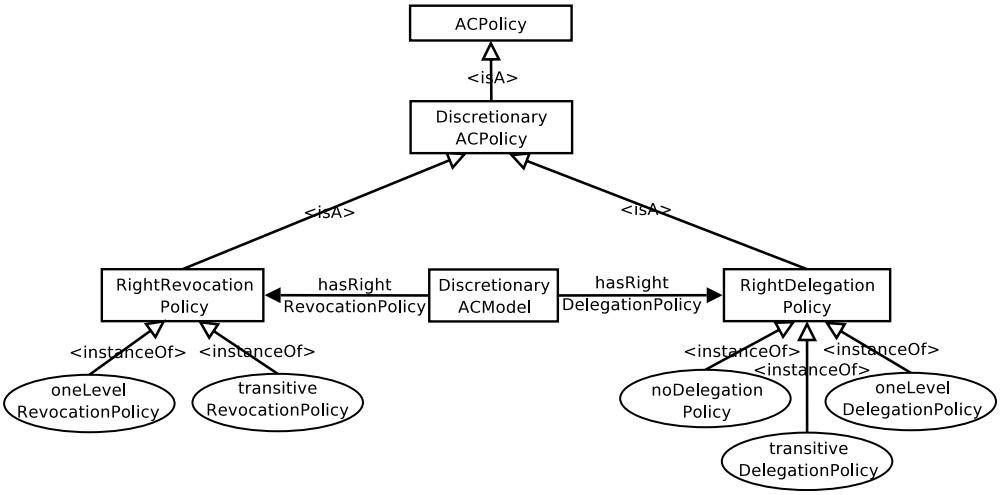

Abbildung 4.30: Strategien wahlfreier Systeme.

Eine *Weitergabestrategie* (RightDelegationPolicy) legt fest unter welchen Voraussetzungen ein Subjekt Rechte weitergeben darf. Jedes wahlfreie Zugriffskontrollmodell muss, wie in Abbildung 4.30 dargestellt, eine Referenz zu einer *Weitergabe-* und *Rücknahmestrategie* aufweisen. Konkrete *Weitergabestrategien* können die Weitergabe von Rechten komplett verbieten (*noDelegationPolicy*), eine Weitergabe von Rechten nur dem Besitzer eines Objektes erlauben (*oneLevelDelegationPolicy*) oder jedem Subjekt, das eine Operation auf dem jeweiligen Objekt ausführen darf, die Weitergabe gestatten (*transitiveDelegationPolicy*). Gerade die zuletzt beschriebene beliebige Weitergabe von Zugriffsrechten stellt hohe Anforderungen an die Rücknahme von Weitergaben.

Um die Reichweite der Rücknahmen von Delegationen festzulegen, werden *Rechterücknahmestrategien* (RightRevocationPolicies) beschrieben. Rücknahmen von Rechten in Systemen, die beliebige Weitergaben erlauben, können transitiv (*transitiveRevocationPolicy*) oder nichttransitiv (*oneLevelRevocationPolicy*) erfolgen. Während bei *nichttransitiven Rücknahmestrategien* nur unmittelbare Weitergaben zurückgenommen werden, umfasst die transitive Rücknahme auch Weitergaben, die von Empfängern einer Delegation durchgeführt wurden.

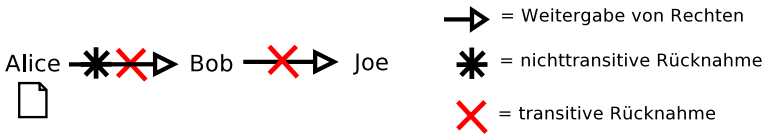

Abbildung 4.31: Transitive Rücknahme von weitergegebenen Rechten.

Abbildung 4.31 zeigt ein Beispiel transitiver und nichttransitiver Rücknahme von delegierten Rechten. Gegeben sind drei Subjekte Alice, Bob und Joe. Alice hat zuvor ein Recht an Bob weitergegeben, der dies seinerseits an Joe delegierte. Zieht Alice nun ihre Berechtigungen zurück, so wird die Weitergabe von Bob an Joe nur im Falle einer *transitiven Rücknahmestrategie* widerrufen.

Historie der Rechteweitergabe (RightDelegationHistory): Die Rücknahme von Rechten ist keine triviale Angelegenheit wie das in Abbildung 4.32 dargestellte Szenario vor Augen führen soll.

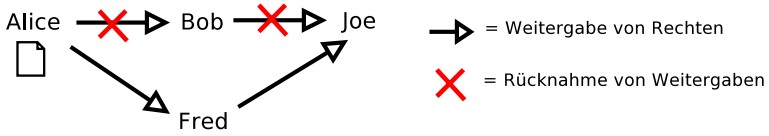

Abbildung 4.32: Beispiel von Weitergaben und transitiven Rücknahmen von Zugriffsrechten.

In diesem Beispiel ist Alice Eigentümer einer Datei. Sie delegiert Zugriffsrechte auf der Datei an Bob und Fred. Bob und Fred führen unabhängig voneinander Weitergaben dieses Rechts an Joe durch. Da Joe die Berechtigung auf dem Objekt durch Rechteweitergaben von Bob und Fred erhalten hat, sollte eine transitive Rücknahme der Delegation von Alice an Bob die Rechte von Joe nicht beeinflussen (vgl. Abbildung 4.32). Um dies zu gewährleisten, muss jedoch die komplette Historie der Rechteweitergaben im Modell aufgezeichnet werden. Nur so kann im Falle von Rücknahmen entschieden werden, ob eine konkrete Delegation betroffen ist oder nicht.

Diese Aufzeichnungen werden konzeptionell durch das in Abbildung 4.33 beschriebene Konzept der *Rechteweitergabehistorie* (DiscretionaryRightDelegationHistory) beschrieben. Für jede Weitergabe wird das initiierende Subjekt der Delegation, das weitergegebene Zugriffsrecht und der Empfänger der Weitergabe persistent im Modell abgelegt.

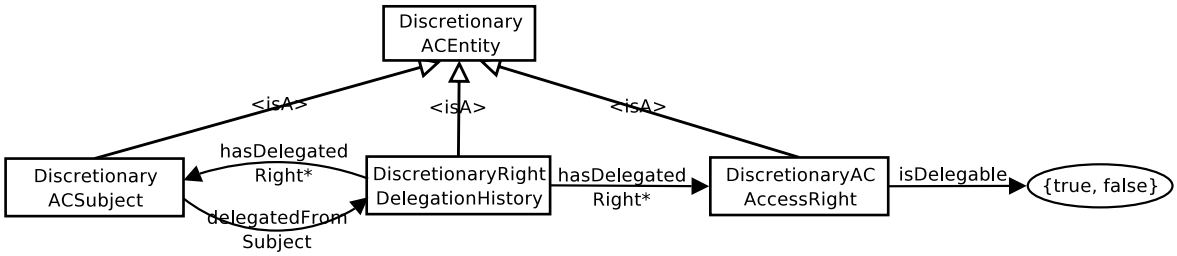

Abbildung 4.33: Konzepte zur Beschreibung der Historie der Rechteweitergabe.

Wahlfreie administrative Operationen (DiscretionaryAdminOperation): Wie bereits beschrieben, erlauben wahlfreie Systeme im Gegensatz zu anderen Zugriffskontrollsysteme jedem Eigentümer die Vergabe von Berechtigungen. Jeder Benutzer ist somit ein Administrator seiner eigenen Objekte und kann den Zustand der Wissensbasis durch Ausführung von Operationen verändern. Bevor eine administrative Operation ausgeführt werden darf, muss durch entsprechende Regeln (vgl. Abschnitt 5.4.2) überprüft werden, ob der Anfragesteller eine Berechtigung zur Durchführung dieser Operation besitzt. Ist dies erfüllt, muss die Modelländerung durch Aufruf einer geeigneten Methode der in Abschnitt 6.2.2 beschriebenen Admin API ausgeführt werden.

Operationen zur Weitergabe und Rücknahme von Rechten sowie zum Wechsel des Eigentümers sind grundlegende Eigenschaften wahlfreier Systeme. Diese werden als

Instanzen des Konzepts der *wahlfreien Administrationsoperationen* (DiscretionaryAdminOperation) modelliert. Abbildung 4.34 gibt eine grafische Übersicht der Instanzen.

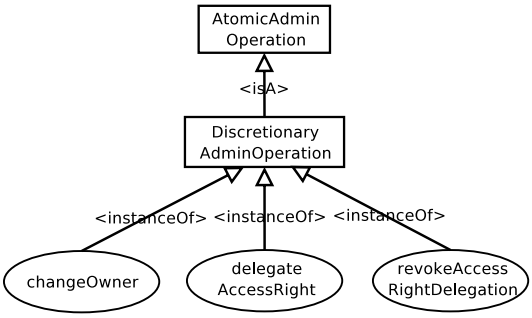

Abbildung 4.34: Instanzen zur Beschreibung wahlfreier administrativer Operationen.

Die administrative Operation zum Wechsel des Eigentümers (`changeOwner`) erlaubt dem Eigentümer die Übertragung sämtlicher Rechte auf einem Objekt an ein anderes Subjekt. Diese Aktion darf nur durchgeführt werden, wenn das durchführende Subjekt der Eigentümer des Objekts ist.

Soll eine Weitergabe von Rechten durchgeführt werden, wird eine zweistellige Operation namens `delegateAccessRight` aufgerufen. Argumente dieser Operation sind das Empfängersubjekt und das zu delegierende Recht. Bevor diese Operation ausgeführt werden kann, muss überprüft werden, ob das Recht delegierbar ist und ob die Strategien des wahlfreien Zugriffskontrollmodells die Weitergabe von Rechten zulassen. Abhängig von der konkreten Weitergabestrategie muss das Subjekt, welches die Weitergabe veranlasst hat, entweder der Eigentümer des Objekts sein oder aber zumindest das weiterzugebende Recht durch Delegation erhalten haben (vgl. Regeln 5.36 und 5.37). Die Ausführung dieser Operation erzeugt eine neue Berechtigungsinstanz für den Empfänger der Weitergabe und die dazugehörige Instanz der Delegationshistorie.

Um Rücknahmen delegierter Rechte zu ermöglichen, wurde die Operation `revokeAccessRightDelegation` modelliert. Auch hier sind die Argumente das Subjekt, dem das Recht entzogen werden soll, und das zu entziehende Zugriffsrecht. Eine Rücknahme von Berechtigungen ist nur durch das Subjekt möglich, dass dieses Recht zuvor auch weitergegeben hat. Abhängig davon ob es sich um eine transitive Rücknahme handelt, werden bei der Ausführung dieser Operation nicht nur direkte Rechteweitergaben sondern alle Weitergaben von delegierten Rechten zurückgenommen. Welche Subjekte von dieser Rücknahme betroffen sind, kann mittels Regeln festgestellt werden. Der eigentliche Löschvorgang kann jedoch nur durch Programmcode erfolgen, da die entsprechenden Fakten aus der Wissensbasis entfernt werden müssen.

4.4.3 Rollenbasierte Systeme

Wie in Abschnitt 2.3.2 bereits erwähnt, ist die Grundidee rollenbasierter Systeme Aufgabenbereiche in Organisationen abzubilden. Die wichtigsten Elemente eines rollenbasierten Zugriffskontrollmodells werden im Folgenden dargestellt.

[San96] definiert eine Rolle als eine „*benannte Ansammlung von Benutzern und Berechtigungen und evtl. anderen Rollen*“. Obwohl eine *Rolle* (Role) aus Anwendungssicht mit einer Benutzergruppe i.A. nicht vergleichbar ist, wird diese als Erweiterung des Konzepts der Subjektgruppe (CompositeSubject) modelliert. Aufgrund der Eigenschaften von Subjektgruppen kann eine Rolle aus Benutzern, Benutzergruppen oder anderen Rollen zusammengesetzt sein. Um die zusätzliche Zuweisung von Zugriffsrechten zu einer Rolle zu erlauben, besitzt eine Rolle eine zusätzliche Relation `#hasGrantedRight`. Jede Rolle wird durch eine Berechtigungsinstanz, die Rolle und zugewiesene Zugriffsrechte vereint, im Modell abgebildet. Wird einer Rolle ein neues Zugriffsrecht zugewiesen, entspricht dies dem Hinzufügen eines zusätzlichen Rechts zur zugehörigen Berechtigung. Abbildung 4.35 stellt die Beziehungen zwischen einer Rolle und einem Zugriffsrecht grafisch dar.

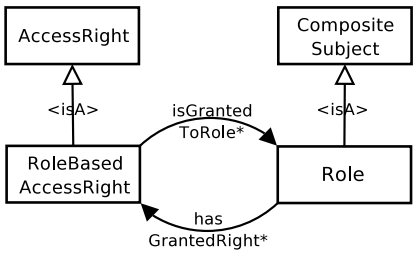

Abbildung 4.35: Beziehung zwischen Rollen und rollenbasierten Zugriffsrechten.

Damit Rollen die üblicherweise hierarchischen Organisationsstrukturen von Unternehmen nachbilden können, wird das Konzept der *hierarchischen Rolle* (HierarchicalRole) definiert. Hierbei handelt es sich um ein Subkonzept einer *Subjekthierarchie*. Da in einer *Rollenhierarchie* Zugriffsrechte stets von allgemeinen Rollen an spezialisiertere weitergegeben werden, ist die Weitergabestrategie von Berechtigungen auf die Instanz *propagationUpPolicy* beschränkt. Die Struktur dieser Rollenhierarchien zeigt Abbildung 4.36.

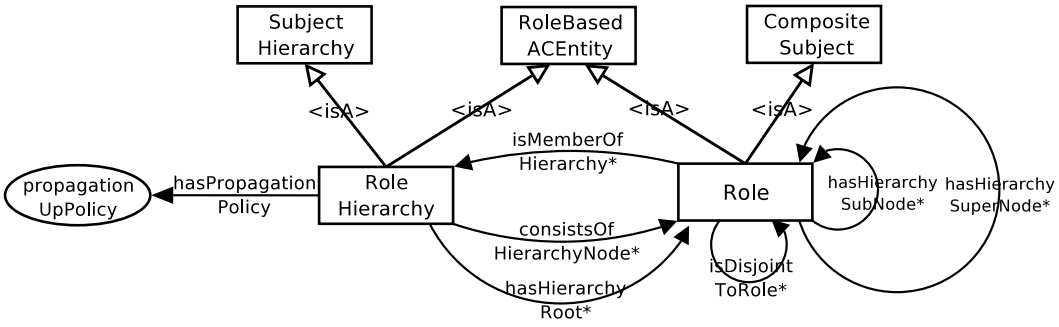

Abbildung 4.36: Konzepte zur Modellierung von Rollen und Rollenhierarchien.

In rollenbasierten Systemen können zusätzliche Einschränkungen auf Rollen definiert werden. So lässt sich beispielsweise eine Trennung von Zuständigkeiten (engl.: Separation of Duties) definieren, so dass wichtige Aufgaben nicht von einer einzigen sondern von mehreren unterschiedlichen Personen durchgeführt werden müssen. Um dies zu modellieren werden Rollen, die für eine Aufgabe zuständig sind, durch die Relation `#isDisjointToRole` disjunkt zueinander definiert. Ein Subjekt, dass Mitglied einer Rolle ist, kann somit nicht

Mitglied einer als disjunkt definierten Rolle sein. Häufig werden auch Kardinalitätsbedingungen spezifiziert, die beispielsweise eine maximale Anzahl von Subjekten pro Rolle festlegen. Aus administrativer Sicht wäre eine Beschränkung der Zuordnung von Subjekten zu Rollen derart denkbar, dass ein Subjekt nur dann einer Rolle zugewiesen werden kann, wenn dieses bereits Mitglied einer anderen Gruppe ist. Dies ist nur ein kleiner Auszug möglicher Beschränkungen, die im Rahmen dieser Arbeit nicht erschöpfend behandelt werden.

Zugriffsanfragen in rollenbasierten Systemen können entweder direkt für eine Person oder aber für eine Rolle, innerhalb derer ein Subjekt eine Operation ausführen möchte, gestellt werden. Letzteres bietet bei hierarchischen Rollen die Möglichkeit eine Tätigkeit nicht unter der explizit zugewiesenen Rolle sondern auf untergeordneten Rollen auszuführen. Da *Rollen* als Subkonzepte von *Subjektgruppen* und *Rollenhierarchien* als Subkonzepte von *Subjekthierarchien* modelliert sind, können die in Abschnitt 5.2.2 besprochenen Regeln ohne Anpassungen auf rollenbasierte Systeme angewendet werden.

4.4.4 Kontextbasierte Systeme

Ein kontextbasiertes Modell nutzt Informationen von Subjekten und Objekten um anhand bestimmter Eigenschaften Autorisierungsanfragen zu beantworten. Da ein solches System natürlich sehr vielgestaltig sein kann, ist ein allgemeines Konzept schwer zu beschreiben. Im Allgemeinen kann ein kontextbasiertes System durch Erweiterungen der Konzepte Kern- und Ergänzungsebenen beschrieben werden. Typischerweise untergliedert sich die Umsetzung eines kontextbasierten Systems in folgende Schritte:

1. **Definition von Elementgruppen:** Subjekte oder Objekte eines kontextbasierten Systems werden anhand von individuellen Eigenschaften Elementgruppen zugeordnet. Wollte man beispielsweise den Zugriff auf Ressourcen jugendlichen Benutzern verbieten, so könnte man die Benutzergruppen *Adults* und *Minors* einführen, in die Personen zur Anfragezeit in Abhängigkeit ihres Alters eingeordnet werden.

2. **Erstellen von Berechtigungen:** Durch Definition von Berechtigungen für die jeweiligen Elementgruppen wird den Mitgliedern dieser Gruppe der Zugriff gestattet oder verboten.

3. **Definition von Regeln:** Der zentrale Schritt ist nun die Definition von Regeln, die Elemente anhand ihrer Eigenschaften den obigen Gruppen zuordnen. Am Beispiel der Altersbeschränkung müsste eine Regel das Geburtsdatum des Benutzers mit dem aktuellen Datum vergleichen und alle volljährigen Benutzer der Gruppe *Adults* zuordnen.

Abhängig von dem jeweiligen konkreten Szenario ist dieses Vorgehen jedoch nicht trivial.

Kapitel 5

Modellierung II – Regeln zur Modellauswertung

Regeln werden verwendet, um über einfache Relationen hinaus Beziehungen zwischen Konzeptinstanzen zu beschreiben. Während das zuvor vorgestellte ontologiebasierte Modell der Beschreibung von Elementen eines Zugriffskontrollsystems dient, erlauben Regeln die Interpretation des Modells durch Schlussfolgerungen auf den Fakten der Wissensbasis. Diese Auswertung des Modells durch Regeln entspricht der Beschreibung der Logik zur Beantwortung von Zugriffsanfragen. Die Verwendung von Regeln ermöglicht aus explizit in der Wissensbasis enthaltenen Fakten neue Fakten abzuleiten. Für die Modellierung eines Zugriffskontrollsystems werden Regeln vor allem für die Beantwortung von Zugriffsanfragen verwendet. Durch Regeln lassen sich beispielsweise Konflikte zwischen negativen und positiven Berechtigungen durch Auswertung von Strategien auflösen.

Wie bereits erwähnt, trägt die Auswertung des Modells durch Regeln zur Flexibilität des Modells bei, da Programmcode zur Beantwortung von Anfragen nicht notwendig ist.

5.1 Konzept zur Modellinterpretation durch Regeln

Hauptaugenmerk der Modellinterpretation liegt auf der Beantwortung von Zugriffsanfragen. Eine Berechtigungsanfrage wird hierbei durch einfache Abfrage des Nicht-OWL-Prädikats `accessGranted` gestartet. Vereinfacht beschrieben wird diesem Prädikat ein Tripel <*subject, object, operation*> übergeben und die Auswertung der zugehörigen Regeln liefert einen Wahrheitswert als Ergebnis zurück. Dieser Wahrheitswert beschreibt, ob ein Zugriff erlaubt ist oder nicht.

Die Beantwortung von Zugriffsanfragen erfolgt unter Verwendung von Regeln, die schrittweise die Wissenbasis interpretieren. In den folgenden Abschnitten werden diese Regeln in einem Bottom-Up Ansatz erklärt. Aufsetzend auf den in der Wissensbasis explizit formulierten Fakten werden neue Fakten abgeleitet, die ihrerseits wiederum in weiteren Regeln Verwendung finden. Dies wird solange fortgesetzt, bis alle für eine Zugriffsentscheidung relevanten Informationen auf das Prädikat `accessGranted` abgebildet wurden.

Im Allgemeinen lassen sich die Regeln in drei unterschiedliche Aufgabenbereiche unterteilen:

1. *Regeln zur Übertragung von Berechtigungen aggregierter Elemente:* Aggregierte Ele-

mente vereinfachen die Administration des Zugriffskontrollmodells. Statt Berechtigungen auf jedem Element einzeln zu vergeben, werden diese zu größeren Einheiten, wie Gruppen oder Hierarchien, zusammengefasst. Da Systemanfragen typischerweise auf Ebene atomarer Elemente, wie z.B. einer konkreten Person, durchgeführt werden, müssen zur Entscheidungsfindung Berechtigungen, die auf zusammengesetzten Elementen vergeben wurden, auf enthaltene atomare Elemente übertragen werden. Am Beispiel von Hierarchien können auch Weitergabestrategien zwischen Ebenen der Hierarchie definiert werden. Zur weiteren Vereinfachung der Administration werden zudem einem speziellen Benutzer namens „*Root*“ Berechtigungen auf allen Elementen des Zugriffskontrollsystems zugestanden.

Die in diesen Fällen notwendige Berechtigungsweitergabe von zusammengesetzten auf atomare Elemente wird durch Regeln formalisiert, die in Abschnitt 5.2 vorgestellt werden.

2. *Regeln zur Konfliktauflösung:* Durch die Möglichkeit positive und negative Berechtigung zu vergeben, kann es bei Anfragen zu Konflikten kommen. Für zwei unterschiedliche Gruppen wurde beispielsweise eine positive bzw. eine negative Berechtigung auf demselben Objekt vergeben. Ist ein Subjekt nun Mitglied beider Gruppen, so tritt bei einer entsprechenden Anfrage ein Konflikt auf. Konflikte werden unter Auswertung geeigneter Strategien, die im Abschnitt 4.3.3 auf Seite 46 erläutert werden, durch Regeln aufgelöst.

   Grundidee der Konfliktauflösung ist es, Konflikte am Ort ihrer Entstehung zu beheben, z.B. innerhalb einer Hierarchie. Ist dies jedoch nicht möglich, wird dieser Konflikt an die nächst höhere Ebene weitergegeben und dort ausgewertet. Das Modell unterstützt derzeit Regeln zur globalen und hierarchischen Konfliktauflösung. Weitere Ebenen oder zusätzliche lokale Konfliktauflösungen sind jedoch denkbar.

   Konflikte innerhalb von Hierarchien können beispielsweise durch Weitergabe von Berechtigungen von niederen auf höhere Hierarchieebenen entstehen. Zur Auflösung von Hierarchiekonflikten gibt es hierarchische Konfliktauflösungsstrategien, die z.B. besagen, dass Berechtigungen auf höheren Schichten dominieren. Zur hierarchischen Konfliktauflösung würden entsprechende Regeln, abhängig von der jeweiligen Strategie, Berechtigungen auf über- bzw. untergeordneten Ebenen betrachten und ggf. eine positive oder negative Entscheidung an die globale Ebene der Konfliktauflösung weitergeben. Kann ein Konflikt innerhalb einer Hierarchie nicht aufgelöst werden, da z.B. dem angefragten Element direkt ein positives und negatives Recht zugewiesen wurde, wird der Konflikt an die globale Konfliktauflösungsebene weitergereicht. Durch diese schrittweise Weitergabe von Konflikten von lokalen auf globale Konfliktauflösungsebenen wird gewährleistet, dass Mitgliedschaften in unterschiedlichen Hierarchien bzw. Gruppen berücksichtigt werden.

   Auf globalem Niveau werden dann Konflikte durch einfache Regeln aufgelöst, die beispielsweise besagen, dass ein positives Recht bedeutender ist als ein negatives. Zudem bestimmen Regeln das Antwortverhalten, wenn keinerlei Informationen für eine Zugriffsanfrage im Modell vorhanden sind.

   Die vollständige Konfliktauflösung wird durch Regeln beschrieben. Zur Weitergabe von Rechten von lokaler auf globale Ebene und der Auflösung von Konflikten werden

eine Reihe zusätzlicher KAON2 Nicht-OWL-Prädikate eingesetzt, die in Abschnitt 5.3 sukzessive eingeführt werden.

Die Regeln zur Konfliktauflösung, allen voran für Konflikte auf Hierarchien, sind sehr komplex und beeinflussen die Laufzeit des automatischen Schließens negativ. Diese Regeln sind jedoch nur dann notwendig, wenn ein gemischtes Zugriffskontrollmodell mit positiven und negativen Berechtigungen beschrieben wird. Dies ist in praktischen Anwendungen jedoch eher selten. Diese Regeln verdeutlichen jedoch sehr anschaulich, welche Mächtigkeit semantischen Technologien in Form von Ontologien und Regeln bei der Beschreibung komplexer Sachverhalte besitzen.

3. *Regeln der Systemschicht:* Auf der in Abschnitt 4.4 auf Seite 48 vorgestellten Systemebene werden Erweiterungen zur Beschreibung konkreter Zugriffskontrollmodelle eingeführt. Regeln werden nun dazu verwendet Eigenheiten dieser Systeme auf die allgemeinen Elemente der Kern- und Ergänzungsebene abzubilden und Zugriffsentscheidungen herzuleiten. Diese Regeln werden in Abschnitt 5.4 vorgestellt.

4. *Regeln zur Konsistenzprüfung:* Durch direkten Zugriff des Administrators auf das Zugriffskontrollmodell kann es zu Inkonsistenzen kommen. Ein Beispiel hierfür wäre die Relation `hasPermissionSubject`, die einer Berechtigung eine Subjektinstanz zuweist. Würde der Administrator dieser Relation jedoch ein Objekt zuweisen, ist das Modell inkonsistent, was zu Problemen beim Reasoning führen kann. Um dies zu verhindern, werden zusätzliche Regeln definiert, die solche Inkonsistenzen überprüfen und ggf. einen Fehler generieren. Ein Fehler wir in Form eines Nicht-OWL-Prädikats modelliert, das durch eine entsprechende Anfrage abgeleitet werden kann. Eine kurze Betrachtung der Regeln zur Konsistenzprüfung des Modells findet in Abschnitt 5.5 statt.

Die Darstellung der Regeln der folgenden Abschnitte erfolgt in einer abstrakten Syntax, die an die Darstellung des SWRL Plugins von Protégé angelehnt ist. Einstellige Prädikate, die mit einem Großbuchstaben beginnen, stellen Klassenatome dar. Beispielsweise bedeutet `Subject(?x)`, dass `x` eine Instanz des Konzepts `Subject` sein muss. Zweistellige Prädikate entsprechen OWL Relationen, die in einer Ontologie enthalten sind. Prädikate, die in diesem Kapitel schrittweise neu eingeführt werden, beschreiben Nicht-OWL-Prädikate, die sich durch eine beliebige Stelligkeit von OWL Relationen unterscheiden. Allgemein gilt: Sind alle Prädikate im Rumpf einer Regel erfüllt, gilt die Konklusion.

Bevor die Regeln zur Beantwortung einer Zugriffsanfrage einem Bottom-Up Ansatz entsprechend erläutert werden, veranschaulicht Regel 5.1 die Regelsyntax und die Herleitung des finalen Prädikats `accessGranted` an einem vereinfachten Beispiel. Die ersten drei Prädikate im Rumpf der Regel beschreiben Klassenatome, die Variabeln einem entsprechenden Konzept des Modells zuordnen. Das durch weitere Regeln abgeleitete Nicht-OWL-Prädikat `hasPositiveRight` beschreibt, dass diese Variabeln in einer positiven Berechtigung im Modell referenziert werden. Die Ausführung der Regel überträgt dieses positive Recht auf das Prädikat `accessGranted`, in dem es den enthaltenen Wahrheitswert auf `"true"` setzt und somit den Zugriff gestattet.

```
∀ s, ob, op, ac: accessGranted(?s,?ob,?op,"true") ←
Subject(?s) ∧ Object(?ob) ∧ Operation(?op) ∧
hasPositiveRight(?s,?ob,?op)
```

Regel 5.1: Beispiel einer vereinfachten Regel zur Ableitung von `accessGranted`.

5.2 Regeln zur Übertragung von Berechtigungen aggregierter Elemente

Berechtigungen werden im Allgemeinen nicht auf atomaren Instanzen wie Personen oder Dateien vergeben, sondern auf zusammengesetzten Elementen, wie Subjekt- oder Objektgruppen. Um eine Zugriffsanfrage für ein konkretes Element beantworten zu können, muss in einem ersten Schritt die Zugehörigkeit dieser Elemente zu zusammengesetzten Elementen untersucht werden. Hierzu werden Regeln definiert, die Berechtigungen von zusammengesetzten Elementen auf die enthaltenen atomaren Instanzen übertragen. Hierbei werden Regeln für Gruppen und Hierarchien unterschieden.

5.2.1 Elementgruppen

Elementgruppen bestehen aus atomaren Elementen oder anderen Elementgruppen. Berechtigungen, die auf Elementgruppen definiert werden, gelten für jedes Mitglied einer Gruppe. Regel 5.2 überträgt Berechtigungen einer Subjektgruppe auf alle Mitglieder dieser Gruppe. Für Anfragen auf der Wissensbasis ist es hierbei wichtig, zwischen direkt zugewiesenen Berechtigungen, die in der Ontologie enthalten sind, und propagierten Berechtigungen, die einer Zugriffsentscheidung dienen, unterscheiden zu können. Hierzu wird für Elementgruppen ein zweielementiges Nicht-OWL-Prädikat `isPropagatedPermissionSubject(?s,?p)` eingeführt. In Regel 5.2 bezeichnet Variable `g` eine Subjektgruppe und `s` eine Subjektinstanz. Ist Subjekt `s` Mitglied der in Berechtigung `p` referenzierten Gruppe `g` überträgt Regel 5.2 diese Berechtigung von der Gruppe `g` an das Element `s`.

```
∀ s, g, p: isPropagatedPermissionSubject(?s, ?p) ←
Subject(?s) ∧  CompositeSubject(?g) ∧ Permission(?p) ∧
isMemberOfCompositeEntity(?s,?g) ∧ isPermissionSubject(?g,?p)
```

Regel 5.2: Weitergabe von Berechtigungen auf Subjektgruppen.

Um komplizierte Fallunterscheidungen zwischen direkt zugewiesenen und propagierten Beziehungen bei der Beantwortung von Zugriffsanfragen zu vermeiden, wird Regel 5.3 eingeführt. Diese bildet auch direkt zugewiesene Berechtigungssubjekte auf das neu eingeführte Nicht-OWL-Prädikat ab. Will man wissen, ob ein Subjekt direkt oder durch Mitgliedschaft

in einer Gruppe in einer Berechtigung referenziert wird, reicht eine Anfrage nach dem Prädikat `isPropagatedPermissionSubject` aus.

```
∀ s, g, p: isPropagatedPermissionSubject(?s,?p) ←
isPermissionSubject(?s,?p)
```

Regel 5.3: Abbildung direkt zugewiesener Subjektberechtigungen.

Die Mitgliedschaft von Elementen in Gruppen ist transitiv definiert, d.h. ist ein Element `x` Mitglied in einer Gruppe `y`, die ihrerseits wiederum Mitglied in einer Gruppe `z` ist, so ist `x` aufgrund der Transitivität auch Mitglied in `z`. Dies wird durch die Regel 5.4 formalisiert.

```
∀ x, y, z: isMemberOfCompositeEntity(?x,?z) ←
isMemberOfCompositeEntity(?x,?y) ∧ isMemberOfCompositeEntity(?y,?z)
```

Regel 5.4: Transitivität von Gruppenmitgliedschaften.

Das Modell sieht ebenfalls die Gruppierung von Objekten und Operationen vor. Auch hierfür gibt es zugehörige Regeln, die Berechtigungen von Gruppen auf die Mitglieder der jeweiligen Gruppe übertragen. Die zugehörigen Regeln sind mit denen der Subjektgruppen vergleichbar und werden hier nicht aufgeführt.

5.2.2 Elementhierarchien

Wie in Abschnitt 4.3.1 erläutert, können Elemente auch hierarchisch organisiert sein. Für die Weitergabe von Berechtigungen zwischen Hierarchieebenen sind Weitergabestrategien entscheidend. Berechtigungen gelten also nicht nur für die Ebenen denen sie explizit zugewiesen wurden, sondern können mittels Regeln auf über- bzw. untergeordnete Ebenen übertragen werden. Die unterschiedlichen Strategien werden durch Regeln umgesetzt. Das allgemeine Vorgehen wird am Beispiel von Subjekthierarchien aufgezeigt und kann direkt auf Objekthierarchien übertragen werden.

Auch hier ist es für evtl. notwendige Konfliktauflösungen auf Hierarchien notwendig, direkt zugewiesene Berechtigungen von propagierten Berechtigungen unterscheiden zu können. Hierzu wird, ähnlich wie bei Elementgruppen, für Subjekthierarchien ein Nicht-OWL-Prädikat `isPropagatedPermissionSubject(?s,?p,?h)` eingeführt. Dies besagt, dass Subjekt `s` in der Hierarchie `h` durch Weitergabe in der Berechtigung `p` referenziert wird.

Wird ein hierarchisches Subjekt direkt in einer Berechtigung referenziert, muss diese Relation auf das zuvor beschriebene Nicht-OWL-Prädikat abgebildet werden. Dies wird in Regel 5.5 beschrieben und gilt unabhängig von einer konkreten Weitergabestrategie.

NoPropagationPolicy: Hier gelten Berechtigungen ausschließlich auf der Hierarchieebene, auf denen diese definiert wurden. Da diese Strategie keine Weitergabe von Berechtigungen beschreibt, ist keine zusätzliche Regel notwendig.

```
∀ s, h, p: isPropagatedPermissionSubject(?s,?p,?h) ←
Subject(?s) ∧ SubjectHierarchy(?h) ∧ Permission(?p) ∧
isPropagatedPermissionSubject(?s,?p) ∧ isMemberOfHierarchy(?s,?h)
```

Regel 5.5: Abbildung von expliziten Berechtigungen auf Nicht-OWL-Prädikat in Subjekthierarchien

PropagationUpPolicy: Diese Strategie definiert, dass Berechtigungen von niedereren an höhere Schichten weitergegeben werden. Für Subjekthierarchien wird dies durch Regel 5.6 beschrieben.

```
∀ sub, sup, h, p: isPropagatedPermissionSubject(?sup,?p,?h) ←
Subject(?sup) ∧ Subject(?sub) ∧ SubjectHierarchy(?h) ∧
Permission(?p) ∧ isPropagatedPermissionSubject(?sub,?p) ∧
isMemberOfHierarchy(?sub,?h) ∧ isMemberOfHierarchy(?sup,?h) ∧
hasHierarchySuperNode(?sub,?sup) ∧
hasPropagationPolicy(?h,"#propagationUpPolicy")
```

Regel 5.6: *PropagationUpPolicy* auf Subjekthierarchien.

Der in der Regel verwendete Ausdruck `"#propagationUpPolicy"` bedeutet hierbei, dass eine Konzeptinstanz mit der in Hochkommata angegebenen ID referenziert wird. Abbildung 5.1 veranschaulicht die in der Regel 5.6 verwendeten Instanzen und Relationen.

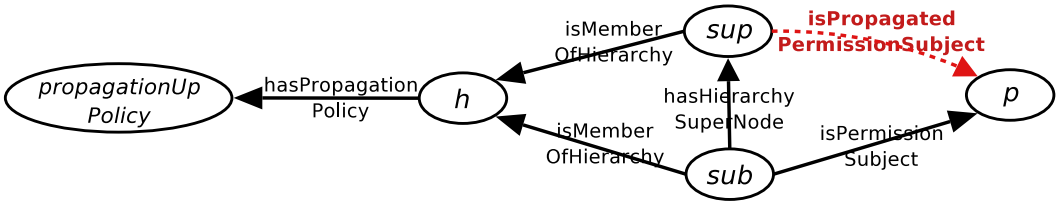

Abbildung 5.1: Veranschaulichung der Instanzen und Beziehungen der Regel 5.6.

Um sicherzustellen, dass Regel 5.6 die Berechtigungen auf alle übergeordneten Ebenen der Hierarchie überträgt, muss die Relation `hasHierarchySuperNode` transitiv sein. Dies wird wiederum durch eine eigene, hier nicht aufgeführte Regel beschrieben.

PropagationDownPolicy: Die Strategie `propagationDownPolicy` beschreibt die Weitergabe von Berechtigungen innerhalb einer Hierarchie von übergeordneten an unterge-

ordnete Schichten. Dies geschieht wie in Regel 5.6 dargestellt, wobei statt der Relation `hasHierarchySuperNode` nun `hasHierarchySubNode` betrachtet wird. Auch hier gilt, dass die Relation `hasHierarchySubNode` transitiv definiert sein muss.

5.2.3 Objektlisten

Wie in Abschnitt 4.3.2 erläutert, werden Objektlisten für listenbasierte Zugriffsrechte verwendet, in denen die Reihenfolge der Operanden von Bedeutung ist. Einer Anfrage nach einem listenbasierten Zugriffsrecht wird eine Liste bestehend aus mehreren Objektinstanzen übergeben. Damit die Beantwortung einer Berechtigungsanfrage für Objektlisten sich nicht von der allgemeinen Vorgehensweise unterscheidet, wird in einem ersten Schritt eine passende Listeninstanz aus der Ontologie extrahiert. Diese Listeninstanz beinhaltet alle der Anfrage übergebenen Elemente in der passenden Reihenfolge.

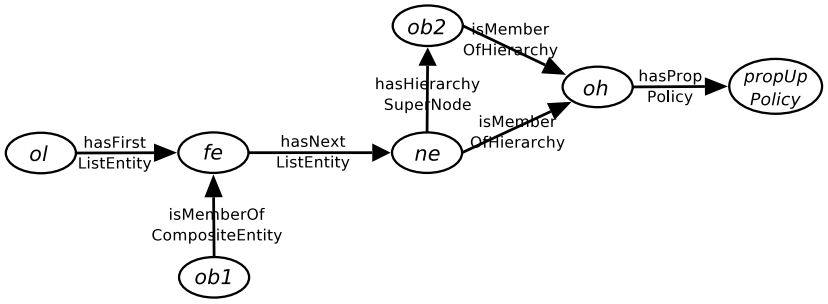

Abbildung 5.2: Weitergaben auf Objektlisten.

Die Suche nach einer geeigneten Objektliste in der Wissensbasis wird dadurch erschwert, dass Objekte einer Liste nicht direkt zugeordnet sein müssen, sondern durch Weitergaben auf Gruppen bzw. Hierarchien übertragen werden können. Dies wird in Abbildung 5.2 beispielhaft dargestellt. `ol` stellt hierbei eine zweielementige Liste dar, die die Elemente `fe` und `ne` enthält. `fe` ist eine Objektgruppe, die Objekt `ob1` als Mitglied hat. `ne` ist Mitglied einer Hierarchie und gibt seine Eigenschaften durch die auf der Objekthierarchie definierte Weitergabestrategie an `ob2` weiter. Aufgrund dieser Weitergaben sind nicht nur die direkt referenzierten Objekte `fe` und `ne` Elemente der Liste, sondern auch `ob1` und `ob2`.

Diese Suche nach geeigneten Objektlisten wird durch Regeln unterstützt. Hierzu werden zwei Nicht-OWL-Prädikate `isPropagatedListObject` und `objectListExists` eingeführt. In einem ersten Schritt werden die möglichen Weitergaben auf Objekten untersucht und auf die gesuchte Objektliste angewendet. `isPropagatedListObject(?ob1,?ob2)` beschreibt, dass Objekt `ob1` durch Aggregationsbeziehungen mit Objekt `ob2` Element der Liste ist. Um in der Suche nach einer geeigneten Objektliste komplizierte Fallunterscheidungen zu vermeiden, ist dieses Prädikat reflexiv definiert. Dies wird durch Regel 5.7 beschrieben. `equal` ist hierbei ein KAON2 internes Prädikat, dass die Gleichheit von Instanzen überprüft.

Die Weitergabe durch Mitgliedschaften in Gruppen bzw. Hierarchien unter Berücksichtung der Weitergabestrategie formalisieren Regeln 5.8 bzw. 5.9.

Nachdem die möglichen Weitergaben beschrieben wurden, kann nun die eigentliche Suche nach einer geeigneten Objektliste in der Wissensbasis durchgeführt werden. Um eine solche Anfrage zu formulieren, nimmt das Prädikat `objectListExists` die gesuchten Lis-

```
∀ ob1, ob2: isPropagatedListObject(?ob1,?ob2) ←
Object(?ob1) ∧ Object(?ob2) ∧ equal(?ob1,?ob2)
```

Regel 5.7: Objektlisten: Reflexive Weitergabe auf Objekten.

```
∀ ob, og: isPropagatedListObject(?ob,?og) ←
Object(?ob) ∧ CompositeObject(?og) ∧ isMemberOfCompositeEntity(?ob,?og)
```

Regel 5.8: Objektlisten: Weitergabe auf Objektgruppen.

```
∀ sub, sup, oh: isPropagatedListObject(?sup,?sub) ←
Object(?sup) ∧ Object(?sup) ∧ ObjectHierarchy(?oh) ∧
isMemberOfHierarchy(?sub,?oh) ∧ isMemberOfHierarchy(?sup,?oh) ∧
hasPropagationPolicy(?oh,"#propagationUpPolicy") ∧
hasHierarchySuperNode(?sub,?sup)
```

Regel 5.9: Objektlisten: Weitergabe auf Objekthierarchien.

tenobjekte entgegen und gibt ggf. eine passende in der Ontologie enthaltene Objektliste zurück. In dem in Abbildung 5.2 dargestellten Beispiel, würden dem Prädikat die Objekte `ob1` und `ob2` übergeben. Das Ergebnis der Anfrage wäre die Listeninstanz `ol`. KAON2 Nicht-OWL-Prädikate besitzen eine fest definierte Stelligkeit. Daher können generische Regeln für beliebige Listenanfragen nicht formuliert werden, sondern müssen in Abhängigkeit der Anzahl der Elemente einer Liste durch neue Regeln angepasst werden. Für allgemeine Fälle listenbasierter Zugriffsrechte sollten Objektlisten mit maximal 3 Elementen ausreichend sein. Regel 5.10 stellt die Suche nach einer zweielementigen Objektliste unter Verwendung der zuvor beschriebenen Prädikate beispielhaft vor.

5.2.4 Regeln zur Beschreibung von Platzhaltern (engl. Wildcards)

In Abschnitt 4.3.4 wurde erläutert, wie einem ausgezeichneten Benutzer namens „*Root*“ Berechtigungen auf allen Objekten innerhalb des Systems vergeben werden. Hierzu definieren die Instanzen `#allACObjects` und `#allACOperations` Elementgruppen, die alle Objekte bzw. alle Operationen des Modells umfassen.

Da eine Objektgruppe nur atomare und zusammengesetzte Objekte, aber keine Objekthierarchien enthalten kann, ist für die Berechnung der Mitglieder der Gruppe `#allACObjects` eine Unterteilung auf zwei Regeln notwendig. Regeln 5.11 und 5.12 machen alle atomaren Objekte und Objektgruppen zu Mitgliedern der Objektgruppe `#allACObjects`.

```
∀ ol, ob1, ob2: objectListExists(?ol,?ob1,?ob2) ←
Object(?ob1) ∧ Object(?ob2) ∧ ObjectList(?ol) ∧
hasFirstListEntity(?ol,?fe) ∧ hasNextListEntity(?fe,?ne1) ∧
¬ hasNextListEntity(?ne1,?ne2) ∧ isPropagatedListObject(?ob1,?fe) ∧
isPropagatedListObject(?ob2,?ne1)
```

Regel 5.10: Überprüfen der Existenz zweielementiger Objektlisten.

```
∀ ob: isMemberOfCompositeEntity(?ob,"#allACObjects") ←
AtomicObject(?ob)
```

Regel 5.11: Zuordnung von atomaren Objekten zur Objektgruppe *#allACObjects*.

```
∀ ob: isMemberOfCompositeEntity(?ob,"#allACObjects") ←
CompositeObject(?ob)
```

Regel 5.12: Zuordnung von Objektgruppen zur Objektgruppe *#allACObjects*.

Alle im System vorhandenen Operationen werden der Instanz `allACOperations` durch Regel 5.13 zugewiesen.

```
∀ op: isMemberOfCompositeEntity(?op,"#allACOperations") ←
Operation(?op)
```

Regel 5.13: Zuordnung von Operationsinstanzen zur Operationsgruppe *#allACOperations*.

5.3 Regeln zur Beantwortung von Anfragen

Autorisierungsanfragen werden durch ableiten des Prädikats `accessGranted` beantwortet. Dieser Abschnitt beschreibt die einzelnen Schritte, die zur Ableitung dieses Prädikats durch Auswertung von Informationen der Wissensbasis notwendig sind. Dazu wird untersucht, ob für eine Anfrage entsprechende Zugriffsinformationen vorliegen. In Abhängigkeit dieser Informationen müssen daraufhin evtl. vorhandene Konflikte aufgelöst werden oder aber fehlende Zugriffskontrollinformationen geeignet interpretiert werden.

Zu Beginn muss untersucht werden, ob für das Subjekt, das Objekt und die Operation einer Anfrage Berechtigungen im System vorhanden sind. Regel 5.14 führt das Nicht-OWL-Prädikat `permissionReference(?s,?ob,?op,?p)` ein. Dieses Prädikat vereint die in einer Berechtigung `p` referenzierten Subjekte, Objekte und Operationen und erlaubt eine einfachere Weiterverarbeitung der in einer Berechtigung enthaltenen Informationen.

```
∀ s, ob, op, p, ar: permissionReference(?s,?op,?ob,?p) ←
Subject(?s) ∧ Object(?ob) ∧ Operation(?op) ∧
Permission(?p) ∧ AccessRight(?ar) ∧ isPermissionRight(?ar,?p) ∧
isPropagatedRightObject(?ob,?ar) ∧ isPropagatedRightOperation(?op,?ar) ∧
isPropagatedPermissionSubject(?s, ?p)
```

Regel 5.14: Vereinigung der Elemente einer Berechtigung in Nicht-OWL-Prädikat.

Um eine Zugriffsanfrage beantworten zu können, muss zwischen positiven von negativen Berechtigungen unterschieden werden. Hierzu werden zwei Nicht-OWL-Prädiktate `hasPositiveRight(?s,?ob,?op)` bzw. `hasNegativeRight(?s,?ob,?op)` eingeführt. Diese Prädikate beschreiben, dass Subjektinstanz `s` eine positive bzw. negative Berechtigung für die Ausführung der Operation `op` auf dem Objekt `ob` besitzt.

Regel 5.15 stellt beispielhaft dar, wie das Prädikat `hasPositiveRight` generiert wird. Wichtig ist hierbei die Abgrenzung von Subjekten und Objekten die Mitglieder einer Hierarchie sind, da Konflikte auf Hierarchien, wie in Abschnitt 5.3.4 beschrieben, zunächst lokal betrachtet werden.

```
∀ s, ob, op, p: hasPositiveRight(?s,?op,?ob) ←
PositivePermission(?p) ∧ permissionReference(?s,?op,?ob,?p) ∧
¬ isMemberOfHierarchy(?s,?h) ∧ ¬ isMemberOfHierarchy(?ob,?oh)
```

Regel 5.15: Erzeugung von globalen positiven Berechtigungen.

Für negative globale Rechte ist die Vorgehensweise identisch. Die entsprechenden Elemente werden jedoch in einer negativen Berechtigungsinstanz referenziert.

Anhand dieser zusätzlichen Prädikate können für eine Anfrage mehrere Fälle unterschieden werden, die in den nächsten Abschnitten genauer betrachtet werden.

5.3.1 Offene und geschlossene Systeme

Liegt weder eine positive noch eine negative Berechtigung für eine Anfrage vor, so liefert Regel 5.15 kein Ergebnis und die Prädikate `hasPositiveRight` und `hasNegativeRight`

können im Modell nicht abgeleitet werden. In diesem Falle wird die Antwort anhand einer dem Zugriffskontrollmodell zugewiesenen Standardstrategie (*DefaultRulePolicy*) ermittelt. Man unterscheidet geschlossene (*openWorldAssumption*) und offene Systeme (*closedWorldAssumption*).

Offene Systeme erlauben jeden Zugriff der nicht explizit verboten ist. Liegt für eine Anfrage keine Berechtigung vor, so wird der Zugriff erlaubt, d.h. der Wahrheitswert des Prädikats `accessGranted` wird auf `"true"` gesetzt. Dies wird durch Regel 5.16 beschrieben.

```
∀ s, ob, op, ac: accessGranted(?s,?ob,?op,"true") ←
Subject(?s) ∧ Object(?ob) ∧ Operation(?op) ∧
ACModel(?ac) ∧ hasBaseACModel(?s,?ac) ∧
hasDefaultRulePolicy(?ac,"#openWorldAssumption") ∧
¬ hasPositiveRight(?s,?ob,?op) ∧ ¬ hasNegativeRight(?s,?ob,?op)
```

Regel 5.16: Vergabe von Berechtigungen in offenen Systemen.

Bei geschlossen Systemen hingegen ist jeder Zugriff verboten, der nicht ausdrücklich erlaubt ist. Liegen somit keine positiven und negativen Rechte vor, wird der Wahrheitswert von `accessGranted` im Gegensatz zu Regel 5.16 in geschlossenen Systemen auf `"false"` gesetzt.

5.3.2 Konfliktfreie Beantwortung von Anfragen

Eine eindeutige Situation liegt genau dann vor, wenn für eine Anfrage alle Rechte positiv oder aber negativ sind. Vor allem in Zugriffskontrollmodellen, die ohnehin nur positive oder negative Berechtigungen erlauben (*PositiveRightsACModel*- bzw. *NegativeRightsACModel*-Instanzen), sind Konflikte nicht möglich. Somit können für diese Modelle Autorisierungsanfragen ohne zusätzliche Konfliktauflösung direkt beantwortet werden. Regel 5.17 zeigt dies am Beispiel eines positiven Zugriffskontrollmodells auf. Durch diese Regel wird bei Vorliegen mindestens einer positiven Berechtigung der Zugriff gestattet.

```
∀ s, ob, op, ac: accessGranted(?s,?ob,?op,"true") ←
Subject(?s) ∧ Object(?ob) ∧ Operation(?op) ∧
PositiveRightsACModel(?ac) ∧ hasBaseACModel(?s,?ac) ∧
hasPositiveRight(?s,?ob,?op)
```

Regel 5.17: Vergabe von Berechtigungen in positiven Zugriffskontrollmodellen.

Für negative Zugriffskontrollmodelle ist das Vorgehen analog. Auch bei gemischten Modellen (*MixedRightsACModel*) gibt es eindeutige Fälle, die ohne Konfliktauflösung beant-

wortet werden können. Die Regeln zur Umsetzung der Konfliktauflösungsstrategien enthalten diese einfachen Fälle und werden im nächsten Abschnitt vorgestellt.

5.3.3 Globale Konfliktauflösung

Zu Konflikten kann es in gemischten Zugriffskontrollmodellen durch Zuweisung einer positiven und negativen Berechtigung zu einem Element kommen. Wie in Abschnitt 5.1 dargestellt, unterscheidet man globale und hierarchische Konfliktauflösung. Die globale Konfliktauflösung trifft Entscheidungen auf systemweiter Ebene und ist die letzte Instanz, die eine widersprüchliche Zugriffsanfrage durch Erzeugung eines geeigneten `accessGranted` Prädikats entscheidet. Ein Konflikt tritt auf, wenn für eine Anfrage je mindestens ein `hasPositiveRight` und `hasNegativeRight` abgeleitet werden kann. Wie in Abschnitt 4.3.3 auf Seite 46 beschrieben, unterscheidet man die globalen Konfliktauflösungsstrategien *GrantTakesPrecedence* und *DenialTakesPrecedence*.

Die Strategie *GrantTakesPrecedence* gibt positiven Berechtigungen ein höheres Gewicht. Liegt mindestens ein positives Recht vor, wird ein Zugriff gestattet. Bei dieser Strategie wird eine Anfrage nur dann verneint, wenn alle vorliegenden Rechte negativ sind. Die Regeln 5.18 und 5.19 setzen dieses Verhalten um.

```
∀ s, ob, op, ac: accessGranted(?s,?ob,?op,"true") ←
Subject(?s) ∧ Object(?ob) ∧ Operation(?op) ∧
MixedRightsACModel(?ac) ∧ hasBaseACModel(?s,?ac) ∧
hasConflictResolutionPolicy(?ac,"#grantTakesPrecedence") ∧
hasPositiveRight(?s,?ob,?op)
```

Regel 5.18: Globale Konfliktauflösung: GrantTakesPrecedence-Strategie mit mindestens einem positiven Recht.

```
∀ s, ob, op, ac: accessGranted(?s,?ob,?op,"false") ←
Subject(?s) ∧ Object(?ob) ∧ Operation(?op) ∧
MixedRightsACModel(?ac) ∧ hasBaseACModel(?s,?ac) ∧
hasConflictResolutionPolicy(?ac,"#grantTakesPrecedence") ∧
¬ hasPositiveRight(?s,?ob,?op) ∧ hasNegativeRight(?s,?ob,?op)
```

Regel 5.19: Globale Konfliktauflösung: GrantTakesPrecedence-Strategie mit ausschließlich negativen Rechten.

Die Strategie *DenialTakesPrecedence* gewichtet Verbote auf Anfragen höher. Umgekehrt zu Regeln 5.18 und 5.19 wird hier ein Zugriff verboten, wenn mindestens eine negative

Berechtigung vorliegt bzw. der Zugriff gestattet, wenn alle Rechte einer Anfrage positiv sind.

5.3.4 Hierarchische Konfliktauflösung

Wie in Abschnitt 5.2.2 auf Seite 65 beschrieben, können Rechte zwischen Ebenen einer Hierarchie weitergegeben werden. Zur Auflösung von evtl. auftretenden Konflikten in einer Hierarchie gibt es die in Abschnitt 4.3.3 auf Seite 46 vorgestellten hierarchischen Konfliktauflösungsstrategien. Um Konflikte auf Hierarchien zu erkennen, sind eine Vielzahl neuer Nicht-OWL-Prädikate notwendig.

Zusätzliche Komplexität der hierarchischen Konfliktauflösung ergibt sich durch die Möglichkeit von Konflikten innerhalb von Objekt- und Subjekthierarchien, die unabhängig voneinander betrachtet werden müssen. Durch die Unterscheidung ob nur das Objekt oder nur das Subjekt oder sowohl Subjekt als auch Objekt einer Anfrage in einer Hierarchie enthalten sind, ergeben sich eine Menge von Regeln, die sich nur durch kleine Details voneinander unterscheiden.

Die nun folgenden Regeln zeigen bis auf wenige Ausnahmen das exemplarische Vorgehen für Anfragen auf, in denen nur das Subjekt jedoch nicht das Objekt Mitglied einer Hierarchie ist.

Wie bei der globalen Konfliktauflösung werden auch hier zwei zusätzliche Nicht-OWL-Prädikate `hasPropPositiveSubjectHierRight(?s,?ob,?op,?h)` und `hasPropNegativeSubjectHierRight(?s,?ob,?op,?h)` eingeführt. `s` ist hierbei ein Subjekt, dass Mitglied einer Hierarchie `h` ist. Die Nicht-OWL-Prädikate beschreiben, dass Subjekt `s` ein positives bzw. negatives Recht zur Ausführung der Operation `op` auf dem Objekt `ob` entweder direkt oder durch Weitergabe innerhalb der Subjekthierarchie besitzt.

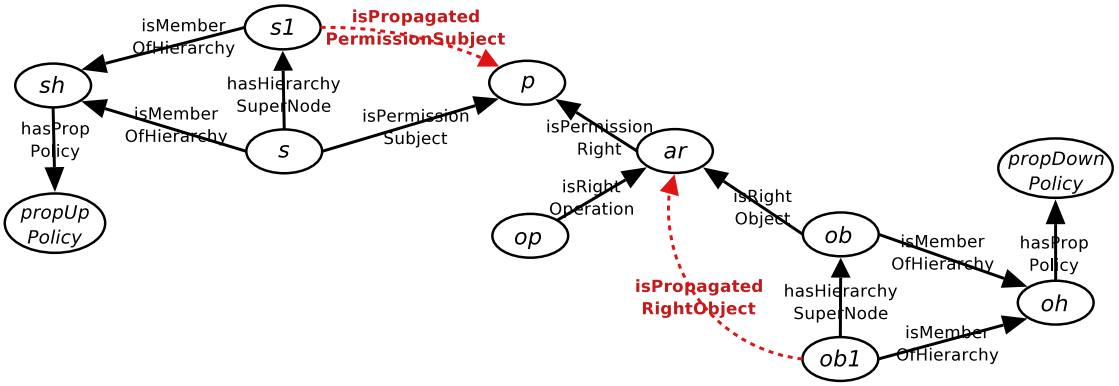

Abbildung 5.3: Weitergabe von Rechten bei Subjekt- und Objekthierarchien.

Sind sowohl Subjekt als auch Objekt einer Anfrage hierarchisch organisiert, erschwert dies die Suche nach einer passenden Berechtigung für ein Subjekt, da nicht nur direkt zugewiesene Objekte eines Zugriffsrechtes analysiert werden müssen, sondern auch Weitergaben auf der Objekthierarchie. Abbildung 5.3 zeigt ein Beispiel für eine Berechtigung die hierarchische Subjekte und Objekte referenziert. Eine Anfrage nach den Elementen `s1` und `ob1` kann nicht direkt beantwortet werden. Zuvor muss die Weitergabestrategie auf den Hierarchien betrachtet werden. Erbt das jeweilige Element die Berechtigungen, werden die in der

Abbildung rot dargestellten Prädikate abgeleitet. Ist das Objekt hingegen wie in Regel 5.20 nicht Mitglied einer Hierarchie, werden nur direkt referenzierte Objekte eines Zugriffsrechts analysiert.

Das Vorgehen zur Herleitung des positiven hierarchischen Rechts für Subjekte zeigen Regeln 5.20 und 5.21. Diese Regel verwenden die in Abschnitt 5.2.2 eingeführten Nicht-OWL-Prädikate für weitergegebene Rechte auf Hierarchien. Während in Regel 5.20 nur das Subjekt Mitglied einer Hierarchie ist, ist in Regel 5.21 auch das Objekt hierarchisch organisiert.

```
∀ s, ob, op, h: hasPropPositiveSubjectHierRight(?s,?ob,?op,?h) ←
Subject(?s) ∧ AccessRight(?ar) ∧ Operation(?op) ∧
Object(?ob) ∧ PositivePermission(?p) ∧ SubjectHierarchy(?h) ∧
isMemberOfHierarchy(?s,?h) ∧ isPropagatedPermissionSubject(?s,?p,?h) ∧
isPermissionRight(?ar,?p) ∧ isRightOperation(?op,?ar) ∧
isRightObject(?ob,?ar)
```

Regel 5.20: Propagierte hierarchische Rechte bei Subjekthierarchien.

```
∀ s, ob, op, sh, oh: hasPropPositiveSubjectHierRight(?s,?ob,?op,?sh) ←
Subject(?s) ∧ AccessRight(?ar) ∧ Operation(?op) ∧
Object(?ob) ∧ PositivePermission(?p) ∧ SubjectHierarchy(?sh) ∧
ObjectHierarchy(?oh) ∧isMemberOfHierarchy(?s,?sh) ∧
isMemberOfHierarchy(?ob,?oh) ∧ isPermissionRight(?ar,?p) ∧
isPropagatedPermissionSubject(?s,?p,?sh) ∧ isRightOperation(?op,?ar) ∧
isPropagatedRightObject(?ob,?ar,?oh)
```

Regel 5.21: Propagierte hierarchische Rechte bei Subjekt- und Objekthierarchien.

Ist das Objekt Mitglied einer Hierarchie sind die Rümpfe beider Regeln erfüllt. Regel 5.20 leitet jedoch in diesem Falle eine Teilmenge der Ergebnisse der Regel 5.21 ab. Die Regeln stehen somit in keinem Widerspruch zueinander. Würde man in Regel 5.20 jedoch hinzufügen, dass Objekt `ob` nicht Mitglied einer Hierarchie sein darf, führt das zur Nichtstratifizierbarkeit des logischen Programms. Details der Stratifizierbarkeit logischer Programme können in Abschnitt B des Anhangs nachgelesen werden.

Wie bildet man nun diese hierarchischen Rechte auf die in den letzten Abschnitten besprochenen globalen Rechte ab? Der einfachste Fall liegt wiederum vor, wenn es keine Konflikte innerhalb einer Hierarchie gibt. Das bedeutet, dass entweder nur positive oder nur negative hierarchischen Rechte für eine Anfrage vorliegen. In diesem Fall kann das hierarchische Recht einfach auf das globale Niveau weitergegeben werden. Dies geschieht durch

Regel 5.22. Liegen ausschließlich negative Berechtigungen vor, wird dies durch eine ähnliche Regel abgebildet.

```
∀ s, ob, op, h: hasPositiveRight(?s,?ob,?op) ←
Hierarchy(?h) ∧ hasPropPositiveSubjectHierRight(?s,?ob,?op,?h) ∧
¬ hasPropNegativeSubjectHierRight(?s,?ob,?op,?h)
```

Regel 5.22: Konfliktfreie Abbildung von hierarchischen auf globale Rechte.

Treten Konflikte durch positive oder negative Berechtigungen in einer Anfrage nach einem hierarchischen Subjekt auf, muss eine hierarchische Konfliktauflösungsstrategie verwendet werden. Man unterscheidet die Strategien *AncestorTakesPrecedence* und *DescendantTakesPrecedence*, in denen Rechte übergeordneter bzw. untergeordneter Hierarchieknoten dominieren. Ein Konflikt tritt auf, wenn für eine Anfrage sowohl `hasPropPositiveSubjectHierRight` und `hasPropNegativeSubjectHierRight` abgeleitet werden kann. Regel 5.23 benennt einen solchen Konflikt durch ein neues Prädikat `hierarchicalConflict(?s,?ob,?op,?h)`. Dieses Prädikat besagt, dass innerhalb der Hierarchie `h` für Subject `s`, Objekt `ob` und Operation `op` ein Konflikt auftritt.

```
∀ s, ob, op, h, ac: hierarchicalConflict(?s,?ob,?op,?h) ←
SubjectHierarchy(?h) ∧ MixedRightsACModel(?ac) ∧
hasBaseACModel(?s,?ac) ∧ hasPropPositiveSubjectHierRight(?s,?ob,?op,?h) ∧
hasPropNegativeSubjectHierRight(?s,?ob,?op,?h)
```

Regel 5.23: Einführung eines Prädikats für hierarchische Konflikte.

Zur Auflösung eines Konflikts müssen direkt zugewiesene hierarchische Rechte von den durch Weitergabe erhaltenen hierarchischen Rechten unterscheidbar sein. Hierzu werden die beiden Prädikate `hasPositiveSubjectHierRight` und `hasNegativeSubjectHierRight` definiert. Diese werden ähnlich wie die propagierten hierarchischen Rechte generiert, wobei hier jedoch nur direkt zugewiesene Berechtigungen entscheidend sind. Hierzu wird das in Regel 5.14 eingeführte Prädikat `permissionReference` abgefragt, dass nur direkte Referenzen einer Berechtigung auswertet. Regel 5.24 stellt dies am Beispiel negativer Berechtigungen beispielhaft dar. Auch hier muss, wie bei den Regeln der propagierten hierarchischen Rechte, wiederum unterschieden werden, ob das Objekt einer Anfrage in einer Hierarchie vorliegt.

Durch die hierarchischen Konfliktauflösungsstrategien entscheiden jeweils Rechte über- bzw. untergeordneter Knoten über das Ergebnis einer Zugriffsanfrage. Die eigentliche Ursache eines Konflikts ist nicht von Interesse, sondern nur wie dieser aufgelöst werden kann. Im Folgenden wird ausschließlich die Strategie *AncestorTakesPrecedence* betrachtet, in der

$$\forall\ s, ob, op, p, h: \texttt{hasNegativeSubjectHierRight}(?s,?ob,?op,?h) \leftarrow \texttt{NegativePermission}(?p) \wedge \texttt{SubjectHierarchy}(?h) \wedge \texttt{permissionReference}(?s,?ob,?op,?p) \wedge \texttt{isMemberOfHierarchy}(?s,?h)$$

Regel 5.24: Hierarchische (nicht propagierte) negative Rechte.

Berechtigungen übergeordneter Hierarchieebenen entscheiden. Um nun im Falle eines Konflikts eine Entscheidung fällen zu können, benötigt man Regeln, die Berechtigungen übergeordneter Hierarchieknoten analysieren. Von Interesse sind hierbei nur Berechtigungen maximaler Superknoten, da deren Rechte über die Zugriffsberechtigung entscheiden.

Abbildung 5.4 zeigt einige Möglichkeiten zu Weitergaben auf Subjekthierarchien auf. + bzw. - zeigen an, dass der entsprechende Hierarchieknoten in einer positiven bzw. negativen Berechtigung referenziert wird, während ++ bzw. - - durch Weitergabe zugewiesene Berechtigungen aufzeigen. In (i) wird der maximale Superknoten in einer negativen Berechtigung direkt referenziert und ist somit negativ, in (ii) sowohl positiv als auch negativ und in (iii) weder positiv noch negativ.

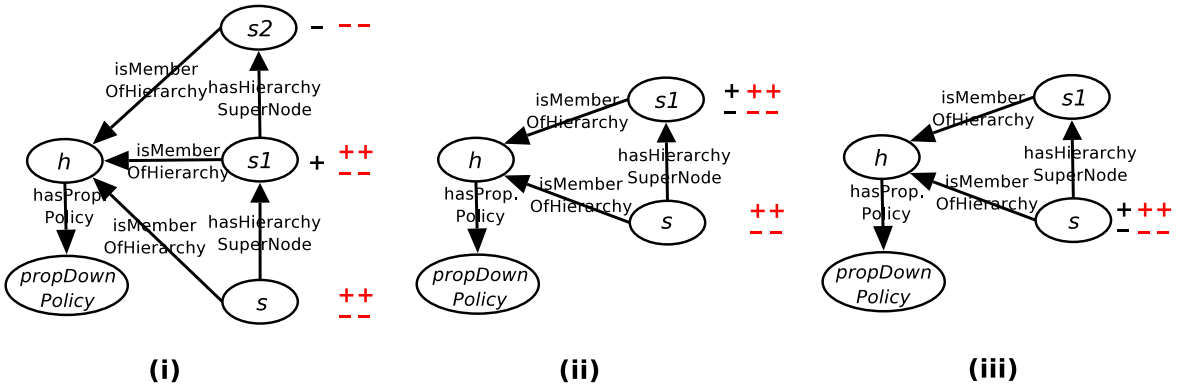

Abbildung 5.4: Beispiele für Superknotenbeziehungen auf Hierarchien.

Die Bestimmung eines maximalen Superknotens erfolgt durch einen Zwischenschritt. Bevor die Berechtigung des maximalen Superknotens bestimmt wird, werden die Rechte aller übergeordneten Knoten ausgewertet und in den Prädikaten `hasPositiveSubjectSuperNode(?sub,?sup,?ob,?op,?h)` bzw. `hasNegativeSubjectSuperNode(?sub,?sup,?ob,?op,?h)` gespeichert. `sup` ist hierbei der Superknoten von `sub`. Hierzu werden für jeden Superknoten die direkt referenzierten Berechtigungen analysiert. Die Regeln 5.25 und 5.26 zeigen das Vorgehen beispielhaft. Da die Relation `hasHierarchySuperNode` transitiv definiert ist, werden in diesen Regeln alle übergeordneten Hierarchieebenen berücksichtigt.

Diese Regeln würden für Beispiel (i) in Abbildung 5.4 ausgehend von Knoten `s` den negativen Superknoten `s2` und den positiven Superknoten `s1` ableiten und für Beispiel (ii) den positiven und negativen Superknoten `s1`.

```
∀ sub,sup,ob,op,h: hasPositiveSubjectSuperNode(?sub,?sup,?ob,?op,?h) ←
Subject(?sub) ∧ Subject(?sup) ∧ SubjectHierarchy(?h) ∧
isMemberOfHierarchy(?sub,?h) ∧ isMemberOfHierarchy(?sup,?h) ∧
hasHierarchySuperNode(?sub,?sup) ∧
hasPositiveSubjectHierRight(?sup,?ob,?op,?h)
```

Regel 5.25: Extraktion aller positiven Superknoten einer Hierarchie.

```
∀ sub,sup,ob,op,h: hasNegativeSubjectSuperNode(?sub,?sup,?ob,?op,?h) ←
Subject(?sub) ∧ Subject(?sup) ∧ SubjectHierarchy(?h) ∧
isMemberOfHierarchy(?sub,?h) ∧ isMemberOfHierarchy(?sup,?h) ∧
hasHierarchySuperNode(?sub,?sup) ∧
hasNegativeSubjectHierRight(?sup,?ob,?op,?h)
```

Regel 5.26: Extraktion aller negativen Superknoten einer Hierarchie.

Zur Bestimmung der Berechtigung des maximalen Superknotens werden diese Zwischenprädikate verwendet. So ist ein maximaler positiver Superknoten ein positiver Superknoten, der seinerseits keine negativen Superknoten besitzt. In Beispiel (i) der Abbildung 5.4 ist `s1` kein maximaler positiver Superknoten von `s`, da er seinerseits einen negativen Superknoten `s2` besitzt. `s2` ist hingegen maximaler negativer Superknoten der Knoten `s` und `s1`.

Dies wird durch die Regeln 5.27 und 5.28 und die neuen Prädikate `hasMaxPositiveSuperNode(?s,?ob,?op,?h)` und `hasMaxNegativeSuperNode(?s,?ob,?op,?h)` umgesetzt. Diese Prädikate beschreiben, dass Subjekt `s` in Hierarchie `h` einen maximalen positiven bzw. negativen Superknoten für Operation `op` und Objekt `ob` besitzt.

```
∀ s, s1, s2, ob, op, h: hasMaxPositiveSuperNode(?s,?ob,?op,?h) ←
SubjectHierarchy(?h) ∧ hasPositiveSubjectSuperNode(?s,?s1,?ob,?op,?h) ∧
¬ hasNegativeSubjectSuperNode(?s1,?s2,?ob,?op,?h)
```

Regel 5.27: Bestimmung des maximalen positiven Superknotens.

Nun sind alle Informationen, die für die Auflösung eines hierarchischen Konflikts benötigt werden, ausgewertet. Bei der Konfliktauflösungsstrategie *AncestorTakesPrecedence* legt die Berechtigung des maximalen Superknotens fest, ob ein Zugriff gestattet wird oder nicht. Ist der maximale Superknoten positiv bzw. negativ, wird ein positives bzw. negatives globa-

```
∀ s, s1, s2, ob, op, h: hasMaxNegativeSuperNode(?s,?ob,?op,?h) ←
SubjectHierarchy(?h) ∧ hasNegativeSubjectSuperNode(?s,?s1,?ob,?op,?h) ∧
¬ hasPositiveSubjectSuperNode(?s1,?s2,?ob,?op,?h)
```

Regel 5.28: Bestimmung des maximalen negativen Superknotens.

les Recht abgeleitet. Regel 5.29 beschreibt diese Abbildung von hierarchischen auf globale Rechte unter Verwendung der zuvor eingeführten Nicht-OWL-Prädikate.

```
∀ s, ob, op, h: hasPositiveRight(?s,?ob,?op) ←
Hierarchy(?h) ∧ hierarchicalConflict(?s,?ob,?op,?h) ∧
hasMaxPositiveSuperNode(?s,?ob,?op,?h) ∧
hasHierarchicalConflictResolutionPolicy(?h,"#ancestorTakesPrecedence")
```

Regel 5.29: Konfliktauflösung durch *AncestorTakesPrecedence* mit positivem maximalen Superknoten.

Für negative maximale Superknoten existiert eine ähnliche Regel. Für Beispiel (i) in Abbildung 5.4 auf Seite 76 generiert diese Regel für Subjekt `s` ein negatives globales Recht, da der maximale Superknoten von `s` positiv ist. Da in Fall (ii) der maximale Superknoten sowohl positiv als auch negativ ist, wird sowohl ein negatives als auch ein positives globales Recht erzeugt. Der hierarchische Konflikt wird somit an die globale Ebene weitergegeben und muss dort aufgelöst werden.

Was geschieht jedoch in Fall (iii) der Abbildung? Subjekt `s` hat keinen Superknoten der in einer Berechtigung referenziert ist und obige Regeln erlauben somit keine direkte Aussage. Besitzt ein Knoten keine Berechtigungen auf Superknoten, werden die direkt zugewiesenen Berechtigungen des Knotens betrachtet. Regel 5.30 beschreibt dieses Verhalten im Falle direkt zugewiesener negativer Berechtigungen.

Im Falle fehlender maximaler positiver bzw. negativer Superknoten und einer direkt zugewiesenen positiven Berechtigung, gibt es eine analoge Regel. In Beispiel (iii) der Abbildung 5.4 besitzt Subjekt `s` sowohl negative als auch positive direkt zugewiesene Berechtigungen. Regel 5.30 und das positive Pendant geben auch hier einen hierarchischen Konflikt durch Erzeugung von globalen positiven und negativen Rechten an die globale Ebene der Konfliktauflösung weiter.

Abbildung 5.5 stellt einen weiteren möglichen Fall dar. `s` ist hierbei ein Element das weder eigene Berechtigungen besitzt, noch übergeordnete Superknoten aufweist. Eine Anfrage führt jedoch durch Weitergabe von untergeordneten Ebenen zu einem Konflikt. Da unter der betrachteten Konfliktauflösungsstrategie *AncestorTakesPrecedence* Rechte übergeordneter Hierarchieebenen entscheiden, wird in einer solchen Situation kein globales Recht erzeugt, da die gewählte Konfliktauflösungsstrategie keine Entscheidung herbeiführen kann.

```
∀ s, ob, op, h: hasNegativeRight(?s,?ob,?op) ←
Hierarchy(?h) ∧ hierarchicalConflict(?s,?ob,?op,?h) ∧
¬ hasMaxPositiveSuperNode(?s,?ob,?op,?h) ∧
¬ hasMaxNegativeSuperNode(?s,?ob,?op,?h) ∧
hasNegativeSubjectHierRight(?s,?ob,?op,?h) ∧
hasHierarchicalConflictResolutionPolicy(?h,"#ancestorTakesPrecedence")
```

Regel 5.30: Konfliktauflösung durch *AncestorTakesPrecedence* ohne Superknoten.

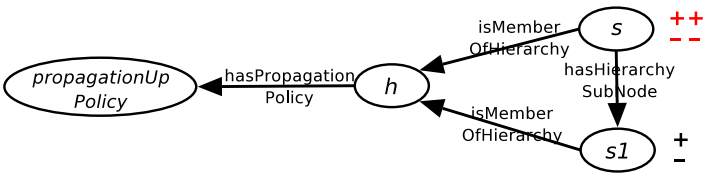

Abbildung 5.5: Hierarchieknoten ohne eigene Berechtigungen.

Die Entscheidung wird somit der Standardstrategie (*DefaultRulePolicy*) auf globaler Ebene überlassen.

Dieses Kapitel hat die allgemeine Vorgehensweise zur hierarchischen Konfliktauflösung anhand von Subjekthierarchien und der Strategie *AncestorTakesPrecedence* beschrieben. Für Subjekthierarchien und die Konfliktstrategie *DescendantTakesPrecedence* erfolgt die Beschreibung durch Regeln auf ähnliche Art und Weise unter Wiederbenutzung der hier eingeführten Prädikate. Auch auf die Auswertung von Konflikten auf Objekthierarchien kann die hier vorgestellte Vorgehensweise übertragen werden.

5.4 Regeln zur Interpretation der Systemschicht

Wie in Abschnitt 4.4 erläutert, beschreibt die Systemebene Erweiterungen, die eine einfache Beschreibung von in der Literatur unterschiedenen Zugriffskontrollmodellen ermöglicht. Diese modellspezifischen Erweiterungen gilt es in diesem Abschnitt auf die in den übergeordneten Schichten eingeführten allgemeinen Konzepte abzubilden. Ziel der Beschreibung systemspezifischer Regeln ist es, Eigenheiten dieser Modelle ohne Anpassung der Zugriffsanfrage modellieren zu können. Das bedeutet, dass einem regelbasierten System nach wie vor ein Tupel `<subject,object,operation>` übergeben wird, aber die Berechnung einer Antwort statt auf konkreten Berechtigungsinstanzen nun z.B. auf Vergleichen von Klassifikationen beruhen. Aus Sicht externer Anwendungen ist diese interne Zugriffskontrollmodellierung völlig transparent und somit ohne Codeanpassungen umsetzbar.

Regeln, die zur Interpretation regelbasierter und wahlfreier Systeme notwendig sind, werden in den folgenden Abschnitten beschrieben. Während rollenbasierte Systeme keine zusätzlichen Regeln zur Beantwortung von Zugriffsanfragen benötigen, werden diese bei kontextbasierten Modellen individuell durch den Administrator zur Verfügung gestellt.

5.4.1 Regelbasierte Systeme

Bei regelbasierten Systemen werden Berechtigungen, wie in Abschnitt 4.4.1 beschrieben, aufgrund von Klassifikationen der Objekte und Subjekte und entsprechenden Strategien auf den Operationen vergeben. Subjekte und Objekte werden aufgrund der ihnen zugeordneten Sicherheitsstufen und Kategorien geeigneten Knoten in den entsprechenden Klassifikationshierarchien zugewiesen. Klassifikationshierarchien erlauben hierbei eine einfache Auswertung regelbasierter Strategien durch Regeln.

Regelbasierte Zugriffskontrollstrategien formulieren Bedingungen zwischen den Klassifikationsbeziehungen von Subjekten und Objekten einer Anfrage und sind für die Vergabe von Berechtigungen ausschlaggebend. Diese Bedingungen werden durch Regeln ausgewertet. Sind die Bedingungen erfüllt, wird eine Zugriffsanfrage durch Ableiten des im allgemeinen Modell verwendeten Nicht-OWL-Prädikats `hasPositiveRight` beantwortet. Da regelbasierte Systeme positive Zugriffskontrollmodelle sind und somit ausschließlich positive Berechtigungen vergeben können, erlaubt die Vergabe dieses positiven Rechtes aufgrund der in Abschnitt 5.3.3 definierten Regeln den Zugriff.

Um eine einfache Vergleichbarkeit der Subjekt- und Objektklassifikationen zu ermöglichen, wird ein zusätzlicher Zwischenschritt definiert, der die zu vergleichenden Sicherheitsstufen von Objekt und Subjekt einer Anfrage in dem Prädikat `compareSecLabel(?s,?sl,?ob,?ol)` kombiniert. Dieses, durch Regel 5.31 abgeleitete Prädikat, überprüft, ob das Subjekt und Objekt einer Anfrage in einer gemeinsamen Klassifikation vorkommen. Ist dies der Fall, werden die Sicherheitsstufen des Subjekts `s` im Argument `sl` und die des Objekts `ob` in `ol` abgelegt. Zur Auswertung der regelbasierten Strategien muss daraufhin nur noch überprüft werden, in welcher Beziehung die Sicherheitsstufen von Subjekt und Objekt im Prädikat `compareSecLabel` stehen. Abbildung 5.6 gibt eine Übersicht der in Regel 5.31 verwendeten Instanzen und deren Relationen.

```
∀ s, sl, o, ol, scl, ocl, cat: compareSecLabel(?s,?sl,?o,?ol) ←
MandatoryACSubject(?s) ∧ MandatoryACObject(?ob) ∧
MandatoryACSecurityLabel(?sl) ∧ MandatoryACSecurityLabel(?ol) ∧
CompositeObject(?cat) ∧ ClassificationNode(?scl) ∧
ClassificationNode(?ocl) ∧ hasSubjectClassification(?s,?scl) ∧
hasSecurityLabel(?scl,?sl) ∧ hasObjectClassification(?s,?ocl) ∧
hasSecurityLabel(?ocl,?ol) ∧ hasCategory(?scl,?cat) ∧
hasCategory(?ocl,?cat)
```

Regel 5.31: Prädikat `compareSecLabel` zum Vergleich von Objekt- und Subjektklassifikationen.

Jede regelbasierte Operation referenziert eine Strategie, die notwendige Beziehungen von Subjekt- und Objektklassifikation zur Vergabe einer Zugriffsberechtigung beschreibt. Ausgehend von den Strategien werden die folgenden Regeln unterschieden:

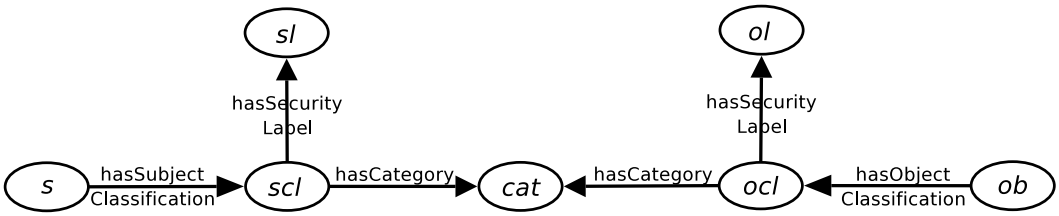

Abbildung 5.6: Übersicht der in Regel 5.31 verwendeten Instanzen.

ExecutionOnSameLevel: Diese Strategie erlaubt den Zugriff genau dann, wenn Subjekt- und Objektklassifikation übereinstimmen. Übertragen auf das in Regel 5.31 abgeleitete Prädikat `compareSecLabel` bedeutet dies, dass ein Zugriff genau dann gestattet wird, wenn die Sicherheitsstufen von Subjekt und Objekt identisch sind. Dies wird durch Regel 5.32 beschrieben. Da auch alle anderen Strategien den Zugriff auf der selben Klassifikationsstufe erlauben, handelt es sich hier um den Basisfall aller Strategien. Diese Regel wird somit auf sämtliche Instanzen regelbasierter Operationen angewendet. `equal` ist ein KAON2 Prädikat zum Vergleich der Gleichheit von Instanzen.

```
∀ s, ob, op: hasPositiveRight(?s,?ob,?op) ←
MandatoryACOperation(?op) ∧ compareSecLabel(?s,?sl,?o,?ol) ∧
equal(?sl,?ol)
```

Regel 5.32: Berechtigungsvergabe bei gleichen Subjekt- und Objektsicherheitsstufen.

NoExecutionDown: Diese Strategie definiert, dass ein Zugriff erlaubt ist, wenn die Klassifikation des Objekts die des Subjekts dominiert. Regel 5.33 untersucht zur Umsetzung dieser Strategie, ob die Sicherheitsstufe des Subjekts ein Superknoten der Sicherheitsstufe des Objekts ist und vergibt ggf. ein positives Recht.

```
∀ s, ob, op, sl, ol: hasPositiveRight(?s,?ob,?op) ←
MandatoryACOperation(?op) ∧ compareSecLabel(?s,?sl,?o,?ol) ∧
hasClassificationPropagationPolicy(?op,"#noExecutionDown") ∧
hasHierarchySubNode(?ol,?sl)
```

Regel 5.33: Berechtigungsvergabe bei Dominanz der Objektklassifikation.

NoExecutionUp: *NoExecutionUp* beschreibt die Umkehrung obiger Strategie. Nur wenn die Klassifikation des Subjekts die des Objekts dominiert, ist der Zugriff gestattet. Somit erlaubt die zugehörige Regel 5.34 einen Zugriff nur dann, wenn die Sicherheitsstufe des Objekts einen Subknoten der Sicherheitsstufe des Subjekts darstellt.

```
∀ s, ob, op, sl, ol: hasPositiveRight(?s,?ob,?op) ←
MandatoryACOperation(?op) ∧ compareSecLabel(?s,?sl,?o,?ol) ∧
hasClassificationPropagationPolicy(?op,"#noExecutionUp") ∧
hasHierarchySubNode(?sl,?ol)
```

Regel 5.34: Berechtigungsvergabe bei Dominanz der Subjektklassifikation.

5.4.2 Wahlfreie Systeme

Bei der wahlfreien Zugriffskontrolle werden dem Eigentümer eines Objektes besondere Rechte eingeräumt. So kann er alle im System vorhandenen Rechte auf seinen Objekten ausführen und Berechtigungen au diesen Objekten an andere Subjekte weitergeben.

Die Rechte des Eigentümers auf seinen Objekten werden nicht explizit in der Wissensbasis gespeichert, sondern zur Anfragezeit durch Regeln bestimmt. Außer der Delegation von Zugriffsrechten, die von zusätzlichen Strategien des jeweiligen Zugriffskontrollmodells abhängig ist, darf der Eigentümer alle Operationen auf seinen Objekten ausführen. Regel 5.35 beschreibt dies. Durch die Vergabe eines positiven Rechts wird dem Eigentümer eines Objekts der Zugriff gestattet, wenn nicht durch explizite Verbote im Modell auf dem entsprechenden Objekt ein Konflikt abgeleitet werden kann.

```
∀ s, ob, op: hasPositiveRight(?s,?ob,?op) ←
DiscretionaryACSubject(?s) ∧ DiscretionaryACObject(?ob) ∧
Operation(?op) ∧ hasOwner(?ob,?s) ∧
¬ equal(?op,"#delegateAccessRight")
```

Regel 5.35: Berechtigungsvergabe für den Besitzer eines Objektes.

5.4.2.1 Regeln zur Delegation von Rechten

In wahlfreien Systemen hat der Besitzer die Möglichkeit Rechte an seinen Objekten an andere Subjekte weiterzugeben. Diese Weitergabe bewirkt, dass dem Empfänger der Weitergabe eine positive Berechtigung für das delegierte Zugriffsrecht in der Wissensbasis hinzugefügt wird. Zudem wird die Weitergabe in der Delegationshistorie (RightDelegationHistory) vermerkt.

Bevor eine Weitergabe ausgeführt werden kann, muss sichergestellt werden, dass der Benutzer eine Berechtigung zur Ausführung dieser Weitergabe besitzt. Die Berechtigung zur Weitergabe ist abhängig von sogenannten Weitergabestrategien (*RightDelegationPolicy*).

Die Weitergabe von Rechten ist als zweistellige Operation modelliert. Argumente sind das Empfängersubjekt der Delegation und das Zugriffsrecht das weitergegeben werden soll. Abbildung 5.7 zeigt die Struktur einer Objektliste, die einer Anfrage übergeben wird. Die in

der Abbildung dargestellten Abkürzungen der Elemente verdeutlichen hierbei die in nachfolgenden Regeln verwendeten Bezeichner von Instanzen.

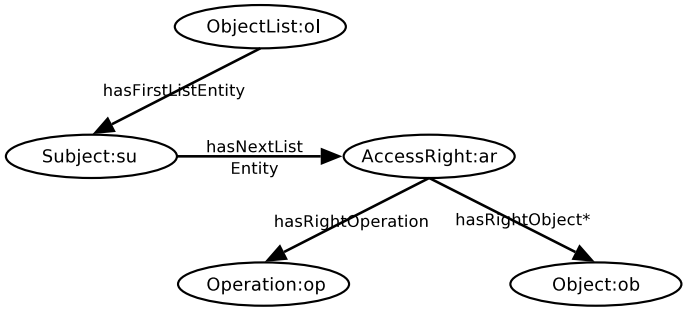

Abbildung 5.7: Struktur einer Objektliste zur Modellierung der Weitergabe von Rechten.

Unter Verwendung dieser Objektlisten werden abhängig von den jeweiligen Strategien für die Rechteweitergabe die folgenden Regeln definiert:

Rechtweitergabe nur durch Besitzer: Mit dieser Strategie ist die Weitergabe von Rechten nur erlaubt, wenn das weiterzugebende Recht als delegierbar klassifiziert wurde und wenn das zugrunde liegende wahlfreie Zugriffskontrollmodell die Weitergabe von Rechten erlaubt. Ist dies erfüllt, wird durch Regel 5.36 dem Eigentümer des Objekts der Zugriff gestattet.

$$
\begin{aligned}
&\forall\ \texttt{s, ol, op, ob, ac, ar: hasPositiveRight(?s,?ol,?op)} \leftarrow \\
&\texttt{DiscretionaryACSubject(?s)} \wedge \texttt{DiscretionaryACObject(?ob)} \wedge \\
&\texttt{DiscretionaryAdminOperation(?op)} \wedge \texttt{DiscretionaryACModel(?ac)} \wedge \\
&\texttt{ObjectList(?ol)} \wedge \texttt{hasBaseACModel(?s,?ac)} \wedge \\
&\neg\ \texttt{hasRightDelegationPolicy(?ac,"\#noDelegationPolicy")} \wedge \\
&\texttt{hasOperationName(?op,"\#delegateAccessRight")} \wedge \\
&\texttt{hasFirstListEntity(?ol,?su)} \wedge \texttt{hasNextListEntity(?su,?ar)} \wedge \\
&\texttt{isDelegable(?ar,"true")} \wedge \texttt{hasRightObject(?ar,?ob)} \wedge \\
&\textit{hasOwner(?ob,?s)}
\end{aligned}
$$

Regel 5.36: Berechtigungskontrolle zur Rechtedelegation durch Besitzer.

Weitergabe von delegierten Rechten: Die Strategie *TransitiveDelegationPolicy* erlaubt die Weitergabe von Rechten auch durch Subjekte, die dieses Recht durch Delegation erhalten haben. Die zugehörige Regel 5.37 überprüft nun, ob das Modell die Weitergabe delegierter Rechte erlaubt und ob der Anfragesteller das Recht durch Weitergabe erhalten hat. Trifft all dies zu, ist der Zugriff gestattet.

```
∀ s, ol, op, ob, ac, ar, dh: hasPositiveRight(?s,?ol,?op) ←
DiscretionaryACSubject(?s) ∧ DiscretionaryACObject(?ob) ∧
DiscretionaryAdminOperation(?op) ∧ DiscretionaryACModel(?ac) ∧
ObjectList(?ol) ∧ DiscretionaryRightDelegationHistory(?dh) ∧
hasOperationName(?op,"#delegateAccessRight") ∧ hasBaseACModel(?s,?ac) ∧
hasRightDelegationPolicy(?ac,"#transitiveDelegationPolicy") ∧
hasFirstListEntity(?ol,?su) ∧ hasNextListEntity(?su,?ar) ∧
isDelegable(?ar,"true") ∧ hasDelegatedRightHistory(?s,?dh) ∧
hasDelegatedRight(?dh,?ar)
```

Regel 5.37: Berechtigungskontrolle zur Rechtedelegation bei *TransitiveDelegationPolicy*.

5.4.2.2 Regeln für Rücknahme von weitergegebenen Rechten

Die Rücknahme von Berechtigungen ist nur Subjekten gestattet, die diese Rechte zu einem früheren Zeitpunkt weitergegeben haben. Die Operation `revokeAccessRightDelegation` beschreibt die für Rücknahmen zuständige Operation. Das dieser Operation zugeordnete Zugriffsrecht ist ebenfalls listenbasiert und die Struktur einer der zugehörigen Anfrage übergebenen Objektliste entspricht Abbildung 5.7. Auch hier ist die beschriebene Regel nur für die Zugriffskontrolle von Anfragen zuständig. Die eigentliche Durchführung der Operation erfolgt durch Aufruf einer zugehörigen Methode der Admin API.

Bei der Vergabe von Zugriffsberechtigungen wird überprüft, ob das Subjekt, dem das Recht entzogen werden soll, dieses überhaupt besitzt. Um die Anwendung der Regel 5.38 nur auf solche Zugriffskontrollmodelle zu beschränken, die Weitergaben von Berechtigungen gestatten, wird überprüft, ob die Strategie zur Rechteweitergabe dies erlaubt. Sind all diese Bedingungen erfüllt, wird durch Auswertung der Historie der Rechteweitergabe überprüft, ob der Anfragesteller `s` das zurückzunehmende Recht `ar` zuvor an das Subjekt `su` delegiert hat und ggf. der Zugriff gestattet. Regel 5.38 stellt das Vorgehen transitiver Rücknahmen dar. Für die Strategie *OneLevelRevocationPoliy* gestaltet sich das Vorgehen ähnlich.

5.5 Regeln zur Überprüfung der Modellkonsistenz

Die verwendete Ontologiesprache OWL sieht bereits einige Elemente zur Konsistenzprüfung vor [Pro04]. So können beispielsweise Klassen zueinander disjunkt definiert werden. Ist ein Individuum Instanz zweier disjunkter Klassen ist ein Modell inkonsistent. Auch Protégé [Pro00] bietet entsprechende Funktionalitäten zur Konsistenzprüfung von Ontologien an.

OWL bietet die Möglichkeit den Definitions- (engl.: Domain) und Zielbereich (engl.: Range) von Relationen durch Angaben von Konzepten zu spezifizieren. Ein Beispiel wäre die Relation `hasPet` deren Definitionsbereich dem Konzept `person` und deren Zielbereich dem Konzept `animal` zugeordnet ist. Das intuitive Verständnis dieser Domains und Ranges ist, dass die Relation `hasPet` nur zwischen Personen und Tieren definiert werden kann.

```
∀ s, ol, op, ob, ac, ar, su, dh: hasPositiveRight(?s,?ol,?op) ←
DiscretionaryACSubject(?s) ∧ DiscretionaryACSubject(?s) ∧
DiscretionaryACObject(?ob) ∧ DiscretionaryAdminOperation(?op) ∧
DiscretionaryACModel(?ac) ∧ ObjectList(?ol) ∧
DiscretionaryRightDelegationHistory(?dh) ∧ hasBaseACModel(?s,?ac) ∧
hasRightRevocationPolicy(?ac,"#transitiveRevocationPolicy") ∧
hasOperationName(?op,"#revokeAccessRightDelegation") ∧
¬ hasRightDelegationPolicy(?ac,"#noDelegationPolicy") ∧
hasFirstListEntity(?ol,?su) ∧ hasNextListEntity(?su,?ar) ∧
isDelegable(?ar,"true") ∧ hasDelegatedRightHistory(?su,?dh) ∧
hasDelegatedRight(?dh,?ar) ∧ delegatedFromSubject(?dh,?s)
```

Regel 5.38: Berechtigungskontrolle zur Rücknahme von weitergegebenen Rechten.

Dies ist jedoch nicht korrekt! OWL interpretiert die Angabe von Domains und Ranges nicht als Beschränkungen, die es einzuhalten gilt, sondern vielmehr als Axiome für das automatische Schließen. Würde die obige Relation beispielsweise einer Instanz des Konzepts `car` zugewiesen werden, würde dies keinen Fehler ergeben. Vielmehr würde ein OWL Reasoner daraus schließen, dass `car` Subklasse von `person` ist, vorausgesetzt die Konzepte von `car` und `person` wurden nicht disjunkt zueinander definiert. Diese Eigenschaft von OWL kann zu unerwünschten Klassifikationsergebnissen beim Reasoning führen. Deshalb sollten Domains und Ranges nur sehr bedacht eingesetzt werden.

In OWL können Relationen auch als funktional bzw. invers-funktional definiert werden. Dies bedeutet, dass eine Relation höchstens einmal auf einer Instanz definiert sein kann. Ein Beispiel für eine solche funktionale Relation wäre `hasMother`. Referenziert nun z.B. Instanz `Jon` `Alice` und `Peggy` in der Relation `hasMother`, so ist dies im Sinne von OWL nicht fehlerhaft. OWL schließt daraus, dass `Alice` und `Peggy` identische Instanzen sind. Nur wenn die beiden Individuen explizit als ungleich definiert werden würden, ergeben sich Inkonsistenzen.

Im Kontext von Zugriffskontrollsystemen macht diese Art der Verwendung obiger Eigenschaften keinen Sinn. Die Kombination von Regeln und Nicht-OWL-Prädikaten in KAON2 ermöglicht den Einsatz dieser OWL Eigenschaften im intuitiven Sinne. Das bedeutet, Domains, Ranges und funktionale bzw. invers-funktionale Relationen werden zu Beschränkungen transformiert, die bei falscher Verwendung einen Fehler erzeugen. Diese Erzeugung von Fehlern hilft die Konsistenz des Modells bei fehlerhafter Administration zu sichern.

Hierzu wurde im Rahmen dieser Arbeit ein entsprechender Konverter gebaut, der die zuvor genannten Eigenschaften einer OWL Ontologie in Regeln übersetzt. Die Umsetzung dieses Konverters wird in Abschnitt 6.1.1 detailliert betrachtet. Das Ergebnis der Transformation wird im Folgenden beispielhaft vorgestellt.

Domains und Ranges von Relationen: Als Beispiel wird hier die Relation `isPermis-`

sionSubject betrachtet, die eine Subjektinstanz einer Berechtigungsinstanz zuweist. Die Domäne dieser Relation ist auf Instanzen des Konzepts Subject und die Range auf Instanzen des Konzepts Permission beschränkt. Wird nun beispielsweise dieser Relation als Domäne eine Instanz des Konzepts Object zugewiesen, soll ein Fehler erzeugt werden. Hierzu wird das Prädikat domainError eingeführt. Argumente dieses Prädikats sind der Name der Relation deren Domäne nicht korrekt ist und die zugehörigen Quell- und Zielinstanzen. Regel 5.39 beschreibt die Erzeugung eines solchen Fehlers.

$$\forall\ o., t.: \texttt{domainError("\#isPermissionSubject",?origin,?target)} \leftarrow \texttt{isPermissionSubject(?origin,?target)} \wedge \neg\ \texttt{Subject(?origin)}$$

Regel 5.39: Konsistenzprüfung: Domain Error

Für die Überprüfung der Range einer Relation wird das Prädikat rangeError eingeführt, dessen Herleitung in Regel 5.40 beispielhaft dargestellt wird.

$$\forall\ o., t.: \texttt{rangeError("\#isPermissionSubject",?origin,?target)} \leftarrow \texttt{isPermissionSubject(?origin,?target)} \wedge \neg\ \texttt{Permission(?target)}$$

Regel 5.40: Konsistenzprüfung: Range Error

OWL Domains und Ranges können auch aus Vereinigungen von Konzepten bestehen. Die Relation isMemberOfCompositeEntity ist ein solches Beispiel. Die Domäne umfasst hier Instanzen der Klassen AtomicEntity und CompositeEntity. Liegt eine solche Vereinigung vor, erzeugt die Transformation Regeln, die die Zugehörigkeit einer Instanz zu allen Konzepten der Vereinigung überprüft.

Funktionale Eigenschaften: Um zu überprüfen, ob zu einer funktionalen Eigenschaft auch tatsächlich höchstens ein Element zugewiesen wurde, wird das Prädikat functionalError eingeführt. Dessen Argumente beschreiben den Namen der Relation, die Ausgangsinstanz und die zwei Zielinstanzen, die die funktionale Eigenschaft verletzen. Regel 5.41 formuliert eine konkrete Transformation am Beispiel der funktionalen Relation hasBaseACModel.

Damit zur Überprüfung der Konsistenz eines Zugriffskontrollmodells nicht jedes Prädikat einzeln abgefragt werden muss, wurde ein allgemeines Prädikat namens error eingeführt. Durch Regeln wie 5.42 und 5.43 werden die zuvor vorgestellten spezifischen Fehler auf dieses allgemeine Fehlerprädikat abgebildet. Um die Konsistenz eines konkreten Zugriffskontrollmodells zu prüfen, reicht eine einfache Anfrage nach dem error Prädikat aus. Das erste Argument beschreibt hierbei die Art des Fehlers, alle anderen Argumente

```
∀ origin, t1, t2: functionalError("#hasBaseACModel",?origin,?t1,?t2) ←
hasBaseACModel(?origin,?t1) ∧ hasBaseACModel(?origin,?t2) ∧
¬ equal(?t1,?t2)
```

Regel 5.41: Konsistenzprüfung: Functional Error

werden direkt von den jeweiligen Fehlerprädikaten übernommen. Da die Prädikate der unterschiedlichen Fehler sich in der Anzahl ihrer Argumente unterscheiden, müssen wie in Regel 5.42 ggf. Nullelemente als Platzhalter hinzugefügt werden.

```
∀ prop, o, t: error("#domainError",?prop,?o,?t,"#null") ←
domainError(?prop,?o,?t)
```

Regel 5.42: Abbildung von Domainfehlern auf allgemeine Fehler.

```
∀ prop, o, t1, t2: error("#functionalError",?prop,?o,?t1,?t2) ←
functionalError(?prop,?o,?t1,?t2)
```

Regel 5.43: Abbildung von funktionalen Fehlern auf allgemeine Fehler.

Kapitel 6

Implementierung

In diesem Kapitel werden wesentliche Aspekte und Besonderheiten der Umsetzung des semantischen Zugriffskontrollsystems betrachtet. Die Beschreibung des semantischen Modells erfolgt durch kombinierten Einsatz verschiedener Frameworks und Hilfswerkzeuge. Diese werden in Abschnitt 6.1 beschrieben. Um eine einfache Verwendbarkeit des Zugriffskontrollmodells zu gewährleisten, werden programmatische Schnittstellen zur Verfügung gestellt. Diese erlauben das initiale Einrichten und die Administration des Modells, sowie die Beantwortung von Zugriffsanfragen. In Abschnitt 6.2 werden diese Schnittstellen und deren interne Realisierung vorgestellt.

6.1 Erstellung des Zugriffskontrollmodells

Das Zugriffskontrollmodell wird durch die Ontologiesprache OWL und Regeln beschrieben. Protégé [Pro00] hat sich im Laufe der Zeit zu einem mächtigen Werkzeug zur Erstellung von Ontologien entwickelt. Eine komfortable grafische Oberfläche und eine Fülle an Erweiterungen zeichnen Protégé aus. Die aktuelle Version des KAON2 Rahmenwerks [KAON05] besitzt keine grafische Oberfläche für die Erstellung von Ontologien. Aufgrund der Komplexität des Modells war eine rein programmatische Modellierung unter Verwendung der KAON2 API nicht möglich. Daher wurde Protégé zur Beschreibung der Ontologien verwendet.

Der Schwerpunkt von Protégé liegt auf der Editierung von Ontologien. Reasoning auf Wissensbasen kann unter Verwendung eines DIG kompatiblen Reasoners, wie z.B. Racer oder Pellet, durchgeführt werden. DIG bezeichnet eine standardisierte XML-Schnittstelle zu Systemen, die auf Beschreibungslogik basieren. Das Reasoning in Protégé unterstützt derzeit ausschließlich Konsistenzprüfungen und Auswertungen von Klassifikationsbeziehungen auf Ontologien. Ein Plugin zur Beschreibung von SWRL [SWRL04] befindet sich derzeit in der Entwicklung. Eine Auswertung solcher SWRL Regeln ist jedoch zum aktuellen Zeitpunkt nicht möglich.

Wie in Abschnitt 2.2 bereits erwähnt, ermöglicht KAON2 automatisches Schließen auf Regeln und erweitert die OWL-Semantik um zusätzliche Logikelemente. Aufgrund der direkten Auswertung von Regeln auf OWL-Ontologien und der Mächtigkeit des Reasonings, wird KAON2 in dieser Arbeit als Reasoner zur Auswertung des Modells eingesetzt.

Somit müssen die in Protégé erstellten Ontologien in KAON2 verwendet werden. Da

beide Rahmenwerke auf standardisiertem OWL beruhen, erfordert die notwendige Konvertierung nur minimale Anpassungen. Im Zuge dieser Konvertierung wurden zusätzliche Änderungen an der Ontologie und das Hinzufügen von KAON2 spezifischen Regeln durch zwei Hilfswerkzeuge durchgeführt. Diese werden im Folgenden kurz vorgestellt.

6.1.1 OWL Constraints Konverter

Bereits in Abschnitt 5.5 wurden Regeln eingeführt, die der Konsistenzprüfung des Modells dienen. Diese Regeln werden aus den in Protégé modellierten OWL-Ontologien abgeleitet. Hierzu werden Domain- und Range-Restriktionen, sowie funktionale und invers-funktionale Eigenschaften von Relationen durch entsprechende Regeln ersetzt. Diese Regeln beschränken die Relationen in Bezug auf die genannten Eigenschaften und erzeugen einen Fehler, falls diese nicht erfüllt sind.

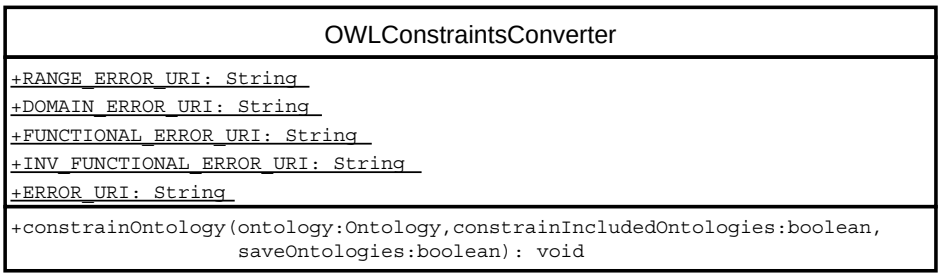

Abbildung 6.1: UML Klassendiagramm des OWL Constraints Konverters.

Die in Abbildung 6.1 dargestellte Klasse `OWLConstraintsConverter` führt diese Transformation durch. Die im Klassendiagramm dargestellten URIs entsprechen den jeweiligen Identifikatoren der abgeleiteten Fehlerklassen, die für die Konsistenzprüfung abgefragt werden können. Eingabe für den Konverter ist eine Ontologie, die Domains, Ranges, funktionale und invers-funktionale Relationen besitzt. Der Konverter analysiert jede in der Ontologie enthaltene Relation, erzeugt ggf. eine geeignete Regel und löscht danach den entsprechende Ausdruck in der Ontologie. Die Ausgabe ist somit eine um oben genannte Eigenschaften bereinigte Ontologie.

Dieser Konverter wird in der Zwischenzeit auch im Projekt OntoGov[1] (Ontology-enabled e-Gov Service Configuration) am FZI verwendet.

6.1.2 Regelparser

Um die Erweiterbarkeit des Modells zu gewährleisten, muss dem Administrator des Systems die Möglichkeit gegeben werden, konzeptionelle Ontologieerweiterungen und die Definition zusätzlicher Regeln durchzuführen. Während die konzeptionelle Erweiterung durch Werkzeuge wie Protégé grafisch erfolgen kann, ist die programmatische Eingabe von Regeln mittels der KAON2 API eher mühsam. Um die Erstellung von Regeln zu vereinfachen, wurde im Verlauf dieser Arbeit ein Regelparser entwickelt. Dieser nimmt eine Regel in Form einer Zeichenkette entgegen und erzeugt daraus ein KAON2 Regelobjekt, dass daraufhin der

[1] http://www.ontogov.com

Zeichen	Bedeutung
>	Trennung von Rumpf und Kopf der Regel
,	konjunktive Verknüpfung von Atomen im Rumpf der Regel
!	nichtomontone Negation
?	Variabeln von Relationen, Klassenatomen und Nicht-OWL-Prädikaten
"x"	`x` ist eine Konstante (Instanz URI oder DataProperty Werte)

Tabelle 6.1: Elemente der Zeichenkette des Regelparsers.

Ontologie hinzugefügt werden kann. Somit ist beispielsweise eine einfache Definition neuer Regeln über eine grafische Oberfläche möglich.

Der Regelparser unterstützt eine Untermenge aller durch KAON2 beschreibbaren Regeln. So werden Regeln auf einfache Hornklauseln beschränkt, deren Kopf nur aus einem einzelnen Atom bestehen kann. Zudem werden *Built-ins*, die zur Ausführung von Berechnungen und Vergleichen von Datenelementen verwendet werden können, bisher nicht unterstützt.

Die Syntax der Regeln lehnt sich an das in SWRL definierte Format an und orientiert sich hierbei in weiten Teilen an der Syntax des SWRL Plugins von Protégé. Die folgende Zeichenkette stellt eine einfache Regel im Eingabeformat des Parsers beispielhaft dar:

```
parent(?x,?y), !male(?y) > mother(?x,?y)
```

Ist `y` ein Elternteil von `x` und keine Instanz der Klasse `male`, so ist `y` die Mutter von `x`. `parent(?x,?y)` beschreibt hierbei eine Relation zwischen zwei Instanzen und `male(?y)` ein sogenanntes Klassenatom, das `y` als Instanz der Klasse `male` kennzeichnet. Zusätzlich zu den genannten Elementen werden vom Parser noch KAON2 Nicht-OWL-Prädikate unterstützt, die sich durch eine beliebige Stelligkeit auszeichnen.

In Tabelle 6.1 sind mögliche Elemente einer Eingabezeichenkette und deren Bedeutung dargestellt.

Zusätzliche Information können der Javadoc Dokumentation der Klasse `RuleParser` entnommen werden.

6.1.3 JUnit basierte Verifikation des Modells

Zur Auswertung des Modells und der Auflösung von Konflikten ist eine Vielzahl komplexer Regeln notwendig. Wie in Kapitel 5 beschrieben, bauen diese teilweise aufeinander auf oder interpretieren widersprüchliche Sachverhalte. Ziel aller Regeln ist es, eine Zugriffsanfrage zu beantworten, d.h. das zuvor vorgestellte Prädikat `accessGranted` abzuleiten. Dies kann jedoch auf vielfältige Art und Weise geschehen. So könnte ein Zugriff beispielsweise aufgrund einer positiven Berechtigung des anfragenden Subjekts oder aber aufgrund fehlender Informationen kombiniert mit einem offenen Zugriffskontrollsystem gestattet werden. Somit ist eine einfache Betrachtung des Endergebnisses nicht ausreichend, da i.A. nicht unmittelbar nachvollziehbar ist, welche Regel zu dem Ergebnis geführt hat. Zudem können minimale Änderungen bestehender Regeln das Verhalten abhängiger Regeln beeinflussen und somit zu unerwünschten Seiteneffekten führen.

Um diese mangelnde Kontrolle der Herleitung von Anfrageergebnissen einzugrenzen, wurde während der Umsetzung des Modells eine automatische Testumgebung unter Verwendung von *JUnit* implementiert. Die Funktionalität jeder Regel wird durch eine eigene Testklasse überprüft. Jede dieser Testklassen generiert durch Instantiierung geeigneter Konzepte einen Testfall, der das spezifische Verhalten der Regel und deren Interaktion mit abhängigen Regeln demonstriert.

Diese sehr aufwändige Vorgehensweise erlaubte es, frühzeitig konzeptionelle Fehler bei der Erstellung des Modells zu erkennen. Die Auswirkungen von späteren Änderungen der Ontologien und Regeln konnten durch einfaches Ausführen der Testfälle überprüft werden. Aufgrund der Komplexität der Regeln und der Vielzahl unterschiedlicher Tests eignen sich diese Testfälle auch zur Überprüfung der Korrektheit neuer Version des KAON2 Frameworks und wurden in die Testumgebung von KAON2 integriert.

6.2 Software Infrastruktur

Der Hauptvorteil der semantischen Modellierung des Zugriffskontrollsystems ist, dass zur Beantwortung von Zugriffsanfragen, außer der Formulierung einer Anfrage an das Inferenzsystem, keinerlei Programmcode notwendig ist. Die Administration des Modells kann jedoch nur in Software erfolgen. Grundsätzlich bieten sich hierzu grafischen Ontologieeditoren wie Protégé an. Ziel der Zugriffskontrollmodellierung ist jedoch, die Zugriffskontrolle auf sich selbst anzuwenden, d.h. Zugriffe die das Modell verändern nur autorisierten Benutzern zu erlauben. Um diese administrativen Zugriffe zu kontrollieren, muss eine entsprechende Software Infrastruktur zur Verfügung gestellt werden.

Die Bestandteile dieser Infrastruktur wurden in Abbildung 3.1 auf Seite 29 bereits dargestellt. Die Query, Setup und Admin API werden in den folgenden Kapiteln erläutert. Es handelt sich hierbei um prototypische Implementierungen, die die allgemeine Vorgehensweise erläutern sollen und für konkrete Anwendungen entsprechend erweitert werden müssen.

6.2.1 Setup API

Die Setup API dient der Initialisierung und der Anpassung des Modells zur Beschreibung organisationsspezifischer Sicherheitsstrategien. Hier legt der in Abbildung 3.1 dargestellte ACModel Designer die grundlegenden Eigenschaften der Zugriffskontrolle fest. Individuelle Anpassungen des Modells auf Konzeptebene werden durch die Setup API nicht unterstützt, sondern können beispielsweise in Protégé vorgenommen werden. Diese Erweiterungen werden in einer separaten Ontologie gespeichert und zum Zeitpunkt des Setups durch importieren dieser Ontologie verfügbar gemacht.

Das Setup legt unter anderem fest, welche Arten von Berechtigungen erlaubt sind, d.h. ob es sich um ein positives, negatives oder gemischtes Zugriffskontrollmodell handelt. Im letztgenannten Fall wird zudem eine globale Konfliktauflösungsstrategie spezifiziert. In jedem Falle muss eine Standardstrategie angegeben werden, die beschreibt wie sich das System verhält, falls keine Informationen für eine Zugriffsanfrage vorliegen. Während des Setup Vorgangs wird eine Ontologie angelegt, in der später alle Instanzen des Zugriffskontrollmodells gespeichert werden. Diese Instanzontologie ist die Grundlage des späteren Reasonings

und importiert alle notwendigen Ontologien, wie z.B. die zuvor beschriebene Ontologie der Ergänzungsebene.

All diese Eigenschaften werden durch Aufruf der Methode `initiateACModel()` der Klasse `ACModelInitiatorCore` festgelegt. Neben der Erzeugung der Instanzontologie bewirkt der Aufruf dieser Methode das Anlegen einer Property-Datei. Diese Datei speichert die wichtigsten Informationen, die für den Zugriff auf das Zugriffskontrollmodell notwendig sind. Dazu gehören die physikalische und logische URI der Instanzontologie und die ID der Instanz des Zugriffskontrollmodells, an die spätere Anfragen gerichtet werden. Zum Zeitpunkt einer Anfrage an das Zugriffskontrollsystem oder der Administration des Modells, werden diese Informationen aus der Datei extrahiert und für den Zugriff auf die Ontologien zur Verfügung gestellt. Die in Abbildung 6.2 dargestellte Klasse `ACModelConnection` wertet die gespeicherten Informationen aus und unterstützt somit einen einfachen Zugriff auf das Zugriffskontrollsystem. Ist eine Verbindung geöffnet, können alle in der Instanzontologie gespeicherten Informationen ausgehend von der referenzierten Zugriffskontrollinstanz durch Aufruf von Methoden abgefragt werden. Die notwendigen Maßnahmen zur Verwaltung der dem Zugriffskontrollmodell zugrunde liegenden Ontologien werden somit durch die Klasse `ACModelConnection` vollständig vor dem Benutzer verborgen.

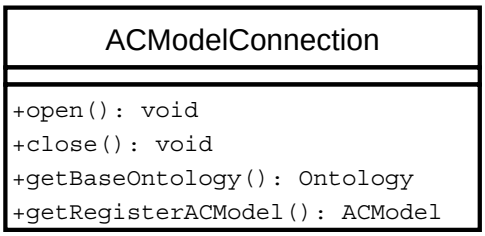

Abbildung 6.2: UML Diagramm der Klasse ACModelConnection.

6.2.2 Administrations API

Aufgabe der Admin API ist es, Funktionalitäten zur Änderung des Zustands des Zugriffskontrollmodells zur Verfügung zu stellen. Beispiele solcher Zustandsänderungen können das Hinzufügen neuer Subjekte und Objekte oder die Rücknahme von Berechtigungen sein. Diese Zustandsänderungen bewirken stets eine Änderung der zugehörigen Ontologien.

Um die Administrierbarkeit des Systems zu gewährleisten, werden die wichtigsten Bestandteile der Zugriffskontrolle in Form von Klassen beschrieben. Im Wesentlichen handelt es sich dabei um Abbildung der in den Ontologien beschriebenen Konzepte und deren Eigenschaften. Zentrales Element ist die in Abbildung 6.3 dargestellte Klasse `ACModel`. Diese repräsentiert die zur Initialisierungszeit beschriebene Modellinstanz und bietet Methoden zur Verifikation des Modells oder der persistenten Ablage etwaiger Änderungen in den zugehörigen Ontologien. Ausgehend von Objekten dieser Klassen können alle Elemente eines Zugriffskontrollsystems durch Aufruf geeigneter Methoden extrahiert werden.

Um eine ausreichende Flexibilität und leichtere Verständlichkeit dieser API zu gewährleisten, wurden Entwurfsmuster eingesetzt deren Einsatz im Folgenden kurz erläutert wird:

Fassade: Eine Fassade (engl. Facade) stellt per Definition Schnittstellen zur Verfügung, die die Benutzung eines Subsystems vereinfachen. In der Implementierung der Ad-

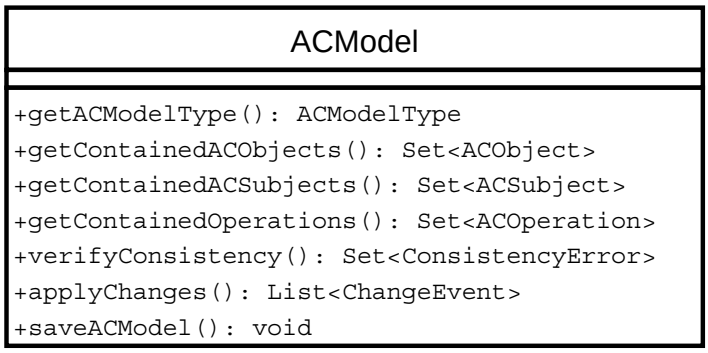

Abbildung 6.3: UML Diagramm der Klasse ACModel.

min und Query API wird dieses Entwurfsmuster eingesetzt, um Ontologiezugriffe vor dem Benutzer zu verdecken. Die Zugriffsschicht des Benutzers besteht aus einer Menge von Schnittstellen, die zumeist nur `get()` Methoden zur Abfrage von Informationen anbieten. Die Klassen, die diese Schnittstellen implementieren, besitzen zusätzliche Funktionalität zur Verwaltung der Ontologieinformationen. Die Fassade dient dazu, die Zugriffsmöglichkeiten des Benutzers auf ein überschaubares, notwendiges Minimum an Methoden zu beschränken. Da die Objekte des Benutzerzugriffs hinter Schnittstellen verborgen sind, können zudem die unterliegenden Implementierungen einfach ausgetauscht werden.

Abstrakte Fabrik: Um die soeben erwähnte Austauschbarkeit von Implementierungen zu unterstützen, werden abstrakte Fabriken (engl. Abstract Factory) eingesetzt. Diese stellen Schnittstellen zur Erzeugung von Familien von Objekten zur Verfügung, ohne die konkreten Klassen dieser Objekte festzulegen. Im Rahmen dieser Arbeit werden Fabriken eingesetzt, um die Austauschbarkeit von Implementierungen zu gewährleisten und somit die Flexibilität zu erhöhen. So können in Abhängigkeit des gewählten Typs des Zugriffskontrollmodells, unterschiedliche konkrete Fabriken implementiert werden, die beispielsweise wahlfreie, regelbasierte oder rollenbasierte Zugriffskontrollelemente erzeugen. Für den Benutzer der Anwendung ist der Austausch dieser Implementierungen durch Verwendung einer abstrakten Fabrik transparent.

Besucher: Ein Besucher (engl. Visitor) dient der Kapselung von Operationen, die auf Objekten ausgeführt werden können. Somit lassen sich neue Operationen durch Erweiterung des Besuchers, ohne Änderungen der entsprechenden Objektklassen, beschreiben. In der Setup API wird dieses Entwurfsmuster verwendet, um Ontologiezugriffe zu implementieren. Alle Operationen, die schreibend oder lesend auf Ontologien zugreifen, sind zentral in einen Besucher ausgelagert. Durch die von den Objekten unabhängige Anpassbarkeit dieses Ontologiezugriffs, wird die Flexibilität des Modells erweitert.

In Abbildung 6.4 werden einige Schnittstellen und Methoden das `elements` Packages beispielhaft dargestellt. Zentrale abstrakte Oberklasse ist `ACEntity`, die die gemeinsamen Eigenschaften aller in einem Zugriffskontrollmodell enthaltenen Elemente beschreibt. Die in dieser Abbildung dargestellten Elemente beschreiben die Zugriffsschicht des Benutzers, die als Fassade über die eigentliche Implementierung gelegt wurde.

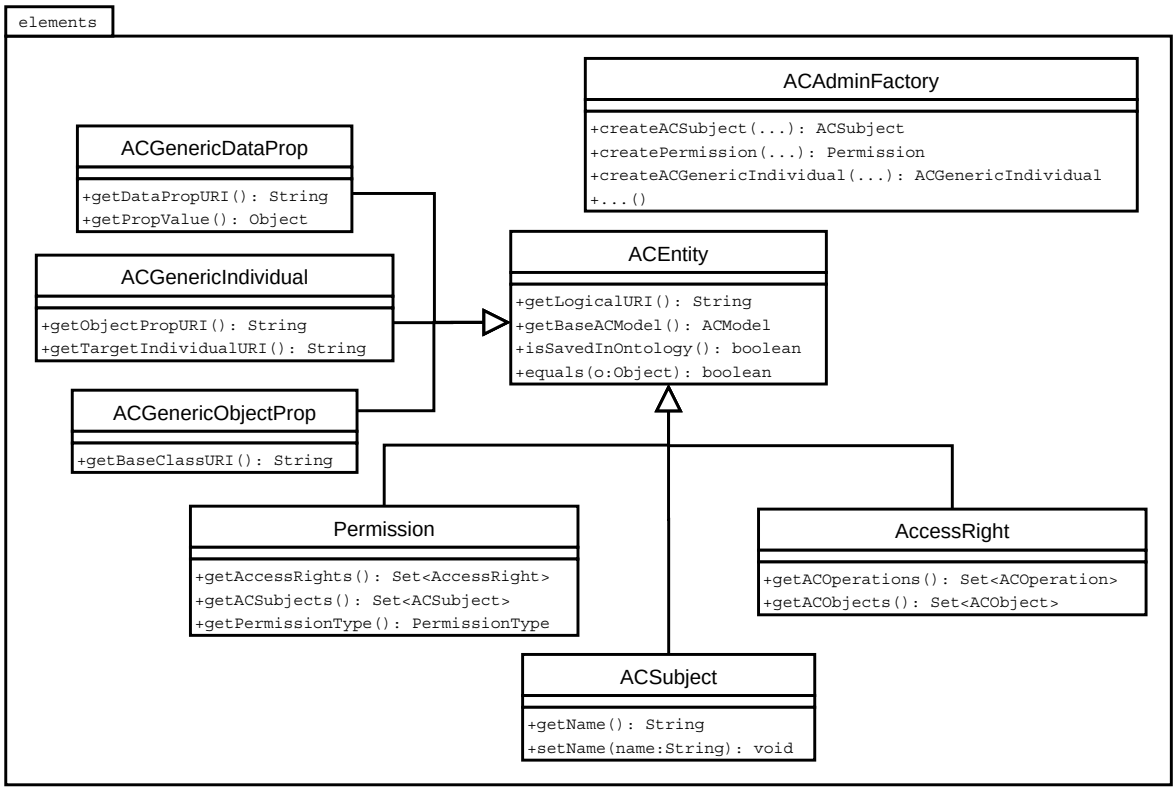

Abbildung 6.4: Struktur des `elements` Packages.

Funktionen zur Durchführung von Änderungen des Modells werden im `change` Package umgesetzt. Dieses stellt generische Events zur Änderung des Ontologiezustandes bereit und ist unter Verwendung des Besuchermusters realisiert. Konkrete Vorgänge, wie z.B. das Hinzufügen eines Subjektes, werden durch kombinierte Ausführung generischer `ChangeEvent` Instanzen beschrieben. Einige der beschriebenen Elemente werden in Abbildung 6.5 dargestellt.

6.2.3 Query API

Die Query API stellt den eigentlichen Kern der Software Infrastruktur dar, da sie für die Beantwortung von Zugriffsanfragen zuständig ist. Hierzu werden entsprechende Anfrage- und Antwortobjekte beschrieben, die alle notwendigen Informationen für die Beantwortung einer Zugriffsanfrage enthalten. Bei den Anfrageobjekten wird zwischen normalen Autorisierungsanfragen (`AuthorisationRequest`) und administrativen Anfragen (`AdministrativeRequest`) unterschieden. Administrative Anfragen bewirken eine Zustandsänderung des Zugriffskontrollmodells. Wird die Ausführung einer administrativen Anfrage genehmigt, muss diese Änderung unter Verwendung der in der Admin API umgesetzten generischen `ChangeEvents` in das Modell eingebracht werden. Die Liste der jeweils notwendigen generischen Events werden unter Verwendung der `calculateChangeEvents()` Methode berechnet. I.A. unterscheidet man Anfragen, die Elemente hinzufügen oder entfernen (`AbstractACEntityRequest`), und Anfragen, die Relationen zwischen zwei Elementen bearbeiten (`AbstractACRelationRequest`). Dies wird in Abbildung 6.6 beispielhaft darge-

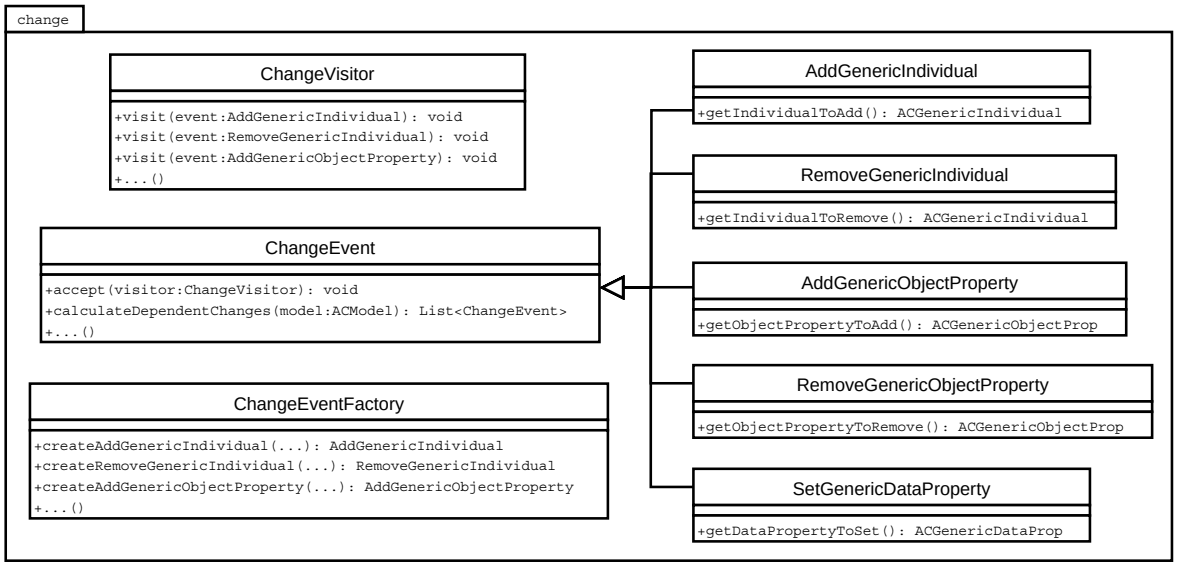

Abbildung 6.5: Struktur des `change` Packages.

stellt.

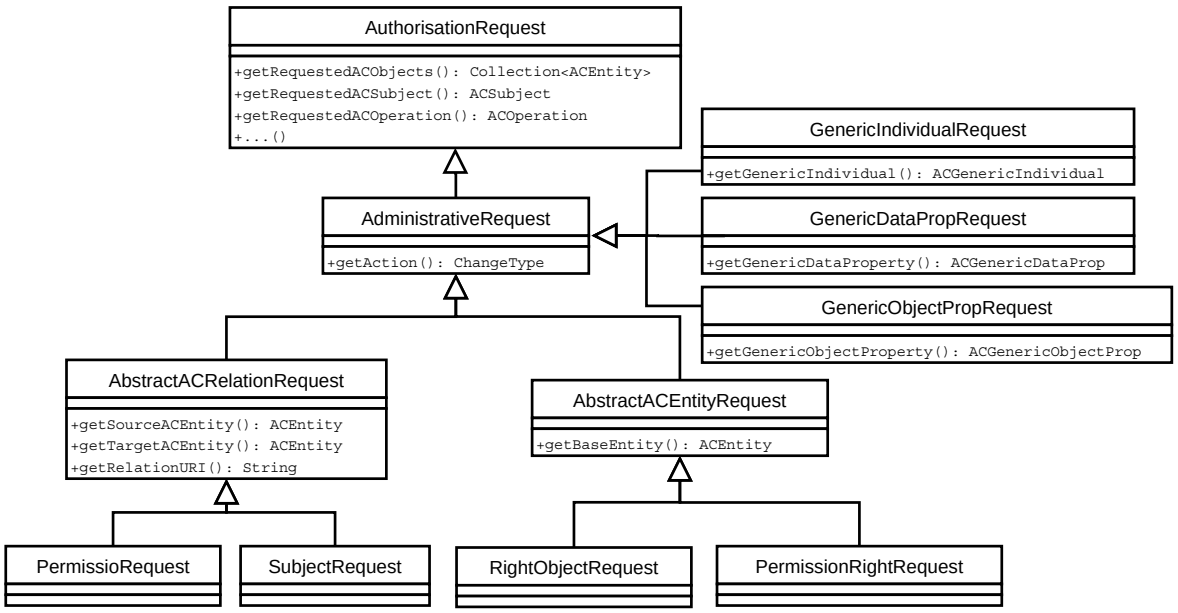

Abbildung 6.6: Vererbungsstruktur administrativer Anfragen.

Die Beantwortung von Zugriffsanfragen erfolgt durch die in Abbildung 6.7 dargestellten Klassen `ACQueryComputer` und `ACAuthorisationComputer`. Mittels `ACAuthorisationComputer` können konkrete Anfragen an das Zugriffskontrollsystem formuliert werden. Diese entscheiden für gegebene Subjekt-, Objekt- und Operationsinstanzen, ob ein Zugriff ausgeführt werden darf oder nicht. Handelt es sich um eine administrative Anfrage wird dies erkannt und bei einem positiven Ergebnis der Anfrage auch gleich auf das Zugriffskontrollmodell, durch Aufruf zuvor berechneter generischer `ChangeEvents`, ausgeführt. Somit stellt diese Klasse auch gleichzeitig den in Abbildung 3.1 auf Seite 29 dargestellten Referenzmo-

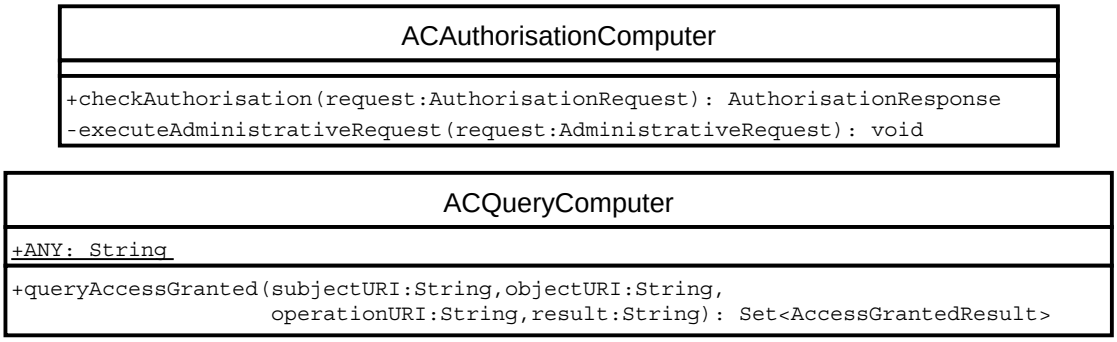

Abbildung 6.7: UML Diagramme der Klassen zur Formulierung von Zugriffsanfragen.

nitor dar. Allgemeinere Anfragen lassen sich unter Verwendung der Klasse `ACQueryComputer` beantworten. Elemente der Anfrage können hierbei durch entsprechende Platzhalter ersetzt werden. Wird z.B. einer Anfrage statt einer konkreten Objektinstanz ein Platzhalter übergeben, bezieht sich diese Anfrage auf alle im System enthaltenen Objektinstanzen. Somit kann man beispielsweise prüfen, auf welche Objekte des Systems ein Subjekt lesend zugreifen darf. In beiden Klassen werden Zugriffsanfragen ausschließlich durch Ableiten des Prädikats `accessGranted` unter Verwendung des KAON2 Reasonings beantwortet.

Kapitel 7

Evaluation

Basierend auf der zuvor vorgestellten Umsetzung der Zugriffskontrollkomponente, betrachtet der folgende Abschnitt die Erreichung der zu Beginn formulierten Ziele kritisch. Im Verlauf dieser Arbeit haben sich immer wieder einzelne Probleme und Beschränkungen ergeben, die die Umsetzung dieser Ziele in unterschiedlichem Maße beeinflussten. Im Gegenzug ergaben sich, vor allem durch die stetige Fortentwicklung der KAON2 Infrastruktur und den dadurch gegebenen zusätzlichen Möglichkeiten des automatischen Schließens, eine Reihe von Verbesserungen.

7.1 Erweiterbarkeit

In dieser Arbeit wird ein Zugriffskontrollsystem als eigenständige Komponente realisiert, die über wohldefinierte Schnittstellen von externen Anwendungen verwendet werden kann. Durch Verwendung dieser Komponente lässt sich eine organisationsspezifische Sicherheitsstrategie an zentraler Stelle über Anwendungs- und Systemgrenzen hinweg beschreiben. Auch Anpassungen aufgrund von Änderungen der Organisationsstruktur eines Unternehmens können an zentraler Stelle durchgeführt werden.

Um eine vielseitige Verwendbarkeit dieser Komponente zu gewährleisten, ist es notwendig individuelle Anpassungen auf einfache Art und Weise vornehmen zu können. Die Verwendung semantischer Technologien in Form von Ontologien und Regeln bietet hier große Vorteile. Anpassungen des Zugriffskontrollsystems können durch einfache Instantiierung vorhandener Konzepte oder der Beschreibung konzeptioneller Erweiterungen und der Definition neuer Regeln zur Interpretation dieser neuen Konzepte durchgeführt werden. Im Allgemeinen ist hierzu keinerlei Codeanpassung notwendig.

Das erstellte Zugriffskontrollmodell ist in der Lage, die wichtigsten Eigenschaften gängiger Zugriffskontrollsysteme zu beschreiben. Darüber hinaus werden auf der Systemebene Erweiterungen definiert, die eine einfache Umsetzung gängiger Zugriffskontrollsysteme ermöglichen. Aus zeitlichen Gründen musste die Implementierung der Systemschicht jedoch leider entfallen, so dass diese über die Schnittstellen der Komponente derzeit nicht verfügbar ist.

Um die Erweiterbarkeit des Modells bewerten zu können, wird die Umsetzung der Unix Zugriffskontrolle betrachtet. Bei der Unix Zugriffskontrolle handelt es sich – ähnlich wie bei den in Abschnitt 4.4.2 besprochenen wahlfreien Systemen – um eine eigentümerbasierte

Zugriffskontrolle. Zusätzlich zu einem Eigentümer wird jedem Objekt eine Subjektgruppe zugeordnet. Für jedes Objekt eines Unixsystems werden Berechtigungen für den Besitzer, die zugewiesene Subjektgruppe und alle anderen Benutzer des Systems vergeben. Abbildung 2.7 auf Seite 18 stellt eine unixtypische Zugriffskontrollliste für ein einzelnes Unixobjekt dar. Die im Folgenden beschriebenen Erweiterungen setzen auf den Konzepten der Ergänzungsschicht des Zugriffskontrollmodells auf.

Eine detaillierte Betrachtung der Umsetzung der Unix Zugriffskontrolle erfolgt in Abschnitt A des Anhangs. An dieser Stelle werden die Ergebnisse der Umsetzung besprochen. Anhand der Beschreibung der Unix Zugriffskontrolle zeigt sich, dass die Beschreibung konkreter Anpassungen des Zugriffskontrollmodells unter geringem Aufwand möglich ist. Die oben genannten zusätzlichen Eigenschaften von Unixobjekten wurden durch drei neue Subkonzepte und zwei zusätzliche Relationen realisiert. Die Rechte des Eigentümers, der Benutzergruppe des Objekts und aller anderen Systembenutzer auf einem Unixobjekt werden durch Abbildung auf entsprechende Instanzen des allgemeinen Berechtigungskonzeptes modelliert.

Aufgrund der zusätzlichen Eigenschaften der Unix Zugriffskontrolle werden drei zusätzliche Regeln benötigt. Im Gegensatz zu anderen Systemen darf jeder Benutzer neue Objekte im System anlegen. Dies wird durch eine entsprechende Regel beschrieben. Zur Beantwortung einer Zugriffsanfrage muss zudem entschieden werden, ob es sich bei dem gegebenen Subjekt um den Besitzer oder ein Mitglied der dem Objekt zugewiesenen Benutzergruppe handelt. Trifft beides nicht zu, wird das Subjekt der „*Gruppe aller anderen Nutzer des Systems*“ zugeordnet. Zur Beschreibung dieses Sachverhalts werden zusätzliche Regeln benötigt. Details zur Umsetzung dieser Regeln können Abschnitt A.3 des Anhangs entnommen werden.

Anhand der Beschreibung der unixspezifischen Zugriffskontrolle wird ersichtlich, dass Anpassungen zur Beschreibung systemspezifischer Sicherheitsstrategien auf dem Zugriffskontrollmodell leicht umzusetzen sind. Unter geringem Aufwand werden zusätzliche Konzepte und Regeln zu deren Interpretation definiert. Diese Regeln gewährleisten, durch Abbildung auf vorhandene Konzepte und Prädikate, dass Anfragen ungeändert an das System gestellt werden können und keinerlei Anpassungen des Programmcodes zur Beantwortung von Zugriffsanfragen notwendig sind.

Die Umsetzung administrativer Operationen, wie `chmod` oder `chown` bei Unixsystemen, erfordert jedoch Anpassungen der Schnittstellen. Die Änderung des Ontologiezustandes durch Aufruf einer administrativen Operation kann ausschließlich durch Programmcode formuliert werden. Aufgrund der erweiterbaren Architektur der Schnittstellen können diese Anpassungen jedoch ohne großen Aufwand implementiert werden.

Zusammenfassend lässt sich sagen, dass die Erweiterbarkeit des Systems aufgrund der Verwendung semantischer Technologien in Form von Ontologien und Regeln als sehr gut bewertet werden kann. Die zu Beginn beschriebenen Anforderungen bezüglich der Erweiterbarkeit der Zugriffskontrollkomponente wurden somit erreicht. Aufgrund der Mächtigkeit des zugrunde liegenden Zugriffskontrollmodells und der Möglichkeit eigener Anpassungen sind Beschreibungen organisationsspezifischer Sicherheitsstrategien auf einfache Art und Weise zu realisieren.

7.2 Mächtigkeit von Anfragen

Das Hauptaugenmerk der Zugriffskontrollkomponente liegt auf der Beantwortung von Zugriffsanfragen. Diese Zugriffsanfragen werden der Query API übergeben und von dieser durch automatisches Schließen, basierend auf den Fakten des semantischen Zugriffskontrollmodells, beantwortet. Dies erfolgt durch Ableiten des in Abschnitt 5.3 vorgestellten Prädikats `accessGranted`. Automatisches Schließen in KAON2 unterstützt Elemente, die über gewöhnliche OWL Axiome hinausgehen. Diese Erweiterungen umfassen Nicht-OWL-Prädikate, die sich durch eine beliebige Stelligkeit auszeichnen und somit die Verknüpfung mehrerer Instanzen in einem Ausdruck ermöglichen. Das Prädikat `accessGranted` verknüpft z.B. Subjekt, Objekt, Operation und das Ergebnis eine Anfrage in Form eines Wahrheitswertes und vereint somit alle Informationen, die zur Beantwortung einer Anfrage notwendig sind. Zudem wird nichtmonotones Schließen in Form der nichtmonotonen Negation (engl.: *Negation as Failure*) unterstützt. Die nichtmonotone Negation ist essentiell für die Beantwortung von Zugriffsanfragen. Man kann somit beispielsweise überprüfen, ob für eine Anfrage eine entsprechende Eigenschaft nicht gilt, d.h. keine Information in der Wissensbasis zu dieser Eigenschaft gefunden werden konnte. In monotoner Logik ist dies nicht beschreibbar.

Die genannten KAON2 Erweiterungen erlauben die vollständige Beantwortung von Zugriffsanfragen durch Regeln. Auch Konflikte, aufgrund sich widersprechender positiver und negativer Berechtigungen, können vollständig durch Regeln aufgelöst werden. Dadurch kann die Beantwortung einer Zugriffsanfrage ohne Programmcode durchgeführt werden, da die Logik zur Beantwortung von Zugriffsanfragen im Modell enthalten ist.

Wie bereits erwähnt, erfolgt die Beantwortung von Zugriffsanfragen durch Ableiten des Prädikats `accessGranted`. Es ist das zentrale Element des Zugriffskontrollmodells und stellt die Schnittstelle des Modells nach außen dar. Die interne Struktur des Zugriffskontrollmodells ist durch die Beschränkung von Anfragen auf das Prädikat `accessGranted` für die Formulierung einer Zugriffsanfrage nicht von Interesse. Im Gegensatz zu datenbankorientierten Ansätzen, in denen man zur Beschreibung von Anfragen stets die zugrundeliegenden Datenbankschemata kennen muss, ist dies ein großer Vorteil.

Konzeptionell lassen sich mit dem Prädikat `accessGranted` beliebige Anfragen formulieren. Häufigste Verwendung ist eine konkrete Autorisierungssanfrage, die mit einem einfachen Wahrheitswert beantwortet wird. Ein Beispiel hierfür ist die Anfrage: „Darf Benutzer *Joe* lesend auf Datei *example.txt* zugreifen?“ Prinzipiell lassen sich jedoch allgemeine Anfragen nach jedem beliebigen Argument von `accessGranted` stellen. So lässt sich beispielsweise eine Anfrage formulieren, die alle Subjekte ermittelt, die auf ein konkretes Objekt zugreifen dürfen. Dazu wird der Anfrage das konkrete Objekt übergeben und der Wahrheitswert des Prädikats auf `true` gesetzt. Das Ergebnis der Anfrage sind eine Menge von Tupeln der Form `<ACSubject, ACOperation>`. Jedes dieser Tupel beschreibt eine Operation mit der das jeweilige Subjekt auf das Objekt der Anfrage zugreifen darf. Auch die Anfrage welche Operationen ein gegebener Benutzer auf einem konkreten Objekt ausführen darf, kann direkt beantwortet werden.

Im Verlauf der Umsetzung des Modells ergaben sich jedoch bei der Beantwortung allgemeiner Zugriffsanfragen Probleme. Abhängig von der Art des verwendeten Zugriffskontrollmodells können allgemeine Anfragen nicht oder nur teilweise beantwortet werden, da KAON2 das zugehörige logische Programm als *nicht stratifizierbar* klassifiziert. Details zur

Stratifikation eines logischen Programms können in Abschnitt B des Anhangs nachgelesen werden. Bei gemischten Zugriffskontrollmodellen, die sowohl positive als auch negative Berechtigungen zulassen und dadurch zusätzliche Regeln zur Konfliktauflösung beim automatischen Schließen benötigen, können aufgrund der mangelnden Stratifizierbarkeit ausschließlich konkrete Autorisierungsanfragen beantwortet werden. Bei den weit verbreiteten positiven bzw. negativen Systemen, die nur die Verwendung positiver bzw. negativer Berechtigungen zulassen, sind beliebige Anfragen möglich. Für die in Abschnitt A des Anhangs beschriebene Erweiterung eines positiven Systems zur Umsetzung der Unix Zugriffskontrolle, gelten wiederum Einschränkungen. Aufgrund der zusätzlichen Regeln und den darin enthaltenen Negationen können Zugriffsanfragen, die Berechtigungen beliebiger Objekte des Systems in Erfahrung bringen wollen, nicht beantwortet werden. Dies bedeutet, dass die Anfrage „mit welchen Operationen darf ein gegebener Benutzer auf die Objekte des Systems zugreifen?“ aufgrund der Nichtstratifizierbarkeit eine Ausnahme erzeugt. Zusammenfassend lässt sich sagen, dass bei gemischten Systemen ausschließlich konkrete Anfragen, bei positiven und negativen Systemen beliebige Anfragen und in der Erweiterung der Unix Zugriffskontrolle ausschließlich Anfragen mit einer konkreten Objektinstanz durch das Zugriffskontrollmodell beantwortet werden können.

Die nahe liegende Ursache der Nichtstratifizierbarkeit des Zugriffskontrollmodells ist das Vorhandensein eines Zyklus mit negativ markierter Kante im Abhängigkeitsgraph des Modells. Die Anwendung des in Abschnitt B des Anhangs vorgestellten Algorithmus zur Überprüfung der Stratifizierbarkeit auf den Regeln des Zugriffskontrollmodells konnte einen solchen Zyklus jedoch nicht finden. Das Zugriffskontrollmodell ist somit stratifizierbar. Nichtsdestotrotz kommt es bei der Beantwortung von Anfragen in oben genannten Fällen zu Ausnahmen aufgrund der Nichtstratifizierbarkeit des zugehörigen logischen Programms. Aufgrund dieser Tatsache muss die Ursache der Ausnahme im Inferencingkern von KAON2 liegen. KAON2 beschreibt logische Programme als Datalog Programme. Zum automatischen Schließen werden diese Programme in sogenannte *Magic Sets* umgewandelt. In der Theorie [Chen97] ist bekannt, dass diese Transformation in speziellen Fällen zur Nichtstratifizierbarkeit eines stratifizierten Programms führen kann.

Im zeitlich begrenzten Rahmen dieser Diplomarbeit war es leider nicht möglich eine Lösung für dieses Problem zu finden, da dies eine Erweiterung des KAON2 Inferencings bedeutet. Spätere Versionen von KAON2, die das Problem bei der internen Umsetzung berücksichtigen, sollten somit die Möglichkeit bieten, beliebige Anfragen an das Zugriffskontrollmodell beantworten zu können.

Die aktuelle Version der Zugriffskontrollkomponente unterstützt somit nur eine Untermenge aller möglichen Anfragen. Nicht ableitbare Anfragen werden durch eine entsprechende Ausnahme gekennzeichnet. Die Beantwortung typischer konkreter Autorisierungsanfragen wird dadurch nicht beeinträchtigt. Jedoch wird die Mächtigkeit der verwendeten semantischen Technologien aufgrund dieses technischen Problems nicht vollständig ausgeschöpft. Nichtsdestotrotz verdeckt das Prädikat `accessGranted` die interne Struktur des Modells bei der Formulierung von Anfragen und ist somit bezüglich der Bedienbarkeit üblichen datenbankbasierten Anwendungen, die stets die internen Datenbankschemata zur Beschreibung von Anfragen kennen müssen, weit überlegen.

7.3 Performanz und Skalierbarkeit

Das wohl entscheidenste Kriterium für die praktische Verwendbarkeit der entwickelten Zugriffskontrollkomponente ist die Performanz des Autorisierungsdienstes. Die beim automatischen Schließen verwendeten Deduktionsalgorithmen sind sehr komplex und deren Umsetzung nicht trivial, was sich in Performanz und Skalierbarkeit bemerkbar macht. In Abhängigkeit der Anzahl der in einem semantischen Modell enthaltenen Instanzen steigt die Laufzeit und der Speicherbedarf des Reasonings stark an.

Beim Zugriffskontrollsystem beeinflusst die Laufzeit des automatischen Schließens die Antwortzeit einer Zugriffsanfrage direkt. Das dem System zugrunde liegende Zugriffskontrollmodell ist auf Ontologieebene betrachtet nicht sehr komplex, da die Anzahl der Konzepte und Relationen beschränkt ist. Zudem werden keinerlei für die Berechnung des automatischen Schließens aufwändige und „teure“ Elemente wie beispielsweise Disjunktionen verwendet. Zur Auflösung von Konflikten und der schrittweisen Ableitung des Prädikats `accessGranted` werden jedoch eine Vielzahl von Regeln eingesetzt. Einige der Regeln sind aufgrund der Vielzahl von Konjunktionen im Rumpf sehr komplex und deren Auswertung entsprechend „teuer“. Die Vielzahl und Komplexität der Regeln erhöht die Anforderungen an das Reasoning.

Die Bearbeitungszeit von Anfragen ist direkt mit der Anzahl der auszuwertenden Regeln korreliert. Tritt während der Beantwortung einer Anfrage ein Konflikt auf, der durch zusätzliche Regeln aufgelöst werden muss, erfordert die Bearbeitung aufgrund der Auswertung zusätzlicher Regeln mehr Zeit. Die Durchführung von Leistungstests auf der beschriebenen Unixerweiterung zeigt, das Zugriffsanfragen in Modellen mit mehreren hundert Instanzen deutlich unter einer Sekunde beantwortet werden. Bei mehreren tausend Instanzen hindoch steigt die Beantwortungszeit merklich und beträgt bei etwa 15.000 Konzeptinstanzen in etwa zwei Sekunden. Nichtsdestotrotz stimmen die gemessenen Werte hoffnungsvoll für die praktische Verwendbarkeit des semantischen Zugriffskontrollmodells. Die Messungen wurden auf einem Pentium M Prozessor mit l,8 GHz und 512 MB Arbeitspeicher ausgeführt.

Neuere Versionen der als Reasoner genutzten KAON2 Infrastruktur brachten jedoch erhebliche Fortschritte in der Laufzeit und im Speicherbedarf mit sich. Benötigte die Beantwortung einfacher Zugriffsanfragen bis vor wenigen Wochen noch bis zu drei Sekunden, ist die gleiche Anfrage mit der aktuellen KAON2 Version in deutlich unter einer Sekunde ausgeführt. Nach Auskunft von Boris Motik, dem Entwickler von KAON2, sollen weitere Optimierungen möglich sein.

Der überwiegende Teil der Regeln dient der Konfliktauflösung bei sich widersprechenden Berechtigungen. Wie in Abschnitt 4.3.2 beschrieben, unterscheidet man positive, negative und gemischte Zugriffskontrollsysteme. Nur bei gemischten Systemen kann es zu Konflikten kommen. Um eine Optimierung der Laufzeit zu erreichen, wurden die für die Konfliktauflösung verwendeten Regeln in eine separate Ontologie ausgelagert. Wird in der Setup API ein gemischtes Zugriffskontrollsystem initialisiert, werden diese Konfliktauflösungsregeln durch Import dieser Ontologie dem Reasoning zugänglich gemacht. Diese Ausgliederung von Konfliktauflösungsregeln führt zu einer spürbaren Beschleunigung der Bearbeitungszeit von Anfragen.

Die Skalierbarkeit der verwendeten semantischen Technologien wird in den nächsten Jahren vermutlich eine der großen Herausforderung für die Umsetzung der Visionen des Semantic Webs sein. Wie bereits erwähnt, steigt die Beantwortungszeit des automatischen

Schließens in Abhängigkeit der Instanzen einer Wissensbasis. Es ist daher davon auszugehen, dass ab einer kritischen Menge von Instanzen die Beantwortung von Zugriffsanfragen zu viel Zeit und Speicherbedarf in Anspruch nehmen wird und die Skalierbarkeit somit nicht gegeben ist.

Zusammenfassend kann gesagt werden, dass für die Umsetzung von Zugriffskontrollsystemen kleiner bis mittlerer Anwendungen die Verwendung des in dieser Arbeit vorgestellten semantischen Ansatzes durchaus denkbar ist und aufgrund der individuellen Erweiterbarkeit und einfachen Verwendbarkeit viele Vorteile bietet. Für große Unternehmen, die Berechtigungen vieler tausend Benutzer und Objekte verwalten, ist der praktische Einsatz der beschriebenen Lösung aufgrund der langen Antwortzeiten nur bedingt möglich. Durch zusätzliche Optimierungen, wie z.B. das Caching von Zugriffsanfragen, kann jedoch auch für große Rechtemanagementsysteme die praktische Verwendbarkeit gewährleistet werden.

Kapitel 8

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde in den Forschungsbereich der semantischen Regeln eingeführt, mit speziellem Augenmerk auf den Einsatz von Ontologien und Regeln. Eine Einführung in die in der Literatur und Praxis bekannten Zugriffskontrollsysteme gaben einen Einblick über die zu modellierende Domäne. Gemeinsamkeiten und Unterschiede der unterschiedlichen Zugriffskontrollmodelle wurden extrahiert, um ein allgemeines Zugriffskontrollsystem beschreiben zu können.

Das Ziel der Arbeit war die Erstellung einer Zugriffskontrollkomponente, die organisationsspezifische Sicherheitsstrategien an zentraler Stelle über Anwendungs- und Systemgrenzen hinaus beschreibt. In der Konzeption der zu erstellenden Komponente wurden Schnittstellen beschrieben, über die der Zugriff externer Anwendungen erfolgt. Diese Schnittstellen ermöglichen den Zugriff auf den eigentlichen Kern des Zugriffskontrollsystems – das semantische Zugriffskontrollmodell.

Dieses Modell wurde durch Ontologien und Regeln realisiert. Die Ontologien beschreiben die zentralen Elemente eines Zugriffskontrollsystems und deren Eigenschaften. Zur besseren Verständlichkeit und intuitiven Benutzbarkeit wurden diese Ontologien schichtenweise organisiert. Ausgehend von einer sehr abstrakten Beschreibung wurden die Elemente und Beziehungen zwischen den Elementen verfeinert, um Eigenheiten spezifischer Zugriffskontrollsysteme beschreiben zu können. Regeln dienen der Interpretation der durch Ontologien beschriebenen Daten und erlauben die Beantwortung von Zugriffsanfragen. Die Zielsetzung war, Zugriffsanfragen ohne jeglichen Programmcode, nur durch die Ausführung entsprechender Regeln, beantworten zu können. Hierzu waren die erweiterten Möglichkeiten des automatischen Schließens in KAON2 eine notwendige Voraussetzung. Vor allem der Möglichkeit der nichtmonotonen Negation und dem Einsatz von Nicht-OWL-Prädikaten kam eine besondere Bedeutung zu.

Durch Umsetzung eines komplexen Regelwerks wurde das Ziel der codefreien Auswertung von Zugriffsanfragen erreicht. Die Regeln beschreiben hierbei Weitergaben auf Elementgruppen oder Elementhierarchien durch unterschiedliche Strategien. Konflikte, die sich aufgrund widersprechender Berechtigungen bei der Beantwortung einer Anfrage ergeben, werden durch Regeln aufgelöst. Liegen für eine Anfrage keine Informationen in der Wissensbasis vor, werden anhand von Strategien entsprechende Antworten abgeleitet. Anhand dieser Beispiele wird erkennbar, welchen zentralen Stellenwert Regeln in dieser Arbeit einnehmen. Ausgehend von den in Ontologien beschriebenen Instanzen dienen alle Regeln dazu, das zentrale Nicht-OWL-Prädikats namens `accessGranted` abzuleiten. Durch Auswer-

tung dieses Prädikats werden Zugriffsanfragen beantwortet. Die interne Struktur des Modells ist für den Benutzer des Systems hinter diesem Prädikat verdeckt. Diese Zentralisierung der Anfrage in einem Prädikat bringt trotz einer einfachen Bedienbarkeit des Systems eine große Mächtigkeit bei Zugriffsanfragen, die jedoch aufgrund von Problemen der Nichtstratifizierbarkeit des zugehörigen logischen Programms derzeit nicht vollständig ausgeschöpft werden können. Die Ursache der Nichtstratifizierbarkeit liegt hierbei nicht am Zugriffskontrollmodell, sondern in einer KAON2 internen Umsetzung des disjunktiven Datalog Programms in sogenannte *Magic Sets*, begründet.

Die entwickelte Software Infrastruktur vereinfacht den Zugriff auf das semantische Modell durch die Umsetzung wohldefinierter Schnittstellen und bildet gemeinsam mit dem semantischen Modell die eigentliche Zugriffskontrollkomponente. Zugriffsanfragen werden durch einfache Methoden der Query API beantwortet die durch Reasoning das Prädikat `accessGranted` ableiten. Ebenfalls wurden Schnittstellen beschrieben, die die Administration des Modells durch Verwaltung der Ontologieinformationen vereinfachen.

Zur Evaluation des entwickelten Systems wurde eine Erweiterung des Zugriffskontrollmodells zur Umsetzung einer Unix Zugriffskontrolle realisiert. Diese Umsetzung zeigte, dass mit dem verfolgten Ansatz viele der ursprünglichen Ziele, wie die Erweiterbarkeit des Modells zur Beschreibung eines organisationsspezifischen Rechtemanagements und eine einfache, mächtige Beantwortung von Zugriffsanfragen ohne Programmcode erreicht wurden. Gesammelte Erfahrungen während der Umsetzung des Systems zeigten, dass das entwickelte Zugriffskontrollsystem in der Lage ist kleine und mittlere Rechtemanagementsysteme zu realisieren. Trotz der Komplexität des zugrunde liegenden Modells erfolgt die Beantwortung von Zugriffsanfragen sehr schnell. Mit zunehmender Anzahl von Instanzen steigt die Berechnungszeit und auch der Speicherbedarf stark an, wodurch eine Skalierbarkeit der verwendeten Technologien nicht gegeben zu sein scheint. Die Realisierung großer Rechtemanagementsysteme erfordert somit zusätzlichen Optimierungsaufwand, z.B. durch Umsetzung von Cachingverfahren.

Im Allgemeinen haben sich Ontologien und Regeln im Verlauf dieser Arbeit als gut geeignet zur Beschreibung eines Zugriffskontrollmodells erwiesen. Durch die intensive Verwendung von Regeln ist das Modell in der Lage anhand der vorliegenden Daten durch Auswertung der Regeln Zugriffsanfragen zu beantworten. Somit entfällt die Notwendigkeit von Programmcode zur Beantwortung von Anfragen und es handelt sich im weitesten Sinne um ein *maschinenverständliches Modell*. Diese Arbeit kann somit den Bemühungen der Umsetzung der Vision des Semantic Web zugeordnet werden.

Weitere Verbesserungen bzw. Veränderungen sind in den folgenden Punkten denkbar:

- Eine besondere Herausforderung für die praktische Verwendbarkeit dieser Komponente stellt die einfache Administration des Modells dar. Das Modell zeichnet sich durch eine Vielzahl von Elementen und Relationen aus. Alle diese Informationen sollten den externen Anwendungen durch die Schnittstellen der Komponente zur Verfügung gestellt werden. In dieser Arbeit wurde dies für die zentralen Elemente des Zugriffskontrollmodells prototypisch durchgeführt. Für alle weiteren Elemente und Relationen werden generische Methoden zur Verfügung gestellt, für deren Verwendung der Anwender die URIs der entsprechenden Ontologieelemente kennen muss. Diese generischen Methoden erlauben direkte Ontologiezugriffe über die Schnittstelle der Komponente hinweg. Da der Anwender sich jedoch üblicherweise nicht mit den Details

der modellinternen semantischen Technologien beschäftigen möchte, sollte diese Umsetzung transparent erfolgen. Eine komplette Beschreibung der Ontologieelemente in Form eines ausführbaren Objektmodells ist aufgrund der Komplexität des Zugriffskontrollmodells nicht möglich. Aktuelle Forschungsprojekte, wie z.B. RDFReactor[1], beschäftigen sich mit der automatischen Abbildung von Ontologien auf ausführbare Objektmodelle. Die Integration eines solchen Ansatzes könnte die Umsetzung der Admin API wesentlich erleichtern. Derzeit erfordern Erweiterungen oder Anpassungen des Modells stets eine Anpassung der Admin API. Die automatische Transformation der Ontologien in Programmcode könnte die Flexibilität des Modells auf die Softwareinfrastruktur übertragen und somit den Aufwand für Modellanpassungen weiter reduzieren.

- Das entworfene Modell zeichnet sich durch eine große Ausdrucksmächtigkeit aus und ist in der Lage verschiedenartige Zugriffskontrollsysteme zu beschreiben. Diese Mächtigkeit des Modells führt jedoch zu einer großen Komplexität, die längere Laufzeiten der Beantwortung von Zugriffsanfragen zur Folge hat. Zusätzliche Optimierungen der Laufzeit lassen sich eventuell durch weitere Modularisierungen, wie dies am Beispiel der Auslagerung von Konfliktauflösungsregeln in eine eigene Ontologie vorgestellt wurde, durchführen. Diese Modularisierungen bewirken eine genauere Anpassung des Modells an konkrete Anwendungsszenarien. Die erreichbaren Laufzeitvorteile werden jedoch aller Voraussicht nach sehr gering ausfallen.

- Die in der Arbeit modellierte Systemschicht zur Umsetzung spezifischer Eigenheiten gängiger Zugriffskontrollsysteme konnte aus zeitlichen Gründen nicht umgesetzt werden. Diese Erweiterungen erlauben eine einfachere Beschreibung konkreter Zugriffskontrollsysteme. Die Umsetzung dieser Erweiterungen erfolgt durch das Hinzufügen der beschriebenen Regeln in das Modell und der Erweiterung der Komponentenschnittstellen zur Administration systemspezifischer Eigenschaften.

- Die praktische Verwendbarkeit der entwickelten Komponente wurde bisher ausschließlich durch Testfälle untersucht. Eine Integration in real existierende Anwendungen wäre daher wünschenswert. Aus derzeitiger Sicht ist die Verwendung der Zugriffskontrollkomponente im Projekt Denkwerkzeug[2] geplant. Hierbei handelt es sich um die Realisierung eines Wiki unter Verwendung semantischer Technologien. Dieser Einsatz der Zugriffskontrollkomponente erfordert Anpassungen des Zugriffskontrollmodells an die Wiki-spezifischen Anforderungen des Rechtemanagements.

[1] http://rdfreactor.ontoware.org
[2] http://www.denkwerkzeug.org

Anhang A

Erweiterung des Modells am Beispiel der Unix Zugriffskontrolle

Durch die Möglichkeit individueller Anpassungen des Zugriffskontrollmodells, ist das in dieser Arbeit entwickelte Zugriffskontrollsystem in der Lage organisationsspezifische Sicherheitsstrategien zu beschreiben. Im Zuge der Evaluation der Flexibilität des Modells wird die Unix Zugriffskontrolle umgesetzt. In einem Unixsystem besitzt jedes Objekt einen eindeutigen Besitzer und ist einer Benutzergruppe zugewiesen. Die Vergabe von Berechtigungen erfolgt für jedes Objekt basierend auf dem Besitzer des Objekts, der Benutzergruppe des Objekts und allen anderen Benutzern des Systems. Abbildung 2.7 auf Seite 18 stellt die in Unix verwendeten Zugriffskontrolllisten grafisch dar.

Zur Beschreibung dieser zusätzlichen Eigenschaften sind Anpassungen der auf der Ergänzungsebene definierten Objektkonzepte notwendig. Diese werden in Abschnitt A.1 erläutert. Zudem werden in Abschnitt A.2 Instanzen eingeführt, die Bestandteil der Unix Zugriffskontrolle sind. Ausgehend von diesen Instanzen wird erläutert, wie die in Unix verwendeten Zugriffskontrolllisten auf die entsprechenden Konzepte des Zugriffskontrollmodells abgebildet werden. Zur Beantwortung von Zugriffsanfragen sind zusätzliche Regeln notwendig, die in Abschnitt A.3 vorgestellt werden. Unixtypische Administrationsaufgaben werden durch die in Abschnitt A.4 beschriebenen Erweiterungen der Query und Admin API ermöglicht.

A.1 Konzeptionelle Erweiterungen

Zusätzliche Eigenschaften von Unixobjekten erfordern die konzeptionelle Erweiterung der Klasse `ACObject`. Abbildung A.1 stellt diese Erweiterungen grafisch dar. Zwei zusätzliche Relationen zum Besitzer eines Objekts und der Benutzergruppe, der ein Objekt zugeordnet ist, beschreiben die notwendigen Anpassungen. Zur Unterscheidung von zusammengesetzten und atomaren Unixobjekten werden die Konzepte `CompositeUnixObject` und `AtomicUnixObject` eingeführt. Im Gegensatz zu den in Abschnitt 4.3.1 vorgestellten atomaren und zusammengesetzten Elementen ist bei Unixsystemen die Weitergabe von Berechtigungen von Gruppen an deren Mitglieder nicht üblich. Berechtigungen sind somit ausschließlich auf den direkt referenzierten Objekten gültig. Um die Weitergabe von Berechtigungen von zusammengesetzten auf atomare Objekte zu unterbinden, dürfen die Konzepte `AtomicUnixObject` bzw. `CompositeUnixObject` nicht als Subkonzepte von `Atomic-`

`Object` bzw. `CompositeObject` modelliert werden. Um die Mitgliedschaft von Unixobjekten in Objektgruppen beschreiben zu können, werden jedoch die Relationen `isMemberOfCompositeEntity` und `consistsOfEntity` der Ergänzungsschicht, wie in Abbildung A.1 dargestellt, wiederverwendet. Da in Unixsystemen keine Hierarchien von Objekten vorgesehen sind, können diese Relationen direkt zwischen den Konzepten `UnixACObject` und `CompositeUnixObject` beschrieben werden.

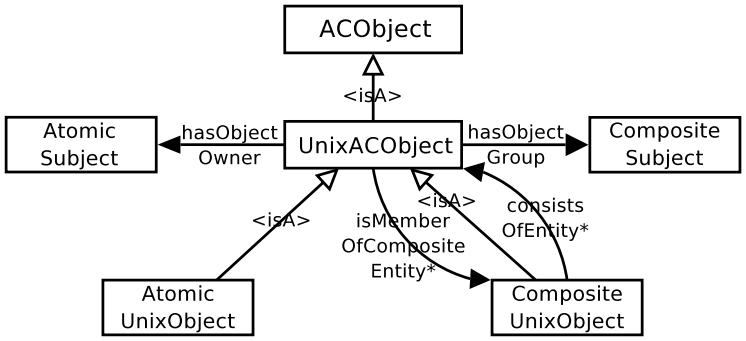

Abbildung A.1: Unix Erweiterung des allgemeinen Objektkonzepts.

A.2 Instantiierung typischer Zugriffskontrollelemente

Dieser Abschnitt beschreibt Instanzen von Zugriffskontrollelementen, die zur Beschreibung der Unix Zugriffskontrolle notwendig sind. Neben der Beschreibung des Systems durch Instantiierung eines geeigneten Zugriffskontrollmodells werden eine Reihe von unixtypischen Operationsinstanzen eingeführt.

Ein Unixsystem erlaubt ausschließlich die Vergabe positiver Berechtigungen. Das zugrunde liegende Zugriffskontrollmodell ist somit eine Instanz des Konzepts `PositiveRightsACModel`. Zudem gilt, dass alles was nicht explizit erlaubt ist, verboten wird. Das Zugriffskontrollmodell referenziert somit die Standardstrategie `ClosedWorldAssumption`. Konflikte zwischen positiven und negativen Berechtigungen können in einem positiven Zugriffskontrollmodell nicht auftreten, so dass keine Konfliktauflösungsstrategie notwendig ist.

In einer Unix Zugriffskontrollliste, die beispielhaft in Abbildung 2.7 auf Seite 18 vorgestellt wird, werden die Rechte zur Ausführung der Operationen `read`, `write` und `execute` für die verschiedenen Subjekte eines Objektes vergeben. Diese Operationen sind Platzhalter für reale Operationen. So entspricht der konkrete Unix Befehl `ls`, der alle Dateien und Unterverzeichnisse eines Verzeichnisses auflistet, einer Ausprägung des `read` Befehls. Hat ein Benutzer das Recht die Operation `read` auf einem Verzeichnis auszuführen, so darf er auch `ls` auf diesem aufrufen. Zur Modellierung dieses Sachverhalts sind Operationsgruppen, beschrieben durch Instantiierung des Konzepts `CompositeOperation`, gut geeignet. Durch die in Kapitel 5.2.1 vorgestellten Regeln werden Berechtigungen, die auf einer Operationsgruppe definiert sind, auf alle Operationen der Gruppe übertragen. Somit werden die Operationen `read`, `write` und `execute` als Operationsgruppen modelliert. Mitglieder der Gruppe `read` sind beispielsweise die Operationen `ls` und `print`.

Aufgrund der zusätzlichen Eigenschaften der Unixobjekte werden zusätzliche Operationen zur Administration dieser Eigenschaften benötigt. Unix sieht unter anderem die Operationen `chmod` zur Veränderung der Rechte auf einem Objekt und `chown` zur Modifikation des Eigentümers eines Objekts vor. `Chmod` und `chown` werden als Instanzen atomarer Operationen beschrieben. Nur der Eigentümer eines Objekts darf diese Operationen ausführen.

Wie bereits erwähnt, werden bei der Vergabe von Berechtigungen auf einem Objekt die Rechte des Eigentümers, der Mitglieder der dem Objekt zugewiesenen Benutzergruppe und aller anderen Subjekte des Systems unterschieden. Um „*alle anderer Subjekte des Systems*“ benennen zu können, wird ein Platzhalter in Form einer Subjektgruppe eingeführt. Diese Subjektgruppe hat den Namen `others`. Die Mitglieder dieser Gruppe werden zum Zeitpunkt einer Anfrage in Abhängigkeit einer konkreten Objektinstanz durch die in Abschnitt A.3 beschriebenen Regeln berechnet.

Die eigentliche Herausforderung liegt jedoch in der Abbildung der in Unix verwendeten Zugriffskontrolllisten auf die im Modell beschriebenen Berechtigungen. Durch die Zugriffskontrolllisten werden jedem Objekt drei Rechte für den Eigentümer, die Gruppe und alle anderen Subjekte vergeben. Jedes dieser drei Rechte wird in eine eigene Berechtigung überführt. Im Folgenden wird dies anhand eines Beispiels erläutert.

Beschrieben werden sollen die Berechtigungen des Objekts `file.txt`. Alice als Eigentümer darf alle Operationen ausführen. Dem Objekt wurde die Benutzergruppe `guests` zugewiesen. Jedes Mitglied dieser Gruppe darf lesend und schreiben auf das Objekt zugreifen. „*Alle anderen Subjekte des Systems*“ dürfen `file.txt` nur lesen. Ausgehend von dieser Beschreibung werden drei Berechtigungsinstanzen erstellt, die in Abbildung A.2 grafisch veranschaulicht werden.

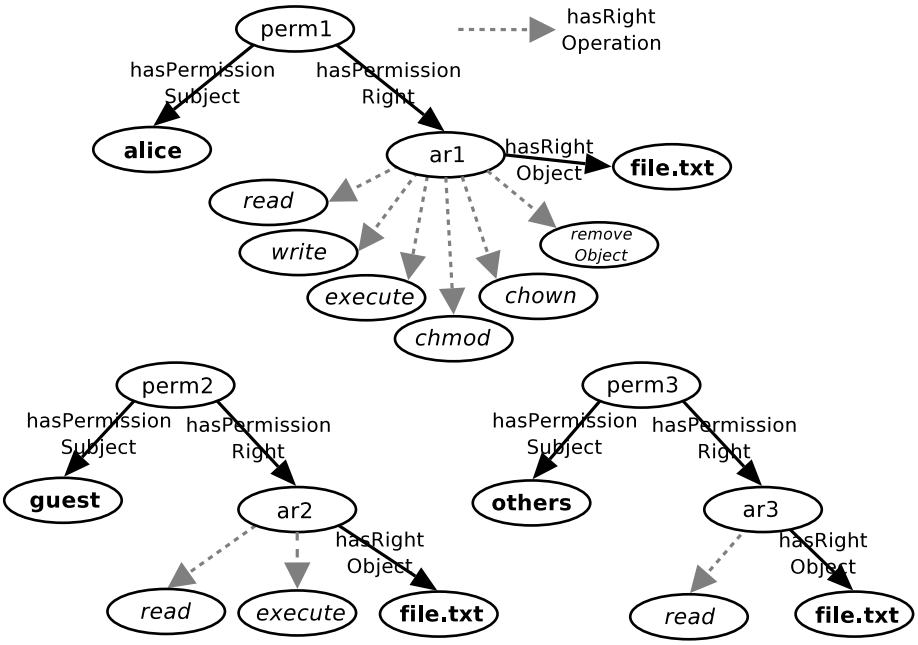

Abbildung A.2: Beispiel der Umsetzung einer Unix Zugriffskontrollliste.

Der Eigentümer darf alle Operationen auf dem Objekt ausführen. Alice darf somit lesend, schreibend und ausführend auf das Objekt zugreifen. Darüber hinaus besitzt sie die

Berechtigung das Objekt zu löschen und den Eigentümer bzw. die Objektrechte zu ändern. Zur Beschreibung dieses Sachverhalts wird eine Zugriffsrechtsinstanz `ar1` erzeugt, die jeder der zuvor genannten Operationen und das entsprechende Objekt referenziert. Dieses Zugriffsrecht wird seinerseits in der Berechtigung `perm1` mit dem Eigentümer des Objekts in Beziehung gesetzt. Die dem Objekt zugewiesene Gruppe darf hingegen nur Lese- und Ausführoperationen auf dem Objekt aufrufen. Alle anderen Subjekte dürfen das Objekt nur lesen. Dies wird in den Berechtigungsinstanzen `perm2` und `perm3` umgesetzt. Aufgrund dieser Berechtigungen können alle Zugriffsanfragen für den Besitzer und für Mitglieder der Gruppe des Objekts ohne Anpassungen der im Kapitel 5.3 beschriebenen Regeln beantwortet werden. Für die Berechtigungen „*aller anderen Subjekte des Systems*“ werden zusätzliche Regeln benötigt. Diese werden im nächsten Abschnitt vorgestellt.

A.3 Zusätzliche Regeln

Zusätzlich zu den in Kapitel 5 definierten Regeln sind für die Interpretation der Unix Zugriffskontrolle weitere Regeln notwendig.

Im Gegensatz zu anderen Systemen hat jeder Benutzer das Recht Objekte anzulegen. Dies lässt sich durch Instantiierung von Berechtigungen nicht ohne weiteres formulieren, da die ID des zu erstellenden Objekts unbekannt ist. Mittels Regeln ist dies jedoch sehr einfach. Regel A.1 stellt die Vorgehensweise dar. Wird eine Anfrage von einem beliebigen Subjekt zur Ausführung der administrativen Operation `addObject` auf einer beliebigen Objektinstanz gestellt, so vergibt Regel A.1 ein positives Recht. Ausgehend von diesem positiven Recht wird daraufhin durch die in Abschnitt 5.3 beschriebenen Regeln das Prädikat `accessGranted` abgeleitet und der Zugriff gestattet. Somit ist gewährleistet, dass jedes Subjekt beliebige Objekte zum System hinzufügen kann. Bei dem in dieser Regel verwendete Prädikat `equal`, handelt es sich wiederum um das KAON2 interne Prädikat zur Überprüfung der Gleichheit von Instanzen.

```
∀ s, ob, op: hasPositiveRight(?s,?ob,?op) ←
AtomicSubject(?s) ∧  Object(?ob) ∧ Operation(?op) ∧
equal(?op,"#addObject")
```

Regel A.1: Positives Recht zur Erzeugung von Unix Objekten.

Darüber hinaus werden Regeln zur Interpretation der zuvor beschriebenen Subjektgruppe `others` benötigt. Diese Subjektgruppe bezeichnet die Menge aller Subjekte des Systems, die nicht Eigentümer und nicht Mitglied der Benutzergruppe eines Objekts sind und ist somit für jedes Unixobjekt anders zusammengesetzt. Die folgenden Regeln berechnen die Zuordnung von Subjekten zu dieser Gruppe in Abhängigkeit des in der Anfrage referenzierten Objekts.

Wird eine Zugriffsanfrage an das System gestellt, muss in einem ersten Schritt festgestellt werden, wie die Beziehungen zwischen angefragtem Objekt und Subjekt sind. Ist das Subjekt Eigentümer oder Mitglied der Benutzergruppe, so ist diese Relation explizit in Berechtigungsinstanzen formuliert und es werden keine zusätzlichen Regeln zur Ableitung

einer Antwort benötigt. Ist ein Subjekt nicht Eigentümer oder Mitglied der Benutzergruppe, so wird es durch Regel A.2 der Subjektgruppe `others` zugeordnet. Hierzu wird das Nicht-OWL-Prädikat `memberOfOthers(?s,?ob)` eingeführt, dass alle Subjekte `s` auflistet, die der Gruppe „*aller anderen Subjekte des Systems*“ der Objektinstanz `ob` angehören.

```
∀ s, g, ob: memberOfOthers(?s,?ob) ←
Subject(?s) ∧ CompositeSubject(?g) ∧ UnixObject(?ob) ∧
¬ hasObjectGroup(?ob,?s) ∧ ¬ hasObjectOwner(?ob,?s)∧
hasObjectGroup(?ob,?g) ∧ ¬ isMemberOfCompositeEntity(?s,?g)
```

Regel A.2: Zuordnung von Subjekten der Anfrage zur Gruppe `others`.

Um eine Anfrage beantworten zu können, muss die für die Subjektgruppe `others` und das Objekt der Anfrage definierte Berechtigungsinstanz ausgewertet werden. In dem in Abbildung A.2 dargestellten Beispiel wäre dies Instanz `perm3`. Regel A.3 überprüft in einem ersten Schritt, ob für `others`, das Objekt und die Operation der Anfrage eine positive Berechtigung existiert. Ist dies der Fall, wird für jedes durch Regel A.2 berechnete Mitglied der Gruppe `others` dieses positive Recht übertragen. Somit ist der Zugriff erlaubt, wenn für die Gruppe `others` eine Berechtigung existiert und das Subjekt nicht der Eigentümer und nicht Mitglied der Gruppe des Objekts ist.

```
∀ s, ob, op: hasPositiveRight(?s,?ob,?op) ←
hasPositiveRight("#others",?ob,?op) ∧ memberOfOthers(?s,?ob)
```

Regel A.3: Positives Recht zur Erzeugung von Unix Objekten.

A.4 Erweiterung der Query API

Um die spezifischen Eigenschaften der Unix Zugriffskontrolle über die Komponentenschnittstellen für externe Anwendungen verwendbar zu machen, müssen Anpassungen der Query und Admin API durchgeführt werden.

Die zusätzlichen Eigenschaften des Unixobjekts werden dabei durch Erweiterung der allgemeinen Objektklasse der Admin API beschrieben. Diese Erweiterungen umfassen Referenzen auf den Eigentümer und die Benutzergruppe des Objekts, sowie die auf dem Objekt vergebenen Rechte. Diese Rechte werden den Komponentenschnittstellen auf unixtypische Art und Weise in Form von Zahlenwerten übergeben. `751` würde beispielsweise dem Besitzer des Objektes Lese-, Schreib- und Ausführrechte geben, der Gruppe des Objekts Lese- und Ausführrechte und allen anderen Benutzern ausschließlich Ausführrechte.

Jeder Benutzer des Systems darf neue Unixobjekte anlegen. Dies erfolgt durch Aufruf einer administrativen Anfrage, die in der Klasse `UnixObjectRequest` umgesetzt ist. Diese erbt von der allgemeinen administrativen Anfrage zum Hinzufügen von Objekten (`ObjectRequest`). Anhand der Informationen des Unixobjektes werden neben der eigentlichen Objektinstanz auch gleich die Referenzen zum Besitzer und der jeweiligen Benutzergruppe des Objekts in die Ontologie eingebracht. Die als Zahlenwert codierten Rechte werden bei der Ausführung der Anfrage, wie in Abbildung A.2 dargestellt, in drei Berechtigungsinstanzen übersetzt.

Unixsysteme stellen eine Reihe von Befehlen zur Verfügung deren Aufruf die Berechtigungsinstanzen eines Objekts ändern. Die bekanntesten dieser Befehle sind `chmod` und `chown`. Die Umsetzung dieser Befehle erfolgt wiederum durch administrative Anfragen, die in den Klassen `ChangeObjectRightRequest` bzw. `ChangeObjectOwnerRequest` implementiert sind. Die Ausführung dieser Anfragen ist nur dem Besitzer des jeweiligen Objektes erlaubt. Nach erfolgreicher Autorisierung greifen die Anfragen auf die Berechtigungsinstanzen des Objekts zu und ändern den Besitzer eines Objekts bzw. die vergebenen Rechte.

Anhang B

Stratifikation bei nichtmonotonem Schließen

Nichtmonotones Schließen (engl. Nonmonotonic Reasoning) unter Verwendung stratifizierter Negation stellt zusätzliche Anforderungen an die Auswertung logischer Programme. Durch die Möglichkeit der Verwendung nichtmonotoner, stratifizierter Negation ist eine Festlegung der Auswertungsreihenfolge notwendig. Die nichtmonotone Negation wird durch Ausführung des Umkehrschlusses berechnet. Dies bedeutet, dass man um eine Negation `not(p)` zu berechnen, versucht zu zeigen, dass `p` erfüllt ist. Gilt `p` ist `not(p)` falsch und umgekehrt. Logische Programme weisen jedoch häufig Rekursionen auf, d.h. durch Verwendung eines Prädikats in Rumpf und Kopf unterschiedlicher Regeln kommt es zu Zyklen zwischen Regeln. Die folgenden Regeln stellen einen Zyklus beispielhaft dar:

$$\texttt{A(x)} \leftarrow \texttt{B(x)}$$
$$\texttt{B(x)} \leftarrow \neg\ \texttt{A(x)}$$

Um sicherzustellen, dass eine nichtmonotone Negation `not(p)` ausgewertet werden kann, muss `p` vollständig berechenbar sein. Um dies zu gewährleisten, legt die Stratifikation (engl. stratification) die Auswertungsreihenfolge fest. Die Grundidee dabei ist, die Prädikat logischer Programme in Partitionen derart einzuteilen, dass keine zwei Prädikate einer Partition negativ voneinander abhängen.

Die folgende Definition aus [Tim03] gibt Anweisungen zur Zerlegung eines Programms in Partitionen:

Ein Programm P ist genau dann stratifizierbar, wenn es in Partitionen $P = P_1 \cup ... \cup P_n$ zerlegbar ist, so dass für i=1,...,n das Folgende gilt:

1. *Jede Regel in P ist in genau einem P_i enthalten;*
2. *Alle definierenden Regeln eines Prädikatsymbols R sind im selben P_i enthalten. Definierende Regeln enthalten das Prädikatsymbol R im Regelkopf.*
3. *Gibt es ein positives Prädikatsymbol R in einer in P_i enthaltenen Regeln, werden alle Regeln die R im Regelkopf enthalten $\bigcup_{j \leq i} P_j$ zugeordnet;*
4. *Gibt es ein negatives Prädikatsymbol R in einer in P_i enthaltenen Regel, werden alle Regeln die R im Regelkopf enthalten $\bigcup_{j < i} P_j$ zugeordnet.*

Anschaulich bedeutet dies, dass ein Programm stratifizierbar ist, gdw. der zugehörige Abhängigkeitsgraph keinen Zyklus enthält in dem eine negativ markierte Kante vorkommt.

Ein Beispiel für ein nichtstratifizierbares Programm wäre:

```
male(X) ← person(X), ¬female(X).
female(X) ← person(X), ¬male(X).
```

Der zugehörige Abhängigkeitsgraph ist in Abbildung B.1 dargestellt. Da dieser Graph einen Zyklus mit negativ markierter Kante aufweist, ist das zugehörige Programm nicht stratifizierbar. Bei einer Auswertung des Geschlechts einer Person kann aufgrund der fehlenden Stratifizierbarkeit die Korrektheit des Ergebnisses nicht garantiert werden.

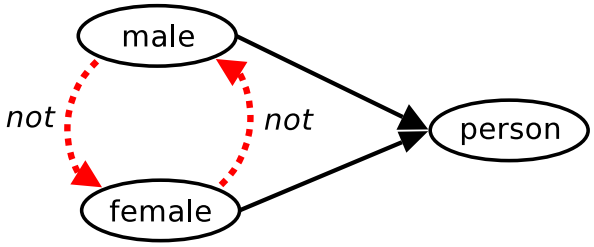

Abbildung B.1: Abhängigkeitsgraph zur Untersuchung der Stratifikation eines Programms.

Aufgrund der Stratifikation muss bei der Erstellung eines nichtmonotonen Programms die Verwendung der Negation sehr bedacht erfolgen. Ansonsten riskiert man Ableitungszyklen, die Schlussfolgerungen auf dem logischen Modell verhindern. Im Verlauf dieser Arbeit mussten an mehreren Stellen Modellanpassungen vorgenommen werden, um die Stratifizierbarkeit des Zugriffskontrollmodells zu gewährleisten.

Anhang C

Software

Die im Verlauf dieser Arbeit erstellte Software wurde in der Programmiersprache Java entwickelt. Zum Einsatz kam die neuste Version Java 1.5. Voraussetzung zur Verwendung der erstellten Zugriffskontrollkomponente ist somit das JDK 1.5[1] von Sun.

Die auf der nächsten Seite beigefügte CD enthält die wichtigsten Bestandteile der im Verlauf dieser Arbeit umgesetzten Lösungen. Die realisierte Zugriffskontrollkomponente liegt sowohl in Form einer binären jar Datei zur einfachen Verwendung durch externe Komponenten, als auch in Form von Quellcode vor. Anhand von erklärenden Beispielen und der Javadoc Dokumentation lässt sich die Verwendung der Komponente und die interne Realisierung des Zugriffskontrollsystems verstehen. Die zur Korrektheit der Regeln verwendeten JUnit Tests liegen ebenfalls im Quellcode vor und können zur Überprüfung der Korrektheit von individuellen Erweiterungen herangezogen werden.

Zudem ist auf der CD die Umsetzung der in Abschnitt A vorgestellten Unix Zugriffskontrolle enthalten. Anhand der Beschreibung und Implementierung der Unix Zugriffskontrolle wird die allgemeine Vorgehensweise der individuellen Anpassung des Zugriffskontrollsystems beispielhaft erklärt. Für die Durchführung von Anpassungen auf Ontologien wird die KAON2[2] Infrastruktur vorausgesetzt.

Die in Abschnitt 6.1 erstellten Werkzeuge in Form des Regelparser zur einfachen Eingabe von KAON2 Regeln in Form von Zeichenketten und des OWL Constraints Konverters zur Übersetzung von speziellen OWL Elementen in Regeln sind ebenfalls im Quellcode beigefügt und werden durch Beispielprogramme erläutert.

Neben dem eigentlichen Programmcode sind auch die Ontologien zur Beschreibung des Zugriffskontrollmodells enthalten. Jede Ontologie liegt hierbei in zwei Versionen vor. Die mit Protégé erstellten Ontologien enthalten Konzepte, Relationen und Instanzen und können zur grafischen Betrachtung eingesetzt werden. Die KAON2 Versionen sind hingegen um Regeln ergänzt und enthalten somit Wissen zur Interpretation des Modells. Aufgrund KAON2 interner Konstrukte, wie z.B. Nicht-OWL-Prädikaten, können diese Ontologien nicht mit Protégé betrachtet werden.

[1] http://java.sun.com/j2se/1.5.0/download.jsp
[2] http://kaon2.semanticweb.org/#download

C.1 Quellcode (CD-ROM)

Abbildungsverzeichnis

Regelverzeichnis

Literaturverzeichnis

[Anu03] CHUTIPORN ANUTARIYA ET AL.: *A Rule-Based XML Access Control Model*, Lecture Notes in Computer Science, Volume 2876, 2003.

[Bor97] WILLEM NICO BORST: *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*, http://doc.utwente.nl/fid/1392, S. 12, 1997.

[Chen97] Y. CHEN: *Magic Sets and Stratified Databases*, Int. Journal of Intelligent Systems, John Wiley & Sons, Ltd., Volume 12, No. 3, S. 203-231, 1997.

[DoD88] US DEPARTMENT OF DEFENSE, NATIONAL COMPUTER SECURITY CENTER: *Glossary of Computer Security Terms*, NCSC-TG-004-88, 1988.

[Dud94] DUDEN: *Das Große Fremdwörterbuch*, Dudenverlag, S. 976, 1994.

[Fen04] DIETER FENSEL: *Ontologies: A Silver Bullet for Knowledge Management*, Springer Berlin Heidelberg, 2004.

[Gru93] GRUBER T.R.: *A translation approach to portable ontology specification*, Knowledge Acquisition, Volume 5, S. 199-220, 1993.

[GLC04] ASUNCIÓN GÓMEZ-PÉREZ, MARIONA FERNÁNDEZ-LÓPEZ, OSCAR CORCHO: *Ontological Engineering*, Springer, S. 11ff, 2004.

[KAON05] BORIS MOTIK: *KAON2 - The KArlsruhe ONtology and semantic web tool suite 2*, http://kaon2.semanticweb.org, 2005.

[Loc03] MARKUS LOCH ET AL.: *First experiences using XACML for access control in distributed systems* , Proceedings of the 2003 ACM workshop on XML security, ACM Press, S. 25ff, 2003.

[MSS04] BORIS MOTIK, ULRIKE SATTLER, RUDI STUDER: *Query Answering for OWL-DL with Rules*, Proceedings of the 3rd International Semantic Web Conference (ISWC 2004), S. 549-563, 2004.

[Nus98] STEFAN NUSSER: *Sicherheitskonzepte im WWW*, Springer Verlag, S. 131ff, 1998.

[OWL04] WORLD WIDE WEB CONSORTIUM (W3C): *Web Ontology Language (OWL)*, http://www.w3.org/2004/OWL/, 2004.

[OAS05] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS): *eXtensible Access Control Markup Language (XACML) 2.0*, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, 2005.

[Pro00] STANFORD MEDICAL INFORMATICS AT THE STANFORD UNIVERSITY SCHOOL OF MEDICINE: *Protégé 2000*, http://protege.stanford.edu, 2000.

[Pro04] MATTHEW HORRIDGE ET AL.: *A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools*, University Of Manchester, http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf, 2004.

[San94] RAVI SANDHU AND P. SAMARATI: *Access Control: Principles and Practice*, IEEE Communications, Volume 32, Number 9, 1994.

[San96] RAVI SANDHU: *Proceedings of the first ACM Workshop on Role-based access control: Roles versus Groups*, Symposium on Access Control Models and Technologies, ACM Press, 1996.

[RBAC96] RAVI SANDHU, EDWARD COYNE, HAL FEINSTEIN AND CHARLES YOUMAN: *Role-Based Access Control Models*, EEE Computer, Volume 29, Number 2, 1996.

[SAP03] IBM BUSINESS CONSULTING SERVICES (SAP PRESS): *SAP-Berechtigungswesen : Design und Realisierung von Berechtigungskonzepten für SAP R/3 und SAP Enterprise Portal*, Galileo Press, 2003.

[SPKR96] SWARTOUT W., PATIL R., KNIGHT K., RUSS T.: *Towards distributed use of large-scale ontologies*, http://ksi.cpsc.ucalgary.ca/-KAW/KAW96/swartout/Banff_96_final_2.html, 1996.

[Str03] THOMAS STRANG: *Technical Report: Vergleich von Wissensmodellen*, http://www.kn.op.dlr.de/∼strang/paper/reports/wissmod.pdf, 2003.

[SWRL04] W3C: *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, http://www.w3.org/Submission/SWRL, 2004.

[Tim03] EKATERINA TIMOSHENKO: *A Classification Theory of Semantics of Normal Logic Programs*, http://www.aifb.uni-karlsruhe.de/WBS/phi/teaching/ws03/manuscripts/slides_timoshenko.pdf, S. 37ff, 2003.

[UnGr96] UNSCHOLD M., GRÜNINGER M.: *Ontologies: Principles, methods and applications*, Knowledge Engineering Review, 11(2), 1996.

[Her04] IVAN HERMAN (W3C): *Semantic Web*, http://www.w3.org/Consortium/Offices/-Presentations/SW_Advanced, 2004.