Zugriffskontrolle für Internet-Informationssysteme:

Ein Lösungsbeitrag für eine maritime Informationsinfrastruktur (MARINFO)

Diplomarbeit

Peter Fecht
Universität Hamburg
Fachbereich Informatik

betreut von

Prof. Dr. Joachim W. Schmidt
Technische Universität Hamburg-Harburg
Arbeitsbereich Softwaresysteme

Prof. Dr. Klaus Brunnstein
Universität Hamburg
Fachbereich Informatik

4. November 1998

Inhaltsverzeichnis

1 Einleitung 1

1.1 Hintergrund und Motivation 1

1.2 Aufgabenstellung und Abgrenzung 2

1.3 Gliederung der Arbeit 5

1.4 Die Entwicklungsumgebung Tycoon-2 6

1.4.1 Das Tycoon-2-System 6
1.4.2 Die Sprache TL-2 7
1.4.3 Generierung von HTML-Dokumenten 8

2 Einführung einer Sicherheitsschicht 10

2.1 Grundlagen der Kryptographie 12

2.2 Das *Secure Socket Layer* – Protokoll 14

2.2.1 Aufbau des Protokollstapels 15
2.2.2 Unterstützte Kryptographie- und MAC-Algorithmen 16
2.2.3 Status der Verbindung und Sitzung 17
2.2.4 Wechsel der kryptographischen Parameter 18
2.2.5 Das SSL-*Handshake* Protokoll 18
2.2.6 Das SSL-*Record*-Protokoll 22

2.3 Beurteilung der Sicherheit 23

2.4 Rechtliche Restriktionen der SSL-Verwendung 25

3 Verwendung von Zertifikaten im Internet 28

3.1 Infrastruktur zur Zertifikatsverwendung 28

3.2 Zertifikate nach X.509-Standard 29

3.3 Zertifikatsautoritäten 32

3.3.1 Beziehungen zwischen Zertifikatsautoritäten 33
3.3.2 Zertifizierungsrichtlinien 35
3.3.3 Verwaltungsaufgaben von Zertifikatsautoritäten 36

3.4 Verwendung von Zertifikaten 38

3.5 Bewertung des Zertifikatskonzeptes 39

4 Modifikation des Tycoon-2-Webservers 40

4.1 Der Tycoon-2-Webserver 40

4.1.1 Struktur des Webservers 40

4.1.2 Funktionsweise des Webservers 42
4.2 SSLeay – Eine SSL-Implementierung 44
4.3 Einbindung von SSLeay in den Tycoon-2-Webserver 46
4.3.1 SSLeay als Tycoon-2-Klasse 46
4.3.2 Aufbau des SSL-Webservers 48
4.3.3 Initialisierung und Konfiguration des Servers 49
4.3.4 Funktionsweise des SSL-Webservers 50
4.3.5 Zugriff auf Zertifikate durch STML-Anwendungen 52
4.4 Beurteilung der Sicherheit 52
5 Autorisierungsmodelle 54
5.1 Definition grundlegender Begriffe 54
5.2 Beschreibung von Modellklassen 55
6 Konzeption eines Autorisierungsdienstes 58
6.1 Anforderungsanalyse 58
6.1.1 Rahmenbedingungen 58
6.1.2 Grundlegende Entwurfsanforderungen 59
6.1.3 Anforderungen an den Autorisierungsdienst 59
6.1.4 Anwendungsfalldiagramm 61
6.1.5 Überblick der Systemstruktur 62
6.2 Spezifikation eines Autorisierungsmodells 63
6.2.1 Subjekte 63
6.2.2 Objekte 65
6.2.3 Zugriffsmodi 67
6.2.4 Verwaltung der Graphen 70
6.2.5 Autorisierungen 71
6.2.5.1 Starke Autorisierungsbasis 71
6.2.5.2 Schwache Autorisierungsbasis 73
6.2.5.3 Ableitung impliziter Autorisierungen 74
6.2.5.4 Beispiel zur Ableitung impliziter Autorisierungen 75
6.2.6 Administration von Autorisierungen 77
6.2.7 Zugriffskontrolle 78
6.3 Entwurf des Autorisierungsdienstes 78
6.3.1 Benutzer 79
6.3.2 Subjekte, Objekte und Zugriffsmodi 79
6.3.3 Autorisierungen 81
6.3.4 Algorithmen zur Autorisierung und Verwaltung 83
6.3.5 Verwaltung der Autorisierungen und Algorithmen 85
6.3.6 Ausführung von Algorithmen 86
6.3.7 Vordefinierte Implikationen und Autorisierungen 87
6.3.8 Ausnahmebehandlung 88
6.3.9 Systemüberblick 89
6.3.10 Integration des Autorisierungsdienstes 89

7 Implementierung des Autorisierungsdienstes in Tycoon-2 91

7.1 Subjekte, Objekte und Zugriffsmodi 91

7.2 Autorisierungen 94

7.2.1 Datenstrukturen 94
7.2.2 Verwaltungsalgorithmen 95
7.2.3 Autorisierungsalgorithmen 96

7.3 Initialisierung und Konfiguration 99

7.4 Integration in Tycoon-2-Anwendungen 99

7.5 Vergleich mit der Zugriffskontrolle von Java 101

8 Integration der realisierten Zugriffskontrollmechanismen 104

8.1 Ausgewählte Dienste des Marinfo-Projektes 104

8.2 Architektur der *BMT Abstract Database* 106

8.3 Entwurf der Zugriffskontrolle 106

8.4 Realisierung der Zugriffskontrolle 107

8.4.1 Authentifizierung und Verschlüsselung 108
8.4.2 Autorisierung 108
8.4.3 Konfiguration des Autorisierungsdienstes 110
8.4.4 Beispiel einer Autorisierung 111

8.5 Erweiterungsmöglichkeiten 112

9 Zusammenfassung 114

9.1 Stand der Implementierung 116

9.2 Ausblick 116

Anhang A Abkürzungsverzeichnis 118

Anhang B Beschreibung ausgewählter Schnittstellen 119

Literaturverzeichnis 125

Abbildungsverzeichnis

Abbildung 1.1: Dienstinfrastrukturen 2
Abbildung 1.2: Zugriffskontrollmechanismen für ein Internet-Informationsystem . 4
Abbildung 1.3: STML – Framework 9
Abbildung 2.1: Digitale Unterschrift 13
Abbildung 2.2: Verwendung von Zertifikaten 14
Abbildung 2.3: Protokollstapel mit SSL 15
Abbildung 2.4: *Handshake*-Protokollverlauf mit neuer Sitzung 19
Abbildung 2.5: *Handshake*-Protokollverlauf mit wiederaufgenommener Sitzung 22
Abbildung 2.6: Paketformat des *Record*-Protokolls 23
Abbildung 3.1: Zertifikatsinfrastruktur 29
Abbildung 3.2: Struktur von X.509-Zertifikaten 30
Abbildung 3.3: Verifizierung der Unterschrift eines Zertifikats 33
Abbildung 3.4: Beispiel eines Zertifizierungspfades 33
Abbildung 3.5: Allgemeine hierarchische Struktur mit Zwischenkanten 34
Abbildung 3.6: Struktur mit zunehmenden Einschränkungen 35
Abbildung 3.7: Lebenszyklus von Zertifikaten 36
Abbildung 4.1: Struktur des Tycoon-2-Webservers 41
Abbildung 4.2: Bearbeitung einer STML-Anfrage 43
Abbildung 4.3: Aufbau der SSLeay-Bibliothek 44
Abbildung 4.4: Verbindungsaufbau mit SSLeay 45
Abbildung 4.5: Einbindung der C-Bibliothek in Tycoon-2 47
Abbildung 4.6: Struktur des SSL-Webservers 48
Abbildung 4.7: Bearbeitung einer STML-Anfrage auf einer SSL-Verbindung 51
Abbildung 6.1: Anwendungsfalldiagramm des Autorisierungsdienstes 61
Abbildung 6.2: Pakete und Kernklassen 62
Abbildung 6.3: Beispiel eines Rollengraphen 64
Abbildung 6.4: Objekttypen 66
Abbildung 6.5: Beispiel eines Objektgraphen 67

Abbildung 6.6: Zugriffskonzept für Daten und Methoden 68
Abbildung 6.7: Beispiel eines Graphen der Zugriffsmodi 69
Abbildung 6.8: Auswertung der Funktion i 72
Abbildung 6.9: Beispiel zur Ableitung von Autorisierungen 76
Abbildung 6.10: Benutzerklasse und Rollenzuordnung 79
Abbildung 6.11: Parametrisierter Graph 80
Abbildung 6.12: Graph zur Aufnahme von Rollen 80
Abbildung 6.13: Verwaltung der Graphen und Benutzer 81
Abbildung 6.14: Hierarchische Struktur von Autorisierungen 82
Abbildung 6.15: Klassenstruktur von Autorisierungen 83
Abbildung 6.16: Klassenhierarchie der Algorithmen 84
Abbildung 6.17: Verwaltung der Autorisierungsbasen und Algorithmen 85
Abbildung 6.18: Sequenzdiagramm zur Durchführung einer Autorisierung 86
Abbildung 6.19: Schablonen für Implikationen und Autorisierungen 87
Abbildung 6.20: Klassen zur Ausnahmebehandlung 88
Abbildung 6.21: Überblick der Klassen des Autorisierungsdienstes 89
Abbildung 6.22: Integration des Autorisierungsdienstes 90
Abbildung 7.1: Beispiel zur Konsistenzbedingung 95
Abbildung 7.2: Ablaufplan des starken Autorisierungsalgorithmus 97
Abbildung 7.3: Beispiel zur Reichweitenauswertung 98
Abbildung 7.4: Sicherheitsmodell des JDK 1.2 101
Abbildung 8.1: Architektur des AdBase-Systems 106
Abbildung 8.2: Überblick der Zugriffskontrolle 107
Abbildung 8.3: Modifikation der *BMT Abstract Database* 109
Abbildung 8.4: Konfiguration der Graphen 110
Abbildung 8.5: Graphen des Beispiels 111
Abbildung 8.6: Abrechnungskomponente und Modelle 112

Tabellenverzeichnis

Tabelle 2.1: Bewertung von Sicherheitsprotokollen 11
Tabelle 2.2: Symmetrische Chiffren des SSL-Protokolls 16
Tabelle 2.3: Parameter des SSL-Sitzungsstatus 17
Tabelle 2.4: Parameter des SSL-Verbindungsstatus 17
Tabelle 2.5: Zustände des SSL-Protokolls 18
Tabelle 2.6: Zeitaufwand für die vollständige Durchsuchung des Schlüsselraums 25
Tabelle 4.1: Konfigurationsparameter des SSL-Webservers 49
Tabelle 6.1: Beispiele zur Mehrfachzuordnung von Rollen 65

Symbolverzeichnis

a	Autorisierung (***a**uthorization*)
AB	Autorisierungsbasis (***a**uthorization **b**ase*)
as	Vorzeichen einer Autorisierung (***a**uthorization **s**ign*)
ASP	Autorisierungsraum (***a**uthorization **sp**ace*)
at	Autorisierungstyp (***a**uthorization **t**ype*)
cm	Kopiermarke (***c**opy **m**arker*)
M	Menge der Zugriffsmodi
m	Zugriffsmodus (*access **m**ode*)
O	Menge der Objekte
o	Objekt (***o**bject*)
P	Reichweite (*scope*)
S	Menge der Subjekte
s	Subjekt (***s**ubject*)
SAB	Starke Autorisierungsbasis (***s**trong **a**uthorization **b**ase*)
WAB	Schwache Autorisierungsbasis (***w**eak **a**uthorization **b**ase*)

Kapitel 1

Einleitung

1.1 Hintergrund und Motivation

Die europäische maritime Industrie ist eine große Gemeinschaft, zu der unter anderem Versicherungen, Klassifizierer[1], Forschungsinstitute, Werften und Reedereien zählen. Zur Sicherung der zukünftigen wirtschaftlichen Entwicklung der Gemeinschaft ist eine Strategie entworfen worden, die in Form des *Research & Development Masterplan*[2] formuliert und strukturiert worden ist. Ein wichtiger Teil des *Masterplan* beschäftigt sich mit dem Aufbau einer offenen, sicheren und effizienten Informationsinfrastruktur, die gegenwärtig nicht existiert, da bislang mit verteilten und heterogenen Systemen gearbeitet wird.

Zur Realisierung einer Informationsinfrastruktur wurde im Rahmen des *ESPRIT*-Programmes das Projekt *Marinfo*[3] (Abk. für *Maritime Industry Information Infrastructure*) ins Leben gerufen, an dem der Arbeitsbereich Softwaresysteme (STS) der Technischen Universität Hamburg-Harburg und Vertreter der europäischen maritimen Industrie beteiligt sind. Das Projekt Marinfo unterteilt sich in vier Bereiche:

Informationsdienste *(information services).* Dabei handelt es sich um multimediale, interaktive Dienste, die Benutzern die enthaltenen Daten präsentieren.

Kommunikationsdienste *(communication services).* Sie dienen der Kommunikation zwischen Benutzern der Informationsinfrastruktur.

Standardisierungsdienste *(standardization services*). Sie unterstützen die effiziente Zusammenarbeit der Gemeinschaft und die einheitliche Realisierung von Diensten.

[1] Klassifizierer sind für die Typklassenfestlegung von Schiffen verantwortlich.

[2] Der *Research & Development Masterplan* ist Bestandteil der Marinfo-Website <http://www.sts.tu-harburg.de/projects/MARINFO/MARINFO>.

[3] Marinfo wird im ESPRIT-Programm unter der Projektnummer 24497 geführt.

Sicherheitsdienste *(security services).* Die Sicherheitsdienste von Marinfo sollen den Zugriff auf Informationen und Dienste der Gemeinschaft kontrollieren.

Zur Realisierung der Sicherheitsdienste werden Mechanismen zur sicheren Datenübertragung, zur Identifizierung von Benutzern und Anbietern, sowie zur gezielten Freigabe von Informationen und Diensten für Benutzer benötigt.

1.2 Aufgabenstellung und Abgrenzung

Das Ziel dieser Arbeit ist die Konzeption und Realisierung einer Zugriffskontrolle für Internet-Informationssysteme. Die Zugriffskontrolle wird exemplarisch in einen Dienst des Marinfo-Projektes integriert.

Ein Internet-Informationssystem entsteht aus dem Zusammenschluß von Internetdiensten zu einer Dienstinfrastruktur. In Abbildung 1.1 [Rudl98] ist ein Informationssystem dargestellt, welches auf einer Dienstinfrastruktur basiert, die sich wiederum aus untereinander verbundenen Informationssystemen ergibt. Die Benutzer der Informations systeme sind jeweils als äußere Kugelhülle dargestellt.

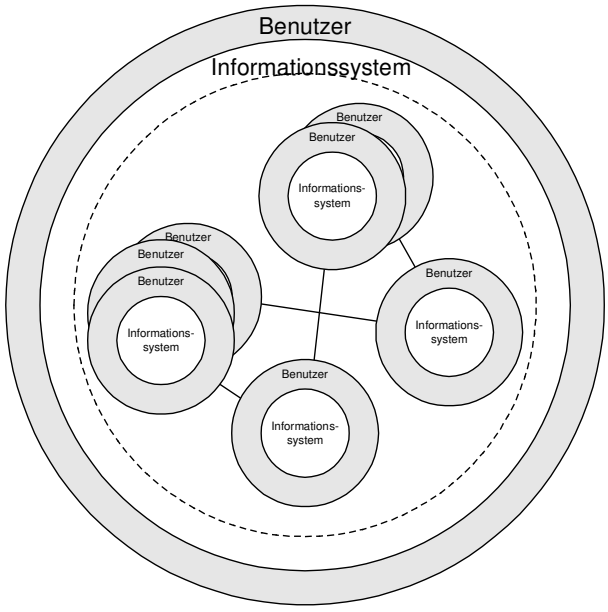

Abbildung 1.1: Dienstinfrastrukturen

Internet-Informationssysteme unterscheiden sich von dem abgebildeten Informationssystemen dadurch, daß das explizite äußere Informationssystem fehlt. Die Dienste der Dienstinfrastruktur sind daher nicht über eine einzelne Schnittstelle ansprechbar, sondern werden von Benutzern direkt über standardisierte Systeme, wie z.B. Internet-Kommunikationsprotokolle, genutzt. [Rudl98]

Damit die Zugriffe der Benutzer auf die Dienste kontrolliert werden können, müssen sie sich zunächst identifizieren. Anschließend kann ihre Dienstnutzung über Nutzungsrechte eingeschränkt werden. Außerdem muß die Kommunikation zwischen Benutzern und Diensten gegenüber Dritten abgeschirmt werden.

Die Zugriffskontrolle für Internet-Informationssysteme umfaßt somit mehrere Aufgabenbereiche und kann nicht über einen einzelnen Sicherheitsmechanismus realisiert werden, sondern erfordert die Kombination mehrerer sich ergänzender Mechanismen.

Zwei grundlegende Anforderungen an eine Zugriffskontrolle sind die Zusicherung der Vertraulichkeit *(confidentiality)* und Integrität (*integrity*) der überwachten Informationen. Die Vertraulichkeit von Informationen gewährleistet, daß Unbefugte keine Kenntnisse über die Bedeutung der Informationen erhalten, selbst dann nicht, wenn der Zugriff auf die Datenträger (z.B. Disketten, Netzwerkpakete) gelingen sollte. Die Integrität von Informationen gewährleistet, daß nur autorisierte Modifikationen an Daten vorgenommen werden, damit die Konsistenz der Datenbasis erhalten bleibt. [Oppl97]

Die Einhaltung dieser Anforderungen kann durch den Einsatz vielfältiger Mechanismen sichergestellt werden, die auf folgenden Konzepten beruhen [ChBe94, HaAt94]:

Identifizierung: Bevor Benutzer Zugang zu Systemen mit sicherheitsrelevanten Informationen erhalten, müssen sie sich identifizieren. Hierzu wird meistens eine Benutzerkennung vergeben, die bei der Systemanmeldung anzugeben ist.

Authentifizierung: Während der Authentifizierung wird die Identität eines Benutzers verifiziert. Dies kann unter anderem durch Eingabe eines Paßworts, Übertragung von Zertifikaten oder Verwendung von Chip-/Magnetkarten geschehen.

Autorisierung: Bei der Autorisierung von Zugriffen wird geprüft, ob der Benutzer das Recht hat, die angegebene Operation auf dem angeforderten Objekt ausführen zu dürfen. Ist das Ergebnis der Autorisierung negativ, wird dem Benutzer der Zugriff verweigert.

Verschlüsselung: Die Verschlüsselung von Daten dient primär der Gewährleistung von Vertraulichkeit. In diesem Bereich verrichten symmetrische Verschlüsselungsalgorithmen gute Dienste. Der Einsatz von asymmetrischen Algorithmen macht die Kryptographie zu einem hervorragenden Werkzeug der Authentifizierung.

Zur Gewährleistung der Zugriffskontrolle müssen Mechanismen, die auf den zuvor beschriebenen Konzepten basieren, in das Internet-Informationssystem integriert werden. Eine mögliche Einordnung der Mechanismen in die grundlegende Architektur des Marinfo-Systems zeigt Abbildung 1.2.

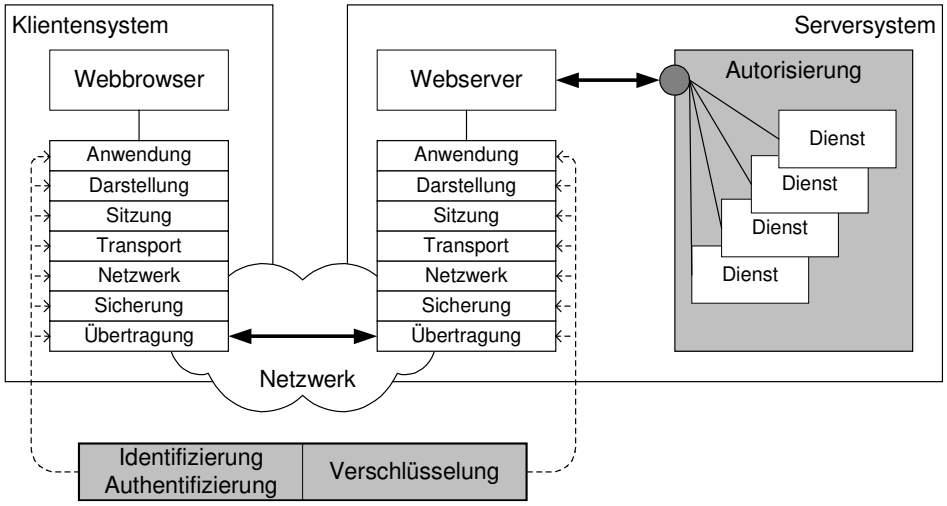

Abbildung 1.2: Zugriffskontrollmechanismen für ein Internet-Informationsystem

Benutzer des Marinfo-Informationssystems kommunizieren über einen Webbrowser mit einem Webserver des Systems, welcher die Anfragen der Benutzer an die Marinfo-Dienste weiterleitet. Die Kommunikation erfolgt über einen Stapel von Protokollen mit unterschiedlichen Aufgabenbereichen. In der Abbildung sind stellvertretend die sieben Schichten des ISO OSI[4]-Modells [Kern92] dargestellt. Ein Protokoll kann die Funktionalität einer oder mehrerer dieser Schichten umfassen. Ein gängiger Internet-Protokollstapel besteht aus dem ***H**yper**t**ext **T**ransfer **P**rotocol (HTTP)*, welches den obersten drei Schichten entspricht, dem ***T**ransmission **C**ontrol **P**rotocol (TCP)* als Transportprotokoll und dem ***I**nternet **P**rotocol (IP)* als Netzwerkprotokoll [ChBe94]. Die Sicherung und Übertragung kann zum Beispiel über ein lokales Netzwerk oder Wählverbindungen erfolgen.

Die zu integrierenden Zugriffskontrollmechanismen beinhalten Dienste zur Identifizierung, Authentifizierung, Verschlüsselung und Autorisierung.

Die Ver- und Entschlüsselung von Daten soll für die Benutzer transparent und anwendungsunabhängig erfolgen. Ebenso soll die Identifizierung und die Authentifizierung unabhängig von Anwendungen sein und nur minimale Benutzeraktionen erfordern. Deshalb bietet sich die Realisierung dieser Dienste als Schicht des Protokollstapels an. Der Vorteil dieses Ansatzes ist, daß die Sicherheitsschicht unabhängig von den übrigen Schichten und den Anwendungen bleibt. Zur Realisierung können zusätzliche Protokolle eingeführt oder bestehende Protokolle um die benötigte Funktionalität erweitert werden.

Der Autorisierungsdienst muß unabhängig von den Kommunikationsmechanismen und den Marinfo-Diensten sein. Dabei soll er flexibel einsetzbar und erweiterbar sein, damit die Anforderungen unterschiedlicher Marinfo-Dienste erfüllt werden können.

[4] ISO ist die Abkürzung für ***I**nternational **S**tandards **O**rganization*; OSI ist die Abkürzung für ***O**pen **S**tandards **I**nterconnection.*

Zusammenfassend ergeben sich aus der Realisierung einer Zugriffskontrolle für Internet-Informationssysteme zwei Aufgabenbereiche:

⇒ Die Integration einer Sicherheitsschicht in bestehende Protokollstapel zur Gewährleistung von Identifizierung, Authentifizierung und Verschlüsselung.

⇒ Die Konzeption und Realisierung eines flexiblen und erweiterbaren Autorisierungsdienstes, der unabhängig von Anwendungen und Kommunikationsmechanismen ist.

Die Zielplattform der Implementierung ist das Tycoon-2-System[5] des Arbeitsbereichs Softwaresysteme der Technischen Universität Hamburg-Harburg (vgl. Kapitel 1.4). Die Benutzer des Marinfo-Informationssystems sollen für den Zugriff Standardbrowser verwenden können, welche bereits eine Sicherheitsschicht enthalten. Als Server dient der Tycoon-2-Webserver, dessen Protokollstapel um eine Sicherheitsschicht erweitert wird. Die ausgewählte Sicherheitsschicht muß von den verwendeten Webbrowsern unterstützt werden. Die realisierten Sicherheitsmechanismen werden auf der Basis eines ausgewählten Informationsdienstes in das Marinfo-Projekt integriert.

Ein Internet-Informationssystem besteht aus Diensten, die auf Systemen ausgeführt werden und über Netzwerke miteinander kommunizieren (vgl. Abbildung 1.2). Die zu realisierende Zugriffskontrolle erstreckt sich über die Bereiche Dienste und Netzwerk. Die Dienste des Informationssystems werden vom Autorisierungsdienst überwacht. Für die Zugriffskontrolle während des Transports über das Netzwerk ist die Sicherheitsschicht zuständig. Zugriffskontrolle für den letzten Bereich, die Systeme der Klienten und Server, ist nicht Gegenstand dieser Arbeit. Dazu muß auf die Sicherheitsmechanismen der Betriebssysteme und der eingebundenen Ressourcen (z.B. Datenbanken) zurückgegriffen werden. Der Autorisierungsdienst kontrolliert daher nur Zugriffe über die Benutzeroberflächen der geschützten Dienste. Direkte Zugriffe auf das Tycoon-2-System müssen vom Betriebssystem kontrolliert werden.

1.3 Gliederung der Arbeit

Nachdem der Hintergrund und die Aufgabenstellung der Arbeit vorgestellt wurden, schließt das erste Kapitel mit einer Beschreibung der Entwicklungsumgebung Tycoon-2. Das Tycoon-2-System wird zur Implementierung von Diensten des Marinfo-Projektes eingesetzt und wird auch zur Realisierung der Zugriffskontrollmechanismen, die im Rahmen dieser Arbeit konzipiert werden, verwendet.

Die Kapitel 2 bis 4 bilden einen zusammenhängenden Teil der Arbeit, welcher die Realisierung von Mechanismen zur Authentifizierung und Verschlüsselung umfaßt. Dazu wird in Kapitel 2 eine Sicherheitsschicht eingeführt, welche die hier benötigten Sicherheitsmechanismen zur Verfügung stellt. Im dritten Kapitel wird eine Infrastruktur beschrieben, welche der Sicherheitsschicht als Grundlage bei der Durchfüh-

[5] Tycoon ist die Abkürzung für ***Ty**ped **C**ommunicating **O**bjects in **O**pen **En**vironments.*

rung ihrer Aufgaben dient. In Kapitel 4 werden die Modifikationen erläutert, die zur Einbindung der Sicherheitsschicht in den Webserver des Tycoon-2–Systems durchgeführt werden müssen.

Der darauffolgende Abschnitt der Arbeit setzt sich aus den Kapiteln 5 bis 7 zusammen, in denen der Autorisierungsdienst realisiert wird. Im fünften Kapitel werden grundlegende Begriffe definiert und eine Klasseneinteilung von Autorisierungsmodellen vorgenommen. Kapitel 6 dient der Konzeption des Autorisierungsdienstes und gliedert sich in die Bereiche Anforderungsanalyse, Spezifikation und Entwurf eines Modells. Dieser Abschnitt der Arbeit endet im siebten Kapitel mit der Implementierung des zuvor entworfenen Autorisierungsmodells in Tycoon-2.

Die Integration der realisierten Zugriffskontrollmechanismen in das Marinfo-Projekt wird in Kapitel 8 vorgenommen. Dazu wird ein Dienst des Projektes ausgewählt, der in Tycoon-2 implementiert ist. Für den ausgewählten Dienst wird eine Zugriffskontrolle, die auf den zuvor realisierten Zugriffskontrollmechanismen basiert, entworfen und prototypisch implementiert.

Das Kapitel 9 enthält eine Zusammenfassung der Ergebnisse dieser Arbeit und geht näher auf den Stand der Implementierung und Erweiterungsmöglichkeiten der Zugriffskontrolle ein.

1.4 Die Entwicklungsumgebung Tycoon-2

Das Tycoon-2-System und seine Programmiersprache **T**ycoon-**L**anguage-2 (TL-2) werden seit etwa 1996 vom Arbeitsbereich Softwaresysteme der TU Harburg[6] und später von der Firma Higher-Order entwickelt. In den folgenden Abschnitten wird das Tycoon-2-System, die Programmiersprache TL-2 und die ***S**tandard **T**ycoon **M**arkup **L**anguage (STML)* erläutert. STML wird zur dynamischen Generierung von ***H**yper**t**ext **M**arkup **L**anguage (HTML)* - Dokumenten verwendet. Die Systembeschreibung basiert auf den Dokumenten [Wegn98, GMSS97, Weik98, Wahl98].

1.4.1 Das Tycoon-2-System

TL-2 Programme werden von einem Compiler in einen maschinenunabhängigen Bytecode übersetzt, der von der virtuellen Maschine des Tycoon-2-Systems interpretiert wird. Die virtuelle Maschine gestattet die effiziente nebenläufige Ausführung von leichtgewichtigen Prozessen *(Threads)*. Alle Threads teilen sich den gemeinsamen Objektspeicher. Der Zugriff auf gemeinsam genutzte Objekte erfordert daher den Einsatz von Synchronisierungsmechanismen, die als Klassen der Standardbibliothek definiert sind.

Alle Daten, der gesamte Bytecode und die Ausführungszustände von Threads werden uniform im Objektspeicher gehalten. Dadurch wird eine effiziente implizite Freispei-

[6] Der Arbeitsbereich STS wurde bis zum Wintersemester 1996/97 an der Universität Hamburg als Arbeitsbereich DBIS geführt.

cherverwaltung ermöglicht, die auf transitiver Erreichbarkeit beruht *(garbage collection)*. Aus der uniformen Datenrepräsentation ergibt sich unmittelbar die Eigenschaft der orthogonalen Persistenz: Die Sicherung des Objektspeichers erhält die Zustände aller Objekte einschließlich der Zustände laufender Prozesse.

Der Kern der virtuellen Maschine implementiert im wesentlichen den Interpreter und den Objektspeicher. Die übrigen Systemkomponenten (z.B. Compiler, Klassenlader und Typprüfer) sind vollständig in TL-2 realisiert und liegen als Bytecode persistent im Objektspeicher vor.

1.4.2 Die Sprache TL-2

Syntaktisch und lexikalisch ist TL-2 relativ stark an C und Java angelehnt. Semantisch und pragmatisch wurden die größten Anleihen bei Smalltalk gemacht [Wegn98].

Klassen, Objekte und Nachrichten

TL-2 ist eine objektorientierte, statisch typisierte Programmiersprache. Jedem Wertausdruck wird zum Zeitpunkt der Übersetzung ein Typ zugeordnet und somit ausgeschlossen, daß zur Laufzeit Typfehler auftreten können (starke Typisierung). Das objektorientierte Paradigma läßt nur Objekte und zwischen ihnen ausgetauschte Nachrichten als Grundlage der Programmierung zu. Jedes Objekt in TL-2 ist ein Exemplar einer Klasse[7] und besitzt eine unabänderbare Objektidentität. Die Objekte werden in einem persistenten Objektspeicher angelegt und unterliegen einer automatischen Freispeicherverwaltung. Alle Variablen stellen nur Verweise auf Objekte dar, die implizit dereferenziert werden.

In Klassen werden Methoden und Variablen definiert, die das Verhalten und die Struktur der Instanzen bestimmen. Methoden können Parameter übernehmen und ein Ergebnis zurückgeben. Bei der Definition von Variablen und Methoden wird zwischen öffentlichen und privaten unterschieden. Falls Variablen oder Methoden als privat deklariert werden, können nur Methoden derselben Klasse auf diese zugreifen.

Vererbung und Metaklassen

Klassen können ihre Variablen- und Methodendefinitionen vererben. Die Vererbung erfolgt von Superklassen an Subklassen, in denen vererbte Definitionen überschrieben werden können (Spezialisierung). Die Definition von aufgerufenen Methoden wird zunächst in der Klasse des aktuellen Objekts und gegebenenfalls in den Superklassen durchgeführt. Da eine Klasse mehrere Superklassen haben kann (Mehrfachvererbung), werden die Superklassen in eine lineare Ordnung gebracht und somit Konflikte während der Durchsuchung vermieden.

TL-2-Klassen sind wiederum Objekte erster Ordnung und gehören deshalb selbst einer Klasse an. Diese sogenannten Metaklassen können Variablen und Methoden beinhalten und sind für die Erzeugung von Instanzen der Klasse zuständig.

[7] Es gibt keine primitiven Datentypen. Auch Zeichenketten und Zahlen werden durch Objekte und Klassen repräsentiert.

Polymorphismus

Das Typsystem von TL-2 basiert auf struktureller Äquivalenz und nicht auf Namensäquivalenz, was z.B. bei C, C++ oder Modula der Fall ist. Als Typ einer Klasse ist die Menge aller Signaturen ihrer Methoden und Variablen definiert. Die Subtypisierungsrelation beruht daher auf strukturellen Vergleichen und bildet die Grundlage für den Subtyppolymorphismus, der es gestattet immer dann, wenn ein Objekt des Typs A erwartet wird, eines des Typs B einzusetzen, sofern B Subtyp von A ist.

Außerdem unterstützt Tycoon den begrenzten parametrischen Polymorphismus (*bounded parametric polymorphism*), der es ermöglicht, Klassen und Methoden mit Typparametern zu versehen, wodurch allgemein formulierte Abstraktionen mit hoher Generizität modelliert werden können.

1.4.3 Generierung von HTML-Dokumenten

Ein Großteil der Benutzeroberflächen von Internet-Informationssystemen basiert auf vordefinierten HTML-Dokumenten. Diese haben den Nachteil, daß sie nur statische Inhalte darstellen. Zur Interaktion mit Benutzern müssen HTML-Dokumente jedoch dynamisch generiert werden. Webserver können über das ***C**ommon **G**ateway **I**nterface (CGI)* mit Skripten und Programmen kommunizieren. Einige Webserver unterstützen zusätzlich den Datenaustausch mit Java-Servlets.

CGI-Skripte und Java-Servlets

Skripte und Programme, die über das CGI aufgerufen werden, generieren unter Berücksichtigung übernommener Parameter ein HTML-Dokument. Das HTML-Dokument wird an den Webserver zurückgegeben, der es zum Klienten überträgt. Nach demselben Prinzip funktionieren Java-Servlets. Sie übernehmen Parameter aus der Anfrage des Klienten und geben ein generiertes HTML-Dokument zurück. [Boes97]

Structured Tycoon Markup Language

Die dynamische Generierung von HTML-Seiten auf der Basis von Tycoon-2-Anwendungen kann unter Einsatz der *Structured Tycoon Markup Language* erfolgen. Die Beschreibung der STML beruht auf den Dokumenten [GaWi97, Wegn98, Rich97].

STML ist eine Erweiterung von HTML und basiert auf der ***S**tandard **G**eneralized **M**arkup **L**anguage (SGML)*. Mit SGML können beliebige Strukturen zur Auszeichnung von Dokumenten definiert werden. Die Struktur wird in einer ***D**ocument **T**ype **D**efinition (DTD)* festgelegt und von Parsern oder Browsern zur Bearbeitung von Dokumenten verwendet.

Zur Integration von TL-2-Sprachelementen definiert die STML-DTD Elemente (*Tags*), die in HTML-Seiten eingebettet werden können. Zu den wichtigsten STML-Tags zählen `tycoon` zur Codeeinbettung, `define` zur Definition von Variablen und `if`/`else` für bedingte Ausführungen.

Standard-Webbrowsern ist die STML-DTD unbekannt. Deshalb werden die STML-Seiten vor der Übertragung an Klienten zu HTML-Seiten transformiert.
Eine Anfrage zur Übertragung einer STML-Seite (*request*) wird vom Webserver an einen HTML-Generator weitergeleitet. Dieser wertet die STML-Tags aus und ruft gegebenenfalls Tycoon-2-Anwendungen auf. Als Ergebnis übergibt der Generator eine dynamisch generierte HTML-Seite an den Webserver, der diese zum Klienten überträgt.

Die zur Verarbeitung von STML-Seiten benötigten Klassen sind in Tycoon-2 realisiert und bilden das Framework aus Abbildung 1.3 [Nied97].

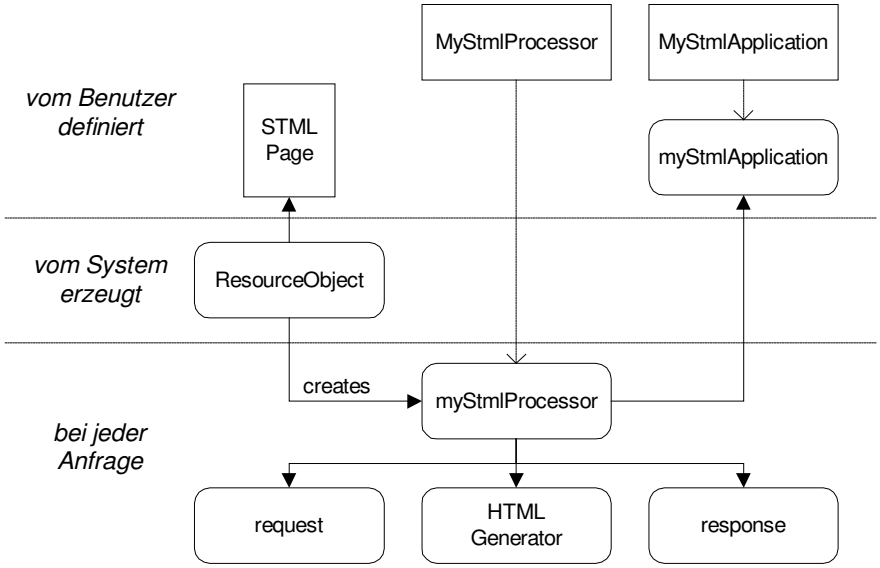

Abbildung 1.3: STML – Framework

Die beiden Klassen `StmlProcessor` und `StmlApplication` sind vom Framework vordefiniert. Der `StmlProcessor` bearbeitet Anfragen und ruft gegebenenfalls Methoden der `StmlApplication` auf. Der Benutzer beerbt die Klassen `StmlProcessor` und `StmlApplication` zur Definition seiner Anwendung (`MyStmlProcessor`, `MyStmlApplication`). Von der STML-Anwendung muß eine Instanz erzeugt werden, die beim Webserver registriert wird.

Außerdem muß der Benutzer STML-Seiten definieren und auf jeder Seite die Klasse des zuständigen STML-Prozessors angeben. Vom System wird jeder STML-Seite ein `ResourceObject` zugeordnet, das bei Anfragen eine Instanz des angegebenen STML-Prozessors erzeugt und diesem das Anfrageobjekt (`request`) übergibt. Das Anfrageobjekt wird vom Webserver erzeugt. Der STML-Prozessor wird also erst beim Aufruf einer zugehörigen STML-Seite erzeugt und beinhaltet Referenzen auf die Anfrage, die Anwendung, einen HTML-Generator und die zu generierende Antwort. Die Tycoon-2-Routinen einer STML-Seite können auf das Prozessorobjekt zugreifen und Anfragedaten auslesen oder Methoden der Anwendung aufrufen.

Kapitel 2

Einführung einer Sicherheitsschicht

Die Funktionen zur Authentifizierung von Kommunikationsteilnehmern und Verschlüsselung von Daten müssen beim Server und beim Klienten vorhanden sein. Damit die Integration der Funktionen in die Systeme der Kommunikationsteilnehmer ohne Anwendungsmodifikationen erfolgen kann, bietet sich die Erweiterung der Protokollstapel um eine unabhängige Sicherheitsschicht an, die mit unterschiedlichen Anwendungen und Transportprotokollen zusammenarbeiten kann.

Das Protokoll der Sicherheitsschicht muß generisch in bezug auf kryptographische Algorithmen sein, damit neue Algorithmen ohne Spezifikationsänderungen eingebunden werden können. Zur Gewährleistung der Interoperabilität zwischen Klienten und Servern soll ein standardisiertes oder weit verbreitetes Protokoll eingesetzt werden.

In Tabelle 2.1 sind drei Sicherheitsprotokolle unterschiedlicher Schichten aufgeführt. Alle Protokolle beinhalten folgende Sicherheitsmechanismen:

- ⇒ Authentifizierung des Servers und / oder des Klienten
- ⇒ Verschlüsselte Datenübertragung
- ⇒ Integritätsprüfung übertragener Daten

Da die benötigten Sicherheitsmechanismen von allen Protokollen zur Verfügung gestellt werden, stützt sich die Bewertung auf die Kriterien Standardisierung, Verbreitung und Unabhängigkeit. Ein standardisiertes Protokoll ist einfacher in bestehende Systeme einzubinden und die Kompatibilität zwischen Implementierungen unterschiedlicher Hersteller ist gewährleistet. Die Verbreitung eines Protokolls ist ein wichtiges Kriterium, um sicherzustellen, daß möglichst viele Benutzer auf geschützte Dienste zugreifen können. Damit das Sicherheitsprotokoll flexibel eingesetzt werden kann, muß es unabhängig von anderen Protokollen und kryptographischen Algorithmen sein.

	Standardisierung	Verbreitung	Unabhängigkeit
IPv6	+	–	+
S-HTTP	–	– –	–
SSL	o	++	+

Tabelle 2.1: Bewertung von Sicherheitsprotokollen

Das ***I**nternet **P**rotocol **V**ersion **6** (IPv6)* wurde von einer Arbeitsgruppe der ***I**nternet **E**ngineering **T**ask **F**orce (IETF)* spezifiziert und soll als Standard etabliert werden [Atki95]. Das Protokoll ist unabhängig von Verschlüsselungsalgorithmen und kann mit unterschiedlichen Authentifizierungsverfahren arbeiten. Die Sicherheitsmechanismen des IPv6 werden bevorzugt zur Kommunikation zwischen Servern eingesetzt, da die Schlüsselverwaltung für Endbenutzer aufwendig ist und entsprechende Standards fehlen.

Das Protokoll ***S**ecure-**HTTP** (S-HTTP)* ist eine Erweiterung des HTTP-Protokolls. Es wurde von der Firma *Enterprise Integration Technologies* entwickelt und wird von der IETF als Draft geführt [ReSc98]. S-HTTP kann nicht mit anderen Anwendungsprotokollen zusammenarbeiten und wird nur von wenigen Webservern und Webbrowsern unterstützt.

Das ***S**ecure **S**ocket **L**ayer (SSL)*- Protokoll der Firma *Netscape* bildet eine zusätzliche Schicht, die von Anwendungsprotokollen genutzt werden kann. Es wurde in zahlreiche Produkte auf mehreren Plattformen integriert und beim *World-Wide-Web-Consortium (W3C)* zur Standardisierung eingereicht [Nets96a]. Das Protokoll arbeitet mit zahlreichen kryptographischen Algorithmen zusammen. Die Überarbeitung des Protokolls durch eine Arbeitsgruppe der IETF führte zur Spezifikation des ***T**ransport **L**ayer **S**ecurity (TLS)*- Protokolls[8], welches zukünftig auch von Netscape unterstützt wird.

Das SSL-Protokoll beinhaltet die benötigten Sicherheitsmechanismen und erfüllt die Anforderungen in bezug auf Standardisierung, Verbreitung und Unabhängigkeit. Es wird deswegen nachfolgend als Grundlage der Zugriffskontrolle für Internet-Informationssysteme dienen.

Bevor auf die Spezifikation des SSL-Protokolls eingegangen wird, erfolgt eine Beschreibung der Grundlagen der Kryptographie. Das Kapitel schließt mit einer Beurteilung der Sicherheit und einer Darstellung der rechtlichen Restriktionen der SSL-Verwendung.

Dieses Kapitel basiert auf der SSL-Spezifikation [Nets96a, Hick95] und den Implementierungsbeschreibungen [HuYo97, BrDa96, Hirs97].

[8] Die Ergebnisse der TLS-Arbeitsgruppe der IETF können unter <http://www.consensus.com/ietf-tls/> eingesehen werden.

2.1 Grundlagen der Kryptographie

Dieses Kapitel bietet eine kurze Einführung in das Gebiet der Kryptographie und stellt einige ihrer Anwendungen vor, die im SSL-Protokoll (vgl. [Nets96b]) eingesetzt werden. Für detaillierte Beschreibungen wird auf [Denn82, Schn94] verwiesen.

Moderne kryptographische Systeme bestehen aus einer Funktion, die einen Klartext P und einen Schlüssel K auf ein Chiffrat C abbildet, und der Umkehrfunktion, die ein Chiffrat C und einen Schlüssel K^{-1} auf einen Klartext P abbildet. Es wird folgende Notation verwendet:

Verschlüsselung: $C \leftarrow K[P]$ Entschlüsselung: $P \leftarrow K^{-1}[C]$

Die Funktionen werden zu Verschlüsselungsalgorithmen zusammengefaßt, die nachfolgend auch als Chiffre bezeichnet werden.

Symmetrische Kryptographie

Symmetrische Kryptographie[9] schließt alle Systeme ein, die denselben Schlüssel zur Ver- und Entschlüsselung benutzen. Es gilt somit:

$$K = K^{-1}$$

Symmetrische Algorithmen werden in Blockchiffren *(block ciphers)* und Datenstromchiffren *(stream ciphers)* unterteilt. Blockchiffren können nur Daten verschlüsseln, deren Länge ein Vielfaches ihrer Blockgröße beträgt. Chiffren können in unterschiedlichen Modi arbeiten, die Auswirkungen auf die Qualität der Verschlüsselung haben. Zur Erhöhung der Qualität können beispielsweise die Datenblöcke von Blockchiffren untereinander verknüpft werden[10].

Die Vorteile symmetrischer Algorithmen sind kleine Schlüssel[11] und schnelle Ver- und Entschlüsselungsprozesse. Der Nachteil ergibt sich aus dem Schlüsselaustausch. Da nur ein Schlüssel verwendet wird, muß dieser sicher übertragen werden.

Asymmetrische Kryptographie

Diese Klasse von Algorithmen benutzt zur Ver- und Entschlüsselung von Daten ein Schlüsselpaar. In solchen Systemen gilt also $K \neq K^{-1}$. Außerdem kann aus dem öffentlichen Schlüssel *(public key)* K nicht der private Schlüssel *(private key)* K^{-1} abgeleitet werden. Es gilt:

Der Besitzer eines Schlüsselpaars verteilt seinen öffentlichen Schlüssel und hält seinen privaten Schlüssel geheim. Ihm kann eine Nachricht gesendet werden, indem sein

[9] Auch als klassische, konventionelle oder *Private-Key* Kryptographie bezeichnet.
[10] Die Verknüpfung von Datenblöcken wird als *cipher block chaining (CBC)* bezeichnet. Der erste Block wird im CBC-Modus mit einem Initialisierungsvektor verknüpft.
[11] Die symmetrischen Algorithmen des SSL-Protokolls arbeiten mit Schlüsselbreiten von 40 bis 168 Bit.

öffentlicher Schlüssel zur Verschlüsselung benutzt wird. Die Nachricht kann nur mit dem zugehörigen privaten Schlüssel entschlüsselt werden.

Asymmetrische Algorithmen arbeiten mit sehr viel größeren Schlüsseln[12] als symmetrische und die Ver- und Entschlüsselungsprozesse dauern wesentlich länger. Aus diesen Gründen werden asymmetrische Algorithmen nicht zur Massendatenverschlüsselung, sondern bevorzugt zum Schlüsselaustausch[13] oder für digitale Unterschriften eingesetzt.

Prüfsumme *(digest)*

Prüfsummen werden als Grundlage nachfolgend beschriebener kryptographischer Anwendungen benötigt. Sie sind das Ergebnis einer Einwegfunktion, die eine Nachricht als Eingabe erhält. Aus der Prüfsumme können keine Rückschlüsse auf die ursprüngliche Nachricht gezogen werden. Für kryptographische Anwendungen werden besondere Einwegfunktionen, sogenannte Hashfunktionen, verwendet, die folgende Eigenschaften haben: Zu einer gegebenen Prüfsumme kann keine passende Nachricht kontruiert werden, und es können keine zwei Nachrichten mit identischen Prüfsummen konstruiert werden.

Digitale Unterschrift *(digital signature)*

Digitale Unterschriften sind eine Anwendung der asymmetrischen Kryptographie und dienen der Authentifizierung des Absenders und der Sicherung der Integrität einer Nachricht.

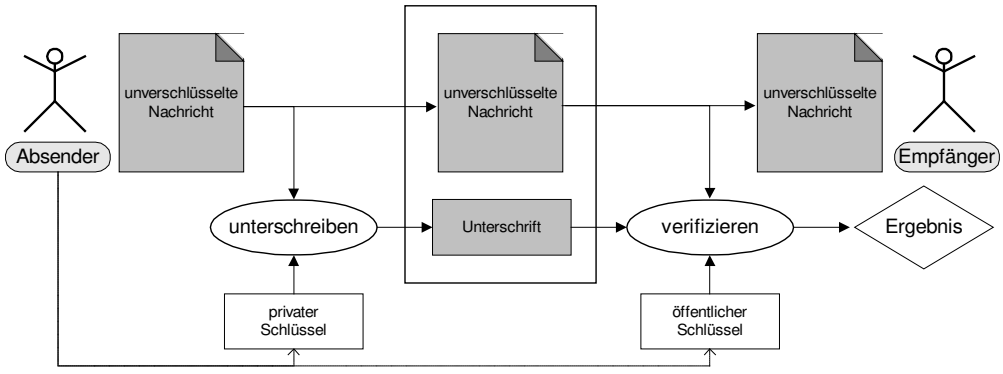

Abbildung 2.1: Digitale Unterschrift

In Abbildung 2.1 [FoBa97] verschlüsselt der Absender eine Prüfsumme der Nachricht und fügt sie als Unterschrift hinzu. Der Empfänger entschlüsselt die Unterschrift mit dem öffentlichen Schlüssel des Absenders. Er berechnet die Prüfsumme der erhaltenen Nachricht und vergleicht das Ergebnis mit der übertragenen Prüfsumme. Fällt der Vergleich positiv aus, kann der Empfänger sicher sein, eine nicht modifizierte Nachricht des Absenders erhalten zu haben.

[12] Die Breite der asymmetrischen Schlüssel des SSL-Protokolls übertreffen die der symmetrischen ungefähr um den Faktor 10.

[13] Die asymmetrisch verschlüsselte Nachricht besteht in diesem Fall aus einem symmetrischen Schlüssel, der zur weiteren Kommunikation verwendet wird.

Zertifikat *(certificate)*

In Zertifikaten werden Daten des Zertifikatsbesitzers und sein öffentlicher Schlüssel gespeichert. Das Zertifikat wird mit dem privaten Schlüssel des Zertifikatsausstellers unterschrieben, der damit garantiert, daß der enthaltene öffentliche Schlüssel dem Zertifikatsbesitzer gehört. Der öffentliche Schlüssel des Ausstellers wird ebenfalls in einem Zertifikat gespeichert und Benutzern zur Verfügung gestellt. Der Empfänger eines Zertifikats verwendet den öffentlichen Schlüssel des Ausstellers zur Prüfung der Unterschrift. Dem Zertifikat kann der Empfänger dann Angaben zur Identität des Zertifikatsbesitzers und dessen öffentlichen Schlüssel entnehmen. (vgl. Abbildung 2.2)

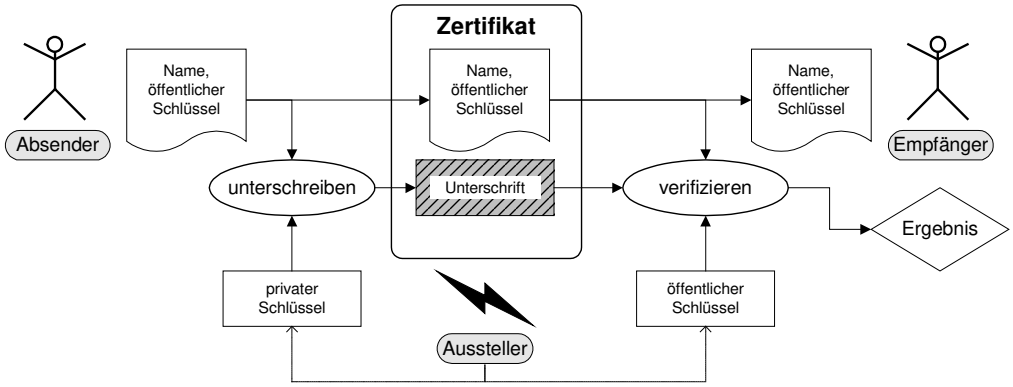

Abbildung 2.2: Verwendung von Zertifikaten

Message Authentication Code

Der ***M**essage **A**uthentication **C**ode (MAC)* ist eine Prüfsumme, die aus einer Nachricht und einem geheimen Schlüssel berechnet wird und dient der Erkennung von Modifikationen an der zugehörigen Nachricht. Der Absender und Empfänger der Nachricht können die Prüfsumme berechnen und mit einem Vergleich die Integrität der Nachricht prüfen. Damit nicht sowohl die Nachricht als auch die Prüfsumme modifiziert werden können, wird zur Berechnung ein Schlüssel verwendet, der nur dem Absender und Empfänger bekannt ist.

2.2 Das *Secure Socket Layer* – Protokoll

Das *Secure Socket Layer*-Protokoll wird zwischen der Anwendungs- und Transportschicht in einen Protokollstapel integriert und ist unabhängig von den umgebenden Protokollen.

Unter Anwendung der zuvor beschriebenen kryptographischen Algorithmen gewährleistet das Protokoll die Authentifizierung der Kommunikationspartner, sowie die vertrauliche Übertragung von Daten und deren Integrität. Zur Authentifizierung werden Zertifikate eingesetzt, deren öffentlicher Schlüssel während des Schlüsselaustausches benutzt wird. Anwendungsdaten werden vor der Übertragung symmetrisch verschlüsselt und mit einem MAC vor Manipulationen geschützt.

Das Ziel des Protokolls, der Schutz von Anwendungsdaten, ist hiermit erreicht. Zusätzliche Algorithmen können ohne Modifikation der Spezifikation in bestehende Implementierungen eingebunden werden. Zur Steigerung der Effizienz können neue Verbindungen mit bereits festgelegten Algorithmen und Parametern früherer Sitzungen aufgebaut werden.

Die folgenden Abschnitte beschreiben den Aufbau und die Funktionsweise des Protokolls. Die vollständige Definition der verwendeten Datenstrukturen und Parametertypen kann der Protokollspezifikation [Nets96a] entnommen werden.

2.2.1 Aufbau des Protokollstapels

Für ein Kommunikationsprotokoll ist es wichtig, konform zu einem bestehenden und etablierten Protokollstapel zu sein. Auf diese Weise kann es einfach eingebunden werden oder andere Protokolle ersetzen. Diese Anforderungen erfüllt das SSL-Protokoll, da es in Verbindung mit jedem Transportprotokoll und mit Anwendungsprotokollen, wie beispielsweise HTTP, FTP oder Telnet, genutzt werden kann.

Die Abbildung 2.3 zeigt das SSL-Protokoll in einem Protokollstapel zwischen den Anwendungsprotokollen und den Transportprotokollen. Das SSL-Protokoll besteht aus den vier Teilprotokollen *Handshake*, *Record*, *Alert* und *ChangeCipherSpec*.

Das *Handshake*-Protokoll handelt kryptographische Parameter aus und führt die Authentifizierung des Klienten und/oder des Servers durch. Das *Record*-Protokoll unterteilt den Datenstrom in Pakete und verschlüsselt diese mit dem ausgehandelten Algorithmus. Zur Neuverhandlung der vom *Record*-Protokoll benötigten kryptographischen Parameter dient das *ChangeCipherSpec*-Protokoll. Das *Alert*-Protokoll wird zur Fehlermeldung verwendet und unterscheidet Warnungen *(warning)* und fatale Fehler *(fatal)*. Letztere führen zum Abbruch einer Verbindung.

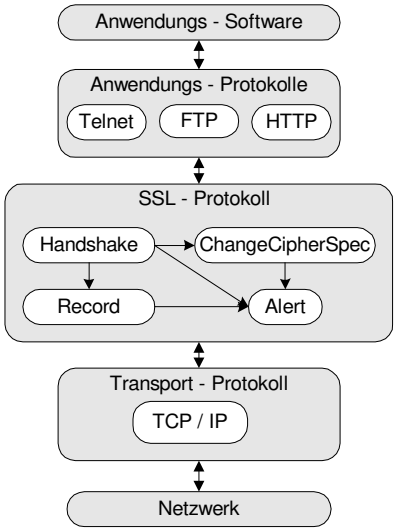

Abbildung 2.3: Protokollstapel mit SSL

2.2.2 Unterstützte Kryptographie- und MAC-Algorithmen

Das SSL-Protokoll kann mit mehreren Algorithmen zur Authentifizierung, Verschlüsselung und MAC-Berechnung zusammenarbeiten und überläßt die Auswahl den Kommunikationspartnern einer Verbindung. Es können keine einzelnen Algorithmen ausgewählt werden, sondern nur vordefinierte Pakete (*CipherSuites)*. Eine *CipherSuite* besteht aus jeweils einem Algorithmus der Bereiche Authentifizierung / Schlüsselaustausch, Datenverschlüsselung und MAC-Berechnung. Die Festlegung gültiger *CipherSuites* erfolgt in der SSL-Protokollspezifikation.

Literaturverweise auf die Spezifikationen der nachfolgend aufgeführten Algorithmen können dem Anhang der SSL-Spezifikation [Nets96a] entnommen werden.

Authentifizierung und Schlüsselaustausch

Zur Authentifizierung und zum Austausch symmetrischer Schlüssel werden asymmetrische Verschlüsselungsalgorithmen eingesetzt. Der Schlüsselaustausch kann mit Diffie-Hellman, RSA oder KEA (Abk. für ***k**ey **e**xchange **a**lgorithm*) erfolgen. Falls RSA für den Schlüsselaustausch ausgewählt wird, erfolgt die Authentifizierung, also die Prüfung von digitalen Unterschriften, ebenfalls mit RSA. Diffie-Hellman benötigt als reines Schlüsselaustauschverfahren zur Durchführung der Authentifizierung den RSA oder DSS (Abk. für ***d**igital **s**ignature **s**tandard*) -Algorithmus. Der KEA-Algorithmus wird nur vom hardwarebasierten Fortezza-System verwendet.

Datenverschlüsselung

Die Verschlüsselung der Nutzdaten des SSL-Protokolls erfolgt durch symmetrische Algorithmen. Es werden folgende Datenstrom- und Blockchiffren unterstützt:

Chiffre	Typ	Schlüsselbreite
RC2	Block	variabel 40-128 Bit
RC4	Datenstrom	variabel 40-128 Bit
DES[14]	Block	56 Bit
Triple-DES	Block	168 Bit
IDEA[15]	Block	128 Bit
Skipjack	Block	96 Bit

Tabelle 2.2: Symmetrische Chiffren des SSL-Protokolls

Der Skipjack-Algorithmus wird nur in Kombination mit Fortezza verwendet.

MAC-Berechnung

Zur Berechnung von *Message Authentication Codes* stehen die beiden Algorithmen ***M**essage **D**igest **5** (MD5)* und ***S**ecret **H**ash **A**lgorithm (SHA)* zur Verfügung.

[14] Abkürzung für ***D**ata **E**ncryption **S**tandard*.
[15] Abkürzung für ***I**nternational **D**ata **E**ncryption **A**lgorithm*.

Eine in der Protokollspezifikation festgelegte *CipherSuite* ist zum Beispiel:

```
SSL_RSA_WITH_RC4_128_MD5
```

Sie arbeitet mit dem RSA-Algorithmus zur Authentifizierung und zum Schlüsselaustausch. Die symmetrische Chiffre RC4 wird 128 Bit breiten Schlüsseln zur Datenverschlüsselung eingesetzt und der Algorithmus MD5 zur MAC-Berechnung verwendet.

2.2.3 Status der Verbindung und Sitzung

Der Zustand eines SSL-Protokollverlaufs ergibt sich aus dem Verbindungs- und Sitzungsstatus. Der Sitzungstatus bleibt über die Lebensdauer einer Verbindung hinaus erhalten und kann zum Aufbau neuer Verbindungen verwendet werden. Er enthält die in Tabelle 2.3 aufgeführten Parameter.

session_identifier	Identifiziert die Sitzung
client_certificate	Das X.509v3-Zertifikat des Klienten (optional)
compression_method	Der verwendete Komprimierungsalgorithmus
cipher_spec	Spezifiziert den Verschlüsselungs- und MAC-Algorithmus
master_secret	Gemeinsame geheime Daten des Servers und Clients
is_resumable	Legt fest, ob Sitzungen wiederaufgenommen werden können

Tabelle 2.3: Parameter des SSL-Sitzungsstatus

Die Parameter des Verbindungsstatus aus Tabelle 2.4 werden von den Verschlüsselungs- und MAC-Algorithmen benötigt.

server & client_random	Vom Server und Klienten für jede Verbindung gewählte Bytesequenzen
server_write_MAC_secret	Ein Attribut des MAC-Algorithmus für Schreiboperationen des Servers
client_write_MAC_secret	Ein Attribut des MAC-Algorithmus für Schreiboperationen des Klienten
server_write_key	Der Schlüssel des vom Server verwendeten symmetrischen Algorithmus
client_write_key	Der Schlüssel des vom Klienten verwendeten symmetrischen Algorithmus
initialization_vectors	Werden von Blockchiffren zur Initialisierung benötigt[16]
sequence_numbers	Der Klient und Server versehen empfangene und versendete Nachrichten mit einer 64 Bit-Sequenznummer

Tabelle 2.4: Parameter des SSL-Verbindungsstatus

[16] Vgl. Kapitel 2.1

2.2.4 Wechsel der kryptographischen Parameter

Das SSL-Protokoll ermöglicht zur Erhöhung der Sicherheit den Austausch von Parametern während einer Verbindung. Der Austausch der Parameter bewirkt eine Änderung des Sitzungs- und Verbindungsstatus. Die Steuerung des Statuswechsels übernimmt das *ChangeCipherSpec*-Protokoll.

Damit der Wechsel verzögerungsfrei erfolgen kann, werden zuvor die neuen Parameter festgelegt und zwischengespeichert. Da die Parameter für Lese- und Schreiboperationen getrennt verwaltet werden, ergeben sich jeweils für den Klienten und den Server die vier Zustände aus Tabelle 2.5.

	Aktueller Status *(current state)*	Schwebender Status *(pending state)*
Lesestatus *(read state)*	Parameter zur Entschlüsselung aktueller Nachrichten	Parameter zur Entschlüsselung zukünftiger Nachrichten
Schreibstatus *(write state)*	Parameter zur Verschlüsselung aktueller Nachrichten	Parameter zur Verschlüsselung zukünftiger Nachrichten

Tabelle 2.5: Zustände des SSL-Protokolls

Die Neuverhandlung der kryptographischen Parameter wird durch Übertragung der Nachrichten `clientKeyExchange` und `certificateVerify`[17] des *Handshake*-Protokolls durch den Klienten ausgelöst. Die Definition dieser Nachrichten erfolgt im Kapitel 2.2.5.

Das *ChangeCipherSpec*-Protokoll verwendet die Nachricht `changeCipherSpec`. Wird diese empfangen, kopiert der Empfänger seinen schwebenden Lesestatus in den aktuellen Lesestatus, da nachfolgende Nachrichten mit den neuen Parametern verschlüsselt werden. Der Absender kopiert seinen schwebenden Schreibstatus in den aktuellen Schreibstatus und verschlüsselt somit nachfolgende Nachrichten mit den neuen Parametern. Für einen vollständigen Zustandswechsel müssen also der Klient und der Server eine `changeCipherSpec`-Nachricht absenden.

2.2.5 Das SSL-*Handshake* Protokoll

Jeder SSL-Verbindungsaufbau beginnt mit der Durchführung des *Handshake*-Protokolls. Im Verlauf des Protokolls werden die Parameter der Sitzung und Verbindung festgelegt; der Server authentifiziert sich gegenüber dem Klienten und optional der Klient gegenüber dem Server. Die Möglichkeit der Wiederaufnahme von Sitzungen führt zu zwei unterschiedlichen Protokollverläufen.

[17] Die Nachricht *certificateVerify* wird nur benötigt, falls der Client sich durch ein Zertifikat authentifiziert hat.

Protokollverlauf mit neuer Sitzung

Die Abbildung 2.4 zeigt den Protokollverlauf des *Handshake*-Protokolls, wenn die Verbindung auf einer neuen Sitzung basieren soll.

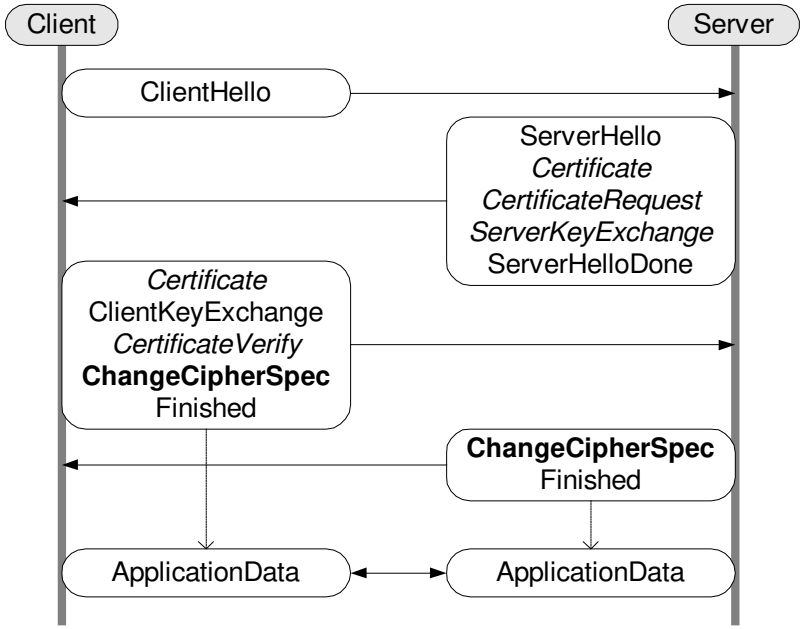

Abbildung 2.4: *Handshake*-Protokollverlauf mit neuer Sitzung

Kontaktaufnahme durch den Klienten

Die erste übertragene Nachricht ist `ClientHello` und wird vom Klienten an den Server geschickt. Als erstes Feld der Nachricht gibt `client_version` die Version des vom Klienten eingesetzten SSL-Protokolls an. Die Datenstruktur `random` wird im weiteren Protokollverlauf benötigt und besteht aus einem Zeitstempel und zufällig generierten Daten. Falls die Sitzungs-ID (`session_identifier`) einen Wert enthält, kann er zu einer früheren, der aktuellen oder anderen aktiven Verbindung gehören. Wird eine frühere oder aktive Verbindung referenziert, ergibt sich der weitere Protokollverlauf aus Abbildung 2.5. Die Angabe der aktuellen Verbindung führt zur Neuverhandlung der verwendeten kryptographischen Parameter. Die Listen `cipher_suite` und `compression_method` enthalten Referenzen auf alle *CipherSuites* und Komprimierungsalgorithmen, die vom Klienten unterstützt werden. Die Reihenfolge der Listenelemente entspricht der Rangordnung der bevorzugten Algorithmen. Als Antwort des Servers erwartet der Klient die Nachricht `ServerHello`. Jede andere Nachricht löst einen Fehler aus, der zum Abbruch der Verbindung führt.

Bearbeitung der Anfrage des Klienten durch den Server

Der Server bearbeitet die `ClientHello`-Nachricht und antwortet mit `ServerHello` oder einer Fehlermeldung. Zur Festlegung der verwendeten Protokollversion beinhaltet die `ServerHello`-Nachricht im Feld `server_version` die höchste Version des Servers, die

auch vom Klienten unterstützt wird. Der Server generiert, unabhängig vom Zufallswert des Klienten, die Datenstruktur des `random`-Feldes. Das Feld `session_id` verweist auf die der Verbindung zugeordneten Sitzung[18], oder bleibt leer, falls die Sitzung nicht wiederaufnehmbar sein soll. Der Server stimmt seine unterstützten *CipherSuites* und Komprimierungsalgorithmen mit den Präferenzen des Klienten ab und trägt seine Auswahl in die Felder `cipher_suite` und `compression_method` ein.

Die Nachricht `Certificate` enthält eine Liste von Zertifikaten. Die Liste entspricht einem Zertifikatspfad, der aus dem Zertifikat des Servers und den Zertifikaten der jeweils ausstellenden Zertifikatsautorität besteht. Auf Zertifikatsautoritäten und Zertifikate wird detailliert in Kapitel 3 „Verwendung von Zertifikaten im Internet“ eingegangen.

Falls der Server kein Zertifikat besitzt oder nur über ein Zertifikat für digitale Unterschriften verfügt, wird ein temporärer öffentlicher Schlüssel des Servers mit der Nachricht `ServerKeyExchange` übertragen. Die Nachricht `Certificate` erübrigt sich in diesem Fall. Der temporäre öffentliche Schlüssel setzt sich, in Abhängigkeit vom verwendeten Algorithmus, aus mehreren Parametern zusammen, die vom Server digital unterschrieben werden.

Ein authentifizierter Server kann von einem Klienten ein Zertifikat anfordern. Dies geschieht mit der Nachricht `CertificateRequest`, die eine Liste mit unterstützten Zertifikatstypen[19] und eine Liste mit Namen von annehmbaren Zertifikatsautoritäten beinhaltet.[20]

Die `ServerHelloDone`-Nachricht markiert das Ende des `ServerHello`-Blocks. Der Server wartet nach ihrer Übertragung auf eine Antwort des Klienten.

Authentifizierung und Berechnung der kryptographischen Parameter

Nach Erhalt der `ServerHelloDone`-Nachricht beginnt der Klient mit der Bearbeitung der erhaltenen Nachrichten. Er entnimmt dem Serverzertifikat oder der Nachricht `ServerKeyExchange` den öffentlichen Schlüssel des Servers und prüft, ob die Parameter der `ServerHello`-Nachricht annehmbar sind.

Mit der Nachricht `Certificate`[21] antwortet der Klient auf einen `CertificateRequest` des Servers. Falls der Klient kein geeignetes Zertifikat besitzt, wird eine Fehlermeldung übertragen, und der Server kann entscheiden, ob er den Verbindungsaufbau fortsetzt oder abbricht.

---

[18] Es handelt sich hierhei entweder um eine neue Sitzung, oder um eine bestehende, für die neue kryptographische Parameter festgelegt werden sollen.

[19] Ein Zertifikatstyp ergibt sich aus den verwendeten Authentifizierungs- und Schlüsselaus tauschalgorithmen. Vgl. Kapitel 2.2.2.

[20] Der Klient hat nicht die Möglichkeit in der Nachricht *ClientHello* bevorzugte Zertifikatsautoritäten anzugeben. Er muß die Auswahl dem Server überlassen. Dieses Vorgehen kann zum Abbruch des Verbindungsaufbaus führen, falls der Server ein Zertifikat einer nicht unterstützten Zertifikatsautorität auswählt.

[21] Es handelt sich um denselben Nachrichtentyp wie beim Serverzertifikat.

Falls der Klient sich mit einem Zertifikat authentifiziert, mit dem digital unterschrieben werden kann, überträgt er die Nachricht CertificateVerify zur expliziten Verifizierung seines Zertifikats. Sie enthält eine mit seinem privaten Schlüssel unterschriebene Prüfsumme.

Die Felder der ClientKeyExchange-Nachricht beinhalten Daten des Klienten, die zur Generierung des symmetrischen Schlüssels benötigt werden, welcher zur Verschlüsselung der Anwendungsdaten eingesetzt wird. Die Struktur der Felder ist vom Schlüsselaustauschalgorithmus abhängig. Der RSA-Algorithmus arbeitet beispielsweise mit einem pre_master_secret, das aus Zufallsdaten und der Protokollversion besteht. Das pre_master_secret wird mit dem öffentlichen Schlüssel des Servers verschlüsselt und im Feld encrypted_pre_master_secret gespeichert.

Der Klient und der Server berechnen aus dem pre_master_secret das master_secret. In diese Berechnung gehen neben dem pre_master_secret die random-Werte der Hello-Nachrichten und vordefinierte Werte ein. Anschließend werden aus dem master_secret und den random-Werten, jeweils für den Klienten und den Server, der Schlüssel des symmetrischen Algorithmus, der geheime Parameter des MAC-Algorithmus und ein Initialisierungsvektor berechnet.

Als nächstes überträgt der Klient die ChangeCipherSpec[22]-Nachricht. Sie gehört nicht zum *Handshake*-Protokoll und veranlaßt den Wechsel der kryptographischen Parameter, so daß folgende Nachrichten mit dem gerade ausgehandelten Algorithmus verschlüsselt werden.

Abschluß des Verbindungsaufbaus

Unmittelbar nach der ChangeCipherSpec-Nachricht überträgt der Klient eine verschlüsselte Finished-Nachricht und beginnt, ohne auf eine Antwort des Servers zu warten, mit der verschlüsselten Übertragung von Anwendungsdaten. Die Finished-Nachricht enthält eine Prüfsumme über alle bisher ausgetauschten *Handshake*-Nachrichten. Der Server berechnet ebenfalls die Prüfsumme und kann, falls die Werte identisch sind, sicher sein, daß der *Handshake* nicht von Dritten manipuliert wurde.

Der Server beendet das *Handshake*–Protokoll mit der Übertragung einer ChangeCipherSpec- und einer Finished-Nachricht. Anschließend beginnt er mit dem Austausch von verschlüsselten Anwendungsdaten.

Protokollverlauf mit wiederaufgenommener Sitzung

Der verkürzte Protokollverlauf aus Abbildung 2.5 ergibt sich, falls die Parameter einer früheren oder anderen aktiven Sitzung für die aufzubauende Verbindung verwendet werden sollen. Dies wird durch einen entsprechenden Eintrag im Feld session_id der Nachricht ClientHello angezeigt.

[22] Vgl. Kapitel 2.2.4

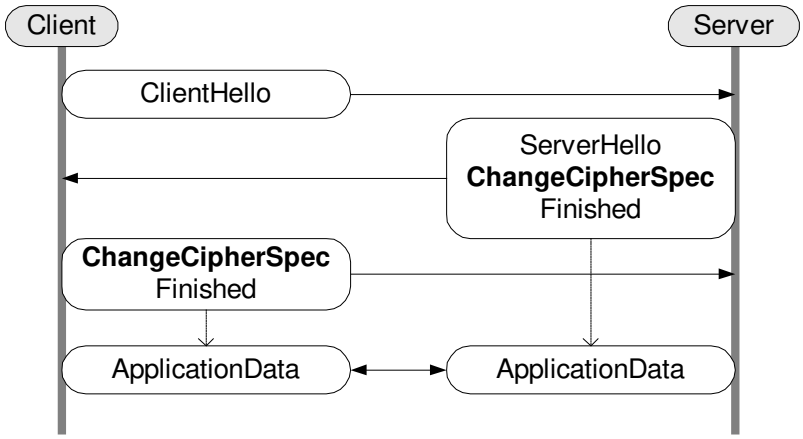

Abbildung 2.5: *Handshake*-Protokollverlauf mit wiederaufgenommener Sitzung

Die Felder `random`, `cipher_suites` und `compression_methods` der `ClientHello`-Nachricht müssen belegt werden, damit ein vollständiger *Handshake* stattfinden kann, falls die Übernahme der Parameter der angegebenen Sitzung scheitert. Dieser Fall kann eintreten, weil Sitzungsparameter nur für einen begrenzten Zeitraum gespeichert werden.

Falls der Server die angeforderte Sitzung noch gespeichert hat, trägt er die Sitzungskennung in das Feld `session_id` der Nachricht `ServerHello` ein. Anderenfalls wird eine neue Sitzungskennung generiert und der vollständige *Handshake* durchgeführt. Die Felder `cipher_suite` und `compression_method` erhalten die Werte der wiederaufgenommenen Sitzung. Die Verbindungsparameter werden unter Verwendung des `master_secret` der wiederaugenommenen Sitzung berechnet.

Der weitere Protokollverlauf entspricht dem des vollständigen *Handshake*. Im Anschluß an den Austausch der `ChangeCipherSpec`-Nachrichten wird der Verbindungsaufbau mit den `Finished`-Nachrichten beendet. Danach kann die Übertragung der Anwendungsdaten beginnen.

2.2.6 Das SSL-*Record*-Protokoll

Das *Record*-Protokoll empfängt Daten beliebiger Länge aus höheren Schichten und zerlegt sie in Fragmente mit einer maximalen Länge von 2^{14} Byte, die anschließend komprimiert, verschlüsselt und an tiefere Schichten übergeben werden. Beim Kommunikationspartner wird dieser Prozeß in umgekehrter Reihenfolge durchgeführt.

Die Abbildung 2.6 zeigt das Format eines Pakets mit fragmentiertem, komprimiertem und verschlüsseltem Inhalt.

Der Kopf des Pakets besteht aus den Feldern `content_type` und `protocol_version`. Sie identifizieren den Inhalt des Pakets (z.B. Nutzdaten, Fehlermeldung, *Handshake*-Nachricht) und die verwendete Version des SSL-Protokolls.

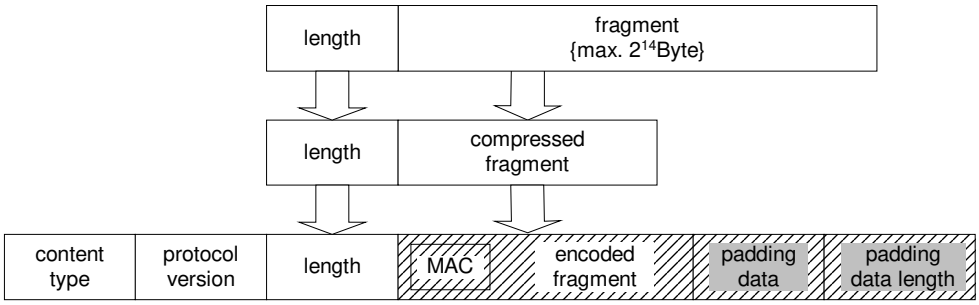

Abbildung 2.6: Paketformat des *Record*-Protokolls

Der Datenteil eines Pakets besteht zunächst aus einem Fragment (`fragment`) und seiner Länge (`length`). Als nächstes werden die Daten komprimiert und die Länge aktualisiert. Im folgenden Schritt werden die Daten verschlüsselt und der `MAC` berechnet. In der Abbildung sind die verschlüsselten Felder schraffiert dargestellt.

Die Felder `padding_data` und `padding_data_length` werden nur benötigt, wenn zur Verschlüsselung eine Blockchiffre verwendet wird. In diesem Fall dienen die `padding_data` zur Verlängerung des Fragments auf ein Vielfaches der Blockgröße der Chiffre. Sollte die Länge des Datenteils bereits ein Vielfaches sein, oder eine Datenstromchiffre verwendet werden, entfallen diese Felder.

Der `MAC` wird vor der Verschlüsselung aus kryptographischen Parametern, dem Fragment, den Fülldaten und der Sequenznummer berechnet.

2.3 Beurteilung der Sicherheit

Die Güte der Sicherheit, die das SSL-Protokoll gewährleisten kann, ergibt sich aus den verwendeten kryptographischen Algorithmen, der Protokollspezifikation und den Implementierungen.

Spezifikation des Protokolls

Schwächen der Spezifikation können erst in späteren Versionen behoben werden und haben starke Auswirkungen auf die Sicherheit. Die Protokollspezifikation sieht Abwehrmaßnahmen vor, die den folgenden Angriffsformen[23] widerstehen können:

⇒ *Dictionary* und *brute-force* Angriffe basieren auf der Durchsuchung des Schlüsselraums, um systematisch den passenden Schlüssel zu ermitteln. Das Protokoll kann mit unterschiedlichen Verschlüsselungsalgorithmen zusammenarbeiten und somit die Größe des verwendeten Schlüsselraums beeinflussen. Bei Auswahl eines starken Algorithmus ist ein erfolgreicher Angriff unwahrscheinlich, kann aber nicht ausgeschlossen werden.

[23] Vgl. [HaAt94, ChBe94, Fech98]

⇒ Bei Wiederholungsangriffen *(replay attack)* werden kopierte Nachrichten zu einem späteren Zeitpunkt erneut versendet. Als Abwehrmaßnahme ist in den Nachrichten die zufällig generierte Sitzungskennung enthalten, so daß die Wiederholung von Nachrichten während anderer Sitzungen scheitert.

⇒ Bei einem Mann-in-der-Mitte-Angriff *(man-in-the-middle attack)* werden Nachrichten abgefangen, gelesen und anschließend weitergeleitet. Die Kombination der symmetrischen und asymmetrischen Verschlüsselung verhindert diesen Angriff, da der symmetrische Schlüssel aus dem verschlüsselten `pre_master_secret` generiert wurde und private Schlüssel nicht übertragen werden, so daß Angreifer sie nicht in ihren Besitz bringen können[24]. Sollten Angreifer die abgefangenen Nachrichten manipulieren, stimmt der `MAC` des *Record*-Protokolls nicht mehr und die Manipulation wird erkannt.

Kryptographische Algorithmen

Unter der Voraussetzung, daß die Algorithmen analytischen Angriffen[25] widerstehen, kann deren Stärke auf Basis der verwendeten Schlüssel beurteilt werden. Die Stärke von Schlüsseln ergibt sich aus ihrer Breite und der Qualität ihrer Generierung, die zufällig erfolgen muß.

Aus der Schlüsselbreite ergibt sich direkt die Widerstandsfähigkeit gegen die Durchsuchung des Schlüsselraumes, da die Mächtigkeit des Schlüsselraumes mit der Schlüsselbreite wächst. Es ist zu beachten, daß die effektive Schlüsselbreite eventuell geringer ist als die logische Schlüsselbreite, da Teile des Schlüssels für Prüfsummen verwendet werden können.[26]

Zur Generierung von Schlüsseln werden echte oder Pseudozufallszahlen verwendet. Sind einige der verwendeten Zufallswerte vorhersagbar, kann der zu durchsuchende Schlüsselraum eingegrenzt werden, und die Sicherheit sinkt.

In der Tabelle 2.6 ist für mehrere Schlüsselbreiten die Zeit angegeben, die der *DES Cracker* der ***E**lectronic **F**rontier **F**oundation (EFF)* zur vollständigen Durchsuchung des Schlüsselraumes braucht. Der *DES Cracker* wurde von der EFF verwendet, um die Verschlüsselung der Nachricht der *DES Challenge II*[27] zu brechen. Die Maschine hat eine Leistungsfähigkeit von $9{,}2*10^{10}$ Verschlüsselungen pro Sekunde. [EFF98]

[24] Eine Ausnahme bildet die Verwendung temporärer öffentlicher Schlüssel, falls der Server kein Zertifikat besitzt. In diesem Fall wird der Server nicht authentifiziert und ein Mann-in-der-Mitte-Angriff kann erfolgreich durchgeführt werden.

[25] Die Analyse von Algorithmen ermöglicht Rückschlüsse auf verwendete Schlüssel.

[26] Der DES-Algorithmus verwendet Schlüssel mit 64 Bits von denen 8 als Paritätsbits genutzt werden. Die effektive Schlüsselbreite beträgt somit 56 Bit.

[27] Die *DES Challenge* ist ein Wettbewerb, der von der Firma RSA ausgeschrieben wird. Der Gewinner ist derjenige, der die Verschlüsselung der Nachricht zuerst bricht.

Schlüsselbreite (Bit)	40	56	64	128
Zeitaufwand (s)	11,95	$7{,}83*10^5$	$2{,}01*10^8$	$3{,}7*10^{27}$

Tabelle 2.6: Zeitaufwand für die vollständige Durchsuchung des Schlüsselraums

Aus der Tabelle ist ersichtlich, daß eine Verschlüsselung mit 40 Bit breiten Schlüsseln Risiken birgt, falls ein Angriff mit entsprechendem Aufwand betrieben wird. Es ist zu beachten, daß die Zeiten sich auf die Durchsuchung des gesamten Schlüsselraums beziehen. Im Durchschnitt wird der Schlüssel in der Hälfte der Zeit gefunden. Bei entsprechender Schlüsselbreite ist es jedoch aufgrund des benötigten Zeitaufwandes unwahrscheinlich, daß die Durchsuchung des Schlüsselraums zum Erfolg führt.

Implementierungen des Protokolls

Das SSL-Protokoll wurde von mehreren Firmen, Organisationen und Privatleuten implementiert[28]. Falls diese Implementierungen Fehler beinhalten, können Auswirkungen auf die Sicherheit des Protokolls nicht ausgeschlossen werden. Als Fehlerquellen kommen zunächst Programmierfehler in Frage, die ein unbeabsichtigtes Verhalten auslösen. Außerdem können sich Fehler aus einer mißverstandenen Protokollspezifikation oder der Verletzung von Rahmenbedingungen ergeben. Eine wichtige Rahmenbedingung ist beispielsweise die zuverlässige Generierung von Zufallszahlen.

2.4 Rechtliche Restriktionen der SSL-Verwendung

Fast alle Staaten beschäftigen sich mit dem Thema der Reglementierung des Vertriebs und der Verwendung von Verschlüsselungstechnologie. Ziel ist es, die Möglichkeit zu erhalten, auf Informationen beliebiger Medien zugreifen zu können. Zur Erreichung des Ziels wurden mehrere Konzepte ausgearbeitet, die auf den folgenden Vorgehensweisen beruhen:

⇒ Der Import, Export und die Verwendung von Verschlüsselungstechnologie wird beschränkt. Als Grundlage der Beschränkung dient die Stärke der verwendeten Verschlüsselungsalgorithmen.

⇒ Die privaten Schlüssel der asymmetrischen Verschlüsselungsalgorithmen müssen bei bestimmten Stellen[29] hinterlegt werden, damit gegebenenfalls Informationen, die mit dem öffentlichen Schlüssel verschlüsselt wurden, wieder entschlüsselt werden können *(key recovery)*.

[28] U.a. von Netscape Communications Corporation, Consensus Development Corporation, Phaos Technology.

[29] In Frage kommen Behörden oder ausgewählte Firmen, wie beispielsweise Zertifikatsautoritäten.

⇒ Die *key escrow* -Technologie sieht vor, Schlüssel mit einer Hintertür zu generieren, damit Informationen entschlüsselt werden können, ohne im Besitz des entsprechenden Schlüssels zu sein.

Das SSL-Protokoll basiert auf Verschlüsselungsalgorithmen und unterliegt somit den obigen Konzepten zur Reglementierung. Allerdings werden die Konzepte sehr unterschiedlich, beziehungsweise überhaupt nicht, umgesetzt. Die folgende Übersicht beschreibt die rechtliche Situation von ausgewählten Gemeinschaften und Staaten und basiert auf den Dokumenten [Koop98, Hof98, GILC98, Hing98, Knap98][30].

USA	
Import/Export	Import ohne Einschränkung; Export von Produkten für den Massenmarkt ist nach einmaliger Prüfung zulässig; für Produkte mit *key recovery*-Verfahren kann eine Genehmigung erteilt werden; Produkte zur Datenverschlüsselung sind bis 40 Bit Schlüsselbreite frei exportierbar, darüber hinaus nur mit Ausnahmegenehmigung
Verwendung	Keine Einschränkungen
Geplante Reglementierungen	Hinterlegung von Schlüsseln *(key recovery)* und Lizenzvergabe für Produkte auf freiwilliger Basis
Europäische Union (EU)	
Import/Export	Import ohne Einschränkung; Export nur mit Genehmigung; ausgenommen sind Produkte für den Massenmarkt und lizenzfreie Software *(public domain)*
Verwendung	Netzbetreiber, die mit Verschlüsselung arbeiten, müssen Gesetzesvertretern Zugriff auf die unverschlüsselten Daten gewähren
Geplante Reglementierungen	Die vorgestellten Konzepte werden geprüft und Vorschläge erarbeitet.
Deutschland	
Import/Export	Entsprechend den Reglementierungen der EU
Verwendung	Keine Einschränkungen
Geplante Reglementierungen	Nur Produkte mit *key escrow*-Verfahren zulässig; Vergabe von Vertriebsgenehmigungen
Frankreich	
Import/Export	Import (außerhalb EU) und Export von Produkten zur Authentifizierung erfordert eine Anmeldung; Produkte zur Datenverschlüsselung erfordern eine Genehmigung
Verwendung	Authentifizierung durch Verschlüsselungsalgorithmen muß angemeldet werden; Datenverschlüsselung erfordert eine Genehmigung (ausgenommen sind Produkte mit *key escrow*–Verfahren); die Verwendung von Produkten mit Schlüsselbreiten von weniger als 40 Bit ist frei
Geplante Reglementierungen	Einführung von ***k**ey **e**scrow **a**gencies (KEA)*; für Benutzer mit Schlüssel einer KEA soll die Verwendung erleichtert werden

[30] Als Grundlage des Kapitels dienen [Koop98] in Version 13.0 und [Hof98] in Version 2.7.

Großbritannien	
Import/Export	Entsprechend den Reglementierungen der EU
Verwendung	Keine Beschränkungen
Geplante Reglementierungen	Hinterlegung von Schlüsseln *(key recovery)* und Lizensierung von Produkten auf freiwilliger Basis

Die beschriebenen Reglementierungen weisen nur die Richtung der Gesetzgebung eines Landes. Es gibt eine Vielzahl von Ausnahmen, die detailliert in den zugrundeliegenden Dokumenten beschrieben werden.

Für Produkte, die das SSL-Protokoll implementieren, ergeben sich aus den beschriebenen Reglementierungen zur Zeit folgende Beschränkungen:

⇒ Produkte aus den USA arbeiten nur mit Schlüsselbreiten bis 40 Bit. Davon sind beispielsweise die Browser und Server von Netscape und Microsoft betroffen. Ausnahmen werden nur für einige Firmen, z.B. Finanzinstitute, gemacht.
⇒ Produkte mit Schlüsselbreiten von mehr als 40 Bit können aus anderen Ländern importiert werden, falls keine Import-/Export-Einschränkungen gelten.
⇒ Die Verwendung der Produkte erfordert, entsprechend der Gesetzgebung der Länder, teilweise eine Anmeldung oder Genehmigung.

Kapitel 3

Verwendung von Zertifikaten im Internet

Der im vorhergehenden Kapitel beschriebene Authentifizierungsmechanismus des SSL-Protokolls basiert auf Zertifikaten und den zugehörigen Zertifikatsautoritäten. In diesem Kapitel wird detailliert auf den Aufbau von Zertifikaten, den Aufgaben von Zertifikatsautoritäten sowie den Problemen der Zertifikatsverwaltung eingegangen. Als Grundlage dienen die Dokumente [FoBa97, HFPS98, HFPS97, RSA93].

3.1 Infrastruktur zur Zertifikatsverwendung

Die Verwendung von Zertifikaten zur Authentifizierung von Kommunikationspartnern im Internet erfordert die Existenz unterstützender Dienste. Die Grundlage der benötigten Infrastruktur bilden Zertifikatsautoritäten *(**c**ertification **a**uthorities, CA's)*, welche die Verwaltungsaufgaben im Zusammenhang mit Zertifikaten übernehmen. Falls die Aufgaben für einzelne Zertifikatsautoritäten zu umfangreich werden, können Aufgabenbereiche an Registrierungsautoritäten abgegeben werden. Außerdem werden gemeinsame Speicher zum Austausch von Zertifikaten und Zertifikatsrücknahmelisten *(**c**ertificate **r**evocation **l**ist, CRL)* benötigt.

Die Abbildung 3.1 gibt einen Vorschlag der Arbeitsgruppe PKIX[31] wieder. Die Zertifikatsbesitzer, Registrierungs- und Zertifikatsautoritäten führen untereinander Verwaltungstransaktionen durch und greifen auf gemeinsame Speicher für Zertifikate und Zertifikatsrücknahmelisten zu. Zertifikatsbesitzer sind nicht nur Endbenutzer, sondern auch Registrierungs- und Zeritifikatsautoritäten, denen wiederum von anderen Zertifikatsautoritäten Zertifikate ausgestellt werden.

[31] ***P**ublic **K**ey **I**nfrastructure (PKIX)* ist eine Arbeitsgruppe der ***I**nternet **E**ngineering **T**ask **F**orce (IETF).*

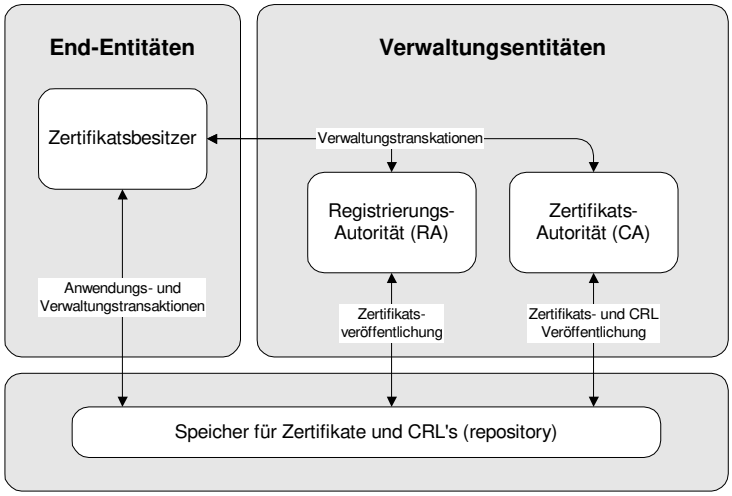

Abbildung 3.1: Zertifikatsinfrastruktur

Für den Entwurf von flexiblen Zertifikatsinfrastrukturen müssen unter anderem folgende Anforderungen berücksichtigt werden:

⇒ Skalierbarkeit, damit die Infrastrukturen in unterschiedlichen Umgebungen eingesetzt werden können und zukünftigen Anforderungen gewachsen sind.
⇒ Unterstützung mehrerer Anwendungen, damit keine proprietären Insellösungen entstehen können.
⇒ Interoperabilität von Infrastrukturen, damit Benutzer nicht auf die Dienste einer Infrastruktur beschränkt sind.
⇒ Die Erfüllung obiger Anforderungen führt zur Definition von Standards, auf denen Infrastrukturen basieren sollten.

3.2 Zertifikate nach X.509-Standard

Ein Zertifikat bindet einen öffentlichen Schlüssel an seinen Besitzer. Die Gültigkeit dieser Bindung wird von einer Zertifikatsautorität gewährleistet, indem sie das Zertifikat digital unterschreibt. Zuvor wird allerdings die Gültigkeit der Daten des Zertifikatsbesitzers, wie zum Beispiel sein Name und öffentlicher Schlüssel, überprüft.

Zertifikate können über unsichere Verbindungen übertragen oder in nicht vertrauenswürdigen Systemen gespeichert werden, da aufgrund der digitalen Unterschrift eine Manipulation erkannt wird.

Der Standard X.509[32] für das Format von Zertifikaten wurde von der ITU-T[33] im Jahr 1988 als Bestandteil der Anforderungen des X.500-Verzeichnisdienstes (vgl.

[32] Der ITU-T Standard X.509 wird von der ISO/IEC als 9594-8 geführt.
[33] Abkürzung für **ITU-T**elecommunication

[ITU97]) festgelegt. Im Juni 1996 wurde die Spezifikation der dritten Version des X.509-Standards (i.f. X509v3) abgeschlossen.

Die folgende Zertifikatsstruktur basiert auf Formatangaben des X.509-Standards und berücksichtigt die Versionen X.509v1 bis X.509v3 [ITU88, ITU95].

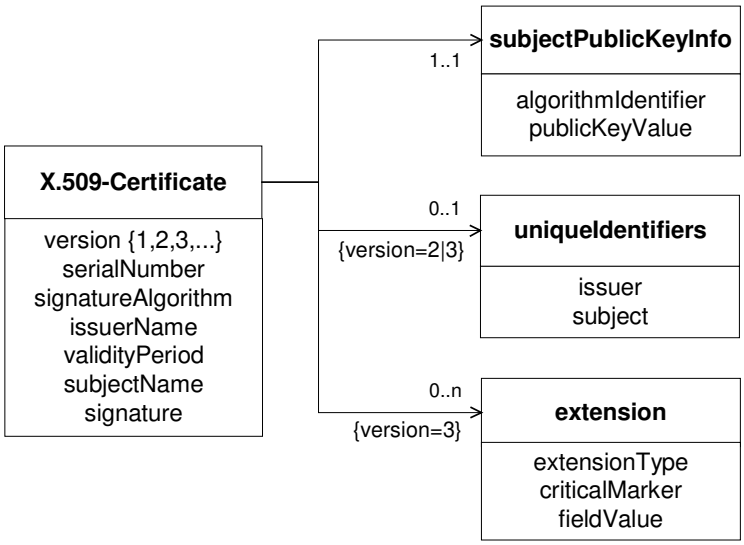

Abbildung 3.2: Struktur von X.509-Zertifikaten

Die Zertifikate der Versionen 1 und 2 bestehen aus einer festen Anzahl von Feldern, während Zertifikate der Version 3 zusätzlich beliebig viele Erweiterungsfelder beinhalten können. Zunächst werden die festen Datenfelder der drei Versionen beschrieben. Anschließend erfolgt die Beschreibung der wichtigsten Typen von Erweiterungsfeldern der dritten Version.

Datenfelder von X.509-Zertifikaten

Der Typ eines Zertifikats ergibt sich aus dem Feld `version`. Zur Identifizierung erhält jedes Zertifikat eine eindeutige Seriennummer (`serialNumber`), die von der Zertifikatsautorität vergeben wird. Die Gültigkeitsdauer (`validity`) besteht aus zwei Werten zur Bestimmung von Beginn und Ende des Gültigkeitszeitraumes.

Das Feld `subjectName` identifiziert die Entität, die mit dem gespeicherten öffentlichen Schlüssel assoziiert ist. Ein Name darf von der ausstellenden Zertifikatsautorität nicht mehrfach vergeben werden, nur der erste Besitzer darf mehrere Zertifikate mit identischen Namen erhalten. Der `issuerName` identifiziert die ausstellende Zertifikatsautorität. In den ersten beiden Versionen dürfen nur X.500-Kennungen zur Identifizierung verwendet werden, während in der dritten Version diese Beschränkung aufgehoben wurde und beliebige Kennungstypen[34] mit eindeutigen Werten zulässig sind.

[34] Gängige Kennungstypen sind beispielsweise Email-Adressen, URL's und IP-Adressen.

Die Unterschrift, mit der die Zertifikatsautorität die Gültigkeit des Zertifikats bestätigt, wird im Feld `signature` gespeichert. Überdies muß der Identifikator des verwendeten Algorithmus (`signatureAlgorithm`) im Zertifikat enthalten sein. Der mit dem Zertifikat assoziierte öffentliche Schlüssel wird zusammen mit dem Identifikator des zugehörigen Algorithmus im Feld `subjectPublicKeyInfo` gespeichert.

In der Version 2 des X.509 Standards wurden die Felder `issuerUniqueIdentifier` und `subjectUniqueIdentifier` hinzugefügt. Wenn die Namen des Ausstellers oder Besitzers des Zertifikats mehrmals verwendet werden[35], ermöglichen diese Felder die eindeutige Zuordnung von Namen zu Zertifikaten.

Erweiterungsfelder der X.509v3-Zertifikate

Die Erweiterungsfelder von X.509v3-Zertifikaten erlauben die Speicherung zusätzlicher Attribute. Anhand des Feldes `extensionType` kann der Inhalt einer Erweiterung identifiziert werden. Typen von Erweiterungsfeldern dürfen frei definiert werden, sollten sich allerdings aus Kompatibilitätsgründen an Standards halten. Ein Zertifikat darf einen Erweiterungstyp nur einmal beinhalten. Falls eine Anwendung einen Erweiterungstyp nicht unterstützt und das `criticalMarker`-Feld der Erweiterung gesetzt ist, muß das Zertifikat zurückgewiesen werden. Ist das `criticalMarker`-Feld nicht gesetzt, darf die Erweiterung ignoriert und die Validierung des Zertifikats fortgesetzt werden. Der Inhalt einer Erweiterung (`fieldValue`) kann eine Datenstruktur sein, deren Semantik sich aus der Typdefinition ergibt.

Von der ISO/IEC und ITU wurden Erweiterungstypen in den X.509v3-Standard aufgenommen, die sich in vier Gruppen unterteilen lassen. Im folgenden wird nur eine Auswahl von Erweiterungstypen beschrieben, die in Kapitel 3.3 verwendet werden. Eine vollständige Auflistung und detaillierte Beschreibung der Felder kann [ITU95, FoBa97] entnommen werden.

⇒ **Informationen zu Schlüssel und Zertifizierungsrichtlinien**
Die Erweiterung `keyUsage` beschreibt den Verwendungszweck des im Zertifikat enthaltenen Schlüssels. Dadurch kann die Anwendung des Schlüssels auf bestimmte Bereiche (z.B. Datenverschlüsselung oder digitale Unterschriften) eingeschränkt werden. In den `certificatePolicies` werden Richtlinien beschrieben, unter denen das Zertifikat ausgestellt wurde und für welche Zwecke es verwendet werden darf. Zertifizierungsrichtlinien werden von Zertifikatsautoritäten aufgestellt und sind in Kapitel 3.3.2 beschrieben.

⇒ **Beschränkungen des Zertifizierungspfades**
In den `basicConstraints` wird angegeben, ob der Besitzer des Zertifikats eine Zertifikatsautorität oder ein Endbenutzer ist. Handelt es sich um eine Zertifikatsautorität, wird die maximal zulässige Länge des nachfolgenden Zertifizierungspfades[36] angegeben. Die `nameConstraints`[37] bestimmen einen Namensraum, inner-

[35] Dieser Fall tritt ein, wenn ein Zertifikat verfällt und das neue unter demselben Namen ausgestellt wird.
[36] Ein Zertifizierungspfad ergibt sich, falls an Zertifikate jeweils das Zertifikat der ausstellenden Zertifikatsautorität angehängt wird. Vgl. Kapitel 3.3.1.

halb dessen alle Namen der Besitzer von Zertifikaten des nachfolgenden Zertifizierungspfades liegen müssen. Zur Eingrenzung des Namensraums können Teilbäume ein- oder ausgeschlossen werden. Die `policyConstraints` definieren Beschränkungen, die zur Beeinflussung der Pfadvalidierung verwendet werden.

⇒ **Erweiterungen für Zertifikatsrücknahmelisten**
Die Erweiterung `CRLDistributionPoint` enthält Informationen zu Verteilknoten von Zertifikatsrücknahmelisten (vgl. Kapitel 3.3.3).

⇒ **Attribute des Zertifikatsbesitzers und -ausstellers**
Die Felder `subjectAlternativeName` und `issuerAlternativeName` gestatten die Angabe alternativer Namen des Zertifikatsbesitzers und -ausstellers. Es können beliebige Formate verwendet werden, und der Name muß nicht eindeutig sein.

3.3 Zertifikatsautoritäten

Eine Zertifikatsautorität *(**c**ertificate **a**uthority, CA)* ist eine vertrauenswürdige Instanz. Sie vergibt Zertifikate zur Identifizierung von Individuen, Systemen und anderen Entitäten, die miteinander über ein Netzwerk kommunizieren.

Die Zertifikatsautorität bindet in einem Zertifikat die Identität des Besitzers an seinen öffentlichen Schlüssel und garantiert durch ihre digitale Unterschrift die Vertrauenswürdigkeit des Zertifikats. Kommunikationsteilnehmer benötigen die öffentlichen Schlüssel von Zertifikatsautoritäten zur Verifizierung der Unterschriften der Zertifikate.

In Abbildung 3.3 hat die Zertifikatsautorität CA1 Zertifikate an die Benutzer a und b vergeben. Wenn Benutzer b mit Benutzer a kommunizieren will, überträgt er zur Authentifizierung sein Zertifikat. Der Benutzer a verwendet den öffentlichen Schlüssel der ausstellenden Zertifikatsautorität CA1 zur Verifizierung der Unterschrift des Zertifikats.

Sind die Kommunikationsteilnehmer, wie im obigen Beispiel, bereits im Besitz der benötigten öffentlichen Schlüssel, können sie die Verifizierung der Unterschriften direkt vornehmen. Falls die Zertifikate jedoch von unbekannten und deshalb zunächst nicht vertrauenswürdigen Zertifikatsautoritäten unterschrieben wurden, werden deren Zertifikate zur Entnahme der öffentlichen Schlüssel benötigt. Damit die unbekannten Zertifikatsautoritäten als vertrauenswürdig angesehen werden können, müssen Beziehungen zu bekannten und vertrauenswürdigen Zertifikatsautoritäten existieren.

[37] Diese Erweiterung wird nur in Zertifikaten verwendet, die an Zertifikatsautoritäten vergeben wurden.

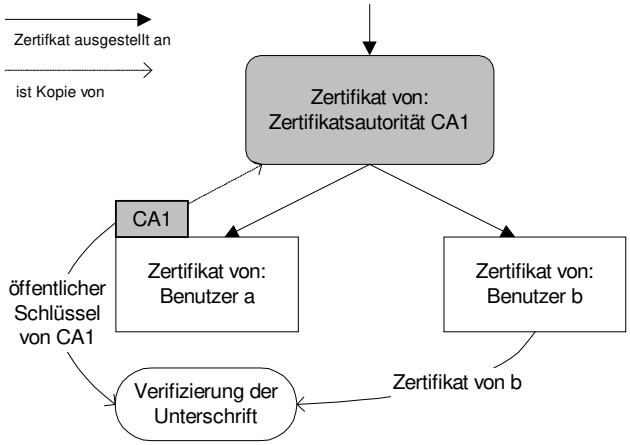

Abbildung 3.3: Verifizierung der Unterschrift eines Zertifikats

3.3.1 Beziehungen zwischen Zertifikatsautoritäten

Beziehungen zwischen Zertifikatsautoritäten ermöglichen Benutzern auf einem vertrauenswürdigem Weg auf Zertifikate unbekannter Benutzer oder Zertifikatsautoritäten zuzugreifen. Aus den Beziehungen ergeben sich Zertifizierungspfade *(certification path)*, die aus untereinander vergebenen Zertifikaten von Zertifikatsautoritäten bestehen. Zertifikatsautoritäten, die ihr Zertifikat selbst unterschreiben, werden als Wurzelzertifikatsautorität *(Root-CA)* bezeichnet.

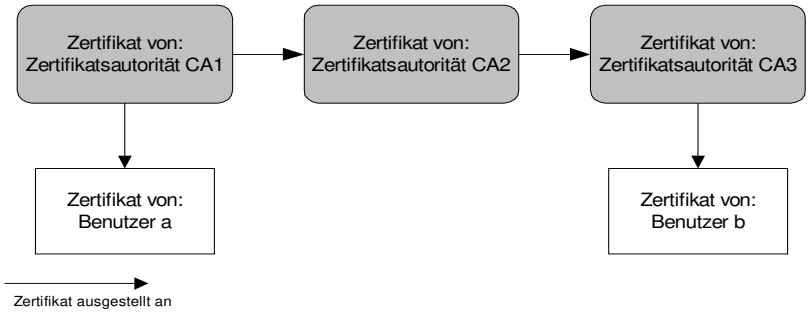

Abbildung 3.4: Beispiel eines Zertifizierungspfades

In Abbildung 3.4 will Benutzer a mit Benutzer b kommunizieren und benötigt zur Validierung des Zertifikats von b den öffentlichen Schlüssel der ausstellenden Zertifikatsautorität CA3. Diese Zertifikatsautorität ist a allerdings unbekannt. Er besitzt nur den öffentlichen Schlüssel der Zertifikatsautorität CA1. Um auf vertrauenswürdigem Weg in den Besitz des Zertifikats von CA3 zu kommen, muß eine Zertifizierungsspfad von CA1 nach CA3 gefunden werden. Im Beispiel führt dieser Pfad über die Zertifikatsautorität CA2, da ihr Zertifikat von CA1 ausgestellt wurde, und sie wiederum das Zertifikat von CA3 ausgestellt hat.

Allgemeine hierarchische Struktur

Die Existenz eines Zertifizierungspfades ist somit die Grundlage einer Zertifikatsvalidierung. Zur Einschränkung des Suchaufwandes können Zertifikatsautoritäten hierarchisch angeordnet werden. Der X.509-Standard legt keine Struktur zwingend fest, beschreibt allerdings folgende Baumstruktur mit Zwischenkanten:

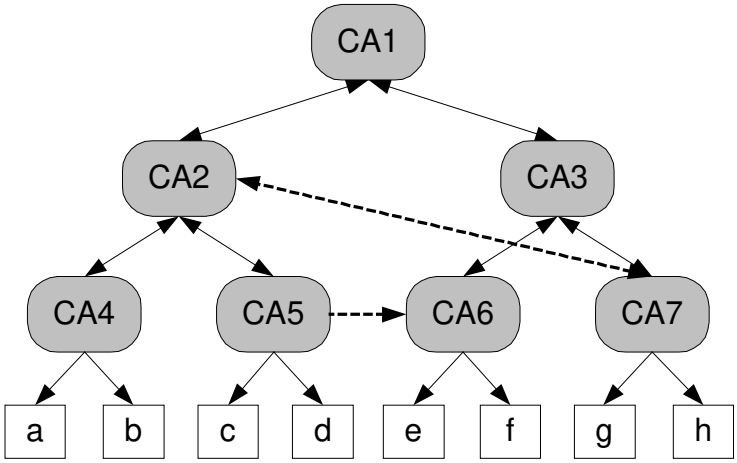

Abbildung 3.5: Allgemeine hierarchische Struktur mit Zwischenkanten

Die Knoten des Baums sind Zertifikatsautoritäten mit gegenseitig ausgestellten Zertifikaten. Endbenutzer werden einer Zertifikatsautorität zugeordnet und stellen somit die Blätter des Baums dar. Das Problem der Baumstruktur sind die langen Zertifizierungspfade, die entstehen können, wenn Knoten höherer Ebenen traversiert werden müssen. Zur Lösung wurden Zwischenkanten eingeführt, die Zertifikate zwischen beliebigen Zertifkatsautoriäten der Hierarchie repräsentieren[38]. Zertifizierungspfade können durch Einfügung geeigneter Zwischenkanten stark verkürzt werden. Zum Beispiel führt der Pfad von d nach e ohne Zwischenkanten über 5 Zertifikatsautoritäten (CA5, CA2, CA1, CA3, CA6). Nach Einführung der Zwischenkante von CA5 nach CA6 nur noch über diese beiden.

Struktur mit bedingter Zertifikatsvergabe

Bei der Suche und Validierung von Zertifizierungspfaden muß jedoch berücksichtigt werden, daß ein validierter Pfad einem Vertrauensverhältnis entspricht. Uneingeschränktes Vertrauen werden die Endbenutzer allerdings nur ausgewählten Zertifikatsautoritäten entgegenbringen. Je länger der Zertifizierungspfad ist, desto weniger vertrauenswürdig wird er für den Endbenutzer.

Die Abbildung 3.6 zeigt eine Struktur mit Zertifikatsautoritäten, deren Zertifikate an Bedingungen geknüpft sind *(progressive-constraint trust model)* [FoBa97]:

[38] Solche Zertifikate werden auch als *cross certificate* bezeichnet.

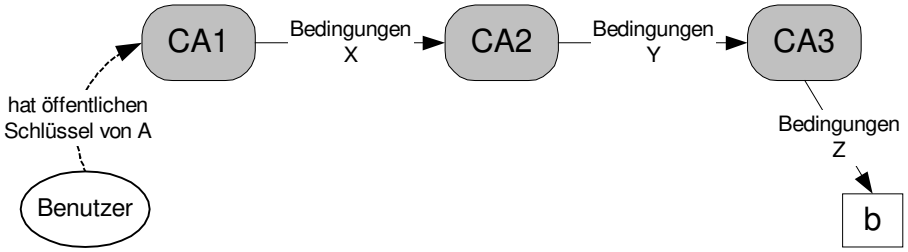

Abbildung 3.6: Struktur mit zunehmenden Einschränkungen

Der Benutzer will mit b kommunizieren und hat den öffentlichen Schlüssel der Zertifikatsautorität CA1, welche somit zum Ausgangspunkt des Zertifizierungspfades wird. Der Zertifizierungspfad verläuft über die Zertifikatsautoritäten CA2 und CA3 und endet schließlich beim Endbenutzer b. Die Zertifikate des Pfades wurden allerdings an Bedingungen X, Y und Z geknüpft, die zur erfolgreichen Validierung erfüllt sein müssen. Die Bedingungen schränken somit die Vertrauenswürdigkeit des Pfades zunehmend ein.

Zur Spezifikation von Bedingungen können beispielsweise die Erweiterungsfelder `certificatePolicies`, `policyConstraints` oder `nameConstraints` verwendet werden.

3.3.2 Zertifizierungsrichtlinien

Bevor ein Zertifikat ausgestellt wird, überprüft die Zertifikatsautorität die Daten des Antragstellers auf ihre Korrektheit. Je umfangreicher diese Prüfung durchgeführt wird, desto vertrauenswürdiger ist die Zertifikatsautorität und das ausgestellte Zertifikat.

Die Verfahren zur Prüfung und Ausstellung von Zertifikaten werden mit Beschreibungen der anderen Verwaltungsaufgaben von Zertifikatsautoritäten zu Zertifizierungsrichtlinien zusammengefaßt. Referenzen auf solche Richtlinien können in den Erweiterungsfeldern von Zertifikaten gespeichert werden und bieten Benutzern die Möglichkeit, Bedingungen der Pfadvalidierung an sie zu knüpfen.

Der Inhalt von Zertifizierungsrichtlinien wurde noch nicht in Standards festgelegt. Im folgenden werden einige Bereiche beschrieben, die bei der Definition von Richtlinien berücksichtigt werden sollten [FoBa97]:

⇒ **Beschränkungen der Gemeinschaft und Anwendbarkeit**
Die Vergabe von Zertifikaten kann auf geschlossene Gemeinschaften beschränkt werden. Außerdem kann die Anwendbarkeit von Zertifikaten auf bestimmte Bereiche eingeschränkt werden.

⇒ **Identifizierung und Authentifizierung**
Zur Identifizierung und Authentifizierung von Zertifikatsantragstellern können unterschiedliche Aufwände betrieben werden.

⇒ **Schlüsselverwaltung**
Die Schlüsselverwaltung erfordert Maßnahmen zum Schutz von privaten Schlüsseln. Es können auch Richtlinien verfaßt werden, die Zertifikatsbesitzern vorschreiben, wie sie ihre Schlüssel zu schützen haben.

⇒ **Operationale Richtlinien**
Die Durchführung von Verwaltungsaufgaben kann in Richtlinien festgelegt werden. Ein Aspekt ist die Häufigkeit der Veröffentlichung von Zertifikatsrücknahmelisten.

⇒ **Lokale Sicherheit**
Die Qualität der Sicherheit einer Zertifikatsautorität umfaßt alle Bereiche von der EDV-Umgebung bis hin zur Personalauswahl.

3.3.3 Verwaltungsaufgaben von Zertifikatsautoritäten

Zertifikatsautoritäten müssen alle Aufgaben übernehmen, die im Zusammenhang mit der Verwaltung von Zertifikaten und Schlüsseln[39] stehen. Aus dem Lebenszyklus von Zertifikaten in Abbildung 3.7 ergeben sich die grundlegenden Aktivitäten, die von Zertifikatsautoritäten durchgeführt werden müssen [FoBa97].

Die Verwaltungsaufgaben werden zu Ausgabe- und Rücknahmeaktivitäten zusammengefaßt. Die Aktivitäten bei der Verwendung von Zertifikaten werden zu einem Validierungsprozeß zusammengefaßt, der in Kapitel 3.4 gesondert beschrieben wird.

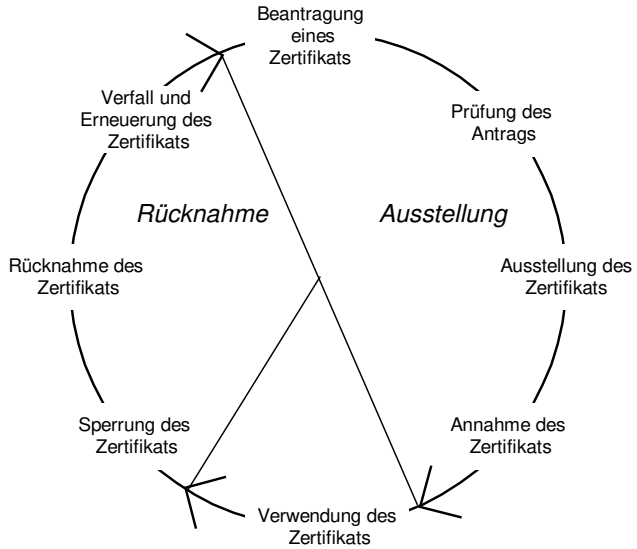

Abbildung 3.7: Lebenszyklus von Zertifikaten

[39] Ausgenommen sind private Schlüssel, falls sie nur vom Benutzer verwaltet werden.

Ausgabeaktivitäten

Dieses Kapitel beschreibt alle Aktivitäten, die bis zur Verwendung eines Zertifikats durchgeführt werden.

Beantragung eines Zertifikats

Ein Benutzer muß sich bei einer Zertifikatsautorität registrieren lassen, bevor ihm ein Zertifikat ausgestellt wird. Dazu müssen der Zertifikatsautorität Daten zur Identitätsprüfung und Zertifikatsdaten übermittelt werden. Zur Identitätsprüfung werden zum Beispiel die Adresse, Ausweisnummern und Fotos verwendet. Zu den Zertifikatsdaten gehört der öffentliche Schlüssel, wenn die Generierung des Schlüsselpaars vom Benutzer übernommen wird. Sie kann auch von der Zertifikatsautorität durchgeführt werden. Dann muß allerdings der private Schlüssel dem Benutzer sicher übermittelt werden.

Prüfung des Antrags

Die Zertifikatsautorität muß die Gültigkeit der Daten des Benutzers überprüfen. Die Art und Weise der Überprüfung wird in den Zertifizierungsrichtlinien einer Zertifikatsautorität festgeschrieben. Grundsätzlich wird zwischen Überprüfungen mit Anwesenheitspflicht des Benutzers und Überprüfungen auf der Basis von Dokumenten unterschieden.

Ausstellung und Annahme des Zertifikats

Die Daten des Benutzers werden in einem Zertifikat gespeichert, welches mit dem privaten Schlüssel der Zertifikatsautorität unterschrieben wird. Anschließend wird das Zertifikat dem Benutzer übermittelt. Ferner kann die Zertifikatsautorität die Veröffentlichung des Zertifikats übernehmen, indem sie es an geeignete Dienste (z.B. X.500-Verzeichnisdienste oder dedizierte Zertifikatsserver) weiterleitet. Eine Bestätigung der Annahme des Zertifikats durch den Antragsteller erfolgt nur auf Anforderung der Zertifikatsautorität.

Rücknahmeaktivitäten

Rücknahmeaktivitäten dienen der Bearbeitung von Ereignissen, welche die Verwendung eines Zertifikats einschränken oder verhindern.

Sperrung und Rücknahme des Zertifikats

Normalerweise sind Zertifikate während ihrer gesamten Lebensdauer gültig. Es kann jedoch vorkommen, daß Zertifikate vor Ablauf ihrer Gültigkeitsdauer nicht mehr akzeptiert werden dürfen. In diesen Fällen können Zertifikate gesperrt[40] oder zurückgenommen werden.

Eine Sperre verhindert temporär die Verwendung eines Zertifikats und wird beispielsweise verhängt, falls Unregelmäßigeiten auftreten[41]. Mögliche Gründe für die

[40] Die Sperrung von Zertifikaten ist eine Erweiterung der ANSI X9 und nicht Bestandteil des X.509-Standards.

[41] z.B. bei finanziellen Transaktionen

endgültige Zertifikatsrücknahme sind Namensänderungen, Wechsel der Zertifikatsautorität oder Erstellung eines neuen Schlüsselpaars.

Im X.509-Standard ist zur Durchführung von Zertifikatssperrungen und -rücknahmen die regelmäßige Veröffentlichung einer Zertifikatsrücknahmeliste (CRL) durch Zertifikatsautoritäten vorgesehen. Bei der Validierung eines Zertifikats muß geprüft werden, ob es in den aktuellen CRL‘s eingetragen ist. Eine CRL kann öffentlich verteilt werden, da sie einen Zeitstempel enthält und von der ausgebenden Zertifikatsautorität digital unterschrieben wird.

Die Struktur einer CRL besteht wie beim Zertifikat aus festen Feldern und einer variablen Zahl von Erweiterungsfeldern. Für jedes zurückgenommene Zertifikat werden die Seriennummer, das Rücknahmedatum und eventuell Zusatzinformationen gespeichert. Bei gesperrten Zertifikaten enthalten die Zusatzinformationen einen entsprechenden Vermerk.

Verfall und Erneuerung des Zertifikats

Jedes Zertifikat hat eine beschränkte Gültigkeitsdauer und muß deshalb, falls der Besitzer dies wünscht, periodisch erneuert werden. Überdies können Schlüssel eine beschränkte Gültigkeitsdauer besitzen, so daß Zertifikate erneuert werden müssen, wenn enthaltene Schlüssel verfallen.

3.4 Verwendung von Zertifikaten

Die Verwendung von Zertifikaten erfordert den Nachweis der Vertrauenswürdigkeit, welcher durch die Validierung von Zertifizierungspfaden erbracht wird. Verläuft die Validierung eines Pfades erfolgreich, darf das zugrundeliegende Zertifikat akzeptiert werden, anderenfalls muß es zurückgewiesen werden.

Zur Validierung eines Pfades muß jedes Zertifikat des Pfades validiert werden, was die Durchführung folgender Aktivitäten erfordert:

1. Prüfung der digitalen Unterschriften der Zertifikate.

2. Vergleich der Namen in den Zertifikaten. Der Besitzer eines Zertifikats muß zugleich der Aussteller des im Pfad folgenden Zertifikats sein. Davon ausgenommen ist das Endbenutzerzertifikat, da es den Abschluß des Pfades bildet.

3. Prüfung des Gültigkeitszeitraumes der Zertifikate.

4. Prüfung, ob Zertifikate in CRL’s eingetragen sind.

5. Prüfung, ob geforderte Zertifizierungsrichtlinien von allen Zertifikaten eingehalten werden.

6. Prüfung, ob die Bedingungen der Erweiterungsfelder `basic constraints` und `name constraints` erfüllt werden.

Im X.509-Standard [ITU88] wird ein Algorithmus definiert, der zur sicheren Validierung von Zertifizierungspfaden eingesetzt werden kann und im wesentlichen auf der Durchführung obiger Aktivitäten basiert.

3.5 Bewertung des Zertifikatskonzeptes

Das Zertifikatskonzept beruht auf der Vertrauenswürdigkeit von Zertifikatsautoritäten und kann deshalb nur Sicherheit gewährleisten, wenn die Benutzer allen beteiligten Zertifikatsautoritäten vertrauen. Dies stellt eine Schwäche des Konzeptes dar, weil Benutzer zu wenig Informationen erhalten, um eine fundierte Entscheidung für oder gegen eine Zertifikatsautorität treffen zu können.

Die Validierung von Zertifizierungspfaden ist für Benutzer nicht nachvollziehbar, da zahlreiche Bedingungen und Einschränkungen geprüft werden. Grundlegende Informationen, wie beispielsweise die Zertifizierungsrichtlinien von Zertifikatsautoritäten, werden den Benutzern nicht angezeigt, obwohl sich aus ihnen die Art und Weise der Überprüfung der im Zertifikat enthaltenen Daten ergibt. Ein weiterer Schwachpunkt ist die periodische Veröffentlichung von CRL's. Denn zurückgezogene Zertifikate können in Abhängigkeit vom Veröffentlichungsintervall der CRL noch verwendet werden.

In der Praxis hat sich gezeigt, daß Zertifikatsautoritäten häufig Zertifikate mit einer unrealistischen Gültigkeitsdauer[42] besitzen. Zertifikate mit hoher Gültigkeitsdauer sind lohnende Ziele für Angriffe.

Damit Benutzer entscheiden können, ob sie einer Zertifikatsautorität vertrauen, benötigen sie umfangreiche Informationen, die vom Validierungsprozeß zur Verfügung gestellt werden müssen. Weiterhin muß für Benutzer nachvollziehbar sein, welche Informationen eines Zertifikats tatsächlich überprüft wurden. Die zuvor beschriebenen Schwächen werden von den aktuellen Standards nicht ausreichend berücksichtigt und schränken den Nutzen von Zertifikaten im Internet stark ein. Innerhalb von geschlossenen Benutzergruppen und festen Zertifizierungspfaden verlieren diese Nachteile jedoch an Bedeutung.

[42] Zum Beispiel ist das *Thawte Server CA*-Zertifikat bis zum Jahr 2021 gültig.

Kapitel 4

Modifikation des Tycoon-2-Webservers

Dieses Kapitel behandelt die Einbindung des vorgestellten SSL-Protokolls in den Webserver des Tycoon-2-Systems. Zunächst wird der Tycoon-2-Webserver beschrieben und auf seine Struktur und Funktionsweise eingegangen. Zur Erweiterung des Webservers wird die lizenzfreie C-Bibliothek SSLeay[43] verwendet. Die Beschreibung des Aufbaus und der Anwendung der Bibliotheksroutinen bildet die Grundlage der Modifikation des Webservers. Das Kapitel schließt mit dem Entwurf des SSL-Webservers.

Die Beschreibungen der Struktur und Verhaltensweise des Webservers ergeben sich aus der Analyse der Implementierung. Zur Modellierung wird die ***U**nified **M**odeling **L**anguage (UML)* verwendet.

4.1 Der Tycoon-2-Webserver

Ein Bestandteil der Tycoon-2-Klassenbibliothek ist ein Webserver, der auf dem *Hypertext Transfer Protocol* aufsetzt und in der Lage ist, STML-Dokumente zu verarbeiten. Im folgenden wird die Struktur des Webservers erläutert und seine Funktionsweise anhand der Bearbeitung einer STML-Anfrage veranschaulicht. Die Konfiguration und Initialisierung wird im Zusammenhang mit dem SSL-Webserver in Kapitel 4.3.3 beschrieben.

4.1.1 Struktur des Webservers

Der Webserver setzt sich aus Klassen zusammen, die zur Kommunikation über Sokkets, Umsetzung des HTTP, Bearbeitung von STML-Dokumenten und Konfiguration benötigt werden. Auf die Darstellung der gesamten Struktur des Servers wird verzichtet, da zur Erweiterung nur einige grundlegende Klassen modifiziert werden müssen.

[43] In dieser Arbeit wird die SSLeay-Version 0.8.1 vom 19.7.1997 verwendet.

Das Diagramm in Abbildung 4.1 zeigt Klassen, Variablen und Methoden, die während der SSL-Erweiterung modifiziert werden oder zum Verständnis der Funktionalität des Servers beitragen.

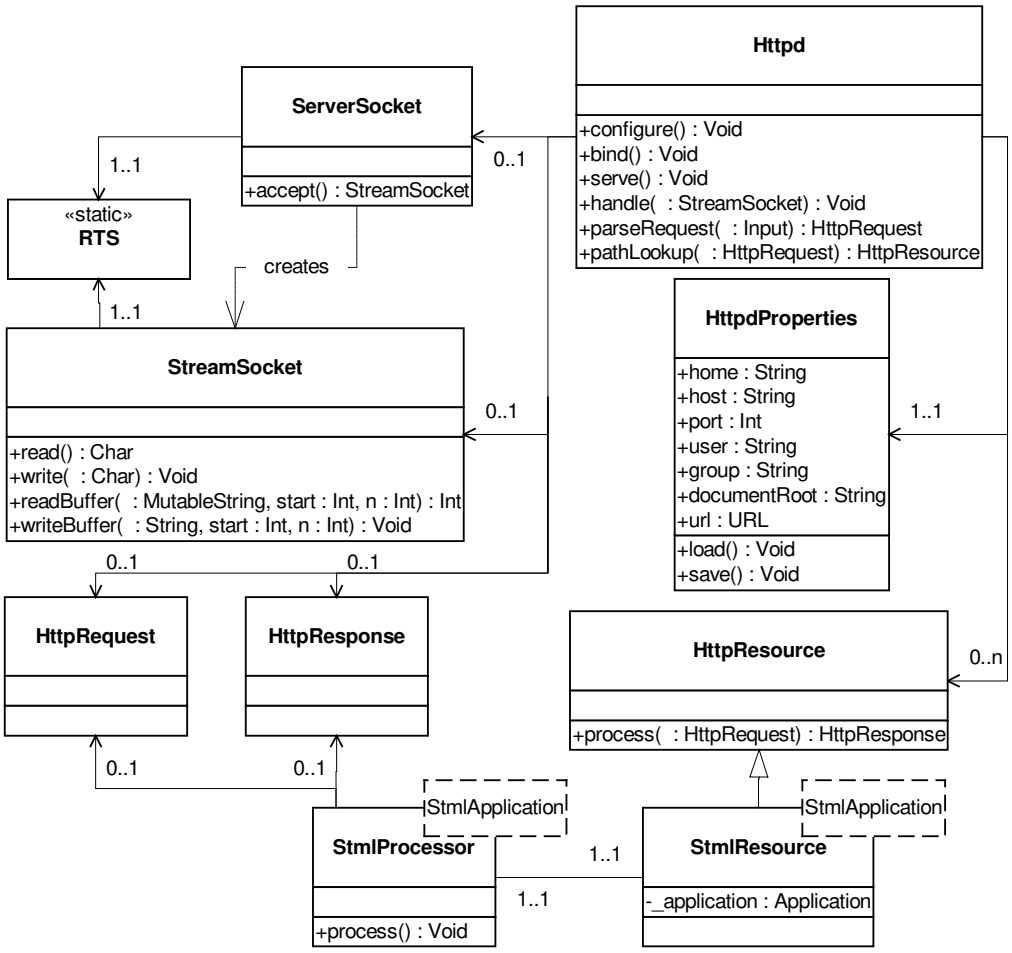

Abbildung 4.1: Struktur des Tycoon-2-Webservers

Die Hauptklasse des Servers ist `Httpd`. Sie enthält Methoden zur Initialisierung des Servers sowie zur Annahme und Bearbeitung von Anfragen. Die Funktionalität zur Verwaltung der Sockets stellt eine C-Bibliothek zur Verfügung, welche über die Klasse `RTS` ***(runtime support)*** angesprochen werden kann. Innerhalb von Tycoon-2 werden Sockets von den Klassen `ServerSocket` und `StreamSocket` repräsentiert, die beide auf `RTS` zugreifen. Nachdem der `ServerSocket` eine Anfrage entgegengenommen hat, erzeugt er eine Instanz der Klasse `StreamSocket`, welche den Datenaustausch übernimmt.

Die Daten der HTTP-Anfrage und HTTP-Antwort werden in den Klassen `HttpRequest` und `HttpResponse` gespeichert. Zur Bearbeitung von Anfragen und Generierung von Antworten werden Ressourcen- und Prozessorklassen benötigt (vgl. Kapitel 1.4.3).In

der Abbildung sind die beiden parametrisierten Klassen StmlResource und StmlProcessor dargestellt, die zur Bearbeitung von STML-Dokumenten verwendet werden und sich gegenseitig referenzieren. Ihre Parametrisierung erfolgt mit der zugehörigen STML-Anwendung (StmlApplication). Die Prozessorklasse greift zur Bearbeitung des STML-Dokuments auf die Klassen HttpRequest und HttpResponse zu. Der Server muß für jede zugeordnete STML-Anwendung eine Referenz auf die entsprechende Ressourcenklasse speichern. Zur Bereitstellung anderer Funktionalität können Klassen basierend auf der abstrakten Superklasse HttpResource definiert werden. Zur Konfiguration des Servers werden in einer Textdatei Parameter gespeichert, die von der Klasse HttpdProperties ausgelesen und dem Server übergeben werden.

4.1.2 Funktionsweise des Webservers

Die Interaktionen der Klassen während der Bearbeitung einer HTTP-Anfrage zeigt das Sequenzdiagramm in Abbildung 4.2. Der Schwerpunkt des Diagrammes liegt auf dem Verbindungsaufbau und dem Datenaustausch. Die Auswertung eines STML-Dokuments und die Generierung des HTML-Dokuments sind von der verwendeten Verbindung unabhängig und werden vereinfacht dargestellt.

Das Diagramm zeigt eine bereits erzeugte Instanz der Klasse Httpd, die einem initialisierten und konfigurierten Webserver entspricht, der jetzt an einen Socket gebunden werden kann. Die Ressourcenklassen der STML-Anwendungen (StmlResource) wurden während der Konfiguration erzeugt und beim Webserver registriert. Die Instanz der Klasse RTS wird vom Tycoon-2-System verwaltet und braucht nicht vom Webserver erzeugt zu werden.

Zunächst wird die Methode bind aufgerufen, welche einen ServerSocket erzeugt, der an die Adresse und den Port der HttpdProperties gebunden wird. Mit serve wird der Server aktiviert und wartet nach Aufruf der accept-Methode des ServerSocket auf eingehende Anfragen.

Trifft eine Anfrage ein, erzeugt der ServerSocket einen StreamSocket und gibt diesen an den Httpd zurück. Zur Bearbeitung der Anfrage wird die Methode handle mit dem StreamSocket als Parameter aufgerufen. Daraufhin liest die Methode bufferedInput des StreamSocket die Daten der Anfrage aus, indem sie auf tysocket_read der RTS-Instanz zugreift. Eine Instanz der Klasse BufferedInput speichert die Anfragedaten und wird an den Httpd zurückgegeben. Zur Zwischenspeicherung der Antwort erzeugt die Methode bufferedOutput eine Instanz der Klasse BufferedOutput und übergibt sie dem Httpd.

Die Analyse der Anfrage wird von der Methode parseRequest durchgeführt. Sie übernimmt den BufferedInput und initialisiert mit den enthaltenen Daten einen HttpRequest. Der HttpRequest wird vom Httpd in der Methode pathLookup zur Bestimmung der zugehörigen StmlResource verwendet. Die process-Methode der StmlResource erzeugt eine HttpResponse-Instanz, welche das generierte HTML-Dokument aufnimmt.

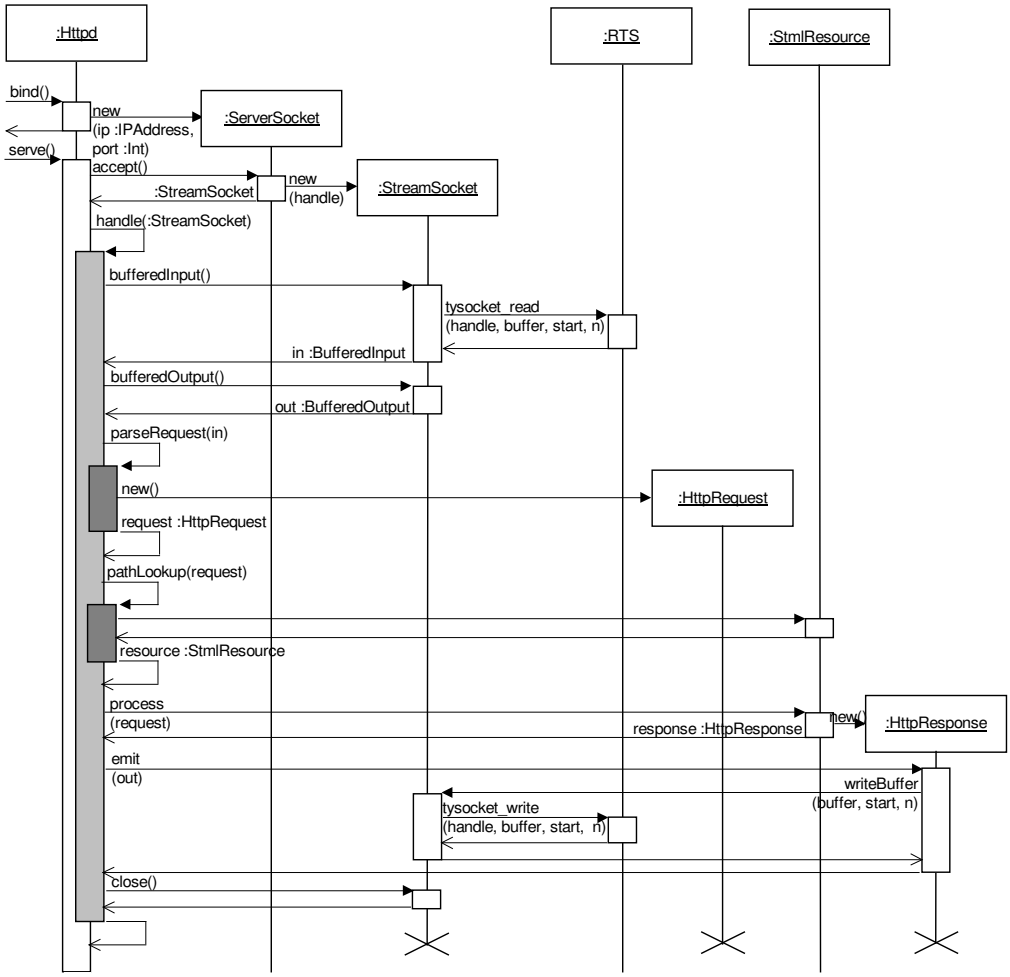

Abbildung 4.2: Bearbeitung einer STML-Anfrage

Zur Übertragung der Antwort wird die Methode `emit` der `HttpResponse` ausgeführt und `BufferedOutput` als Parameter übergeben. Über den `BufferedOutput` wird die Methode `writeBuffer` des `StreamSocket` aufgerufen, die wiederum auf `tysocket_write` der `RTS`-Instanz zugreift.

Die Ausführung der `handle`-Methode endet mit dem Aufruf der `close`-Methode des `StreamSocket`. Die Instanzen der Klassen `StreamSocket`, `HttpRequest` und `HttpResponse` sind an lokale Variablen der `handle`-Methode gebunden und werden nach ihrer Beendigung zerstört. Die Ausführung der `serve`-Methode wird mit einem erneuten Aufruf von `accept` des `ServerSockets` fortgeführt.

Die Struktur und das Verhalten des Webservers aus den Abbildungen 4.1 und 4.2 wird nachfolgend dahingehend modifiziert, daß der Webserver nach Einbindung einer SSL-Bibliothek das SSL-Protokoll unterstützt und Tycoon-2-Anwendungen auf Klientenzertifikate zugreifen können.

4.2 SSLeay – Eine SSL-Implementierung

Die C-Bibliothek SSLeay ist eine Implementierung des *Secure Socket Layer*-Protokolls in der Version 3. Als Grundlage von SSLeay diente die Protokollspezifikation von Netscape [Nets96a] und bereits existierende kryptographische Bibliotheken. SSLeay ist zur Erweiterung diverser Internetprotokolle (z.B. Telnet, HTTP) um SSL-Funktionalität verwendet worden. Die Beschreibung der Bibliothek basiert auf der *Programmers Reference* [HuYo97] und Kommentaren zu den Quelltexten der verwendeten Version 0.8.1.

Aufbau der SSLeay-Bibliothek

Die SSLeay-Bibliothek besteht aus mehreren Modulen, die sich in drei Schichten anordnen lassen:

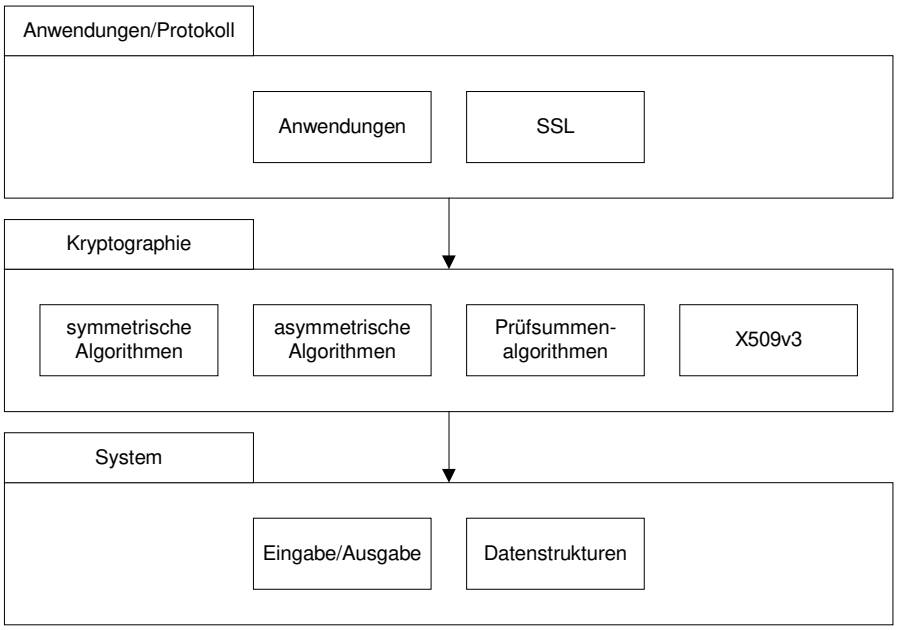

Abbildung 4.3: Aufbau der SSLeay-Bibliothek

Die unterste Schicht der Bibliothek bildet die Systemschicht. Sie enthält grundlegende Datenstrukturen und Eingabe-/ Ausgaberoutinen.

Kryptographische Routinen sind in der mittleren Schicht zusammengefaßt. Sie enthält die symmetrischen Algorithmen DES, RC4 und IDEA. Zur asymmetrischen Verschlüsselung sind die Algorithmen RSA, DSS und Diffie-Hellman vorgesehen. Die Berechnung von Prüfsummen erfolgt mit MD5, MD2 und SHA. Außerdem beinhaltet die Schicht alle Routinen, die zur Verwaltung von X509v3-Zertifikaten benötigt werden.

Die Anwendungs- und Protokollschicht besteht aus der Implementierung des SSL-Protokolls und Anwendungen, die direkten Zugriff auf alle Routinen der Kryptographieschicht ermöglichen.
Die beschriebenen Module werden in den beiden C-Bibliotheken `cryptolib` und `ssllib` zusammengefaßt, welche importiert werden können.

Anwendung der SSLeay-Bibliothek

Das SSL-Protokoll ist als zusätzliche Schicht zwischen der Transportschicht und der Anwendungsschicht realisiert. Diese Positionierung erlaubt die Verwendung des Protokolls mit unterschiedlichen Anwendungen und Netzwerkprotokollen. Die Einbindung des SSL-Protokolls in Anwendungen erfordert somit nur die Modifikation der Schnittstellenaufrufe zwischen der Anwendungs- und Transportschicht.[44]

Die folgende Abbildung illustriert die notwendigen Modifikationen[45] am Quelltext eines Servers am Beispiel eines Verbindungsaufbaus:

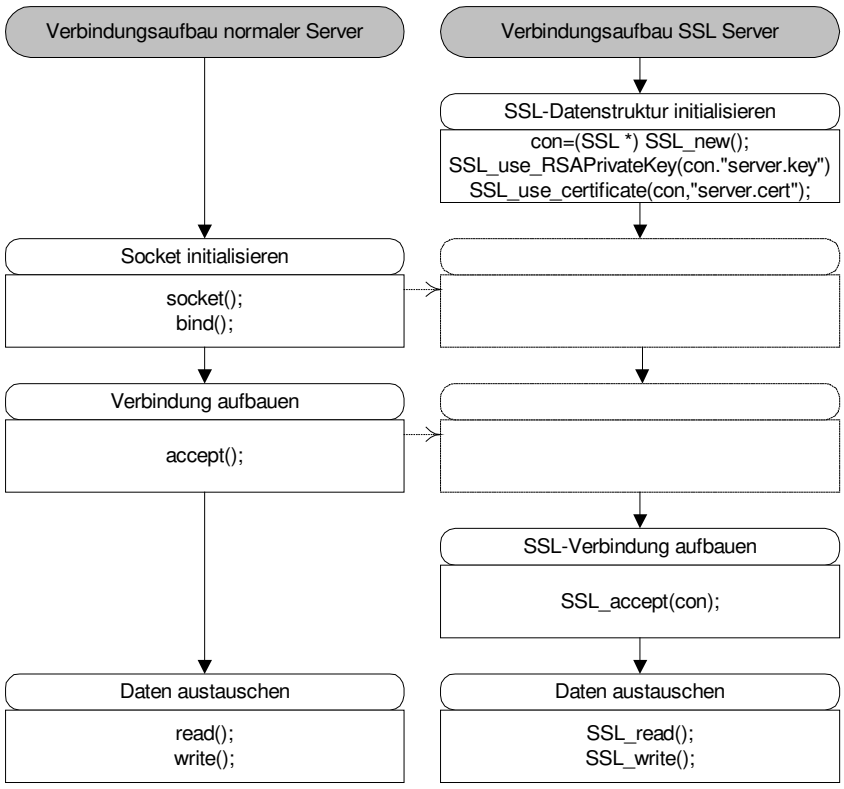

Abbildung 4.4: Verbindungsaufbau mit SSLeay

[44] Die Anwendung muß die beiden Bibliotheken *cryptolib* und *ssllib* importieren.
[45] Beschrieben sind nur die minimal notwendigen Modifikationen. Zur vollständigen Initialisierung und Unterstützung aller Modi des SSL-Protokolls sind weitere Anpassungen erforderlich.

Zunächst erfordert der SSL-Server die Initialisierung der Datenstrukturen der SSLeay-Bibliothek. Die Datenstruktur wird einer Verbindungsvariablen zugewiesen. Ferner müssen das Zertifikat und der private Schlüssel des Servers mit der Verbindung assoziiert werden.

Die nächsten beiden Schritte erfordern keine Modifikationen, da sie mit dem normalen Server identisch sind. Sie betreffen die Initialisierung des Sockets und das Akzeptieren einer Verbindung zu einem Klienten.

Ist die Verbindung fehlerfrei aufgebaut worden, muß vom SSL-Server auf der bestehenden Verbindung eine SSL-Verbindung errichtet werden. Während dieser Phase finden der Zertifikats- und Schlüsselaustausch und Verhandlungen über zu verwendende Algorithmen statt. Die Validierung von Zertifikaten kann durch Registrierung einer benutzerdefinierten *Callback*-Funktion beinflußt werden. Die Funktion wird während der Validierung aufgerufen und kann die Einhaltung frei spezifizierbarer Bedingungen überprüfen. Der Rückgabewert der Funktion entscheidet über die Annahme oder Zurückweisung des Zertifikats.

Ergab der SSL-Verbindungsaufbau keine Fehlermeldung, können der Server und der Klient unter Verwendung der SSL-Routinen zum Lesen und Schreiben jetzt Daten austauschen. Die Routinen des normalen Servers werden nicht mehr benötigt.

4.3 Einbindung von SSLeay in den Tycoon-2-Webserver

Die Einbindung von SSLeay in den Tycoon-2-Webserver erfordert zunächst die Implementierung der C-Bibliothek als Tycoon-2-Klasse. Anschließend kann die Tycoon-2-Klasse in den Webserver integriert und notwendige Modifikationen vorgenommen werden.

Im folgenden werden anhand von Klassen- und Sequenzdiagrammen die Unterschiede zu dem zuvor beschriebenen Webserver aufgezeigt. Überdies werden Zugriffsmöglichkeiten von STML-Anwendungen auf Zertifikate betrachtet.

4.3.1 SSLeay als Tycoon-2-Klasse

Die Implementierung der SSLeay-Bibliothek als Tycoon-2-Klasse erfolgt durch Verwendung der Klasse `DLL`[46]. Sie ermöglicht den Aufruf von C-Routinen als Methoden einer Tycoon-2-Klasse.

Zur Vereinfachung der Schnittstelle zwischen Tycoon-2 und SSLeay werden die benötigten Routinen zu folgenden fünf Methoden zusammengefaßt:

[46] DLL ist die Abkürzung für ***d**ynamic **l**ink **l**ibrary*.

`ssl_init(client_verify :Int, server_cert :String, ca_cert :String) :Int`

Diese Methode führt die Initialisierung der SSL-Datenstrukturen durch und legt Parameter fest. Der Parameter `client_verify` bestimmt, ob der Klient einer SSL-Verbindung anhand eines Zertifikats authentifiziert werden soll. Die Strings `server_cert` und `ca_cert` geben die Pfade zu den Server- und CA-Zertifikaten an. Das Serverzertifikat wird zur Authentifizierung des Servers verwendet, während die CA-Zertifikate bei der Verifizierung der Zertifikate von Klienten benötigt werden. Der Rückgabewert der Methode wird als Fehlerstatus interpretiert.

`ssl_connect(handle :Int)`

Sie nimmt das `handle` auf einen Socket entgegen und baut eine SSL-Verbindung auf. Die Authentifizierungen und Algorithmusverhandlungen finden während dieser Phase statt.

`ssl_read(buffer :MutableString, bufferLength :Int) :Int`

Auf der SSL-Verbindung eintreffende Daten werden ausgelesen und entschlüsselt im Puffer (`buffer`) gespeichert. Ist die Kapazität des Puffers (`bufferLength`) erschöpft, bricht die Methode ab und gibt den Fehlerstatus zurück.

`ssl_write(buffer :String, bufferLength :Int) :Int`

Im Puffer enthaltene Daten werden verschlüsselt und auf der SSL-Verbindung übertragen. Nach Beendigung der Übertragung gibt die Methode den Fehlerstatus zurück.

`get_oneline_cert() :String`

Diese Methode liest einige Daten des Zertifikats eines Klienten aus und gibt sie in Form einer Zeichenkette (`String`) zurück.

Die Methoden werden mit den Bibliotheken `cryptolib` und `ssllib` zur Bibliothek `ssl_server_socket` zusammengefaßt und in die Klasse `SslDll` eingebunden:

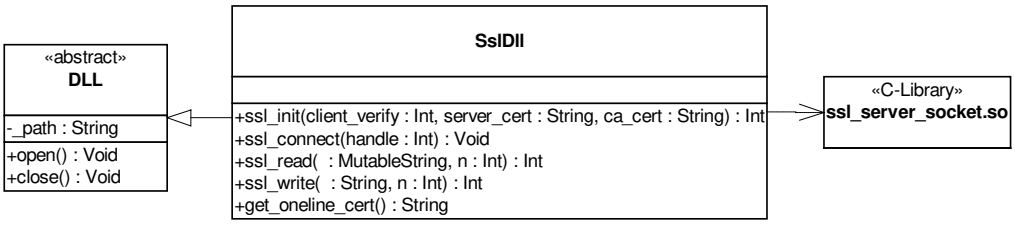

Abbildung 4.5: Einbindung der C-Bibliothek in Tycoon-2

Die Klasse `SslDll` erbt von der Klasse `DLL` und beinhaltet lediglich die Signaturen der C-Methoden. Nachdem eine Instanz erzeugt wurde, muß die angegebene Bibliothek (`_path`) mit `open` geöffnet werden. Anschließend können die C-Routinen als Methoden der Tycoon-2-Klasse aufgerufen werden.

4.3.2 Aufbau des SSL-Webservers

Die Einbindung des SSL-Protokolls erfordert die Modifikation von Klassen des Tycoon-2-Webservers. Das folgende Klassendiagramm basiert auf dem Klassendiagramm des Webservers aus Abbildung 4.1. Es zeigt modifizierte und neu definierte Klassen. Bei modifizierten Klassen ist dargestellt, von welcher Klasse des Webservers sie erben.

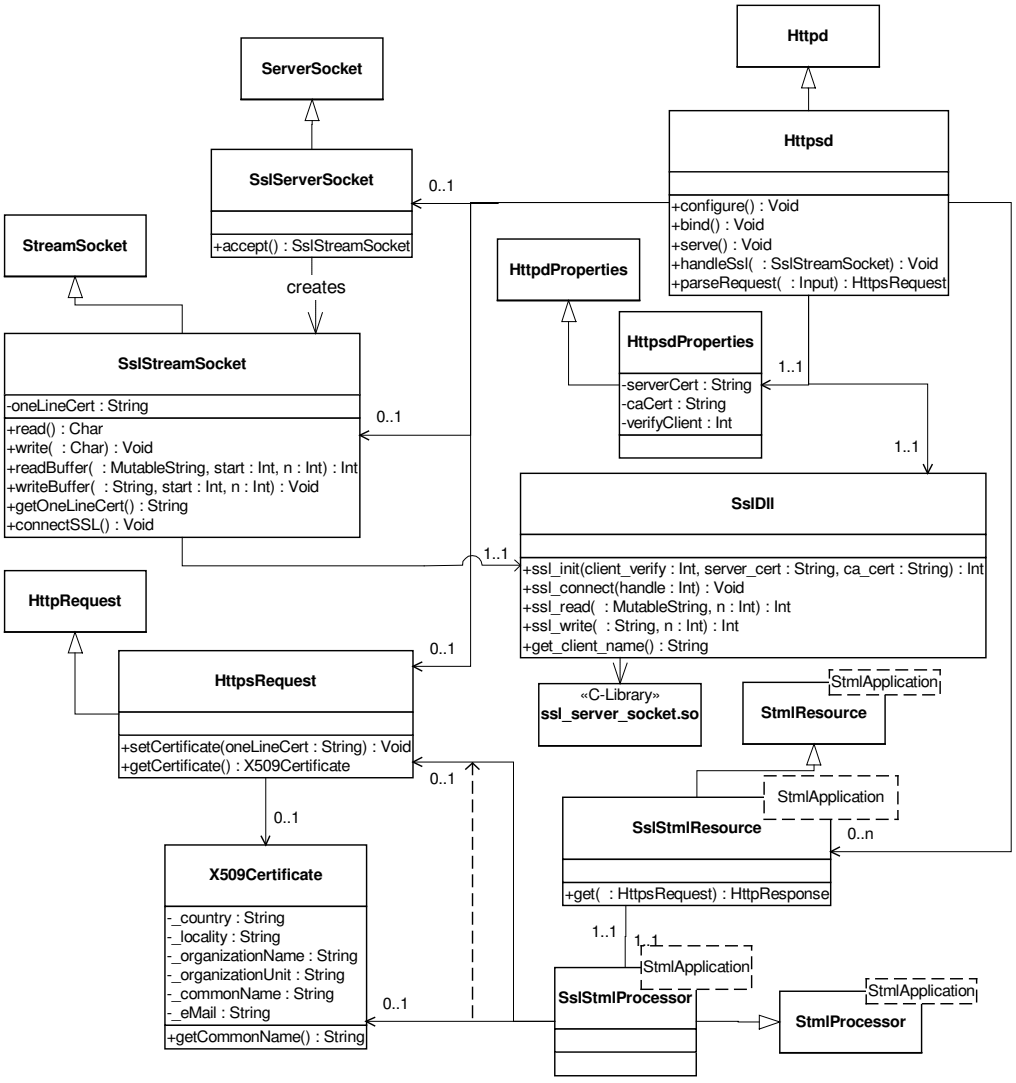

Abbildung 4.6: Struktur des SSL-Webservers

Die Klasse `Httpsd` beinhaltet modifizierte Methoden der Superklasse und die neue Methode `handleSsl`.

Die Klasse `SslStreamSocket` hat zwei neue Methoden zum Aufbau einer SSL-Verbindung (`connectSSL`) und Auslesen des Zertifikats eines Benutzers (`getOneLineCert`) erhalten.

Zur Bereitstellung von SSL-Funktionalität greifen die Klassen `Httpsd` und `SslStream-Socket` auf die bereits beschriebene Klasse `SslDll` zu.

Die Klasse `HttpsRequest` kann das Zertifikat eines Benutzers speichern. Die Methode `setCertificate` nimmt das Zertifikat in Form eines Zeichenkette[47] entgegen und trägt die enthaltenen Daten in eine Instanz der Klasse `X509Certificate` ein. Ein `X509Certificate` kann neben dem Namen und der Email-Adresse auch Angaben zu Land, Landkreis, Organisation und Abteilung des Zertifikatsbesitzers aufnehmen.

In der Klasse `SslStmlResource` wurde die Methode `get` modifiziert, welche von `process` aufgerufen wird. Der Klasse `SslStmlProcessor` wurde die öffentliche Variable `x509Certificate` hinzugefügt, damit zugehörige STML-Anwendungen auf das Zertifikat des Klienten zugreifen können.

Die zur Konfiguration des Servers benötigte Klasse `HttpsdProperties` wurde um einige Variablen erweitert, die von der Methode `ssl_init` der Klasse `SslDll` als Parameter benötigt werden. Die Bedeutung der Variablen wird im nachfolgenden Kapitel erläutert.

4.3.3 Initialisierung und Konfiguration des Servers

Zur Initialisierung des Servers muß eine Instanz der Klasse `Httpsd` erzeugt und das Heimatverzeichnis in der Variablen `home` abgelegt werden. Die Konfiguration wird von der Methode `configure` durchgeführt. Die benötigten Tycoon-2-Anweisungen lauten:

```
define tyhttpd :Httpsd,
tyhttpd := Httpsd.new,
tyhttpd.home := "HOME_DIRECTORY",
tyhttpd.configure;
```

Die Methode `configure` greift auf die Klasse `HttpsdProperties` zu, welche folgende Konfigurationsparameter aus einer Textdatei ausliest:

`DocumentRoot`	Basisverzeichnis der Dokumente des Servers
`Host`	URL des Servers
`Port`	Port des Servers
`User`	Benutzer, der den Server startet
`Group`	Gruppe des Benutzers
`ServerCert`	Datei mit dem Zertifikat des Servers
`CaCert`	Datei mit Zertifikaten der unterstützten Zertifikatsautoritäten
`VerifyClient`	Enthält den Wert 1, falls der Klient authentifiziert werden soll; sonst 0

Tabelle 4.1: Konfigurationsparameter des SSL-Webservers

[47] Die Zeichenkette ist der Rückgabewert der *get_oneline_cert*-Methode der Klasse *SslDll*.

Die Dokumente des Webservers werden im DocumentRoot-Verzeichnis gespeichert. Aus den Parametern Host und Port ergibt sich die Adresse des Webservers. Die Angaben zum Benutzer und seiner Gruppe werden vom Betriebssystem zur Autorisierung von Dateizugriffen benötigt. Diese Parameter sind mit denen des normalen Webservers identisch.

Der SSL-Webserver benötigt zusätzliche Parameter, die auf das Serverzertifikat (ServerCert) und auf eine Datei mit den Zertifikaten der unterstützten Zertifikatsautoritäten (CaCert) verweisen. Die Anforderung eines Klientenzertifikats durch den Server ist optional und kann mit dem Parameter VerifyClient aktiviert werden.

Das Serverzertifikat kann von einer beliebigen Zertifikatsautorität ausgestellt werden. Die Zertifikate der Zertifikatsautoritäten müssen einzeln angefordert und in einer Datei gespeichert werden.

Die Initialisierung und Konfiguration des Webservers ist damit abgeschlossen. Falls der Webserver STML-Dokumente verwalten soll, müssen die zugehörigen Ressourcenklassen registriert werden. Dazu wird eine Ressourcenklasse mit einer Instanz der entsprechenden STML-Anwendung initialisiert und einem Unterverzeichnis des Heimatverzeichnisses des Webservers zugeordnet.

4.3.4 Funktionsweise des SSL-Webservers

Das Sequenzdiagramm in Abbildung 4.7 zeigt die Bearbeitung einer STML-Anfrage auf einer SSL-Verbindung mit dem Schwerpunkt des SSL-Verbindungsaufbaus und Datenaustausches. Da die Klasseninteraktionen mit denen des normalen Webservers teilweise übereinstimmen, werden nur die Abweichungen näher beschrieben.

Die Instanzen der Klassen Httpsd, SslDll und StmlResource werden während der Initialisierung und Konfiguration des Webservers erzeugt. Im Gegensatz zu RTS wird die SslDll nicht vom Tycoon-System, sondern vom Webserver verwaltet.

Die Methode configure führt die Konfiguration des Webservers durch und initialisiert mit ssl_init die SSL-Bibliothek. Abgesehen von der Erzeugung einer Instanz der Klasse SslServerSocket, werden die Methoden bind und serve wie beim normalen Webserver aufgerufen.

Wenn eine Anfrage eintrifft, wird eine Instanz der Klasse SslStreamSocket erzeugt und während der Initialisierung die Methode ssl_connect aufgerufen, sowie gegebenenfalls das Klientenzertifikat ausgelesen (nicht in Abbildung 4.7 dargestellt). Anschließend wird der SslStreamSocket an den Httpsd übergeben.

Die Bearbeitung der Anfrage erfolgt durch die handleSsl-Methode, welche zunächst über bufferedInput die Anfragedaten ausliest. Um dies zu tun, greift der SslStreamSocket mit ssl_read auf die SSL-Verbindung zu. Die zurückgegebene BufferedInput-Instanz wird von der parseRequest-Methode entgegengenommen, welche eine Instanz der Klasse HttpsRequest erzeugt, die zur Aufnahme der Anfragedaten und des Klientenzertifikats dient. Falls ein Klientenzertifikat übertragen wurde, erzeugt HttpsRe-

quest eine Instanz der Klasse `X509Certificate` und übergibt ihr die ausgelesenen Daten.

Nachdem mit `pathLookup` die zugehörige Instanz der `SslStmlResource` bestimmt wurde, startet `process` die Auswertung des STML-Dokuments und führt zur Erzeugung einer Instanz der Klasse `HttpResponse`. Die Übertragung der Antwort geschieht über `emit` und `writeBuffer` mit dem Aufruf der Methode `ssl_write` der `SslDll`.

Die Bearbeitung endet mit dem Aufruf von `close` des `SslStreamSockets` und führt zur Löschung der nicht mehr referenzierten Instanzen.

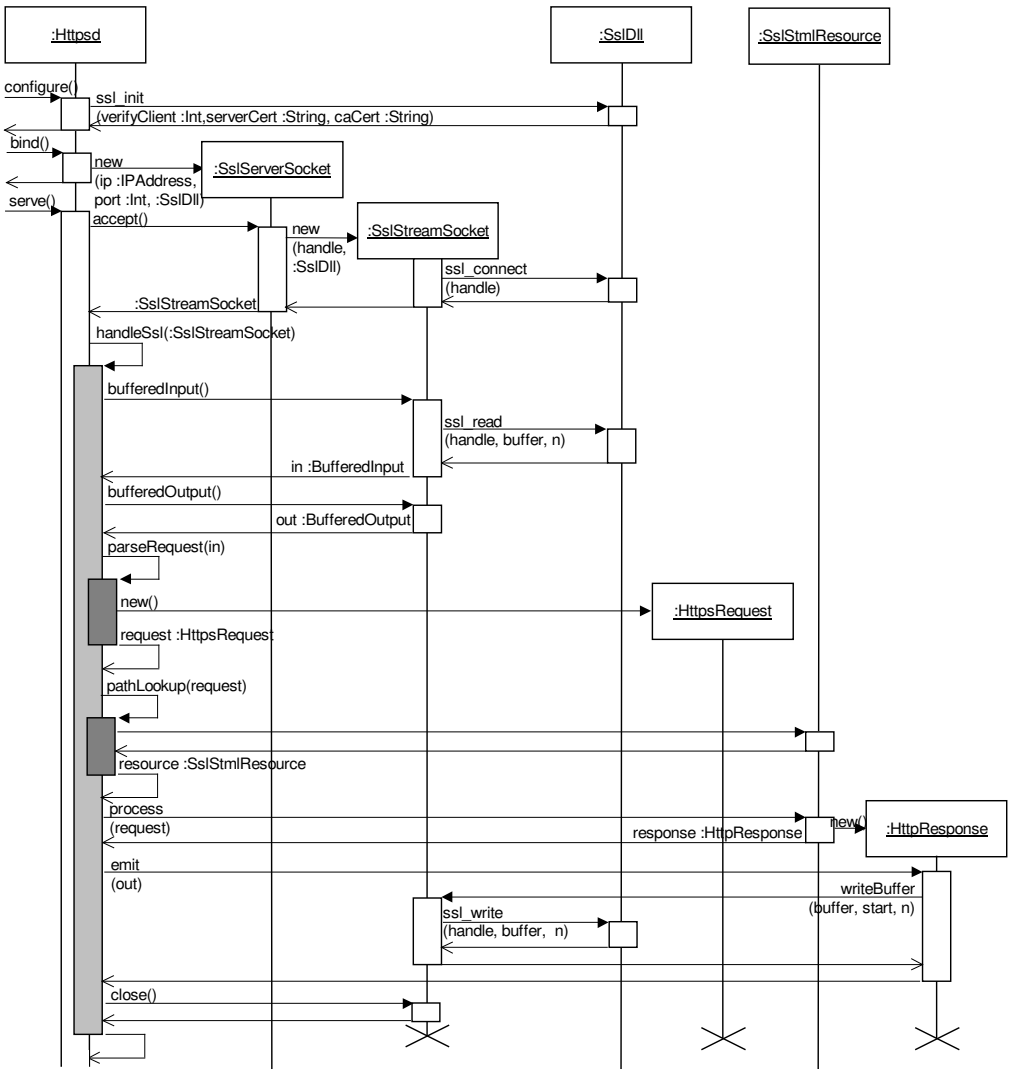

Abbildung 4.7: Bearbeitung einer STML-Anfrage auf einer SSL-Verbindung

4.3.5 Zugriff auf Zertifikate durch STML-Anwendungen

Damit STML-Anwendungen die Daten eines Benutzers weiterverarbeiten können, muß der Zugriff auf sein Zertifikat möglich sein. Die erforderliche Verbindung zwischen Zertifikaten und STML-Anwendungen kann in STML-Dokumenten geschaffen werden.

Wie aus den Abbildungen 1.3 und 4.6 hervorgeht, speichert eine Prozessorklasse des SSL-Webservers Referenzen auf die zugehörige STML-Anwendung (`application`) und das Zertifikat des aktuellen Klienten (`x509Certificate`). Tycoon-2-Anweisungen innerhalb von STML-Dokumenten können damit auf die Anwendung und das Zertifikat zugreifen. Das folgende Textfragment zeigt den Beginn eines STML-Dokuments:

```
<!doctype stml system>
<stml processor=MyProcessor>
<tycoon>
self.application.aMethod(self.x509Certificate)
</tycoon>
```

Die erste Zeile legt den Typ des Dokuments als STML fest. Danach erfolgt die Zuordnung eines Prozessors, der zur Generierung der HTML-Dokumente benötigt wird. Das `tycoon`-Tag beinhaltet eine Anweisung, die einer Methode der STML-Anwendung das Zertifikat als Parameter übergibt. Der Bezeichner `self` referenziert den Prozessor. Von der Anwendung können die Felder des Zertifikats dann ausgelesen und weiterverarbeitet werden. Auf den Namen der Benutzer kann zum Beispiel mit `getCommonName` zugegriffen werden.

4.4 Beurteilung der Sicherheit

Die vom SSL-Webserver gewährleistete Sicherheit ist abhängig von dem SSL-Protokoll und der unterstützenden Infrastruktur. Die Schwächen des SSL-Protokolls sind hauptsächlich in der Unterstützung schwacher Verschlüsselungsalgorithmen begründet (vgl. Kapitel 2.3). In der verwendeten SSLeay-Bibliothek können schwache Algorithmen deaktiviert werden. Allerdings erlauben rechtliche Restriktionen teilweise nur die Verwendung von Schlüsseln bis zu 40 Bit Breite (vgl. Kapitel 2.4), so daß sich dieser Schwachpunkt nicht generell beheben läßt.

Schwerwiegendere Sicherheitsmängel ergeben sich aus der unterstützenden Infrastruktur (vgl. Kapitel 3.5), da sie nicht in der Protokollspezifikation festgeschriebenen ist. Implementierungen von Validierungsalgorithmen für Zertifikate können deshalb erheblich voneinander abweichen. Die Vertrauenswürdigkeit von Zertifikaten und Zertifikatsautoritäten kann jedoch nur basierend auf einer umfangreichen Validierung gewährleistet werden. Die Validierungsalgorithmen der SSLeay-Bibliothek können um selbstdefinierte Funktionen ergänzt werden, so daß der Umfang der Validierung variabel ist und vom Systembetreiber festgelegt werden kann.

Nicht beinflußbar sind Sicherheitsmängel der Klientensoftware der Benutzer. Da die Kommunikation über SSL gleiche Funktionalität beim Klienten und beim Server erfordert, ist die Sicherheit vom schwächsten Glied der Kette abhängig. Wenn ein Klient nur schwache Algorithmen anbietet, müssen diese vom Server akzeptiert werden, sofern die Verbindung aufgebaut werden soll.

Abgesehen von rechtlichen Einschränkungen bildet das SSL-Protokoll eine zuverlässige und sichere Basis zur Authentifizierung und Verschlüsselung im Internet. Zur Anpassung der Sicherheit an individuelle Anforderungen kann die SSLeay-Bibliothek erweitert werden.

Kapitel 5

Autorisierungsmodelle

Ein Autorisierungsmodell ist ein konzeptionelles, von Implementierungen unabhängiges Modell, das eine reichhaltige Semantik zur Beschreibung der funktionalen und strukturellen Eigenschaften eines Sicherheitssystems bietet [CMF+95]. Bevor auf die Eigenschaften von Autorisierungsmodellen eingegangen wird, erfolgt die Definition relevanter, modellunabhängiger Begriffe.

5.1 Definition grundlegender Begriffe

Basierend auf den Dokumenten [CFM+95, Kers95, NyOs94, Kaß95] werden im folgenden Begriffe definiert, die für das Verständnis der Struktur und des Verhaltens von Autorisierungsmodellen erforderlich sind.

Subjekt und Rolle: Subjekte sind die aktiven Entitäten eines Systems und fordern den Zugriff auf Objekte. Als Subjekte können beispielsweise Benutzer, Prozesse oder Rollen in Erscheinung treten. Rollen sind benannte Mengen von Zugriffsrechten, die Benutzern zugeordnet werden können.

Objekt und Zugriffsmodus: Objekte sind die passiven Entitäten des Systems. Sie repräsentieren Daten, Prozesse und Systemressourcen, die vor unberechtigten Zugriffen oder Modifikationen geschützt werden müssen. Die Zugriffsmodi des Systems repräsentieren die Zugriffsmöglichkeiten der Subjekte auf Objekte. Die Ausführung eines Zugriffsmodus verursacht einen bidirektionalen Informationsfluß zwischen Subjekten und Objekten.

Zugriffsanforderung: Sie besteht aus dem anfordernden Subjekt, dem Objekt und dem Zugriffsmodus, der ausgeführt werden soll.

Autorisierung und Autorisierungszustand: Eine Autorisierung beschreibt einen Zugriffsmodus, den ein Subjekt auf einem Objekt des Systems ausführen darf. Die Menge aller Autorisierungen bildet den Autorisierungszustand des Systems.

Rechtepropagierung: Die Weitergabe eigener Rechte an andere Subjekte wird als Rechtepropagierung bezeichnet.

Konsistenzbedingung: Modifikationen des Autorisierungszustandes dürfen nur durchgeführt werden, wenn der resultierende Zustand nicht gegen die Konsistenzbedingungen des Systems verstößt.

5.2 Beschreibung von Modellklassen

Es gibt verschiedene Autorisierungsmodelle, die sich im Hinblick auf ihre Eigenschaften unterscheiden und zu Klassen zusammengefaßt werden können. Nachfolgend werden die grundlegenden Modellklassen beschrieben und einige der dazugehörigen Autorisierungsmodelle vorgestellt. Da eine Modellklasse nicht immer alle Anforderungen eines Systems erfüllt, können Autorisierungsmodelle auf kombinierten Eigenschaften mehrerer Klassen beruhen. Die Einteilung und Beschreibung der Klassen basiert auf [CFM+95, Kers95].

Diskrete Modelle *(discretionary models)*

Diskrete Autorisierungsmodelle kontrollieren den Zugriff von Subjekten auf Objekte anhand der Identität der Subjekte. Das zugreifende Subjekt kann sowohl ein Benutzer, als auch ein Systemprozeß sein. Die Zugriffsforderung wird über den Autorisierungszustand des Systems geprüft. Falls für den gewünschten Zugriff des Subjekts auf das angeforderte Objekt eine Autorisierung existiert, wird der Zugriff gestattet, anderenfalls wird er verweigert. Diskrete Modelle können dem Benutzer die Propagierung von Autorisierungen erlauben.

⇒ Das *HRU-Access-Matrix* Modell[48] wurde ursprünglich von Lampson [Lamp71] aufgestellt und mehrfach modifiziert. In den Zeilen und Spalten der Zugriffsmatrix werden die Subjekte und Objekte des Systems angeordnet. Die Zellen nehmen die Zugriffsmodi auf, die ein Subjekt bezüglich eines Objekts ausführen darf. Das Modell kann in unterschiedlichen Systemen verwendet werden, so daß die Definition der Zugriffsmodi vom Einsatzgebiet abhängt.

⇒ Das *Take-Grant* Modell basiert auf dem *HRU-Access-Matrix* Modell. Es wurde von Jones [JLS76] aufgestellt und später erweitert. Zur Darstellung der Autorisierungen wird statt einer Matrix ein Graph verwendet. Der Schwerpunkt des Modells liegt bei der Propagierung von Autorisierungen.

Diskrete Autorisierungsmodelle sind flexibel und können deshalb von unterschiedlichen Systemen und Anwendungen genutzt werden. Allerdings können die Zugriffsbeschränkungen, trotz der Autorisierung jedes einzelnen Zugriffs, leicht umgangen werden, da die Verwendung der Informationen nicht kontrolliert wird.

[48] *HRU* ist die Abkürzung für ***H**arrison-**R**uzzo-**U**llman*, die das Modell als erste formal beschrieben haben (1976).

Regelbasierte Modelle ***(mandatory models)***

Regelbasierte Modelle überwachen Zugriffe auf der Basis von Klasseneinteilungen der Subjekte und Objekte des Systems. Der Zugriff wird gestattet, falls zwischen den Klassen des Subjekts und des Objekts, unter Berücksichtigung des Zugriffsmodus, eine Beziehung besteht.

⇒ Das *Bell-LaPadula* Modell [BeLa73] ist eine Erweiterung des Zugriffsmatrixmodells und orientiert sich an den Schutzbedürfnissen komplexer Systeme. Das Modell basiert auf der Klassifizierung der Elemente des Systems. Jedem Element wird eine Sicherheitsstufe zugeordnet, die aus einer Klassifikation und einer Menge von Kategorien besteht. Das Modell verhindert den Informationsfluß von höheren zu niedrigeren Sicherheitsstufen.

Der Einsatz regelbasierter Modelle ist nur möglich, wenn die Subjekte und Objekte eines Systems in Klassen eingeteilt werden können. Die Flexibilität ist daher geringer als bei diskreten Modellen, da diese Anforderung nicht von allen Systemen erfüllt wird. Als Vorteil erweist sich, daß regelbasierte Modelle nicht nur den Zugriff, sondern auch den Fluß von Informationen kontrollieren können.

Objektorientierte Modelle

Diese Modellklasse erlaubt die Verwendung objektorientierter Datenmodelle und unterstützt deren reichhaltige Semantik. Die Granularität der Zugriffskontrolle reicht deshalb von Klassen über Instanzen oder Instanzgruppierungen bis hin zu Variablen und Methoden. Außerdem werden im Autorisierungsmodell Beziehungen zwischen Objekten berücksichtigt. Dazu gehören unter anderem Vererbungsbeziehungen zwischen Klassen oder Assoziationen zwischen Objekten [Atki89, BeMa91, EvKe90, Spoo89].

Zur Spezifikation von Autorisierungen stellen objektorientierte Modelle eine erweiterte Semantik zur Verfügung, die im wesentlichen auf folgenden Konzepten basiert [CMF+95]:

⇒ *Explizit / implizit*. Durch Anwendung von Ableitungsregeln können implizite Autorisierungen aus expliziten abgeleitet werden.
⇒ *Positiv / negativ*. Während positive Autorisierungen den Zugriff gestatten, wird er von negativen Autorisierungen explizit verweigert.
⇒ *Stark / schwach*. Schwache Autorisierungen können von anderen schwachen oder starken Autorisierungen überschrieben werden.

Objektorientierte Modelle können sowohl auf diskreten als auch auf regelbasierten Autorisierungsmodellen beruhen.

⇒ Ein Autorisierungsmodell für objektorientierte Datenbanken wurde von Rabitti im *ORION / ITASCA*–Projekt entworfen. Das Modell basiert auf einer diskreten Zugriffskontrolle, die zusätzlich Beziehungen zwischen Subjekten, Objekten und Zugriffsmodi berücksichtigt. Diese Beziehungen werden zur Ableitung neuer Autorisierungen verwendet. [BOS94, CMF+95, RBKW91]

⇒ Im *IRIS*-Autorisierungsmodell werden Variablen und Methoden einer Klasse als geschützte Funktionen interpretiert. Das objektorientierte Prinzip der Datenkapselung wird somit aufrechterhalten. Ferner wird ein umfangreiches Konzept zur Propagierung von Autorisierungen definiert. [Ahad92]

⇒ Das *Data-Hiding*-Modell beruht ebenfalls auf der Kapselung von Daten. Die Methoden von Klassen können zusätzlich in öffentliche und private unterteilt werden. [Bert92]

Objektorientierte Autorisierungsmodelle bieten umfangreiche Modellierungsmöglichkeiten, die maßgeblich zur Aufwandsverringerung bei der Spezifikation von Autorisierungen beitragen. Da sie direkt auf objektorientierten Datenstrukturen basieren, können die Eigenschaften objektorientierter Systeme in die Modellierung einbezogen werden.

Die Beschreibung der Eigenschaften von Modellklassen wird im nachfolgenden Kapitel bei der Auswahl von Autorisierungsmodellen, die in die Spezifikation und den Entwurf des Autorisierungsdienstes eingehen, verwendet.

Kapitel 6

Konzeption eines Autorisierungsdienstes

In diesem Kapitel wird ein Autorisierungsdienst konzipiert, der als Bestandteil der Zugriffskontrolle von Internet-Informationssystemen eingesetzt werden kann. Dazu wird zunächst eine Anforderungsanalyse durchgeführt. Anschließend erfolgt die Spezifikation eines formalen Autorisierungsmodells. Das Kapitel schließt mit der Umsetzung des formalen Modells in ein objektorientiertes Klassenmodell.

6.1 Anforderungsanalyse

Das Ergebnis der Anforderungsanalyse ist die Beschreibung der Aufgaben und Eigenschaften des Autorisierungsdienstes. Sie vermittelt die Grundzüge des umzusetzenden Konzeptes und legt die Grenzen und Beschränkungen des Dienstes fest. Im wesentlichen werden Anforderungen betrachtet, die sich aus der Integration des Dienstes in bestehende Systeme ergeben. Die Analyse beschränkt sich auf die Untersuchung qualitativer Anforderungen. Quantitative Anforderungen können zusätzlich zur Verbesserung der Systemleistung herangezogen werden.

6.1.1 Rahmenbedingungen

Die Rahmenbedingungen ergeben sich aus der Systemumgebung, in die der Autorisierungsdienst integriert werden soll. Aus der Beschreibung der Aufgabenstellung und der Entwicklungsumgebung (vgl. Kapitel 1.2 und 1.4) geht hervor, daß bei der Konzeption des Dienstes die Eigenschaften objektorientierter Systeme und die Anforderungen von Internet-Informationssystemen berücksichtigt werden müssen.

Objektorientierte Systeme bieten eine Reihe von Eigenschaften, die zur Realisierung von wiederverwendbarer, erweiterbarer und wartbarer Software führen können, sofern sie sinnvoll umgesetzt werden. Diese Eigenschaften werden bei der Festlegung von grundlegenden Entwurfsanforderungen einbezogen.

Die Eigenschaften von Internet-Informationssystemen haben Auswirkungen auf die Konzeption eines Autorisierungsdienstes, da sie die Menge von geeigneten Autorisierungsmodellen einschränken. Es gibt Modelle, die auf der Kontrolle des Informationsflusses zwischen Benutzern des Systems beruhen[49]. Dieser Ansatz kann in der beschriebenen Systemumgebung nicht umgesetzt werden, da Benutzer über Standardsoftware auf das Informationssystem zugreifen. Die Verwendung der angeforderten Informationen, insbesondere die Weitergabe, kann somit nicht vom Autorisierungsdienst kontrolliert werden. Zur Kontrolle des Informationsflusses sind Erweiterungen der Standardsoftware, beziehungsweise der Einsatz proprietärer Software nötig.

Der Autorisierungsdienst soll mit unterschiedlichen Realisierungen von Internet-Informationssystemen zusammenarbeiten können und muß deshalb unabhängig von konkreten Protokollen und Kommunikationsmechanismen sein (vgl. Abbildung 1.2).

6.1.2 Grundlegende Entwurfsanforderungen

Als grundlegende Entwurfsanforderungen werden Eigenschaften verstanden, die auf alle Komponenten des Autorisierungsdienstes zutreffen sollen. Die Anforderungen ergeben sich aus den Eigenschaften objektorientierter Systeme.

Generizität: Der Autorisierungsdienst muß unabhängig von spezifischen Anwendungseigenschaften sein, so daß weitere Anwendungen ohne konzeptionelle Änderungen des Autorisierungsdienstes mit Zugriffskontrolle versehen werden können. Desweiteren soll der Schutz einer Anwendung nur minimale Änderungen an ihren Klassen erfordern.

Homogenität: Der Autorisierungsdienst soll auf einem homogenen Konzept beruhen, darf jedoch Spezialfälle und Abweichungen vom Normverhalten nicht ausschließen. Das erfordert die Verwendung generischer Datentypen und den Aufbau einer schlüssigen Klassenhierarchie. Die Vorteile eines homogenen Konzeptes liegen in der leichteren Erweiterbarkeit und Anpassungsfähigkeit des Autorisierungsdienstes.

Erweiterbarkeit: Die Einhaltung der beiden zuvor genannten Anforderungen sichert die Konzeption eines erweiterbaren Autorisierungsdienstes zu, der mit geringem Zeitaufwand an neue Erfordernisse angepaßt werden kann. Denkbar ist die Erweiterung des zugrundeliegenden Autorisierungsmodells um Konzepte, die eine flexiblere Modellierung von Autorisierungen ermöglichen.

6.1.3 Anforderungen an den Autorisierungsdienst

Die folgenden Anforderungen müssen vom Autorisierungsmodell und dessen Umsetzung erfüllt werden. Sie sollen die Konzeption eines flexiblen, anpassungsfähigen und leicht einsetzbaren Autorisierungsdienstes gewährleisten. Hierbei werden allgemeine Anforderungen an Zugriffskontrollsysteme aus [CFM+95, Hart81, LoSc87] zugrundegelegt.

[49] Vgl. Autorisierungsmodell *Bell-LaPadula* aus Kapitel 5.2.

Skalierbarkeit

Der Grad der Skalierbarkeit eines Dienstes ist maßgeblich für seine Anpassungsfähigkeit an veränderte Einsatzbedingungen. Solche Änderungen können sich etwa aus erhöhten Datenmengen oder neuen Anwendungen ergeben. Die Anforderung der Skalierbarkeit erstreckt sich somit über mehrere Gebiete.

Die Objektgranularität muß skalierbar sein, um den Schutzbedürfnissen unterschiedlicher Anwendungen gerecht zu werden. Der Autorisierungsdienst soll deshalb alle auftretenden Objekttypen als schützbar akzeptieren und Zugriffskontrolle sowohl für einzelne Instanzen als auch für alle Instanzen eines Objekttyps vergeben können.

Der Autorisierungsdienst muß die Verwendung mehrerer Autorisierungsalgorithmen unterstützen, die sich hinsichtlich optimierter Strategien unterscheiden. Die Auswahl von geeigneten Algorithmen soll sich zur Laufzeit aus den Werten der Autorisierungsdaten ergeben.

Es soll die Möglichkeit bestehen, Untermengen der Objekte einer Anwendung zu schützen. Auf diese Weise werden unnötige Autorisierungsaufrufe vermieden, da sich die Zugriffskontrolle des Autorisierungsdienstes auf schutzbedürftige Objekte beschränkt. Überdies wird die Zahl der erforderlichen Anwendungsmodifikationen gering gehalten, weil nur von der Zugriffskontrolle geschützte Klassen angepaßt werden müssen.

Flexible Modellierung von Autorisierungen

Die flexible Modellierung von Autorisierungen ermöglicht dem Autorisierungsdienst die Anpassung an unterschiedliche Anforderungen. Zur Modellierung von Autorisierungen können mehrere Konzepte kombiniert werden.

Die Vergabe von Zugriffsrechten für Mengen von Subjekten verringert den Administrationsaufwand und steigert die Übersichtlichkeit. Subjektmengen können durch Zusammenfassung zu Gruppen oder Zuordnung von Rollen gebildet werden.

Das objektorientierte Konzept der Vererbung läßt sich auch auf Autorisierungen anwenden, indem aus explizit angegebenen implizite Autorisierungen abgeleitet werden können. Der Autorisierungsalgorithmus wird dazu um Regeln zur Ableitung von Autorisierungen erweitert.

Die Ableitung von Autorisierungen kann zur Vergabe unerwünschter Zugriffsrechte führen. Eine Lösung dieses Problems besteht in der Überschreibbarkeit von Autorisierungen. Die benötigte Funktionalität wird von den objektorientierten Konzepten der schwachen / starken und negativen / positiven Autorisierungen zur Verfügung gestellt.

Dynamische Modifizierbarkeit

Der Autorisierungszustand des Dienstes ändert sich während der Laufzeit von Anwendungen. Auslöser von Änderungen sind zum Beispiel der Eintrag neuer Objekte oder die Anpassung von Autorisierungen bestehender Objekte. Diese Änderungen müssen zur Aufrechterhaltung der Zugriffskontrolle sofort wirksam werden und erfordern die dynamische Modifikation zur Laufzeit des Autorisierungsdienstes.

Unabhängigkeit von Anwendungen

Der Entwurf des Autorisierungsdienstes muß unabhängig von Anwendungen erfolgen. Anpassungen an die Schutzbedürfnisse von Anwendungen sollen ausschließlich über die Konfiguration des Dienstes während der Initialisierungsphase und zur Laufzeit durchgeführt werden.

6.1.4 Anwendungsfalldiagramm

In Anwendungsfalldiagrammen werden die Beziehungen zwischen Benutzern eines Systems, nachfolgend auch Akteure genannt, und Anwendungsfällen dargestellt. Ein Anwendungsfall beschreibt eine typische Interaktion zwischen Akteuren und dem System [Oest97]. Dieses Analysemodell soll die Berücksichtigung aller relevanten Benutzer-System-Interaktionen während des Entwurfs sicherstellen.

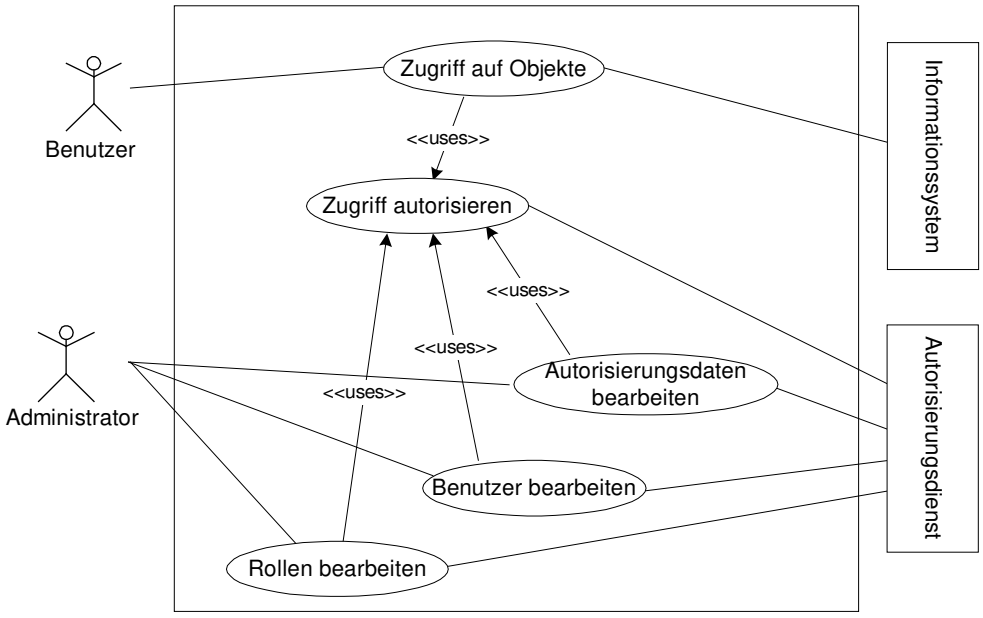

Abbildung 6.1: Anwendungsfalldiagramm des Autorisierungsdienstes

Akteure

Das *Informationssystem* stellt Dienste zur Verfügung, die von *Benutzern* angefordert werden. Diese Dienste können vom *Autorisierungsdienst* geschützt werden. Der Autorisierungsdienst speichert die Autorisierungsdaten, stellt Methoden zur Datenbearbeitung zur Verfügung und führt Autorisierungen von Zugriffsanforderungen durch. Benutzer greifen auf Objekte zu, die vom Autorisierungsdienst geschützt werden. *Administratoren* bearbeiten die Datenbasis des Autorisierungsdienstes, verwalten Benutzer und weisen ihnen Rollen zu.

Anwendungsfälle

Zugriff autorisieren. Dies ist der zentrale Anwendungsfall, der von allen anderen Anwendungsfällen verwendet wird. Er übergibt dem Autorisierungsdienst die Identitäten des zugreifenden Akteurs und des angeforderten Objekts, sowie den Zugriffsmodus. Das Resultat der Autorisierung führt zur Gewährung oder Verweigerung der Zugriffsanforderung.

Zugriff auf Objekte. Ein Benutzer fordert den Zugriff auf ein Objekt des Informationssystems. Diese Zugriffsanforderung wird in dem Anwendungsfall *Zugriff autorisieren* bearbeitet und in Abhängigkeit vom Resultat der Autorisierung entweder gewährt oder verweigert.

Die Anwendungsfälle *Autorisierungsdaten, Benutzer* und *Rollen bearbeiten* werden vom Administrator zur Verwaltung der Daten des Autorisierungsdienstes benötigt.

6.1.5 Überblick der Systemstruktur

Die Anforderungsanalyse schließt mit einem Überblick der Kernklassen des Systems und ihrer Gruppierung zu Paketen. Die Struktur ergibt sich aus den Ergebnissen der Anforderungsanalyse und wird während des Entwurfs als grundlegendes Klassendiagramm verwendet.

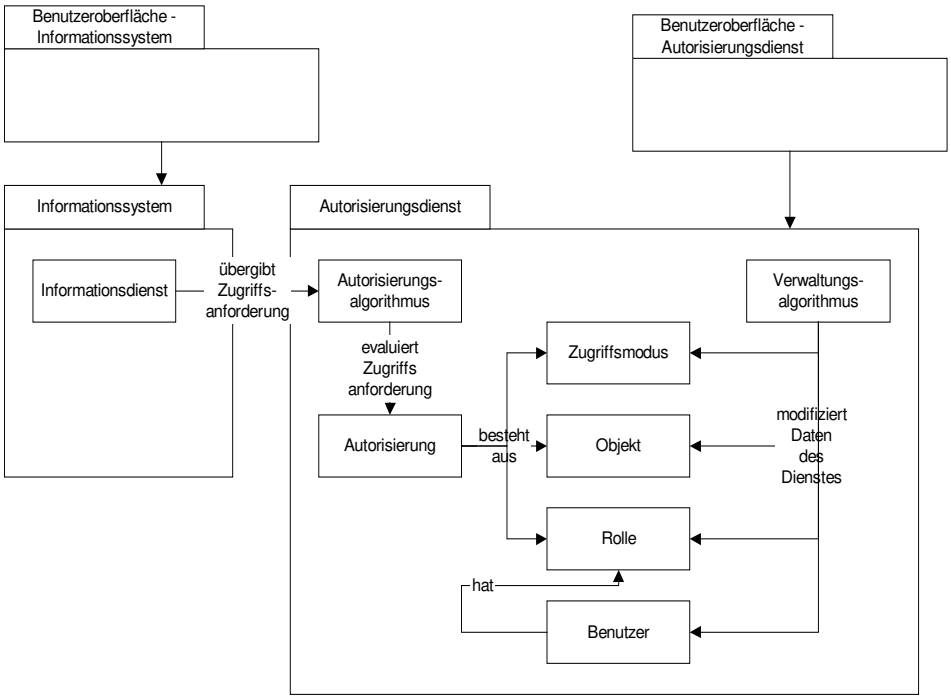

Abbildung 6.2: Pakete und Kernklassen

Das Informationssystem bietet mehrere Dienste an, die zur Autorisierung von Benutzerzugriffen den Autorisierungsalgorithmus benötigen. Die Benutzeroberfläche des

Autorisierungsdienstes wird nur zur Administration benötigt, so daß sich eine getrennte Realisierung von der Benutzeroberfläche des Informationssystems anbietet. Diese Vorgehensweise unterstützt die zuvor festgelegten Anforderungen der Erweiterbarkeit und Wartbarkeit des Systems.

Der Autorisierungsdienst bietet seine Dienste über die Schnittstellen der Verwaltungs- und Autorisierungsalgorithmen an. Seine interne Struktur ist für aufrufende Informationsdienste transparent und erlaubt somit eine anwendungsunabhängige Realisierung. Eine Autorisierung besteht in klassischen Autorisierungsmodellen aus den Komponenten Subjekt, Objekt und Zugriffsmodus. Das Subjekt wurde zugunsten der flexibleren Modellierung von Autorisierungen durch ein Benutzer/Rollen-Konzept ersetzt.

6.2 Spezifikation eines Autorisierungsmodells

Die Spezifikation dient der formalen Beschreibung der Struktur und des Verhaltens eines Modells, das die Anforderungen der zuvor durchgeführten Analyse erfüllt. Zunächst erfolgt die Auswahl einer geeigneten Modellklasse auf der Basis der Rahmenbedingungen. Damit der Autorisierungsdienst den Anforderungen objektorientierter Systeme genügt, wird ein objektorientiertes Modell spezifiziert. Außerdem muß ein diskretes Autorisierungsmodell umgesetzt werden, da die Flußkontrolle regelbasierter Modelle in Internet-Informationsystemen, unter Einhaltung der Rahmenbedingungen, nicht durchführbar ist (vgl. Kapitel 6.1.1).

Die Autorisierungsmodelle der objektorientierten, diskreten Klasse erfüllen jeweils nur einen Teil der Anforderungen. Deshalb basiert die Spezifikation auf der Kombination ausgewählter Eigenschaften der Modelle *ORION*, *IRIS* und *Data-Hiding*.

6.2.1 Subjekte

Subjekte sind die aktiven Elemente eines Informationszugriffs, indem sie die Ausführung einer Aktion auf einem Objekt fordern. Als Subjekte können Benutzer des Systems, Prozesse oder Objekte in Erscheinung treten. Die direkte Umsetzung dieses Sachverhalts erfordert die Definition eines Subjekts für jeden Benutzer, Prozeß oder Objekt. Eine flexiblere Lösung bietet die indirekte Zuordnung über Rollen [NyOs94]. Der Vorteil rollenbasierter Systeme liegt in der vereinfachten Administration, da sich die Zahl der Subjekte verringert und die Vergabe von Zugriffsrechten an Benutzer nur die Zuordnung von entsprechenden Rollen erfordert. Desweiteren kann die Rollenvergabe an die Auswertung von Bedingungen geknüpft werden, die auf der Ebene von einzelnen Zugriffsrechten zu aufwendig ist. Als Bedingungen kommen einander gegenseitig ausschließende Rollen[50] oder eine maximale Anzahl von Benutzern, die einer Rolle zugeordnet sein dürfen, in Frage [Nuss98].

[50] Dies ist ein Konzept, um die Vereinigung zu umfangreicher Zugriffsrechte in einem Subjekt zu verhindern.

Die Zugriffskontrolle des ORION-Modells [CFM+95, BOS94], welche nachfolgend beschrieben wird, definiert Subjekte auf der Basis von Rollen. Es gibt vielfältige Möglichkeiten, Rollen zu definieren. Üblicherweise ergeben sich Rollen aus Aufgabengebieten, Tätigkeiten, Verantwortungsbereichen oder Gruppenzugehörigkeiten der Benutzer. Allerdings ist die Festlegung von Rollen und deren Zugriffsrechten von Anwendungen und Sicherheitspolitiken abhängig, so daß sich durchaus andere Einteilungen ergeben können.

Die Rollen des Modells stehen in einer Implikationsbeziehung zueinander. Eine Rolle R_1 impliziert die Rolle R_2 (formal $R_1 > R_2$) genau dann, wenn alle Autorisierungen von R_2 auch für R_1 gelten. Wird einem Benutzer die Rolle R_1 zugeordnet, erhält er somit implizit die Zugriffsrechte der Rolle R_2.

Aus den Implikationsbeziehungen zwischen den Rollen ergibt sich ein Rollengraph. Ein möglicher Aufbau ist in Abbildung 6.3 dargestellt.

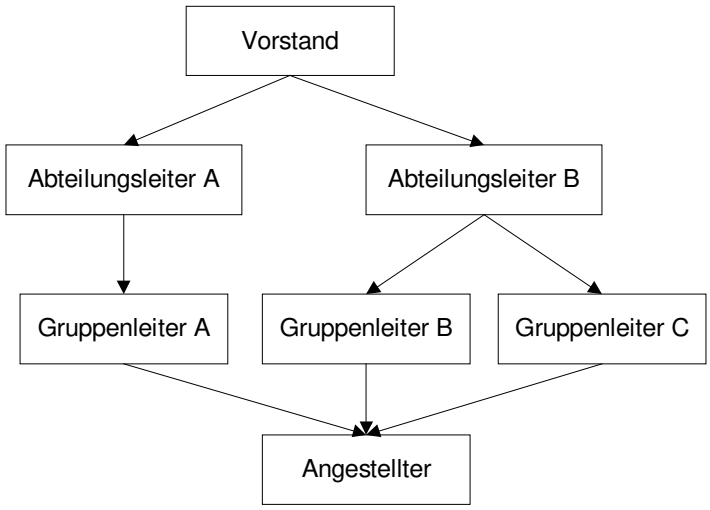

Abbildung 6.3: Beispiel eines Rollengraphen

Eine Kante zwischen zwei Knoten zeigt eine Implikationsbeziehung an. Daraus folgt, daß die Rolle an der Wurzel des Graphen die Autorisierungen sämtlicher anderer Rollen beinhaltet, und die Rolle auf der untersten Stufe des Graphen nur mit minimalen Zugriffsrechten[51], ausgestattet ist. Basierend auf der Implikationsbeziehung wird über den Subjekten eine Teilordnung mit dem Symbol „≥“ definiert:

Seien s_i und s_j zwei Subjekte. Es gilt $s_i \geq s_j$, falls im Rollengraph ein Pfad von s_i zu s_j führt. Daraus folgt

$s_i \geq s_j$, falls $s_i = s_j$
oder $s_i > s_j$
oder es Subjekte $s_1, s_2,..,s_n$ mit $s_i > s_1 > s_2 > ... > s_n > s_j$ existieren.

[51] Die Zuordnung von minimalen Zugriffsrechten wird als *least privilege*–Konzept bezeichnet [CFM+95].

Aus dem Graphen ergeben sich zum Beispiel folgende gültige Beziehungen:

Vorstand > Abteilungsleiter B > Gruppenleiter C > Angestellter
Vorstand ≥ Angestellter

Benutzern können mehrere Rollen zugeordnet werden. Aus der Mehrfachzuordnung ergeben sich jedoch Zugriffskonflikte, falls unterschiedlichen Rollen gegensätzliche Zugriffsrechte erteilt wurden. Zur Auflösung dieses Konflikts werden die Rollen des Benutzers ausgewertet, bis eine Rolle den Zugriff gestattet. Besitzt keine Rolle die erforderlichen Rechte, wird der Zugriff verweigert.

	Autorisierung 1	Autorisierung 2
1. Rolle	Verbot	Verbot
2. Rolle	Verbot	Verbot
3. Rolle	Erlaubnis	Verbot
4. Rolle	nicht durchgeführt	Verbot
Ergebnis	**Erlaubnis**	**Verbot**

Tabelle 6.1: Beispiele zur Mehrfachzuordnung von Rollen

In Tabelle 6.1 sind zwei Beispiele zur Auswertung von Autorisierungen bei mehrfacher Rollenzuordnung enthalten. Die erste Autorisierung ergibt eine Zugriffserlaubnis, da der dritten Rolle der Zugriff erlaubt wird. Die Zugriffsverbote der ersten beiden Rollen werden nicht berücksichtigt und die vierte Rolle braucht nicht ausgewertet zu werden. Die zweite Autorisierung führt zu einem Zugriffsverbot, da keiner der Rollen eine Zugriffserlaubnis zugeordnet ist.

Zur Beschreibung weiterer Eigenschaften und der Verwaltungsoperationen des Graphen wird auf Kapitel 6.2.4 verwiesen.

6.2.2 Objekte

Das ORION-Autorisierungsmodell wurde im Kontext einer objektorientierten Datenbank entwickelt, die Zugriffskontrolle für alle gespeicherten Typen von Objekten bietet. Die Objekte sind hierarchisch angeordnet und umfassen vom System über Klassen bis hin zu den Attributen[52] der Instanzen alle schützbaren Entitäten des Systems [BOS94, CFM+95, RBKW91].

In der Abbildung 6.4 sind die Typen des ORION-Modells und in die Spezifikation übernommene Typen dargestellt.

In der Anforderungsanalyse sind nur Instanzen von Klassen als schützbare Objekte vorgesehen. Das System und Datenbanken können über Instanzen geschützt werden. Zugriffschutz für Klassen wird nicht benötigt, weil Benutzer generell nicht auf Klassendefinitionen zugreifen dürfen und die Einhaltung dieser Restriktion über Schnitt-

[52] In der Spezifikation des ORION-Modells sind die Variablen und Methoden einer Klasse ihre Attribute.

stellendefinitionen des Informationssystems zugesichert wird. Im Gegensatz zum ORION-Modell wird nicht der direkte Zugriff auf Daten der Klassen geschützt, sondern die Methoden der Klassen als Zugriffsmodi modelliert[53] [FLG94, EvKe90]. Der Zugriff auf eine Instanz erfolgt deshalb nur über die Ausführung von Methoden und nicht direkt auf Daten. Aus dem ORION-Modell werden nur die Typen *Instanzgruppe* und *Instanz* übernommen.

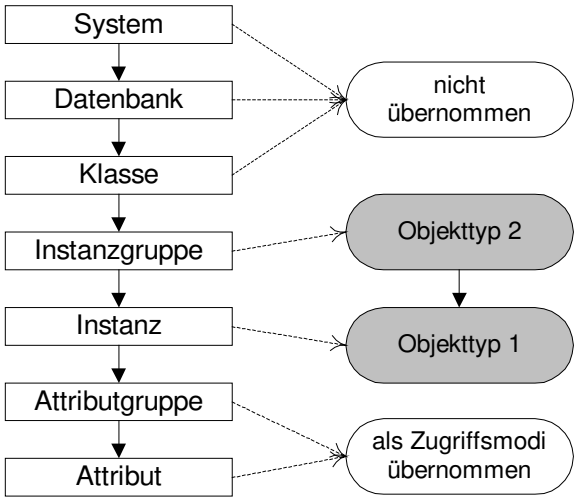

Abbildung 6.4: Objekttypen

Ein Objekt des Typs 1 repräsentiert jeweils eine Instanz einer Klasse. Der Objekttyp 2 repräsentiert alle Instanzen einer Klasse und ermöglicht eine flexiblere und einfachere Administration des Systems. Statt einer Autorisierung pro Instanz wird nur noch eine Autorisierung für die Menge aller Instanzen einer Klasse benötigt.

Der Begriff *geschützte Klasse* wird nachfolgend auf Klassen angewendet, deren Instanzen geschützt sind. Im Gegensatz zur Terminologie des ORION-Modells bezieht sich der Begriff nicht auf den Schutz von Klassendefinitionen. Für jede geschützte Klasse des Informationssystems kann ein Objekt des Typs 2 spezifiziert werden, das alle Instanzen dieser Klasse repräsentiert.

Zwischen den Objekten wird, analog zu den Subjekten, eine Implikationsbeziehung definiert. Eine Implikationsbeziehung zwischen den Objekten o_1 und o_2 (formal: $o_1 > o_2$) gestattet Subjekten, die Zugriffsrechte auf o_1 haben, auch auf das Objekt o_2 zuzugreifen.

Die Instanzen einer Klasse (Objekttyp 1) stehen in einer Implikationsbeziehung zum entsprechenden Objekttyp 2, der die Menge aller Instanzen der Klasse repräsentiert. Zusätzlich können beliebige Implikationen zwischen den Objekttypen spezifiziert werden. Die Abbildung 6.5 zeigt zum Beispiel Implikationen zwischen Instanzen und

[53] Die Modellierung der Zugriffsmodi erfolgt in Kapitel 6.2.3.

Instanzmengen unterschiedlicher Klassen (Implikation a) und zwischen Instanzmengen zweier Klassen (Implikation b).

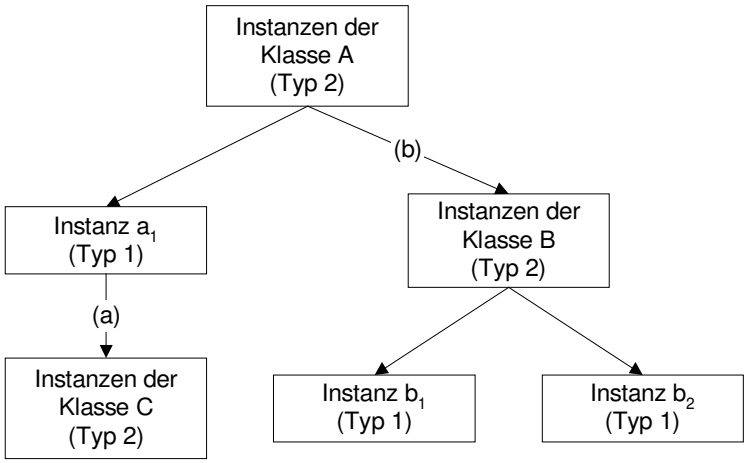

Abbildung 6.5: Beispiel eines Objektgraphen

Basierend auf der Implikationsbeziehung wird über den Objekten eine Teilordnung (formal $o_i \geq o_j$) definiert:

Seien o_i und o_j zwei Objekte. Es gilt $o_i \geq o_j$, falls im Objektgraph ein Pfad von o_i zu o_j führt. Daraus folgt
$o_i \geq o_j$, falls $o_i = o_j$
oder $o_i > o_j$
oder es Objekte $o_1, o_2,..,o_n$ mit $o_i > o_1 > o_2 > ... > o_n > o_j$ existieren.

Die Vererbungsbeziehungen der Klassenhierarchie des zugrundeliegenden objektorientierten Systems werden im Autorisierungsmodell nicht berücksichtigt, da gemäß der Anforderungsanalyse Schutz für Klassen nur selektiv vergeben wird. Subklassen von geschützten Klassen sind ebenfalls geschützt, sie müssen aber explizit in den Objektgraphen eingetragen werden. Sofern die Vererbungsbeziehungen zwischen Klassen zur Vergabe von Autorisierungen erforderlich sind, müssen sie explizit als Implikationen zwischen Instanzmengen modelliert werden.

6.2.3 Zugriffsmodi

Der Zugriff auf Objekte muß auf der Methodenebene erfolgen, da sonst gegen das objektorientierte Prinzip der Datenkapselung verstoßen wird, und Attribute direkt modifizierbar wären [VoBr94, MBCC+90]. Das ORION-Modell faßt Variablen und Methoden von Klassen unter dem Objekttyp Attribut zusammen und gestattet den direkten Zugriff, der über elementare Zugriffsmodi wie zum Beispiel *lesen* und *schreiben* gesteuert wird. Dieses Konzept ist daher zur Umsetzung der Anforderungen ungeeignet.

Unter den Gesichtspunkten der Einhaltung objektorientierter Eigenschaften und Anwendungsunabhängigkeit bietet sich die Umsetzung des *Data-Hiding*-Modells [CFM+95] an. Es gewährleistet Zugriffsschutz für Objekte auf Methodenebene und unterscheidet zwischen öffentlichen und privaten Methoden. Der Zugriff auf private Methoden ist nach dem objektorientierten Paradigma nur innerhalb einer Klasse erlaubt und braucht deshalb vom Autorisierungsdienst nicht kontrolliert zu werden[54]. Die Modifikation von Attributen ist nur über Methoden gestattet. Aus den Anforderungen geht hervor, daß Zugriffsschutz selektiv vergeben wird und deswegen eventuell nur einige der öffentlichen Methoden geschützt sind. Die Methoden eines Objektes können also in private, geschützte[55] und öffentliche Methoden unterteilt werden (vgl. Abbildung 6.6).

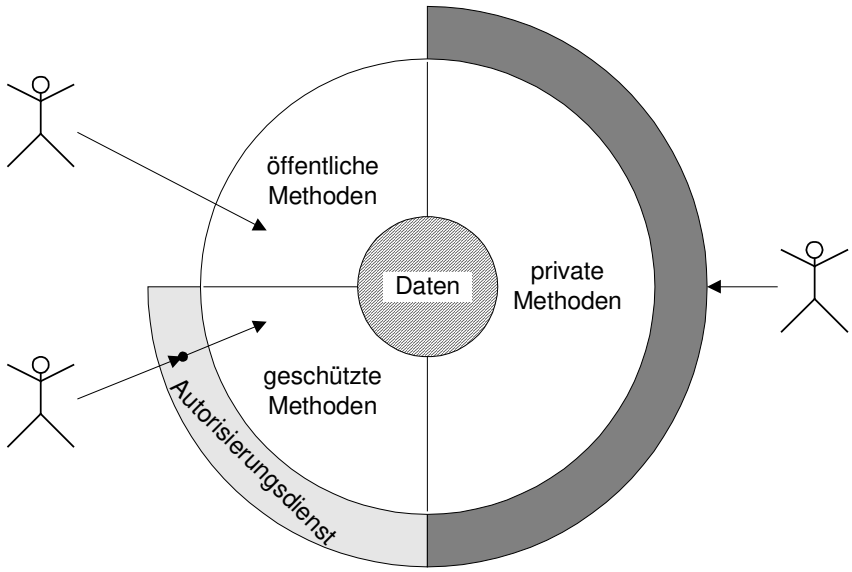

Abbildung 6.6: Zugriffskonzept für Daten und Methoden[56]

Private Methoden dürfen von Benutzern nicht ausgeführt werden. Der Zugriff auf geschützte Methoden wird vom Autorisierungsdienst kontrolliert und abhängig von spezifizierten Autorisierungen gestattet oder verweigert. Auf öffentliche Methoden können Benutzer direkt zugreifen.

Die Zugriffsmodi des Modells sind somit alle geschützten Methoden der geschützten Klassen. Es ist möglich, neu definierte, vererbte und redefinierte Methoden einer Klasse zu schützen. Einschränkungen können sich aus den Eigenschaften der Entwicklungsumgebung und der Implementierung ergeben.

[54] Diese Einschränkung gilt nur, falls der externe Zugriff auf private Methoden vom System unterbunden wird.

[55] Der Begriff *geschützt* steht in keinem Zusammenhang mit dem Schlüsselwort *protected*, welches in einigen objektorientierten Sprachen verwendet wird.

[56] Dargestellt sind nur externe Zugriffe. Der Methodenschutz greift natürlich auch, falls der Aufruf innerhalb der Klasse erfolgt.

Zwischen den Zugriffsmodi wird wiederum eine Implikationsbeziehung definiert. Eine Implikationsbeziehung zwischen den Methoden m_1 und m_2 (formal: $m_1 > m_2$) gestattet Subjekten, die Zugriffsrechte auf m_1 haben, auch die Methode m_2 auszuführen.

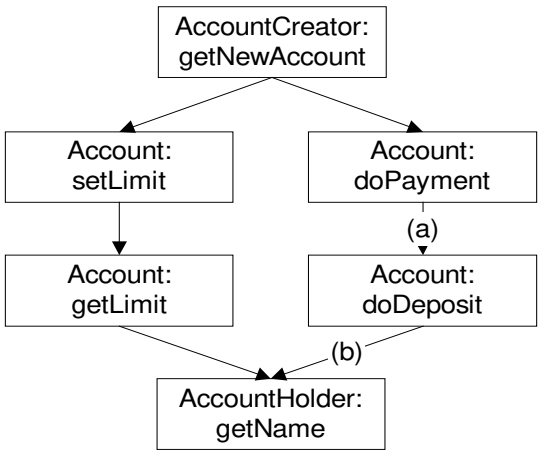

Abbildung 6.7: Beispiel eines Graphen der Zugriffsmodi

Die Abbildung 6.7 zeigt einen Ausschnitt eines Zugriffsmodigraphen, der Methoden mehrerer Klassen über Implikationsbeziehungen verbindet. Die Bezeichnung einer Methode ist nur im Kontext der dazugehörigen Klasse eindeutig. Im Graph werden deshalb die Methoden zusammen mit den Namen ihrer Klassen aufgeführt.

Das Beispiel basiert auf den drei Klassen `Account`, `AccountCreator` und `AccountHolder`. Ein `Account` ist ein Konto, das vom `AccountCreator` für einen Kunden, den `AccountHolder`, eingerichtet wird. Die Methode `getNewAccount` dient zum Einrichten eines Kontos und ist geschützt. Außerdem impliziert `getNewAccount` die Methoden `setLimit` und `doPayment` der Klasse `Account`. Diese Beziehungen ermöglichen Subjekten, die Kontos einrichten dürfen, implizit deren Verwaltung. Innerhalb der Klasse `Account` existieren Implikationen zwischen `setLimit` und `getLimit`, sowie zwischen `doPayment` und `doDeposit`. Eine Autorisierung zur Limitsetzung impliziert somit das Recht zur Limitauslesung. Weiterhin dürfen Subjekte mit dem Recht Auszahlungen vorzunehmen auch Einzahlungen entgegennehmen. Die Implikationen zwischen `getLimit`, `doDeposit` und `getName` ermöglichen Subjekten, die Konten verwalten, den Namen des Kontoinhabers auszulesen.

Im Graph der Zugriffsmodi lassen sich zwei Typen von Implikationsbeziehungen unterscheiden: Implikationen zwischen Methoden einer Klasse (z.B. Implikation a) und klassenübergreifende Implikationen (z.B. Implikation b). Die Kombination beider Typen erlaubt die Spezifikation mächtiger Autorisierungen und trägt zur Erfüllung der Systemanforderungen bei. Zum Beispiel können die erforderlichen Zugriffsrechte zur Durchführung einer Benutzeraktion, bei entsprechender Modellierung der Zugriffsmodi- und Objektimplikationen, über eine einzelne Autorisierung vergeben werden.

Die Implikationsbeziehung dient wieder der Definition einer Teilordnung ≥ über den Methoden:

Seien m_i und m_j zwei Methoden. Es gilt $m_i \geq m_j$, falls im Zugriffsmodigraph ein Pfad von m_i zu m_j führt. Daraus folgt:
$m_i \geq m_j$, falls $m_i = m_j$
oder $m_i > m_j$
oder es Subjekte $m_1, m_2,..,m_n$ mit $m_i > m_1 > m_2 > ... > m_n > m_j$ existieren.

6.2.4 Verwaltung der Graphen

Über den Mengen der Subjekte, Objekte und Zugriffsmodi sind Implikationsbeziehungen definiert, die sich, wie in den Abbildungen 6.3, 6.5 und 6.7 dargestellt, als Graph repräsentieren lassen.

Die Graphen werden zur Ableitung impliziter Autorisierungen[57] benötigt und müssen folgende Eigenschaften erfüllen, um die korrekte und effiziente Durchführung von Ableitungs- und Verwaltungsoperationen gewährleisten zu können [NyOs94, BOS94].

Azyklische Struktur: Der Graph darf keine Zyklen beinhalten. Ein Zyklus widerspricht der Intention der Implikationsbeziehung, die Elemente höherer Stufen mit denen niedrigerer verknüpfen soll. Eine Implikation von einer niedrigeren zu einer höheren Stufe führt jedoch zur Gleichsetzung aller Elemente des Zyklus, da jedes Element die übrigen impliziert.

Redundanzfreiheit: Die Redundanzfreiheit gewährleistet die korrekte und effiziente Durchführung von Operationen auf dem Graphen. Eine Kante ist redundant, falls ihre Endknoten über einen Pfad im Graphen verbunden sind. Redundante Kanten können bei der Durchführung von Verwaltungsoperationen entstehen und werden zur Steigerung der Effizienz von Operationen entfernt.

Maximaler und minimaler Knoten: Maximale und minimale Knoten von Graphen werden zur Spezifikation von Autorisierungen mit maximalen, beziehungsweise minimalen, Zugriffsrechten verwendet. Maximale Zugriffsrechte können zum Beispiel den Administratoren des Systems zugeordnet werden, während minimale Zugriffsrechte[58] allen Benutzern gewährt werden können.

Die Verwaltung des Graphen erfordert die Hinzufügung und Entfernung von Kanten und Knoten. Der Graph darf nach Beendigung einer Operation nicht gegen die zuvor beschriebenen Eigenschaften verstoßen oder dazu führen, daß die Konsistenzbedingungen der Autorisierungsbasen nicht mehr erfüllt sind (vgl. Kapitel 6.2.5).

[57] Vgl. Kapitel 6.2.5.3.
[58] Vgl. *least privilege*-Konzept aus [CFM+95].

6.2.5 Autorisierungen

Eine Autorisierung beschreibt das Recht eines Subjekts, eine Methode eines Objekts ausführen zu dürfen. Es werden explizite und implizite Autorisierungen unterschieden. Explizite Autorisierungen werden in Autorisierungsbasen gespeichert und vom Autorisierungsdienst zur Ableitung impliziter Autorisierungen verwendet. Die Ableitung von Autorisierungen basiert auf den zuvor beschriebenen Implikationsbeziehungen der Subjekte, Objekte und Zugriffsmodi. Die Beschreibung der Autorisierungsbasen und der Ableitung von Autorisierungen basiert auf den Dokumenten [CFM+95, RBKW91].

Zur flexibleren Modellierung von Autorisierungen werden zwei zusätzliche Konzepte eingeführt (vgl. Kapitel 5.2):

⇒ Positive und negative Autorisierungen
⇒ Starke und schwache Autorisierungen

Damit die Anwendung der beiden Konzepte bei der Ableitung von Autorisierungen nicht zu Konflikten führt, werden über den Autorisierungsbasen Invarianten definiert. Diese sollen verhindern, daß Autorisierungen eingefügt werden, die zu widersprüchlichen Zugriffsrechten führen können.

Unter Berücksichtigung der beiden hinzugefügten Konzepte wird der Autorisierungsraum ASP *(**a**uthorization **sp**ace)* wie folgt definiert:

$$ASP = S \times O \times M \times \{+,-\} \times \{s,w\}^{59}.$$

Eine Autorisierung $a \in ASP$ ist somit ein 5-Tupel (s,o,m,as,at) mit s als Subjekt, o als Objekt und m als Zugriffsmodus. Das Vorzeichen wird durch as *(**a**uthorization **s**ign)* festgelegt und der Typ durch at *(**a**uthorization **t**ype)* als stark oder schwach bestimmt.

Für starke und schwache Autorisierungen gelten unterschiedliche Konsistenzkriterien und Ableitungsregeln. Sie werden daher in disjunkten Autorisierungsbasen verwaltet.

6.2.5.1 Starke Autorisierungsbasis

Die starke Autorisierungsbasis *(**s**trong **a**uthorization **b**ase, SAB[60])* beinhaltet alle positiven und negativen starken Autorisierungen und ist daher eine Teilmenge des Autorisierungsraumes. Starke Autorisierungen haben die Form (s, o, m, as, „s“) und werden nachfolgend in der Kurzform [s, o, m, as], ohne explizite Angabe des Typs, dargestellt[61]. Aus den expliziten Autorisierungen werden implizite Autorisierungen abgeleitet. Die Implikationsbeziehung zwischen starken Autorisierungen wird mit dem Symbol „→“ gekennzeichnet.

---

[59] Die Symbole + und − kennzeichnen positive und negative Autorisierungen. Für starke und schwache Autorisierungen werden die Symbole s *(strong)* und w *(weak)* eingeführt.

[60] In [CFM+95] wird die starke Autorisierungsbasis mit AB abgekürzt. Hier wird die Abkürzung SAB bevorzugt.

[61] Die Schreibweisen (s, o, m, as, „s“) und [s, o, m, as] sind äquivalent. Für schwache Autorisierungen wird die Kurzform <s, o, m, as> verwendet.

Der Autorisierungsalgorithmus muß zur Auswertung einer Zugriffsanforderung prüfen, ob eine explizite oder abgeleitete implizite Autorisierung den Zugriff gestattet oder verweigert. Hierzu wird eine Funktion i definiert:

$$i : S \times O \times M \rightarrow (\mathit{true}, \mathit{false}, \mathit{undecided})$$

Für eine Zugriffsanforderung der Form (s, o, m) ergibt sich:

$$i((s,o,m)) = \begin{cases} \mathit{true}, \text{ falls } [s,o,m,+] \in SAB \vee \exists [s_1,o_1,m_1,+] \in SAB : [s_1,o_1,m_1,+] \rightarrow [s,o,m,+] \\ \mathit{false}, \text{ falls } [s,o,m,-] \in SAB \vee \exists [s_1,o_1,m_1,-] \in SAB : [s_1,o_1,m_1,-] \rightarrow [s,o,m,-] \\ \text{sonst } \mathit{undecided} \end{cases}$$

Die Funktion i nimmt als Parameter ein Subjekt, Objekt und Zugriffsmodus entgegen und gibt einen der folgenden Werte zurück: `true`, falls die entsprechende positive Autorisierung in der Autorisierungsbasis enthalten ist oder abgeleitet werden kann; `false`, wenn die entsprechende negative Autorisierung oder eine Ableitung enthalten ist; `undecided`, falls keine explizite Autorisierung spezifiziert ist und aus explizit spezifizierten keine implizite abgeleitet werden kann. Der Funktionsauswertung wird in Abbildung 6.8 verdeutlicht.

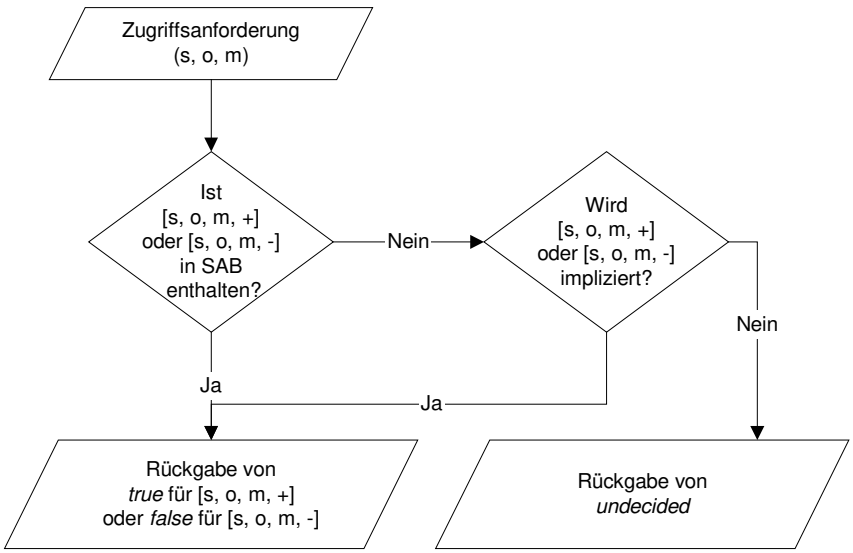

Abbildung 6.8: Auswertung der Funktion i

Die starke Autorisierungsbasis muß folgende Invarianten erfüllen:

⇒ Konsistenz der starken Autorisierungsbasis

$$\forall [s,o,m,as] \in SAB :$$
$$\text{falls } \exists [s_1,o_1,m_1,as] \in SAB : [s,o,m,as] \rightarrow [s_1,o_1,m_1,as]$$
$$\Rightarrow \neg \exists [s_2,o_2,m_2,\neg as] \in SAB : [s_2,o_2,m_2,\neg as] \rightarrow [s_1,o_1,m_1,\neg as]$$

Diese Eigenschaft sichert zu, daß keine zwei Autorisierungen simultan existieren können, aus denen sich eine implizite positive und deren komplementäre negative ableiten lassen.

$\Rightarrow$ Redundanzfreiheit der starken Autorisierungsbasis

$$\forall [s,o,m,as] \in SAB :$$
$$(\forall [s_1,o_1,m_1,as] : [s,o,m,as] \rightarrow [s_1,o_1,m_1,as]) \Rightarrow [s_1,o_1,m_1,as] \notin SAB$$

Diese Eigenschaft verhindert das Einfügen redundanter Autorisierungen in die starke Autorisierungsbasis. Eine Autorisierung ist redundant, falls sie von einer bereits spezifizierten Autorisierung impliziert wird.

6.2.5.2 Schwache Autorisierungsbasis

Die schwache Autorisierungsbasis *(**w**eak **a**uthorization **b**ase, WAB)* enthält die positiven und negativen schwachen Autorisierungen. Zur Darstellung schwacher Autorisierungen wird nachfolgend die Kurzform <s, o, m, as> verwendet. Implizite Autorisierungen werden, wie bei der starken Autorisierungsbasis, aus expliziten abgeleitet. Die Ableitung kann den Autorisierungstyp nicht ändern, so daß implizite Autorisierungen, die aus schwachen Autorisierungen abgeleitet werden, ebenfalls schwach sind. Das Symbol „$\mapsto$“ kennzeichnet Implikationsbeziehungen zwischen schwachen Autorisierungen.

Zur Auswertung von Zugriffsanforderungen wird auf der schwachen Autorisierungsbasis eine Funktion d definiert:

$$d : S \times O \times M \rightarrow (\mathit{true}, \mathit{false}).$$

Die Auswertung ergibt für die Zugriffsanforderung (s, o, m):

$$d((s,o,m)) = \begin{cases} \mathit{true}, \text{ falls } \langle s,o,m,+\rangle \in WAB \vee \exists \langle s_1,o_1,m_1,+\rangle \in WAB : \langle s_1,o_1,m_1,+\rangle \rightarrow \langle s,o,m,+\rangle \\ \mathit{false}, \text{ falls } \langle s,o,m,-\rangle \in WAB \vee \exists \langle s_1,o_1,m_1,-\rangle \in WAB : \langle s_1,o_1,m_1,-\rangle \rightarrow \langle s,o,m,-\rangle \end{cases}$$

Im Gegensatz zur Funktion i der starken Autorisierungsbasis, kann die Funktion d nicht den Wert `undecided` zurückgeben. Für jede Zugriffsanforderung (s, o, m) muß somit entweder eine positive oder negative Autorisierung spezifiziert oder ableitbar sein.

Die folgende Invariante sichert dies zu:

$\Rightarrow$ Vollständigkeit der schwachen Autorisierungsbasis

$$\forall (s,o,m) : \left(\exists \langle s_1,o_1,m_1,as\rangle \in WAB : \langle s_1,o_1,m_1,as\rangle \mapsto \langle s,o,m,as\rangle\right)$$

Eine weitere Invariante verhindert Inkonsistenzen der schwachen Autorisierungsbasis und ist äquivalent zur Konsistenzbedingung der starken Autorisierungsbasis:

$\Rightarrow$ Konsistenz der schwachen Autorisierungsbasis

$$\forall\langle s, o, m, as\rangle \in WAB :$$
$$\text{falls } \exists\langle s_1, o_1, m_1, as\rangle \in WAB : \langle s, o, m, as\rangle \rightarrow \langle s_1, o_1, m_1, as\rangle$$
$$\Rightarrow \neg\exists\langle s_2, o_2, m_2, \neg as\rangle \in WAB : \langle s_2, o_2, m_2, \neg as\rangle \rightarrow \langle s_1, o_1, m_1, \neg as\rangle$$

Im Gegensatz zur starken Autorisierungsbasis sind in der schwachen Autorisierungsbasis Redundanzen zulässig. Deshalb können schwache Autorisierungen, die von existierenden Autorisierungen impliziert werden, trotzdem in die Autorisierungsbasis eingefügt werden.

Die letzte Invariante des ORION-Modells beschreibt die Koexistenz der starken und schwachen Autorisierungsbasis. Diese Eigenschaft wird im theoretischen Modell von der mengenalgebraischen Vereinigung der beiden Autorisierungsbasen gefordert (vgl. [CFM+95]). Da die starke Autorisierungsbasis vor der schwachen Autorisierungsbasis ausgewertet wird, können in der Praxis jedoch keine Inkonsistenzen auftreten (vgl. Kapitel 6.2.7). Die Koexistenzbedingung wird deshalb zur Verringerung des Verwaltungsaufwandes im weiteren Entwurf nicht berücksichtigt.

6.2.5.3 Ableitung impliziter Autorisierungen

Die Ableitung impliziter Autorisierungen ergibt sich aus der Anwendung von Implikationsregeln, die auf den Implikationsbeziehungen zwischen Subjekten, Objekten und Zugriffsmodi basieren.

Regeln zur Ableitung positiver und negativer Autorisierungen

Die Ableitung von Autorisierungen kann in Richtung der Implikationsbeziehungen (i.f. als abwärts bezeichnet) und entgegengesetzt (i.f. als aufwärts bezeichnet) erfolgen. Im Subjektgraphen erfolgt die Ableitung bei positiven Autorisierungen aufwärts und bei negativen Autorisierungen abwärts. Mit Bezug auf Abbildung 6.3 erhält ein Abteilungsleiter die Zugriffsrechte seiner Gruppenleiter, und die Gruppenleiter erhalten ein Zugriffsverbot, falls auch ihr Abteilungsleiter nicht zugreifen darf.

Im Objektgraphen erfolgt die Ableitung immer abwärts. Daher gelten die positiven und negativen Autorisierungen der Objekte implizit für die Objekte auf niedrigeren Stufen. Es können somit ganze Teilbäume des Objektgraphen mit einer Autorisierung sowohl gesperrt als auch freigegeben werden.

Die Ableitung von Zugriffsmodi erfolgt bei positiven Autorisierungen abwärts und bei negativen Autorisierungen aufwärts. Eine Zugriffserlaubnis für höhere Zugriffsmodi gilt daher auch für Zugriffsmodi auf niedrigeren Stufen, und Zugriffsverbote für niedrige Zugriffsmodi wirken sich auch auf höhere Stufen aus.

Die nachfolgenden Regeln zur Ableitung von starken und schwachen Autorisierungen basieren auf den Regeln zur Ableitung positiver und negativer Autorisierungen.

Regeln zur Ableitung starker Autorisierungen

Positive Autorisierungen lassen sich für Subjekte auf höheren Stufen und für Objekte und Zugriffsmodi auf niedrigeren Stufen der jeweiligen Graphen ableiten.

Regel für positive Autorisierungen:

$$\forall s_l, s_k \in S, o_i, o_j \in O, m_n, m_m \in M:$$
$$s_k \geq s_l, o_i \geq o_j, m_n \geq m_m \Rightarrow [s_l, o_i, m_{n,+}] \rightarrow [s_k, o_j, m_{m,+}]$$

Negative Autorisierungen lassen sich für Subjekte und Objekte auf niedrigeren Stufen und für Zugriffsmodi auf höheren Stufen der jeweiligen Graphen ableiten.

Regel für negative Autorisierungen:

$$\forall s_l, s_k \in S, o_i, o_j \in O, m_n, m_m \in M:$$
$$s_k \geq s_l, o_i \geq o_j, m_n \geq m_m \Rightarrow [s_k, o_i, m_{m,-}] \rightarrow [s_l, o_j, m_{n,-}]$$

Regeln zur Ableitung schwacher Autorisierungen

Die Regeln zur Ableitung schwacher Autorisierungen beruhen auf denen der starken Autorisierungsbasis. Zusätzlich muß allerdings berücksichtigt werden, daß schwache Autorisierungen überschrieben werden dürfen. Die Ableitung einer impliziten schwachen Autorisierung endet deshalb bei einer überschreibenden expliziten Autorisierung.

Zur Abgrenzung des Gültigkeitsraumes von impliziten Autorisierungen wird der Begriff Reichweite P *(scope)* einer schwachen Autorisierung eingeführt. Die Reichweite P umfaßt alle abgeleiteten Autorisierungen, die nicht von einer expliziten Autorisierung überschrieben werden:

$$\forall \langle s_l, o_i, m_n, as \rangle \in WAB:$$
$$P(\langle s_l, o_i, m_n, as \rangle) =$$
$$\{ \langle s, o, m, as \rangle \mid s \in S, o \in O, m \in M \wedge \langle s_l, o_i, m_n, as \rangle \rightarrow \langle s, o, m, as \rangle \wedge$$
$$(\neg \exists \langle s_k, o_j, m_m, as \rangle \in WAB : \langle s_l, o_i, m_n, as \rangle \rightarrow \langle s_k, o_j, m_m, as \rangle \wedge \langle s_k, o_j, m_m, as \rangle \rightarrow \langle s, o, m, as \rangle) \}$$

Die folgende Ableitungsregel ergibt sich nach Berücksichtigung der Reichweite einer schwachen Autorisierung:

$$\forall s_l, s_k \in S, o_i, o_j \in O, a_n, a_m \in M, as \in \{+,-\}:$$
$$\text{falls } \langle s_l, o_i, a_n, as \rangle \in WAB \wedge \langle s_k, o_j, a_m, as \rangle \in P(\langle s_l, o_i, a_n, as \rangle) \Rightarrow \langle s_l, o_i, a_n, as \rangle \mapsto \langle s_k, o_j, a_m, as \rangle$$

6.2.5.4 Beispiel zur Ableitung impliziter Autorisierungen

Die Regeln zur Ableitung impliziter Autorisierungen werden nachfolgend anhand eines Beispiels veranschaulicht. Die Abbildung 6.9 zeigt die Graphen der Subjekte, Objekte und Zugriffsmodi, welche für das Beispiel verwendet werden. Bei den Graphen der Subjekte und Zugriffsmodi handelt es sich um Ausschnitte der Graphen aus den Abbildungen 6.3 und 6.7.

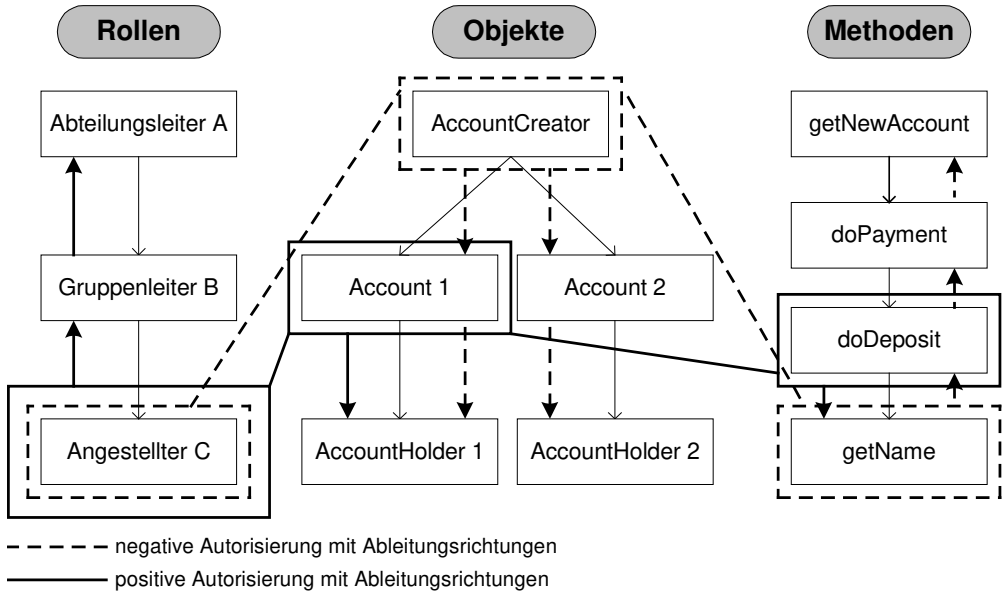

Abbildung 6.9: Beispiel zur Ableitung von Autorisierungen

In der schwachen Autorisierungsbasis sind die folgenden beiden Autorisierungen enthalten:

(1) <Angestellter C, AccountCreator, getName, – >
(2) <Angestellter C, Account 1, doDeposit, + >

Die beiden Autorisierungen sind zusammen mit ihren Ableitungsrichtungen in Abbildung 6.9 eingezeichnet. Anhand der schwachen Autorisierungsbasis soll nun folgende Zugriffsanforderung autorisiert werden:

(Angestellter C, AccountHolder 1, getName)

Dazu versucht der Autorisierungsalgorithmus, die Zugriffsanforderungen aus den expliziten schwachen Autorisierungen abzuleiten. Die Reichweite einer schwachen Autorisierung wird zunächst vernachlässigt.

zu (1) Es wird die Ableitungsregel für negative Autorisierungen benötigt. Das Subjekt und der Zugriffsmodus stimmen überein. Im Objektgraphen kann aus dem AccountCreator der AccountHolder 1 abgeleitet werden (gestrichelte Pfeile). Das Ergebnis der Autorisierung ist daher ein **Zugriffsverbot**.

zu (2) Es wird die Ableitungsregel für positive Autorisierungen benötigt. Das Subjekt stimmt mit der Zugriffsanforderung überein. Der AccountHolder 1 kann aus dem AccountCreator abgeleitet werden (durchgezogener Pfeil). Der Zugriffsmodus getName ist aus dem Modus doDeposit ableitbar (durchgezogener Pfeil). Das Ergebnis der Autorisierung ist daher eine **Zugriffserlaubnis.**

Dieser Konflikt kann durch Betrachtung der Reichweite der Autorisierungen aufgelöst werden. Da die Attribute der zweiten Autorisierung in Ableitungsrichtung der Attri-

bute der ersten Autorisierung liegen, beziehungsweise mit ihnen identisch sind, wird die erste Autorisierung von der zweiten überschrieben (vgl. Kapitel 6.2.5.3). Die Zugriffsanforderung befindet sich daher nicht innerhalb der Reichweite der negativen Autorisierung, und das Ergebnis ist somit die Zugriffserlaubnis der zweiten Autorisierung.

6.2.6 Administration von Autorisierungen

Zur Administration von Autorisierungen werden Strategien zur Erteilung, Weitergabe[62] (i.f. Propagierung) und Rücknahme von Autorisierungen, sowie zur Konsistenzsicherung der Autorisierungsbasen benötigt. Mögliche Strategien reichen von einem zentralen Administrator über das dezentrale Eigentümerkonzept bis hin zum *Laissez-Faire*-Konzept [Nuss98]. Im ersten Ansatz werden die Rechte zur Verwaltung von Autorisierungen nur der Administratorenrolle zugesprochen. Die dezentrale Verwaltung überläßt den Erzeugern von Objekten die Administration von Autorisierungen. Im *Laissez-Faire*-Konzept hingegen wird der Eigentümer nicht berücksichtigt. Stattdessen dürfen alle Subjekte ihre Zugriffsrechte auf Objekte selbständig verwalten.

Das ORION-Modell [BOS94] sieht die Umsetzung eines *Laissez-Faire*-Konzeptes vor. Der Nachteil ist die unkontrollierte Propagierung von Rechten, die im Rahmen eines Informationssystems mit geschützten Bereichen nicht wünschenswert ist. Im IRIS-Modell [CFM+95] wird ein eingeschränktes *Laissez-faire*-Konzept verwendet, indem für Autorisierungen angegeben wird, ob das Subjekt diese propagieren darf. Es kann somit ein skalierbarer Dienst mit zentraler Verwaltung, d.h. Propagierungen sind nicht erlaubt, bis hin zu dezentraler Verwaltung, d.h. alle Subjekte dürfen Rechte propagieren, realisiert werden.

Die Propagierung von Rechten entspricht der Eintragung von expliziten Autorisierungen in die Autorisierungsbasen des Modells. Ein Subjekt darf nur dann eine neue Autorisierung eintragen, falls aus dem Subjekt, dem Objekt und dem Zugriffsmodus der neuen Autorisierung eine positive Autorisierung abgeleitet werden kann. Das heißt, wenn ein Subjekt auf ein Objekt zugreifen kann, darf es anderen Subjekten den Zugriff erlauben oder verweigern. Die Propagierung von negativen Autorisierungen ist nicht vorgesehen, da Subjekte ohne Zugriffsrechte auch keine Propagierungen vornehmen dürfen.

Die Umsetzung erfordert eine Erweiterung der Autorisierungsdefinition um eine Kopiermarke *(**c**opy **m**arker)* cm:

$$a = (s, o, m, as, at, cm).$$

Im IRIS-Modell zeigt die Kopiermarke an, ob die Propagierung von Rechten erlaubt ist. Dieses Konzept wird um die Möglichkeit erweitert, die Propagierung des weitergegebenen Rechts zu beschränken. In der erweiterten Definition kann die Kopiermarke drei Werte annehmen:

$$cm \in \{false, true, selectable\}$$

[62] Die Weitergabe von Autorisierungen wird im folgenden auch als Propagierung bezeichnet.

Der Wert `false` verhindert die Propagierung von Rechten. Subjekte können ihre Zugriffsrechte zwar ausüben, aber nicht weitergeben. Der Wert `true` erlaubt die Propagierung von Zugriffsrechten, allerdings wird die Kopiermarke bei der Weitergabe auf `false` gesetzt. Das Subjekt mit dem propagierten Recht darf dieses somit nicht weitergeben. Bei Autorisierungen mit der Kopiermarke `selectable` darf das weitergebende Subjekt den Wert der Kopiermarke in der propagierten Autorisierung bestimmen. Da negative Autorisierungen nicht propagiert werden dürfen, wird ihre Kopiermarke nicht ausgewertet und kann einen NULL-Wert annehmen.

Der Entzug eines Zugriffsrechtes kann von allen Subjekten vorgenommen werden, die das Recht zur Propagierung der zugrundeliegenden Autorisierung haben.

6.2.7 Zugriffskontrolle

Die Zugriffskontrolle prüft Zugriffsanforderungen anhand der Autorisierungsbasen und Ableitungsregeln und gestattet oder verweigert den Zugriff. Zunächst wird geprüft, ob die Zugriffsanforderung einer expliziten oder impliziten starken Autorisierung entspricht. Trifft dies zu, wird der Zugriff, entsprechend des Vorzeichens der Autorisierung, entweder gestattet oder verweigert. Ergab die Prüfung der starken Autorisierungsbasis kein Ergebnis, werden die Autorisierungen der schwachen Autorisierungsbasis überprüft. Aus der Vollständigkeitseigenschaft der schwachen Autorisierungsbasis ergibt sich, daß entweder eine positive oder negative schwache Autorisierung, explizit oder implizit, der Zugriffsanforderung entspricht.

Die Zugriffskontrolle kann als eine Funktion beschrieben werden [CFM+95]:

$$
\begin{aligned}
f(s, o, m) := \quad & \text{if } i(s,o,m) = \textit{undecided} \\
& \text{then } f(s,o,m) = d(s,o,m) \\
& \text{else } f(s,o,m) = i(s,o,m)
\end{aligned}
$$

Die Funktion f nimmt eine Zugriffsanforderung der Form (s, o, m) entgegen und gibt entweder `true` für eine positive Autorisierung oder `false` für eine negative Autorisierung zurück. Zur Auswertung werden die Funktionen i und d der Autorisierungsbasen verwendet.

6.3 Entwurf des Autorisierungsdienstes

Die Umsetzung des zuvor spezifizierten Autorisierungsmodells erfolgt unter Verwendung der *Unified Modeling Language*. In Strukturdiagrammen werden Klassen und dazugehörige Assoziationen dargestellt. Die Klassendefinitionen beinhalten nur ausgewählte Variablen und Methoden, die zum Verständnis der Funktionalität einer Klasse benötigt werden. Eine detailliertere Beschreibung wird in Anhang B vorgenommen. Zur Darstellung der Interaktionen zwischen Klassen werden Sequenzdiagramme eingesetzt. Der Aufbau des Kapitels entspricht dem der Spezifikation des Autorisierungsmodells.

6.3.1 Benutzer

Die Benutzer des Systems werden mit der Klasse `User` modelliert und über ihren Namen identifiziert. Der Name dient zur Auswahl von Benutzern bei der Administration und muß eindeutig sein. Die Rollen eines Benutzers werden von der Benutzerklasse verwaltet (vgl. Abbildung 6.10).

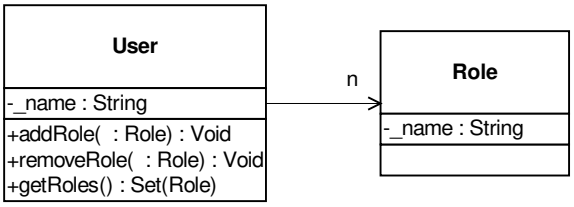

Abbildung 6.10: Benutzerklasse und Rollenzuordnung

Mit den Methoden `add-/removeRole` können einem Benutzer Rollen zugeteilt bzw. entzogen werden. Zur Durchführung von Autorisierungen werden die Rollen eines Benutzers mit `getRoles` ausgelesen.

6.3.2 Subjekte, Objekte und Zugriffsmodi

Die Kernklassen des Autorisierungsmodells sind Subjekte, Objekte und Zugriffsmodi. Aus den Klassen und der Implikationsbeziehung ergibt sich ein Graph mit den in Kapitel 6.2.4 beschriebenen Eigenschaften.

Die Subjekte des Autorisierungsmodells sind Rollen. Im Entwurf werden sie als Klasse `Role` berücksichtigt. Jede Instanz der Klasse `Role` muß im System eindeutig über einen Namen identifizierbar sein. Den Objekten und Zugriffsmodi entsprechen Instanzen der Klassen des Systems und deren Methoden. Deshalb handelt es sich nicht um neu definierte Klassen, sondern um systemabhängige Klassen, deren Bezeichnungen von der Implementierungsumgebung abhängen. Aus dem Kontext der Tycoon-2-Systems ergeben sich die Bezeichner `Object` und `Method`, welche im folgenden verwendet werden. Die Klasse `Object` ist die Wurzel der Tycoon-2 Klassenhierarchie und somit Superklasse aller anderen Klassen. Die Klasse `Method` dient zur Kapselung einer Methode.

Die Graphen der Subjekte, Objekte und Zugriffsmodi des Autorisierungsmodells haben eine einheitliche Struktur mit identischen Eigenschaften. Zur Umsetzung der Graphen werden deshalb parametrisierte Klassen verwendet, die in Abhängigkeit vom Typparameter den Graph für Subjekte, Objekte oder Zugriffsmodi aufnehmen.

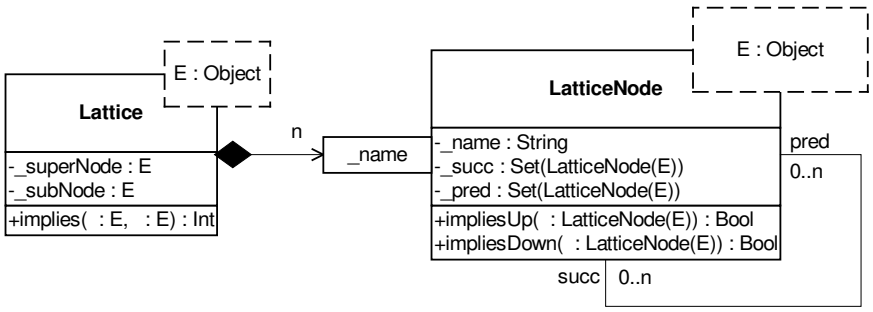

Abbildung 6.11: Parametrisierter Graph

Die Abbildung 6.11 zeigt die Konstruktion eines Graphen aus der Basisklasse `Lattice`[63] und den Knoten `LatticeNode`. Die Klasse `Lattice` beinhaltet Methoden zur Verwaltung der Knoten und Implikationen, sowie zur Auswertung von Implikationen (`implies`). Überdies existieren zwei private Variablen zur Speicherung des maximalen und minimalen Knotens des Graphen (`_superNode, subNode`). Jeder Knoten speichert in zwei Variablen seine Vorgänger und Nachfolger im Graphen (`_succ, _pred`). Zur Unterstützung von positiven und negativen Autorisierungen können für Knoten Implikationsbeziehungen zu Vorgängern und Nachfolgern geprüft werden (`impliesUp, impliesDown`).

In Abbildung 6.12 ist die Struktur eines Graphen, der mit der Klasse `Role` typisiert wurde, dargestellt.

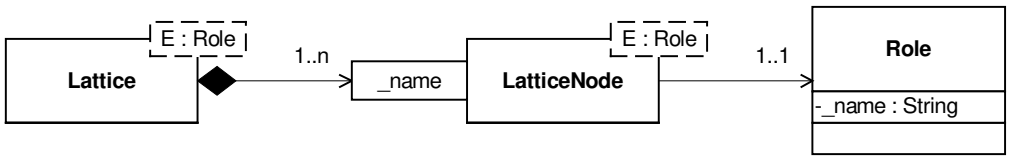

Abbildung 6.12: Graph zur Aufnahme von Rollen

Jedem Knoten des Graphen wird eine Rolle zugeordnet, deren Name zur Identifizierung des Knoten dient. Die Eindeutigkeit des Namen wird von der Klasse `Role` gefordert.

Die Graphen der Objekte und Zugriffsmodi werden mit den Systemklassen `Object` und `Method` parametrisiert. Konzepte zur Vergabe von Namen für Knoten dieser Graphen sind deswegen systemabhängig und werden im Rahmen der Implementierung in Tycoon-2 beschrieben (vgl. Kapitel 7.1).

Eine Instanz der Klasse `LatticeManager` aggregiert die Graphen des Autorisierungsdienstes und die Benutzer des Systems.

[63] Im ORION-Modell [CFM+95] werden die Graphen als *Lattices* bezeichnet.

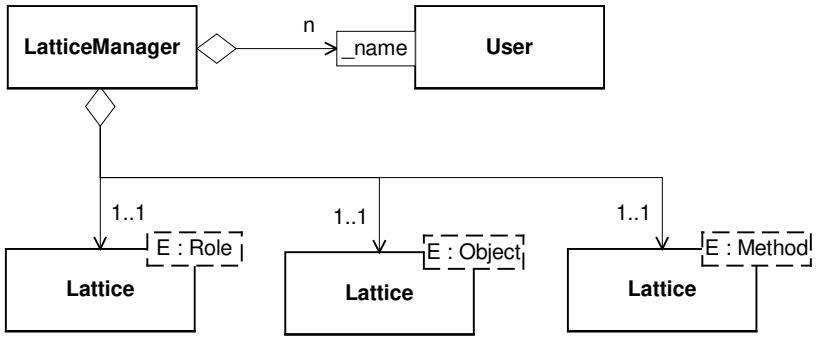

Abbildung 6.13: Verwaltung der Graphen und Benutzer

Der `LatticeManager` aus Abbildung 6.13 stellt für jeden der drei Graphen einen Satz von Verwaltungsmethoden bereit, die auf die Methoden der Klasse `Lattice` zugreifen. Mit den Verwaltungsmethoden der Benutzer können Instanzen der Klasse `User` hinzugefügt, gelöscht und ausgelesen werden.

6.3.3 Autorisierungen

Die Attribute von Autorisierungen werden nicht in einer Klasse gekapselt, sondern auf eine hierarchische Datenstruktur abgebildet. Diese bietet den Vorteil, daß über Schlüssel (Namen oder Objektkennungen) effizient auf die Attribute zugegriffen werden kann.

Aus der Spezifikation von starken und schwachen Autorisierungen ergibt sich die Notwendigkeit getrennter Autorisierungsbasen, weil mit unterschiedlichen Algorithmen und Konsistenzbedingungen gearbeitet wird. Der Autorisierungstyp braucht deshalb nicht explizit gespeichert zu werden, sondern ergibt sich aus der zugehörigen Autorisierungsbasis.

Die Wurzel der hierarchischen Struktur aus Abbildung 6.14 ist eine Autorisierungsbasis. Sie enthält Referenzen auf alle Objekte, für die Autorisierungen spezifiziert sind. In der nächsten Ebene verweisen die Objekte auf zugriffsberechtigte Subjekte, die wiederum Verweise auf erlaubte Zugriffsmodi und zusätzliche Attribute beinhalten.

Die Knoten der hierarchischen Struktur müssen Referenzen auf nachfolgende Knoten oder Blätter aufnehmen. In den Klassen `Object` und `Role`, welche die Objekte und Subjekte repräsentieren, sollen jedoch keine Referenzen gespeichert werden, damit die Unabhängigkeit vom System und den Diensten erhalten bleibt. Es werden daher zusätzliche Klassen benötigt, die Objekte und Subjekte kapseln und Referenzen nachfolgender Knoten und Blätter aufnehmen.

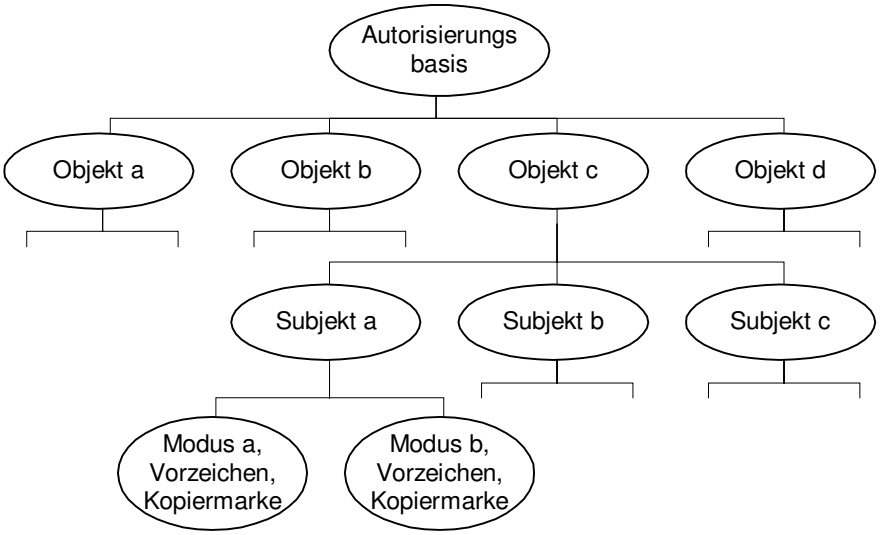

Abbildung 6.14: Hierarchische Struktur von Autorisierungen

Die Struktur und das Verhalten der Autorisierungsbasen wird, entgegen dem objektorientierten Paradigma der Kapselung innerhalb einer Klasse, auf mehrere Klassen verteilt. Dieses Vorgehen gestattet die getrennte Modifikation von Algorithmen und Datenstrukturen und bietet eine Basis zur Implementierung unterschiedlicher Autorisierungsstrategien. Außerdem können die grundlegenden Datenstrukturen zur Effizienzsteigerung um Hilfsstrukturen ergänzt werden, die als Zwischenspeicher dienen oder zusätzliche Indizes aufnehmen[64]. Die Struktur der starken und schwachen Autorisierungsbasis ist identisch und basiert deshalb auf denselben Klassen (vgl. Abbildung 6.15). Das Verhalten der Autorisierungsbasen wird in Kapitel 6.3.4 modelliert.

Eine Autorisierungsbasis wird von einer Instanz der Klasse `AuthorizationBase` repräsentiert, die Referenzen auf Objekte verwaltet. Die Klasse `AuthorizationObject` ist der Container für eine Instanz der Klasse `Object`. Sie enthält Methoden zur Verwaltung der Subjekte (`insert-/removeSubject`) von Autorisierungen. Die Klasse `AuthorizationSubject` kapselt die Rolle und alle Methoden des Objekts, die von der Rolle ausgeführt werden dürfen. Zu jeder Methode wird das Vorzeichen `sign` und die Kopiermarke `copyMarker` der Autorisierung gespeichert (`insert-/removeAccessMode`). Die Werte des Vorzeichens und der Kopiermarke werden als Integer modelliert.

[64] Die Durchführung von Autorisierungen kann die mehrfache Prüfung von Implikationsbeziehungen erfordern, so daß die Zwischenspeicherung der Ergebnisse bereits erfolgter Prüfungen die Effizienz der Algorithmen steigert.

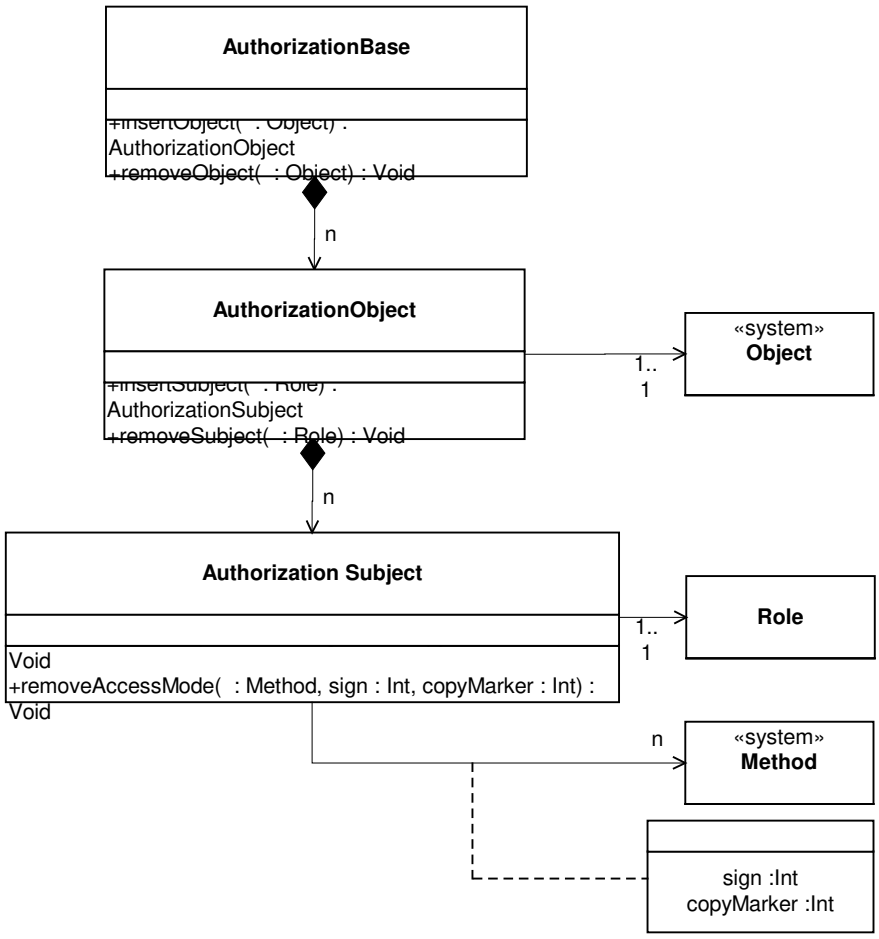

Abbildung 6.15: Klassenstruktur von Autorisierungen

6.3.4 Algorithmen zur Autorisierung und Verwaltung

Die Funktionalität zur Durchführung und Verwaltung von Autorisierungen ist in Algorithmusklassen gekapselt. Alle Algorithmen sind in einer Vererbungshierarchie eingeordnet und erben von abstrakten Klassen. Solche Strukturen werden als Strategiemuster *(strategy pattern,* Gamma [GHJV95]) bezeichnet.

Strategiemuster kapseln Algorithmen mit identischer Schnittstelle, damit unabhängig vom aufrufenden Objekt die Funktionalität variiert werden kann. Die aufrufenden Objekte werden als Parameter übergeben und stellen Schnittstellen zur Verfügung, die es den Methoden der Algorithmusklassen erlauben, auf die Daten der Objekte zuzugreifen.

In Abbildung 6.16 ist eine grundlegende Klassenhierarchie dargestellt, die um spezialisierte Algorithmen erweitert werden kann. Spezialisierungen führen zu optimierten

Algorithmen, die in Abhängigkeit vom Autorisierungszustand und der Zugriffsanforderung ausgewählt werden können.

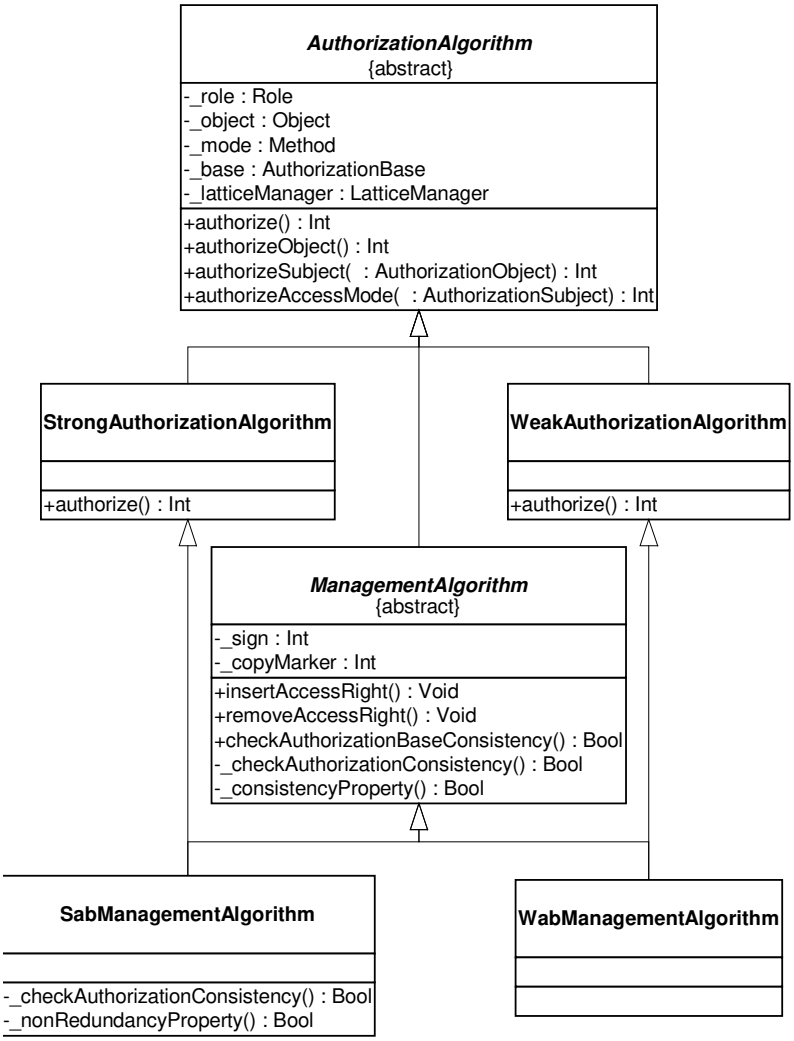

Abbildung 6.16: Klassenhierarchie der Algorithmen

Zur Durchführung von Autorisierungen dienen Algorithmen, die von der abstrakten Klasse `AuthorizationAlgorithm` abgeleitet werden. Ihre Variablen `_role`, `_object` und `_mode` speichern die Attribute der zu autorisierenden Zugriffsanforderung. Außerdem werden Referenzen auf die Autorisierungsbasis und die Graphen benötigt (`_base`, `_latticeManager`). Die Variablen werden während der Initialisierung belegt. Zur Autorisierung werden die Methoden `authorizeObject`, `authorizeSubject` und `authorizeAccessMode` aufgerufen, die für jeweils ein Attribut die Existenz einer passenden expliziten oder impliziten Autorisierung überprüfen und das Ergebnis als Integerwert zurückgeben.

In den Subklassen `StrongAuthorizationAlgorithm` für die starke und `WeakAuthorizationAlgorithm` für die schwache Autorisierungsbasis wird die abstrakte Methode

authorize implementiert. Dabei werden die besonderen Eigenschaften der Autorisierungsbasen, wie zum Beispiel die Reichweite von schwachen Autorisierungen berücksichtigt.

Zur Verwaltung von Autorisierungen wird die abstrakte Klasse ManagementAlgorithm definiert. Ihre Variablen _sign und _copyMarker speichern das Vorzeichen und die Kopiermarke einer Autorisierung, und die Methoden insertAccessRight und removeAccessRight dienen zum Einfügen und Entfernen von Autorisierungen.

Die Methoden checkAuthorizationBaseConsistency und _checkAuthorizationConsistency rufen _consistencyProperty zur Prüfung der Konsistenzbedingungen der Autorisierungsbasen auf. In den Subklassen SabManagementAlgorithm und WabManagementAlgorithm können weitere Bedingungen eingeführt werden. In der Klasse SabManagementAlgorithm wird zum Beispiel die Methode _nonRedundancyCheck zur Prüfung der Redundanzfreiheit der starken Autorisierungsbasis definiert.

Außerdem wird für Konsistenzprüfungen die Funktionalität zur Durchführung von Autorisierungen benötigt. Deshalb erben die Klassen SabManagementAlgorithm und WabManagementAlgorithm von der abstrakten Klasse ManagementAlgorithm und zusätzlich von StrongAuthorizationAlgorithm beziehungsweise WeakAuthorizationAlgorithm..

6.3.5 Verwaltung der Autorisierungen und Algorithmen

Eine Instanz der Klasse AuthorizationManager aggregiert die Autorisierungsbasen und Algorithmen zur Autorisierung und Verwaltung. Die Methoden der Klasse instanziieren Algorithmen und initialisieren diese mit den erforderlichen Parametern.

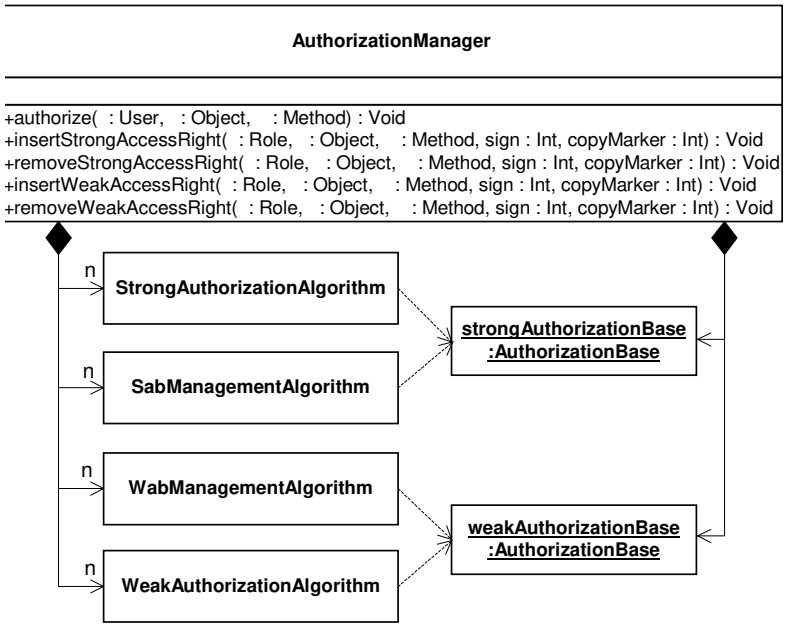

Abbildung 6.17: Verwaltung der Autorisierungsbasen und Algorithmen

Zur Durchführung von Autorisierungen definiert die Klasse `AuthorizationManager` (vgl. Abbildung 6.17) die Methode `authorize`. Sie nimmt als Parameter einen Benutzer, ein Objekt und eine Methode entgegen und führt für die Rollen des Benutzers Autorisierungen durch. Dazu wird ein `AuthorizationAlgorithm` initialisiert und dessen `authorizeObject`-Methode aufgerufen. Der `AuthorizationManager` ist für die Auswahl eines geeigneten starken oder schwachen Algorithmus zuständig. Falls alle Autorisierungen negativ verlaufen, wird eine Ausnahme (vgl. Kapitel 6.3.8) ausgelöst, die vom Informationssystem abgefangen und verarbeitet werden kann.

Zur Verwaltung der Autorisierungen stehen Methoden zur Verfügung, die über einen starken oder schwachen Verwaltungsalgorithmus auf die Autorisierungsbasen zugreifen. Das Hinzufügen und Entfernen von Autorisierungen entspricht der Propagierung bzw. dem Entzug von Zugriffsrechten. Deshalb wird anhand der Kopiermarke geprüft, ob der Benutzer zur Durchführung der Operation berechtigt ist (vgl. Kapitel 6.2.6).

6.3.6 Ausführung von Algorithmen

Algorithmen implementieren nur das Verhalten der Autorisierungsbasen und greifen deshalb zur Durchführung und Verwaltung von Autorisierungen auf die Klassen der Autorisierungsdatenstruktur zu. Als Beispiel zeigt das Sequenzdiagramm in Abbildung 6.18 die Interaktionen zwischen Objekten, die während der Durchführung einer Autorisierung auftreten.

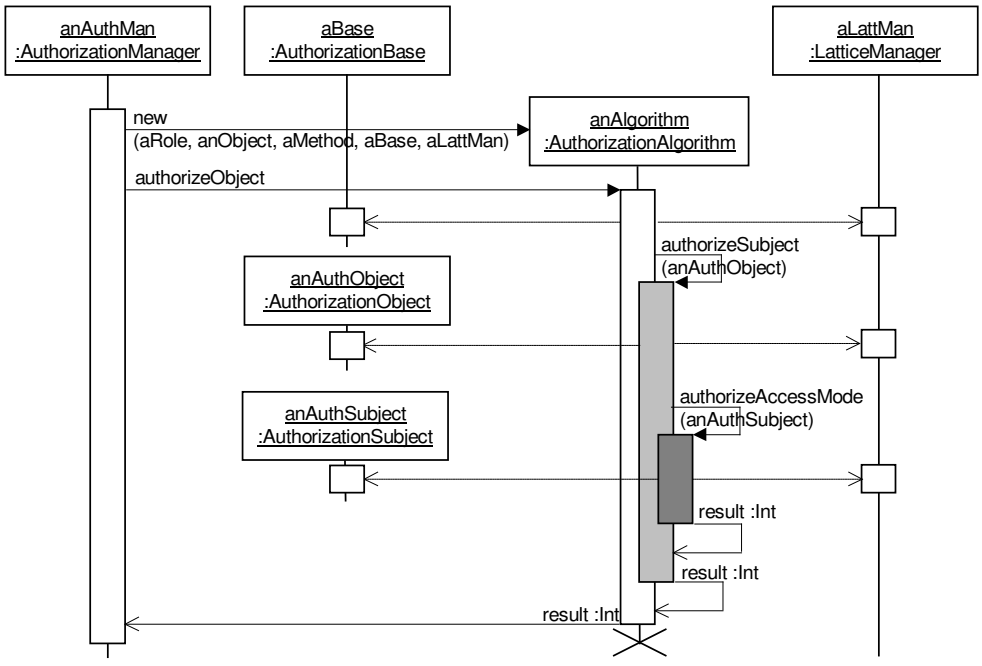

Abbildung 6.18: Sequenzdiagramm zur Durchführung einer Autorisierung

Der `AuthorizationManager` initialisiert eine Instanz der Klasse `AuthorizationAlgorithm` mit den Attributen einer Zugriffsanforderung und den Referenzen auf die Auto-

risierungsbasis und den LatticeManager. Danach ruft er die Methode authorizeObject des Algorithmus auf. Der Algorithmus prüft, ob das Objekt in der Autorisierungsbasis enthalten ist. Ist dies nicht der Fall, wird über den LatticeManager ein Objekt gesucht, welches das angeforderte Objekt impliziert. Die Interaktionen zwischen dem Algorithmus, der Basis und dem LatticeManager sind umfangreich und werden deshalb vereinfacht, ohne Angabe von Methoden, dargestellt. Falls ein explizites oder implizites Objekt existiert, wiederholt sich der Vorgang für das Subjekt und den Zugriffsmodus in den Methoden authorizeSubject und authorizeAccessMode, die verschachtelt aufgerufen werden. Dabei wird nicht mehr auf die Autorisierungsbasis, sondern auf die Containerklassen AuthorizationObject und AuthorizationSubject zugegriffen. Das Ergebnis der Autorisierung wird in Form eines Integerwertes an den AuthorizationManager zurückgegeben und die Instanz des Algorithmus gelöscht.

6.3.7 Vordefinierte Implikationen und Autorisierungen

Für die Automatisierung der Administration des Objektgraphen und der Autorisierungsbasen können Implikationen zwischen Objekten und Autorisierungen vordefiniert und in Schablonen *(templates)* gespeichert werden. Bei der Erzeugung eines neuen Objektes werden die vordefinierten Implikationen und Autorisierungen vom Autorisierungsdienst übernommen.

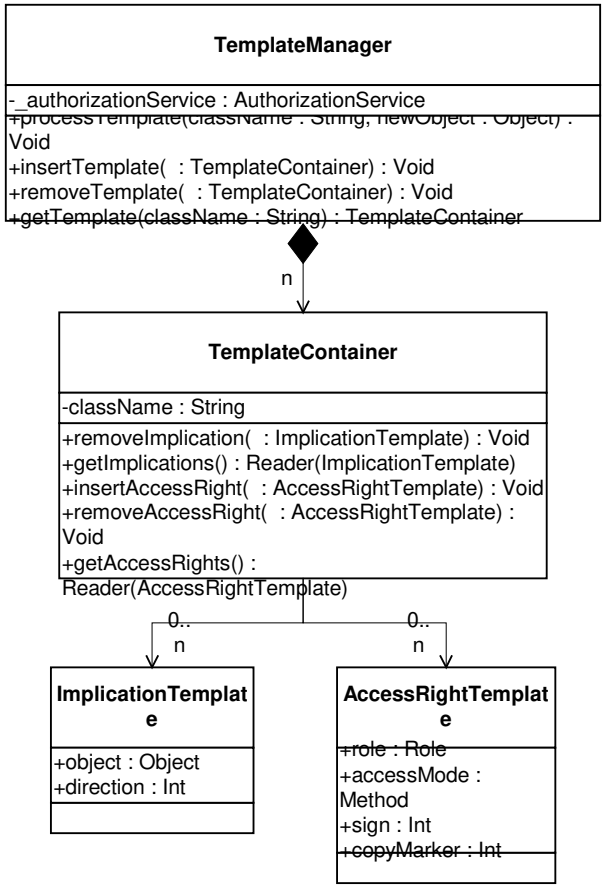

Abbildung 6.19: Schablonen für Implikationen und Autorisierungen

In Abbildung 6.19 sind die zur Verwaltung von Schablonen benötigten Klassen dargestellt. Eine vordefinierte Implikationsbeziehung zwischen Objekten wird in einer Instanz der Klasse `ImplicationTemplate` gespeichert. Sie enthält das beteiligte Objekt (`object`) und eine Variable `direction`, welche die Richtung der Implikation angibt (`newObject > Object` oder `Object > newObject`). Eine Autorisierung wird in einer Instanz der Klasse `AccessRightTemplate` gespeichert. Die Variable `role` verweist auf das Subjekt der Autorisierung. Falls kein Subjekt angegeben ist, wird die Autorisierung für die aktuelle Rolle des Benutzers, der das Objekt erzeugt hat, vergeben. Die Variablen `accessMode`, `sign` und `copyMarker` enthalten die übrigen Attribute der Autorisierung.

Die Klasse `TemplateContainer` aggregiert die vordefinierten Implikationen und Autorisierungen für eine Objektklasse (`className`). Der `TemplateManager` verwaltet die `TemplateContainer` für alle geschützten Klassen. Bei der Erzeugung eines neuen Objektes wird die Methode `processTemplate` aufgerufen, welche die zugehörige Schablone ausliest und über Methoden der Klasse `AuthorizationService` in den Autorisierungsdienst einfügt.

6.3.8 Ausnahmebehandlung

Die Ausnahmebehandlung des Autorisierungsdienstes beruht auf der abstrakten Klasse `Exception`. Alle Klassen des Autorisierungsdienstes können Instanzen der Subklassen von `Exception` erzeugen und somit Ausnahmen auslösen.

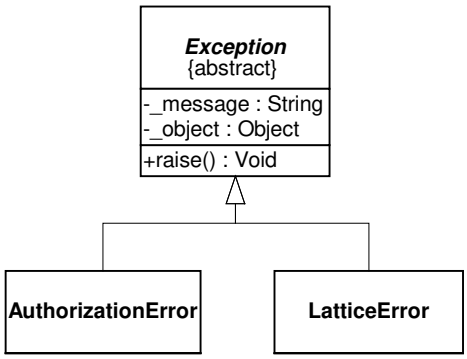

Abbildung 6.20: Klassen zur Ausnahmebehandlung

Die Klasse `Exception` aus Abbildung 6.20 definiert Variablen zur Aufnahme des auslösenden Objektes (`_object`) und einer Nachricht (`_message`), welche die Ausnahme näher beschreibt. Nachdem den Variablen Werte zugewiesen wurden, kann eine Ausnahme durch Aufrufen der Methode `raise` ausgelöst werden. Von der Klasse `Exception` erben die Klassen `AuthorizationError` und `LatticeError`, die zur Auslösung von Ausnahmen in den Autorisierungsbasen beziehungsweise Graphen eingesetzt werden.

6.3.9 Systemüberblick

Die wichtigsten Klassen des Autorisierungsdienstes und ihre Beziehungen werden in Abbildung 6.21 zusammenhängend dargestellt.

Bis auf die neue Klasse `AuthorizationService` wurden alle Klassen und Beziehungen zuvor definiert und beschrieben. Der `AuthorizationService` soll den geschützten Diensten des Informationssystems als Schnittstelle dienen und kapselt zu diesem Zweck jeweils eine Instanz des `LatticeManager` und des `AuthorizationManager`. Die Methoden des `AuthorizationService` entsprechen denen der Manager und leiten Aufrufe an diese weiter.

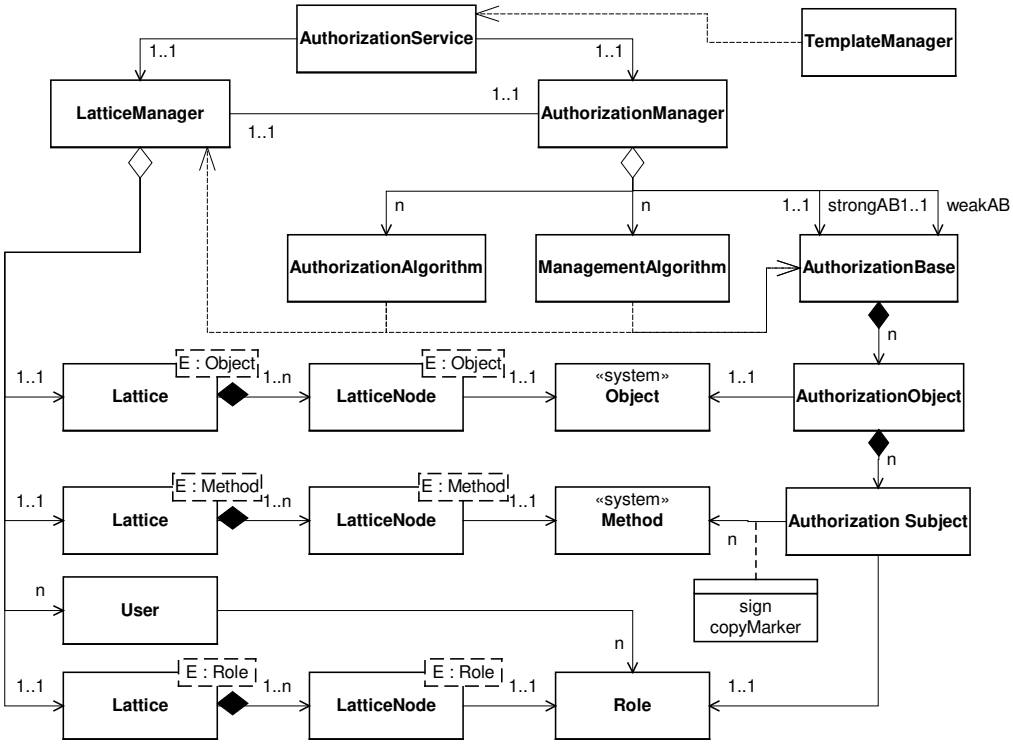

Abbildung 6.21: Überblick der Klassen des Autorisierungsdienstes

6.3.10 Integration des Autorisierungsdienstes

Das zu schützende Informationssystem, beziehungsweise dessen Dienste, benötigen Zugriff auf die Schnittstellen des Autorisierungsdienstes. Damit alle Objekte des Informationssystems zugreifen können, wird der Autorisierungsdienst als globales Objekt instanziiert. Zusätzlich wird eine Schnittstellenklasse definiert, die alle Methoden des Autorisierungsdienstes anbietet und zur Kommunikation mit der Benutzeroberfläche verwendet wird.

In Abbildung 6.22 erbt ein geschützter Dienst von der Schnittstellenklasse `AuthorizationServiceInterface`. Der Dienst kann entweder über die Methoden der Schnittstellenklasse oder direkt auf den globalen Autorisierungsdienst (`anAuthorizationService`) zugreifen. Die Benutzeroberfläche des Autorisierungsdienstes kann ohne Modifikationen in das Informationssystem integriert werden, da ausschließlich Methoden der Schnittstellenklasse aufgerufen werden.

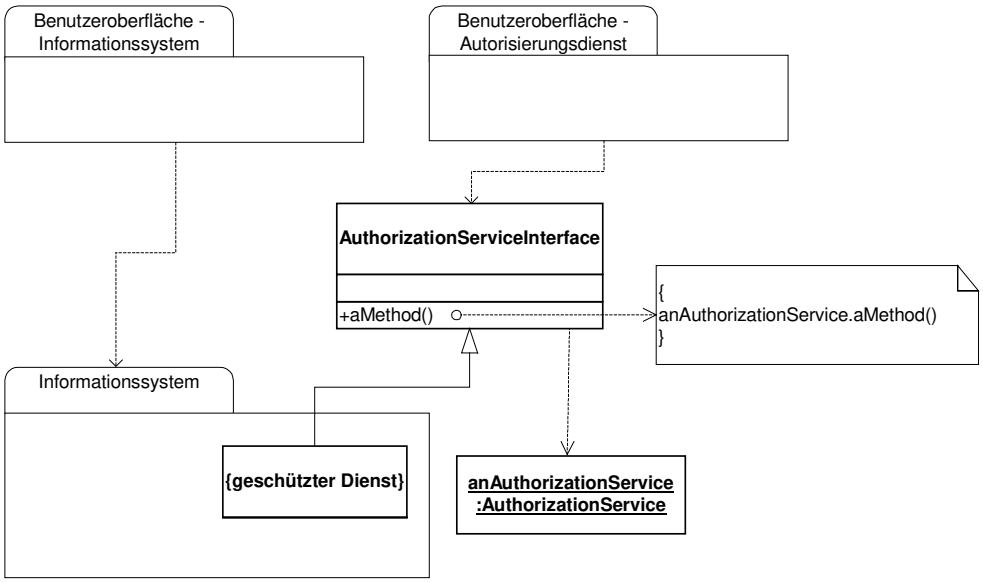

Abbildung 6.22: Integration des Autorisierungsdienstes

Vor der Ausführung geschützter Methoden des Informationssystems muß die `authorize`-Methode zur Autorisierung des Zugriffs aufgerufen werden. Die Durchführung des Aufrufs ist von der Systemumgebung abhängig und kann generell auf zwei Weisen erfolgen: vom System vor der Methodenausführung oder als erste Anweisung innerhalb des Methodenrumpfes. Der erste Ansatz erfordert Modifikationen am System, damit Methodenaufrufe abgefangen und autorisiert werden können. Der Aufruf innerhalb von Methoden ist zwar systemunabhängig, erfordert allerdings die Modifikation aller geschützten Klassen des Informationssystems.

Kapitel 7

Implementierung des Autorisierungsdienstes in Tycoon-2

In diesem Kapitel wird die Implementierung des zuvor entworfenen Autorisierungsdienstes in Tycoon-2 beschrieben. Dabei wird insbesondere auf die Abhängigkeiten des Entwurfs von der Implementierungsumgebung eingegangen. Überdies wird die Funktionsweise der Autorisierungs- und Verwaltungsalgorithmen näher erläutert. Das Kapitel schließt mit einem Vergleich zwischen der Tycoon-2-Implementierung des Autorisierungsdienstes und der Zugriffskontrolle der Java-Programmierumgebung.

7.1 Subjekte, Objekte und Zugriffsmodi

Die Implementierung der Graphen zur Aufnahme der Subjekte, Objekte und Zugriffsmodi erfolgt durch Umsetzung der Klassen `Lattice` und `LatticeNode` (vgl. Abbildung 6.11). Die gemeinsame Benutzerschnittstelle wird in der Klasse `Lattice-Manager` implementiert (vgl. Abbildung 6.13).

Datenstrukturen der Graphen

Die Klasse `Lattice` speichert die Knoten in einem `Dictionary`. Das Tycoon-2-Objekt, welches von der Klasse `LatticeNode` gekapselt wird, dient als Schlüssel für Zugriffe auf das `Dictionary`. Die Namen der Knoten werden in einem zusätzlichen `Dictionary` gespeichert. Dieses wird zur Prüfung der Eindeutigkeit von Namen bei der Hinzufügung von Knoten und zur Benennung von Knoten innerhalb der grafischen Benutzeroberfläche verwendet.

In der Klasse `LatticeNode` werden das zu kapselnde Objekt, sein Name sowie die Vorgänger und Nachfolger im Graphen gespeichert. Zur Speicherung der Vorgänger und Nachfolger wird die ungeordnete Datenstruktur `Set` verwendet, welche Existenzprüfungen erlaubt und sequentiell ausgelesen werden kann.

Algorithmus zur Implikationsprüfung

Der wichtigste Algorithmus der Graphen dient zur Prüfung von Implikationsbeziehungen zwischen Knoten. Er ist in den Methoden `implies` der Klasse `Lattice` sowie `impliesUp` / `impliesDown` der Klasse `LatticeNode` implementiert. Der `implies`-Methode werden als Parameter zwei Knoten übergeben. Als erstes wird ein Identitätsvergleich durchgeführt. Falls dieser erfolgreich verläuft, wird der Integerwert 0 zurückgegeben. Ansonsten ist der zweite Knoten der Ausgangspunkt der weiteren Prüfung.

Den Methoden `impliesUp` und `impliesDown` wird der erste Knoten als Parameter übergeben. Zunächst führen die Methoden mit der eigenen und der übergebenen Instanz einen Identitätsvergleich durch. Falls die Knoten übereinstimmen, wird der Wert `true` zurückgegeben. Anderenfalls wird die Methode `impliesUp` bei den Vorgängern und `impliesDown` bei den Nachfolgern des Knotens rekursiv aufgerufen. Falls kein Identitätsvergleich positiv ausfällt, existiert keine Implikationsbeziehung und es wird der Wert `false` zurückgegeben. Die möglichen Rückgabewerte der Methode `implies` sind:

$$\text{implies}(\text{node1}, \text{node2}) = \begin{cases} 0, \text{falls node1} = \text{node2} \\ 1, \text{falls implies}\mathit{Up}(\text{node1}) = \text{true} \\ -1, \text{falls implies}\mathit{Down}(\text{node1}) = \text{true} \\ \mathit{nil}, \text{sonst} \end{cases}$$

Implementierung der Subjekte, Objekte und Zugriffsmodi

Die Subjekte des Autorisierungsdienstes ergeben sich direkt aus der Implementierung der Klasse `Role`.

Der Objektgraph des Autorisierungsdienstes wird mit dem Typ `Object` parametrisiert. Als Objekte können alle Instanzen von Klassen des Tycoon-2-Systems verwendet werden, da die Klasse `Object` die Wurzel der Tycoon-2-Klassenhierarchie bildet (vgl. Kapitel 6.3.2). Die Klassenobjekte der geschützten TL-2-Klassen werden als Objekte des Typs 2 verwendet. Der Aufruf der `clazz`-Methode gibt das Klassenobjekt des aktuellen Objekts (`self`) zurück:

```
self.clazz
```

Klassenobjekte sind Subklassen von `Object`, so daß sie ohne Modifikationen in den Objektgraphen eingebunden werden können. Es erfolgt deshalb keine Unterscheidung in Typ-1 und Typ-2-Objekte auf Typebene (vgl. Kapitel 6.2.2). Die Implikationsbeziehungen zwischen neuen Objekten und ihren Klassenobjekten werden bei der Hinzufügung der Knoten eingetragen. Auf das Klassenobjekt eines Objekts kann über die Methode `clazz` zugegriffen werden. Die Eintragung der Klassenobjekte erfolgt während der Initialisierung des Objektgraphen.

Die Zugriffsmodi des Autorisierungsdienstes entsprechen den geschützten Methoden der geschützten Klassen. Der Graph der Zugriffsmodi kann Instanzen der Klasse `Method` aufnehmen. Als schützbare Methoden sieht der Entwurf eine Teilmenge der öffentlichen Methoden einer Klasse vor (vgl. Kapitel 6.2.3). Für den Zugriff auf die

Instanzen dieser Methoden werden die reflexiven Eigenschaften des Tycoon-2-Systems ausgenutzt:

```
self.clazz.methodDictionaries[0][Symbol.new("methodName")],
```

Die Klassenmethoden werden unter dem Index 0 in der Feldvariable `methodDictionaries` gespeichert. Der Schlüssel für den Zugriff ist eine Instanz der Klasse `Symbol`, welche mit dem Namen der Methode initialisiert wurde. Das zurückgegebene Objekt ist die Instanz der angegebenen Methode und vom Typ `Method`.

Verwaltung der Graphen und Benutzer

Eine Instanz der Klasse `LatticeManager` initialisiert und speichert die Graphen der Subjekte, Objekte und Zugriffsmodi. Außerdem speichert sie die Benutzer des Autorisierungsdienstes in einem `Dictionary`, welches die Namen der Benutzer als Schlüssel verwendet.

Die Methoden zur Verwaltung der Graphen verweisen auf die entsprechenden Methoden der Klasse `Lattice`. Zusätzliche Logik gibt es lediglich bei der Hinzufügung von Knoten. Wenn für den neuen Knoten kein Name angegeben wird, verwendet der `LatticeManager` die zugehörige Tycoon-2-Objektkennung. Diese wird mit dem Befehl

```
self.identityHash.printString
```

ausgelesen. Die Methode `identityHash` gibt eine Integerwert zurück, welcher mit `printString` in eine Zeichenkette konvertiert wird. Die Speicherung der Objektkennung kann nicht zu Inkonsistenzen mit dem persistenten Objektspeicher führen, da sie in einer Instanz der Klasse `LatticeNode` zusammen mit dem Objekt gespeichert wird. Baumelnde Verweise in Form von ungültigen Objektkennungen können deshalb nicht entstehen.

Eintragung der Superknoten und Subknoten

Die Superknoten und Subknoten der Graphen können während der Initialisierung oder nachträglich festgelegt werden. Zur Initialisierung werden die Knoten der `new`-Methode der Metaklasse von `Lattice` als Parameter übergeben. Die nachträgliche Festlegung erfolgt durch Ausführung der Methoden `setSuperNode` und `setSubNode` der Klasse
`Lattice`.

Als Super- und Subknoten des Subjektgraphen können beliebige Instanzen der Klasse `Role` festgelegt werden. Aus den Rollenbezeichnungen sollte der Verwendungszweck der Knoten hervorgehen. Für Testzwecke wurden dem Superknoten die Bezeichnung `Administrator` und dem Subknoten die Bezeichnung `World` zugewiesen.

Den Zugriffsmodi- und Objektgraphen müssen als Super- und Subknoten Instanzen von Klassen, beziehungsweise Methoden zugewiesen werden. Falls sich aus dem geschützten Dienst keine geeigneten Klassen und Methoden ergeben, können Instanzen der vordefinierten Klassen `SuperNode` und `SubNode` verwendet werden. Die beiden

Klassen enthalten die Methoden `superMethod` und `subMethod`, welche als Super- und Subknoten des Graphen der Zugriffsmodi dienen.

7.2 Autorisierungen

Die Implementierung einer Datenstruktur zur Speicherung von Autorisierungen basiert auf den Klassen `AuthorizationBase`, `AuthorizationObject` und `AuthorizationSubject` (vgl. Abbildung 6.15). Zur Verwaltung und Durchführung von Autorisierungen werden Algorithmen verwendet, die in Subklassen von `AuthorizationAlgorithm` und `ManagementAlgorithm` implementiert sind (vgl. Abbildung 6.16). In den folgenden Abschnitten wird näher auf die verwendete Datenstruktur und die Implementierung der Algorithmen eingegangen.

7.2.1 Datenstrukturen

Der hierarchische Aufbau von Autorisierungen erfordert die Kapselung der Objekte der tieferen Ebene (vgl. Kapitel 6.3.3). Dazu werden Instanzen der Klasse `Dictionary` verwendet:

```
class AuthorizationBase
        private _objects :Dictionary(Object, AuthorizationObject)

class AuthorizationObject
        private _subjects :Dictionary(Role, AuthorizationSubject)

class AuthorizationSubject
        private _accessModes :Dictionary(Method, Int)
```

In der Klasse `AuthorizationBase` werden Instanzen der Klasse `AuthorizationObject` gespeichert. Als Schlüssel für Zugriffe auf das `Dictionary` dient das gekapselte Objekt. Die Klasse `AuthorizationObject` ermöglicht den Zugriff auf Instanzen der Klasse `AuthorizationSubject` über zugeordnete Rollen. Schließlich werden die Methoden in der Klasse `AuthorizationSubject` gespeichert. Sie dienen als Schlüssel für Zugriffe auf das Vorzeichen und die Kopiermarke der Autorisierung, welche als Integerwert gespeichert sind. Aus einem Integerwert i ergeben sich folgende Werte für das Vorzeichen `sign` und die Kopiermarke `copyMarker` (vgl. Kapitel 6.2.6):

$$\text{sign} = \begin{cases} \textit{true}, \text{falls } i > 0 \\ \textit{false}, \text{falls } i < 0 \end{cases} \qquad \text{copyMar ker} = \begin{cases} \textit{false}, \text{falls } |i| = 1 \\ \textit{true}, \text{falls } |i| = 2 \\ \textit{selectable}, \text{falls } |i| = 3 \end{cases}$$

7.2.2 Verwaltungsalgorithmen

Zur Verwaltung der Autorisierungsbasen werden Algorithmen zur Einfügung und Entfernung von Autorisierungen benötigt. Diese Algorithmen sind in den Methoden `insertAccessRight` und `removeAccessRight` der Subklassen von `ManagementAlgorithm` implementiert (vgl. Abbildung 6.16). Sie sichern zu, daß die Ausführung einer Verwaltungsoperation nicht gegen die Bedingungen der Autorisierungsbasen verstößt. Die Bedingungen sind in mehreren Methoden implementiert, die einen Wert vom Typ `Bool` zurückgeben. Eine Verwaltungsoperation wird nur dann durchgeführt, wenn alle relevanten Bedingungen den Wert `true` zurückgeben.

Die schwache und die starke Autorisierungsbasis fordern die Einhaltung unterschiedlicher Bedingungen (vgl. Kapitel 6.2.5.1 und 6.2.5.2), deren Überprüfung durch Algorithmen im folgenden beschrieben wird.

Bedingungen der starken Autorisierungsbasis

Die *Konsistenzbedingung* ist in der Methode `_consistencyProperty` der Klasse `ManagementAlgorithm` implementiert und sichert zu, daß sich aus zwei expliziten Autorisierungen nicht widersprüchliche implizite Autorisierungen ableiten lassen. Vor der Einfügung von Autorisierungen wird deshalb für alle expliziten Autorisierungen und der einzufügenden Autorisierung jeweils geprüft, ob sich gemeinsame Subjekte, Objekte und Zugriffsmodi ableiten lassen. Falls dies zutrifft, und sich die Autorisierungen im Vorzeichen unterscheiden, wird die neue Autorisierung nicht in die Autorisierungsbasis eingefügt und eine Ausnahme mit der Meldung *„consistency check failed“* ausgelöst. Anderenfalls ist die Bedingung erfüllt und die neue Autorisierung darf eingefügt werden. In Abbildung 7.1 ist ein Beispiel zur Konsistenzbedingung dargestellt.

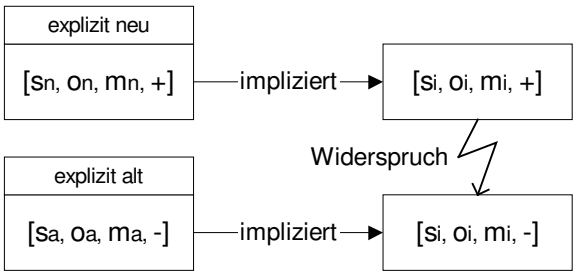

Abbildung 7.1: Beispiel zur Konsistenzbedingung

Aus den expliziten alten und neuen Autorisierungen lassen sich eine gemeinsame implizite Autorisierung ableiten. Da sich die Vorzeichen der expliziten Autorisierungen unterscheiden, und das Vorzeichen bei der Ableitung erhalten bleibt, widersprechen sich die beiden impliziten Autorisierungen. Die neue Autorisierung darf deshalb nicht eingefügt werden.

Die *Redundanzfreiheitsbedingung* soll verhindern, daß bereits implizierte Autorisierungen explizit eingefügt werden. Sie ist in der Methode `_nonredundancyCheck` implementiert, welche die einzutragende Autorisierung an den Autorisierungsalgorithmus

übergibt. Falls das Ergebnis `undecided` ist, wird die Autorisierung nicht impliziert und darf eingetragen werden. Anderenfalls ist sie schon explizit oder implizit enthalten, und es wird eine Ausnahme mit der Meldung *„nonredundancy check failed“* ausgelöst.

Bedingungen der schwachen Autorisierungsbasis

Die *Konsistenzbedingung* der schwachen Autorisierungsbasis beruht auf derjenigen der starken Autorisierungsbasis. Falls die Konsistenzbedingung zwei widersprüchliche Autorisierungen gefunden hat, muß für die schwache Autorisierungsbasis noch geprüft werden, ob sie in einer Implikationsbeziehung zueinander stehen. Ist dies der Fall, liegt nur eine der beiden Autorisierungen innerhalb der Reichweite einer Zugriffsanforderung, und die Konsistenzbedingung ist deshalb nicht verletzt.

Die *Vollständigkeitsbedingung* wird in Kapitel 7.3 „Initialisierung und Konfiguration“ behandelt.

7.2.3 Autorisierungsalgorithmen

Autorisierungsalgorithmen setzen sich aus den Methoden `authorizeObject`, `authorizeSubject` und `authorizeAccessMode` der Klasse `AuthorizationAlgorithm` zusammen. Die Initialisierung von Algorithmen wird in Kapitel 6.3.6 beschrieben.

Autorisierungsalgorithmus der starken Autorisierungsbasis

In Abbildung 7.2 ist der Ablaufplan des Autorisierungsalgorithmus der starken Autorisierungsbasis dargestellt. Die Durchführung der Autorisierung beginnt mit dem Aufruf der Methode `authorizeObject`. Zunächst wird geprüft, ob das Objekt der Zugriffsanforderung in der starken Autorisierungsbasis enthalten ist. Falls dies zutrifft, wird die Methode `authorizeSubject` aufgerufen. Sonst wird iterativ für die Objekte der Autorisierungsbasis geprüft, ob sie das Objekt der Zugriffsanforderung implizieren. Wenn eine Implikationsbeziehung gefunden wird, erfolgt der Aufruf der Methode `authorizeSubject`. Anderenfalls kann die Zugriffsanforderung nicht autorisiert werden und es wird der Wert `undecided` zurückgegeben.

Die Ausführung der Methode `authorizeSubject` verläuft analog zu `authorizeObject`. Falls das Subjekt der Anforderung im `AuthorizationObject` enthalten ist, oder von einem enthaltenen Subjekt impliziert wird, erfolgt der Aufruf von `authorizeAccessMode`. Ansonsten wird in die Methode `authorizeObject` zurückgesprungen und mit dem Auslesen der Objekte fortgefahren.

In der Methode `authorizeAccessMode` wird das Vorzeichen des enthaltenen oder eines implizierten Zugriffsmodus ausgelesen und geprüft, ob die Richtungen der Implikationsbeziehungen zwischen dem Subjekt, Objekt und Zugriffsmodus zum Vorzeichen passen (vgl. Regeln zur Ableitung starker Autorisierungen aus Kapitel 6.2.5.3). Wenn kein passender Zugriffsmodus im `AuthorizationSubject` enthalten ist, wird mit das Auslesen der Subjekte fortgeführt. Anderenfalls wird das Vorzeichen als Ergebnis der Autorisierung zurückgegeben und gestattet (`true`) oder verweigert (`false`) den Zugriff.

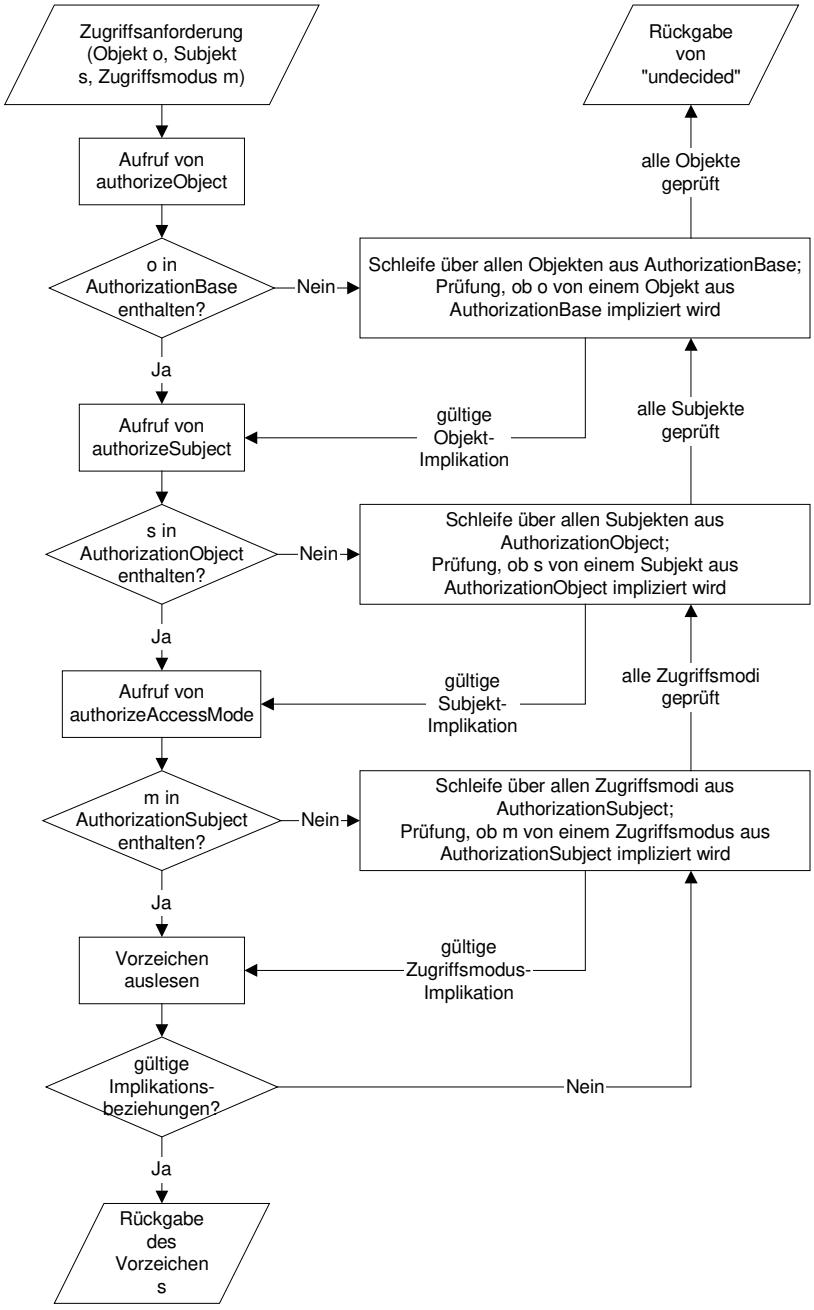

Abbildung 7.2: Ablaufplan des starken Autorisierungsalgorithmus

Autorisierungsalgorithmus der schwachen Autorisierungsbasis

Der Autorisierungsalgorithmus der schwachen Autorisierungsbasis unterscheidet sich in zwei Punkten vom starken Autorisierungsalgorithmus:

⇒ Das Ergebnis der Autorisierung ist immer `true` oder `false` und nicht `undecided` (vgl. Kapitel 6.2.5.2).
⇒ Bei der Ableitung impliziter Autorisierungen muß die Reichweite (*scope*, vgl. Kapitel 6.2.5.3) einer expliziten Autorisierung berücksichtigt werden.

Weil schwache Autorisierungen überschrieben werden dürfen, endet deren Reichweite bei einer spezifischeren Autorisierung. Der Autorisierungsalgorithmus darf daher nicht die erste gefundene explizite Autorisierung, welche die Zugriffsanforderung impliziert, auswerten. Vielmehr müssen alle in Frage kommenden expliziten Autorisierungen gefunden werden und das Vorzeichen derjenigen Autorisierung, die von keiner der anderen Autorisierungen impliziert wird, zurückgegeben werden.

Der schwache Autorisierungsalgorithmus durchläuft den Ablaufplan des starken Autorisierungsalgorithmus solange, bis alle expliziten Autorisierungen, welche die Zugriffsanforderung implizieren, gefunden und in eine Lösungsmenge eingetragen sind. Danach wird die Lösungsmenge nach einer Autorisierung durchsucht, die keine der anderen Autorisierungen impliziert. Die gefundene Autorisierung wird daher nicht überschrieben und ihr Vorzeichen wird als Ergebnis zurückgegeben.

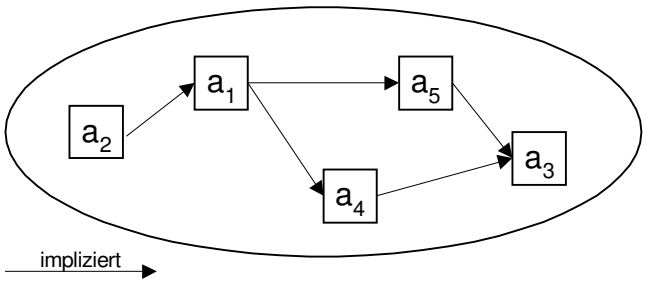

Abbildung 7.3: Beispiel zur Reichweitenauswertung

Das Beispiel in Abbildung 7.3 zeigt eine Lösungsmenge mit fünf expliziten Autorisierungen. Die Überprüfung der Implikationsbeziehungen zeigt, das die dritte Autorisierung keine der anderen Autorisierungen impliziert. Ihr Vorzeichen wird deshalb als Ergebnis des Autorisierungsalgorithmus zurückgegeben.

7.3 Initialisierung und Konfiguration

Die Initialisierung des Autorisierungsdienstes erfolgt durch Bindung einer Instanz der Klasse `AuthorizationService` an eine globale Variable:

```
define authorizationService :AuthorizationService;
authorizationService := AuthorizationService.new;
```

In der Methode `new` werden Instanzen der Klassen `LatticeManager` und `AuthorizationManager` erzeugt und an private Variablen des `authorizationService` gebunden. Während der Initialisierung des `LatticeManager` erfolgt die Auswertung der Textdatei `classes_methods`. Dies führt zur Eintragung von Klassenobjekten in den Objektgraphen und zugehöriger Methoden in den Graphen der Zugriffsmodi.

Zum Abschluß der Konfiguration müssen Rollen in den Subjektgraphen eingetragen und für alle Graphen Implikationsbeziehungen spezifiziert werden. Außerdem muß zur Einhaltung der *Vollständigkeitsbedingung* der schwachen Autorisierungsbasis eine Autorisierung eingefügt werden, die alle Zugriffsanforderungen impliziert. Wenn das *least privilege*-Konzept angewendet wird, muß die Autorisierung negativ sein, damit alle Zugriffsanforderungen verweigert werden. Dazu wird über den Superknoten des Subjekt- und Objektgraphen und dem Subknoten des Graphen der Zugriffsmodi folgende schwache Autorisierung spezifiziert:

(super_subject, super_object, sub_accessMode, –, w, false)

Diese Autorisierung verweigert allen Benutzern den Zugriff, solange sie nicht von anderen schwachen oder starken Autorisierungen überschrieben wird.

7.4 Integration in Tycoon-2-Anwendungen

Die Integration des Autorisierungsdienstes in Tycoon-2-Anwendungen erfordert einige Modifikationen an den Klassen der Anwendungen, die im folgenden beschrieben werden (vgl. Kapitel 6.3.10).

Eintragung neuer Objekte

Neue Instanzen von geschützten Klassen müssen in den Objektgraphen des Autorisierungsdienstes eingetragen werden. Damit dies automatisch während der Erzeugung von Instanzen geschieht, werden die Fabrikmethoden[65] der zugehörigen Metaklassen modifiziert. Das folgende Beispiel zeigt die `new`-Methode der Metaklasse von `MyClass`, welche eine Instanz erzeugt und diese beim Autorisierungsdienst einträgt.

[65] Eine Fabrikmethode erzeugt Instanzen von Klassen (vgl. [GHJV95]).

```
new() :MyClass
        {
        let instance = _new,
        instance._init(),
        authorizationService.insertObject(instance, "name"),
        instance
        }
```

Der Autorisierungsdienst ist an die globale Variable `authorizationService` gebunden. Der Aufruf der Methode `insertObject` führt zur Eintragung des Objektes `instance` vom Typ `MyClass` unter der Bezeichnung `name`.

Aufruf des Autorisierungsalgorithmus

Bevor eine geschützte Methode ausgeführt werden darf, muß der Autorisierungsalgorithmus des Autorisierungsdienstes aufgerufen werden. Dieser Aufruf kann vom Tycoon-2-System oder dem geschützten Dienst aus erfolgen. Die erste Variante erfordert die Modifikation des Tycoon-2-Systems. Damit die Portabilität von Objektspeichern erhalten bleibt, und der Autorisierungsdienst in jedem Tycoon-2-System eingesetzt werden kann, wird diese Alternative nicht weiter verfolgt. Die zweite Variante ermöglicht die Integration des Aufrufs in die Methoden geschützter Dienste. Das folgende Beispiel zeigt den Aufruf der `authorize`-Methode als erste Anweisung innerhalb des Rumpfes der geschützten Methode `protectedMethod`.

```
protectedMethod() :Void
   {
   authorizationService.authorize(
       user,
       self, (* object *)
       self.clazz.methodDictionaries[0][Symbol.new("protectedMethod")]),
   (* body *)
   ...
   }
```

Die Methode `authorize` der Klasse `AuthorizationService` erwartet als Parameter das Subjekt, das Objekt und den Zugriffsmodus. Das Subjekt entspricht einer Instanz der Klasse `User`. Die `User`-Instanz des zugreifenden Benutzers wird während der Authentifizierung anhand der Zertifikatsdaten identifiziert und innerhalb des Dienstes oder in einer globalen Variablen gespeichert. Das Objekt entspricht der aktuellen Instanz und muß daher über `self` referenziert werden. Als Zugriffsmodus wird eine Instanz der Klasse `Method` erwartet. Der erforderliche reflexive Zugriff auf die Klassendefinition wurde in Kapitel 7.1 erläutert.

Wenn die Autorisierung negativ verläuft, wird eine Ausnahme ausgelöst, welche die Ausführung der geschützten Methode abbricht. Anderenfalls werden die übrigen Anweisungen des Methodenrumpfes ausgeführt.

Die geschützten Methoden werden an Subklassen vererbt. Falls eine geschützte Methode in einer Subklasse überschrieben wird, muß als erste Anweisung der Aufruf der

`authorize`-Methode erhalten bleiben. Dies kann durch Ausführung der super-Methode oder durch expliziten Aufruf von `authorize` mit den benötigten Parametern erfolgen.

Benutzeroberfläche

Die Administration des Autorisierungsdienstes erfolgt über das Internet. Dazu werden STML-Dokumente erstellt, welche Aufrufe von Methoden der Schnittstellenklasse `AuthorizationServiceInterface` beinhalten. Die STML-Dokumente zur Administration können nahtlos in die Benutzeroberfläche der geschützten Dienste eingebunden werden.

7.5 Vergleich mit der Zugriffskontrolle von Java

In diesem Kapitel wird die Tycoon-2-Implementierung des Autorisierungsdienstes mit den Zugriffskontrollmechanismen der Java-Programmierumgebung verglichen. Java bietet sich für einen Vergleich an, da es sich um eine weit verbreitete Plattform für Internetprogramme handelt. Im folgenden wird das Sicherheitsmodell des ***J**ava **D**evelopment **K**it (JDK)* der Version 1.2 betrachtet, welches von den Entwicklern in [Gong98, GMPS97 und Sun98] beschrieben wird.

Sicherheitsmodell des JDK 1.2

Bei dem Entwurf des Sicherheitsmodells wurden einige grundlegende Konzepte wie Zugriffskontrollisten *(access control lists)* (vgl. HRU-Zugriffskontrollmatrix aus Kapitel 5.2) und Schutzdomänen *(protection domain)* (vgl. [Salt74]) berücksichtigt. Einen Überblick des Sicherheitsmodells gibt Abbildung 7.4 [Gong98].

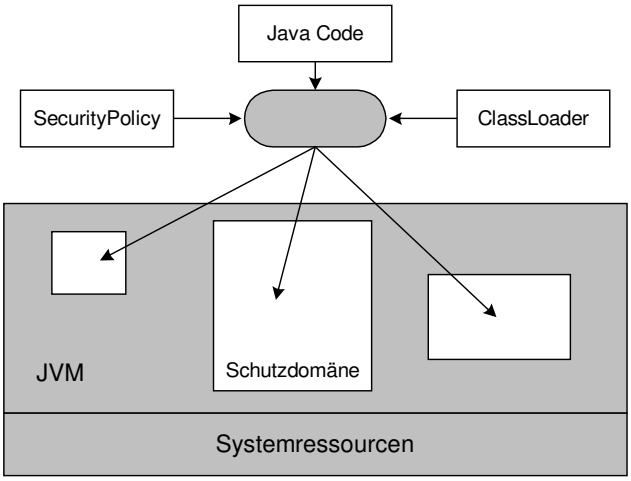

Abbildung 7.4: Sicherheitsmodell des JDK 1.2

Das primäre Ziel des Sicherheitsmodells ist die Kontrolle der Zugriffe von Java Code auf Systemressourcen. Dadurch soll das System der Benutzer vor unerwünschten Zugriffen durch fremde Javaprogramme geschützt werden.

Die Zugriffskontrolle basiert auf der Spezifikation von Sicherheitsrichtlinien (*SecurityPolicy*). Sicherheitsrichtlinien können als Textdatei gespeichert werden und bestehen aus Zugriffskontrollisten. Eine Zugriffskontrolliste beinhaltet für ein Subjekt die erlaubten Kombinationen aus Aktionen, Zielen und Zugriffstypen *(permissions)*. Als Subjekt kann der Speicherort oder der Programmierer des Java Codes eingetragen werden. Während sich der Speicherort direkt aus der Internetverbindung ergibt, werden zur Identifizierung der Programmierer Zertifikate verwendet.

Zugriffstypen ergeben sich aus der Subtypisierung der Klasse `Permission`. Deren Subklassen orientieren sich an den Ressourcentypen der Systemumgebung: zum Beispiel `FilePermission` oder `SocketPermission`. Für Zugriffe des Typs `FilePermission` sind mögliche Aktionen `read` oder `write` und das Ziel gibt den Pfad und Namen der Datei an.

Während des Ladevorgangs eines Javaprogrammes werden seine Zugriffsrechte aus der Sicherheitsrichtlinie ausgelesen und von der ***J**ava **V**irtuellen **M**aschine (JVM)* eine Schutzdomäne erzeugt, innerhalb der das Javaprogramm initialisiert wird.

Die Autorisierung von Zugriffen erfolgt durch Aufruf der Methode `checkPermission` des `AccessControllers`. Sie nimmt als Parameter einen Subtyp der Klasse `Permission` entgegen. Das `Permission`-Objekt enthält die Bezeichnungen des angeforderten Ziels und der auszuführenden Aktion. Falls in den Zugriffsrechten eine Erlaubnis enthalten ist, wird die Ausführung des Programms fortgesetzt. Anderenfalls wird eine Ausnahme ausgelöst.

Zur flexibleren Modellierung von Zugriffsrechten können Implikationsbeziehungen zwischen `Permissions` spezifiziert werden. Dazu implementiert jede Subklasse von `Permission` die Methode `implies`, welche als Parameter eine `Permission` entgegennimmt und einen Wahrheitswert zurückgibt. Außerdem besteht die Möglichkeit, `Permissions` in Gruppen zusammenzufassen. Die Klasse `PermissionCollection` gruppiert `Permissions` gleichen Typs, während die Klasse `Permissions` alle Subtypen von `Permission` akzeptiert.

Vergleich der Zugriffskontrollen

Ein grundlegender Unterschied zwischen den Zugriffskontrollen des Autorisierungsdienstes und der Java-Umgebung besteht in der Zielsetzung: In Java wird der Benutzer vor den Aktionen von Javaprogrammen geschützt, während der Autorisierungsdienst in Tycoon-2 die Objekte vor Benutzerzugriffen schützt.

Der Ansatz in Java ergibt sich aus der Verwendung von Applets, die auf den Systemen der Benutzer ausgeführt werden und nur beschränkten Systemzugriff haben sollen. Bei der Realisierung von Internet-Informationssystemen wird Java allerdings auch auf dem Server eingesetzt werden, so daß die Objekte der Anwendung vor Benutzerzugriffen geschützt werden müssen. Die dazu benötigten Sicherheitsmechanismen können auf den existierenden basieren, indem sich statt Programmierer die Benutzer einer Anwendung über Zertifikate authentifizieren und ihre Zugriffsrechte in Sicherheitsrichtlinien eingetragen werden.

Die Zugriffskontrollisten der Sicherheitsrichtlinien entsprechen den Autorisierungen in Tycoon-2. Die Identifikation von Subjekten, Objekten (Zielen) und Zugriffsmodi (Aktionen) erfolgt anhand von Bezeichnungen und nicht auf objektorientierter Basis. Damit Instanzen von Klassen und deren Methoden geschützt werden können, benötigen sie eindeutige Bezeichnungen, die zur Spezifikation von Zugriffsrechten dienen. Dieses Konzept ermöglicht Inkonsistenzen, da Objekte unabhängig von ihren Bezeichnern in Zugriffskontrollisten existieren.

Es können nur Zugriffsberechtigungen vergeben werden. Weiterführende Konzepte wie positive / negative und starke / schwache Autorisierungen sind im Sicherheitsmodell von Java nicht implementiert. Die Implikationsbeziehungen gelten immer für gesamte Autorisierungen. Eine getrennte Ableitung von Zugriffsrechten aus Subjekten, Objekten und Zugriffsmodi ist deshalb nicht möglich, so daß auch keine Graphen für deren Verwaltung benötigt werden. Es gibt keine zentrale Verwaltung der Implikationsbeziehungen, da sie innerhalb von `Permission`-Objekten gekapselt sind.

Der Autorisierungsaufruf erfolgt in beiden Systemen über ein globales Objekt, das Parameter entgegennimmt und abhängig von deren Auswertung eine Ausnahme auslöst oder die weitere Ausführung der Methode zuläßt.

Zusammenfassend ergibt sich, das das Sicherheitsmodell von Java aufgrund fehlender objektorientierter Eigenschaften zum Schutz von Anwendungen auf der Basis von Methoden nur bedingt geeignet ist. Die Objektidentifikation über Bezeichner und die dezentrale Verwaltung von Implikationen schränken den Nutzen des Modells auf den selektiven Schutz von Systemressourcen ein. Die homogene Integration des Autorisierungsdienstes in Tycoon-2 und seine Flexibilität in Bezug auf die Spezifikation von Autorisierungen kann das Sicherheitsmodell des JDK 1.2 nicht bieten.

Kapitel 8

Integration der realisierten Zugriffskontrollmechanismen

Der SSL-Webserver und der Autorisierungsdienst werden zur Gewährleistung von Authentifizierung, Verschlüsselung und Autorisierung von Zugriffen exemplarisch in einen Dienst des Marinfo-Projektes integriert.

Zunächst erfolgt die Vorstellung einiger Dienste der Marinfo-Projektpartner *Germanischer Lloyd* und ***B**ritish **M**aritime **T**echnology (BMT).* Anschließend werden der Entwurf und die Realisierung einer Zugriffskontrolle für die *BMT Abstract Database* durchgeführt.

8.1 Ausgewählte Dienste des Marinfo-Projektes

Die Dienste des Marinfo-Projektes sind über eine Website[66] zu erreichen, die während der Entwicklungsphase vom Arbeitsbereich STS der TU Harburg verwaltet wird. Auf die meisten Dienste kann unbeschränkt zugegriffen werden, da sie der freien Informationsverbreitung dienen (z.B. Konferenzkalender, Glossardatenbank). Zukünftig sollen vermehrt Geschäftsprozesse der Marinfo-Partner über das Internet abgewickelt werden. Die Geschäftsprozesse dürfen nur für Geschäftspartner der Unternehmen zugreifbar sein und müssen deshalb mit Zugriffskontrolle versehen werden.

Im folgenden werden geplante Dienste des *Germanischen Lloyd* und ein bereits realisierter Dienst der Firma ***B**ritish **M**aritime **T**echnology (BMT)* beschrieben, die Geschäftsprozesse der Unternehmen repräsentieren.

Dienste des Germanischen Lloyd

Der *Germanische Lloyd* ist ein international tätiges Unternehmen zur Klassifizierung von Schiffen. Geschäftspartner und Mitarbeiter des *Germanischen Lloyd* sollen über

[66] <http://www.sts.tu-harburg.de/projects/MARINFO/MARINFO>

das Internet auf ausgewählte Geschäftsprozesse zugreifen können. Im folgenden werden zwei Dienste beschrieben, deren Umsetzung im Rahmen des Marinfo-Projektes geplant ist.

Unterstützung der Zeichnungsprüfung

Damit ein Schiffsneubau entsprechend den Vorschriften des *Germanischen Lloyd* klassifiziert werden kann, müssen von der Bauwerft zahlreiche Unterlagen zur Prüfung und Genehmigung eingereicht werden. Der zu realisierende Dienst soll eine Liste einzureichender bzw. eingereichter Dokumente anschaulich darstellen und dem Kunden die Abfrage des Bearbeitungszustandes seiner Dokumente ermöglichen.

Der Zugriff auf den Dienst soll kontrolliert werden und die Vertraulichkeit und Integrität übertragener Daten muß gewährleistet sein. Die Benutzer des Dienstes werden in Gruppen mit unterschiedlichen Zugriffsrechten eingeteilt.

Zugriff auf administrative, technische und operative Schiffsdaten

Das Schiffsregister des *Germanischen Lloyd* enthält strukturierte Informationen über sämtliche Schiffe, die vom *Germanischen Lloyd* klassifiziert worden sind. Es wird in administrative Informationen (z.B. Schiffsname, Eigner, Heimathafen) und technische Angaben zum Schiff (z.B. Länge, Tiefgang, Ladung) unterteilt. Das Schiffsregister ist Bestandteil eines umfassenderen Informationssystems zur Verwaltung operativer Daten. Zu den operativen Daten zählen z.B. Statusinformationen, Berichte und Besichtigungstermine der fahrenden Flotte. Auf diese administrativen, technischen und operativen Informationen soll über einen Internetdienst zugegriffen werden.

Die geforderte Funktionalität der Zugriffskontrolle entspricht der des zuvor beschriebenen Dienstes.

BMT Abstract Database

Die Firma *British Maritime Technology* betreibt eine Datenbank in der Inhaltsbeschreibungen *(Abstracts)* maritimer Reports gespeichert werden. Kunden von BMT können auf die Inhalte der Datenbank zugreifen. Zur Recherche kann eine CD-ROM genutzt werden, welche die Datenbank enthält, oder es wird auf vierteljährlich erscheinende Ausdrucke der Neuzugänge zurückgegriffen. Außerdem kann über eine Telnet-Sitzung auf den Host zugegriffen und die Datenbank abgefragt werden.

Im Rahmen des Marinfo-Projektes wurde die Datenbank als *BMT Abstract Database* in das Internet-Informationssystem integriert. Hieraus ergeben sich zahlreiche Vorteile, da Kunden über standardisierte Schnittstellen (Webbrowser mit HTML-Seiten) auf die Datenbank zugreifen können. Das Internet bietet flexible Präsentationsmöglichkeiten und direkte Interaktion mit den Kunden. Zur Erweiterung des Angebotes können zusätzliche Internet-Dienste integriert werden.

Zur Zeit bietet die *BMT Abstract Database* Suchschnittstellen an, die eine Volltextsuche und eine Suche auf der Basis ausgezeichneter Felder (z.B. Autor, Sprache, Schlüsselwörter) unterstützt. Die Suchergebnisse werden dem Benutzer seitenweise angezeigt.

Zukünftig soll die *BMT Abstract Database* Funktionalität zur Abrechnung von Datenbankzugriffen beinhalten. Dazu müssen mehrere Abrechnungsmodelle unterstützt werden, die beispielsweise den Einzelzugriff und Zugriff über Abonnements unterscheiden.

Die Grundlage der Diensterweiterung bildet Funktionalität zur Authentifizierung von Kunden, Verschlüsselung von Daten und Autorisierung von Zugriffen. Auf die Realisierung von Abrechnungsmodellen wird im Rahmen dieses Beispiels nicht näher eingegangen.

8.2 Architektur der *BMT Abstract Database*

Die *BMT Abstract Database* basiert auf dem Produkt *Tycoon AdBase* der Firma *Higher-Order*. Das AdBase-System besteht aus einer WAIS-Datenbank, einem Tycoon-2-Webserver und der AdBase-Anwendung mit STML-Benutzeroberfläche. Der Einsatz des Systems als *BMT Abstract Database* erfordert lediglich die Speicherung der Daten in der WAIS-Datenbank und die Erstellung von STML-Dokumenten, die als Schnittstelle zwischen Benutzeroberfläche und Anwendung dienen. [Rich97a, Rich97b]

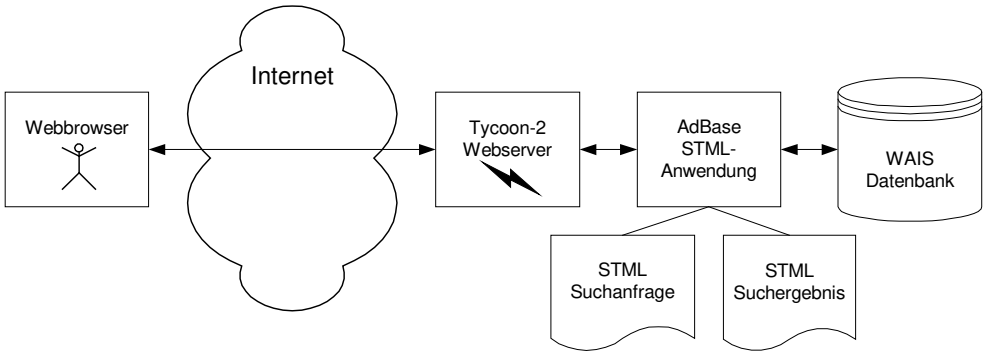

Abbildung 8.1: Architektur des AdBase-Systems

Ein Kunde fordert zunächst beim Webserver das Dokument STML-Suchanfrage an, in das er Suchbegriffe eingeben kann. Das ausgefüllte Formular wird zurückgesendet, und der Webserver leitet die Bearbeitung der Formulardaten durch die AdBase-STML-Anwendung ein. Basierend auf den Formulardaten wird eine Suchanfrage generiert und der WAIS-Datenbank übergeben. Die Suchergebnisse werden von der Anwendung aufbereitet und in das Dokument STML-Suchergebnis eingetragen, welches dem Kunden übermittelt wird.

8.3 Entwurf der Zugriffskontrolle

Nachdem die Benutzer des Dienstes anhand ihrer Zertifikate identifiziert wurden, soll der Zugriff auf die *Abstracts* auf der Basis unterschiedlicher Abrechnungsmodelle

autorisiert werden können. Weil die Integration von Abrechnungsmodellen keine umfangreichen Modifikationen erfordern soll, muß die Vergabe von Autorisierungen auf einem generischen Konzept beruhen. Außerdem müssen Autorisierungen für *Abstracts* gruppiert werden können. Dazu bietet sich die Einführung übergeordneter Objekte an, die in Implikationsbeziehungen zu *Abstracts* stehen. Als übergeordnete Objekte kommen beispielsweise Instanzen von Klassen in Frage, die einen Zeitraum oder ein Themengebiet repräsentieren. In Abbildung 8.2 ist der Verlauf von Zugriffen auf *Abstracts* dargestellt.

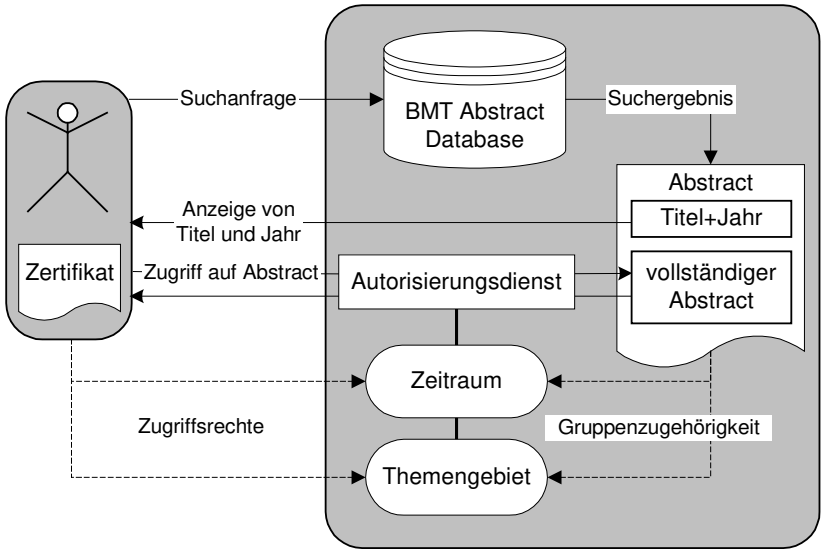

Abbildung 8.2: Überblick der Zugriffskontrolle

Die *Abstracts* der Datenbank sind Gruppen zugeordnet, die in der Abbildung durch Zeiträume und Themengebiete repräsentiert werden. Benutzer können auf der Basis diverser Abrechnungsmodelle Zugriffsrechte auf diese Gruppen erlangen. Nachdem sich ein Benutzer mit seinem Zertifikat identifiziert hat, kann er die Datenbank nach *Abstracts* durchsuchen. Als Suchergebnis werden die Titel und Veröffentlichungsjahre gefundener *Abstracts* angezeigt. Der Benutzer kann für angezeigte Titel die Übertragung des vollständigen *Abstracts* anfordern. Diese Zugriffsanforderung wird vom Autorisierungsdienst überprüft, indem Übereinstimmungen zwischen den Gruppenzuordnungen angeforderter *Abstracts* und den Zugriffsrechten des Benutzers gesucht werden. Falls ein entsprechendes Zugriffsrecht existiert, wird der angeforderte *Abstract* übertragen.

8.4 Realisierung der Zugriffskontrolle

Damit BMT ihren Kunden die geplanten Dienste der *BMT Abstract Database* gezielt und sicher anbieten kann, müssen Dienste zur Authentifizierung, Verschlüsselung und Autorisierung eingebunden werden. Die dazu erforderlichen Erweiterungen und Modifikationen des Systems werden nachfolgend beschrieben.

8.4.1 Authentifizierung und Verschlüsselung

Zur Integration von Funktionalität zur Authentifizierung und Verschlüsselung in die *BMT Abstract Database* wird der Tycoon-2-Webserver des AdBase-Systems gegen den SSL-Webserver ausgetauscht.

Die erforderlichen Modifikationen beschränken sich auf die Ergänzung der SSL-Parameter in der bestehenden Konfigurationsdatei, der Anpassung der Klassenbezeichner im Initialisierungsskript des Servers und einer zusätzlichen Vererbungsbeziehung.

In der Konfigurationsdatei müssen die Pfade zum Server- und CA-Zertifikat angegeben und der Parameter `ClientVerify` auf „1“ gesetzt werden. Zur prototypischen Realisierung des Systems werden bei der Firma *CryptSoft*[67] ein Serverzertifikat und einige Klientenzertifikate beantragt, die allerdings nur zu Demonstrationszwecken verwendet werden dürfen. Außerdem wird noch das Zertifikat von *CryptSoft* benötigt, welches in der `CACert`-Datei abgelegt werden muß. Für spätere kommerzielle Einsätze müssen Zertifikate von einer anerkannten Zertifikatsautorität bezogen werden. Im Initialisierungsskript muß die Klasse `Httpd` durch `Httpsd` ersetzt werden. Damit STML-Anwendungen auf die Daten aus Klientenzertifikaten zugreifen können, muß der STML-Prozessor der Anwendung von der Klasse `SslStmlProcessor` erben (vgl. Abbildung 8.3).

8.4.2 Autorisierung

Der Autorisierungsdienst beruht auf dem Schutz von Objekten und deren Methoden. Bei der Integration des Dienstes in die *BMT Abstract Database* ergibt sich das Problem, daß die *Abstracts* nicht als Tycoon-2-Klassen realisiert sind. Sie werden während der Bearbeitung von Suchergebnissen in transienten Instanzen der Klasse `ClassifiedAd` gekapselt. Weiterhin fehlen Datenstrukturen, die zur Spezifikation von Autorisierungen für Gruppen von *Abstracts* verwendet werden können.

Die Integration des Autorisierungsdienstes erfordert daher Modifikationen an der Anwendungsarchitektur der *BMT Abstract Database*, beziehungsweise des *Tycoon AdBase*-Systems, die im Klassendiagramm aus Abbildung 8.3 dargestellt sind.

Die Klasse `AdBaseApplication` erbt von der Klasse `AuthorizationServiceInterface` und stellt die Funktionalität zur Kommunikation mit dem Autorisierungsdienst zur Verfügung. Aus der Spezialisierung der Klasse `AdBaseApplication` ergibt sich `BMApplication`, welche die Hauptklasse der *BMT Abstract Database* ist[68]. Wie zuvor beschrieben, ist `SslStmlProcessor` die Superklasse von `AdBaseProcessor`.

---

[67] *CryptSoft* wird von den SSLeay-Entwicklern E.A. Young und T.J. Hudson geleitet und ist in der Sicherheitsberatung tätig (<http://www.cryptsoft.com>).

[68] Die Vererbung erfolgt indirekt über die Klassen `GeneralPasswdApplication` oder `RemoteIPApplication`.

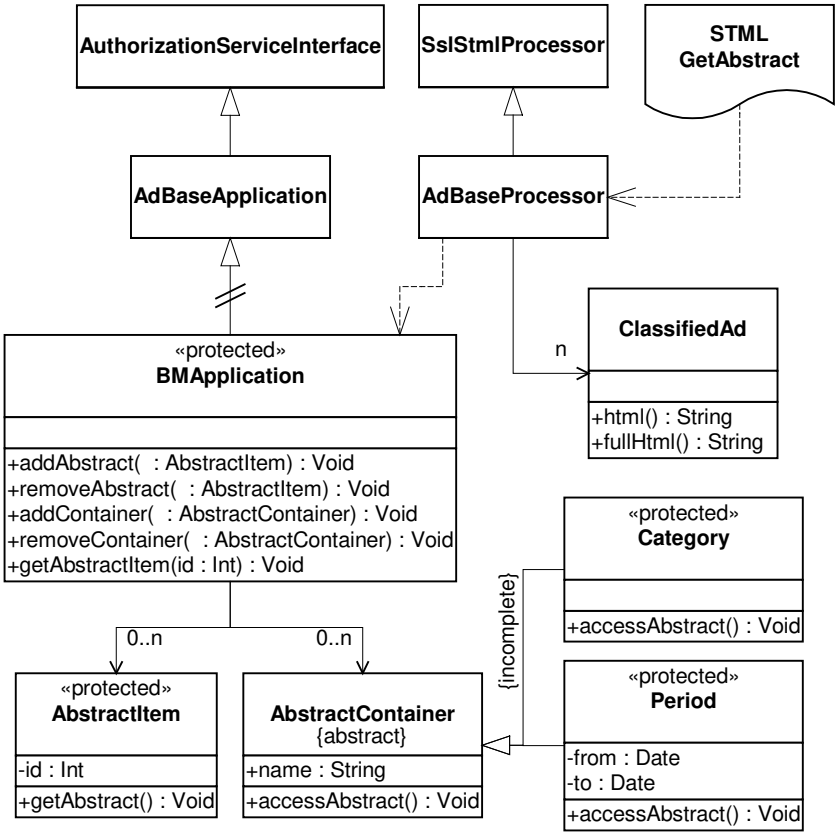

Abbildung 8.3: Modifikation der *BMT Abstract Database*

Damit Autorisierungen für *Abstracts* vergeben werden können, wird in Instanzen der Klasse `AbstractItem` für jeden *Abstract* der WAIS-Datenbank seine Kennung (`id`) gespeichert. Zur Gruppierung von *Abstracts* werden Subklassen von `AbstractContainer` verwendet. Im Klassendiagramm sind beispielhaft die Klassen `Category` und `Period` dargestellt, welche die Gruppierung auf der Basis von Themenbereichen und Veröffentlichungszeiträumen erlauben. Zur Unterstützung anderer Konzepte können weitere Subklassen von `AbstractContainer` definiert werden. Die Zuordnung von *Abstracts* zu Gruppen erfolgt nicht über Assoziationen zwischen Instanzen, sondern innerhalb des Autorisierungsdienstes über Implikationsbeziehungen des Objektgraphen. Die Klassen, deren Instanzen in den Objektgraphen eingetragen werden, sind mit dem Stereotyp `protected` ausgezeichnet.

Zur Benutzeroberfläche wird das Dokument `STML-GetAbstract` hinzugefügt. Es übernimmt aus dem Dokument `STML-Suchergebnis` die `id` des angeforderten *Abstracts* und übergibt sie der Methode `getAbstractItem` der Klasse `BMApplication`, welche die geschützte Methode `getAbstract` der Klasse `AbstractItem` aufruft und den Autorisierungsprozeß auslöst. Falls die Autorisierung erfolgreich verläuft, wird vom Dokument `STML-GetAbstract` die Methode `fullHtml` der Klasse `ClassifiedAd` aufgerufen, welche den *Abstract* im HTML-Format in das Dokument einträgt. Danach erfolgt die Übertragung des Dokuments.

8.4.3 Konfiguration des Autorisierungsdienstes

Bevor die Zugriffskontrolle der *BMT Abstract Database* verwendet werden kann, muß die Datenbasis des Autorisierungsdienstes aufgebaut werden. Sie setzt sich aus den Benutzern, den Graphen der Subjekte, Objekte und Zugriffsmodi, sowie der starken und schwachen Autorisierungsbasis zusammen.

Der erste einzutragende Benutzer ist der Administrator der *BMT Abstract Database*. Er wird mit dem in seinem Zertifikat enthaltenen Namen beim Autorisierungsdienst registriert.

Zur Initialisierung der Graphen werden die vordefinierten Super- und Subknoten verwendet. In den Objektgraphen werden die Instanz der Klasse `BMApplication` und die Metaklassen der geschützten Klassen `AbstractItem`, `Period` und `Category` eingetragen. Der weitere Aufbau des Rollen- und Objektgraphen ist von dem Inhalt der Datenbank abhängig und wird exemplarisch im nächsten Kapitel beschrieben. Die `accessAbstract`-Methoden der Subklassen von `AbstractContainer` und die `getAbstract`-Methode der Klasse `AbstractItem` werden in den Graphen der Zugriffsmodi eingetragen. Nach der Spezifikation einiger Implikationen ergeben sich die drei Graphen aus Abbildung 8.4.

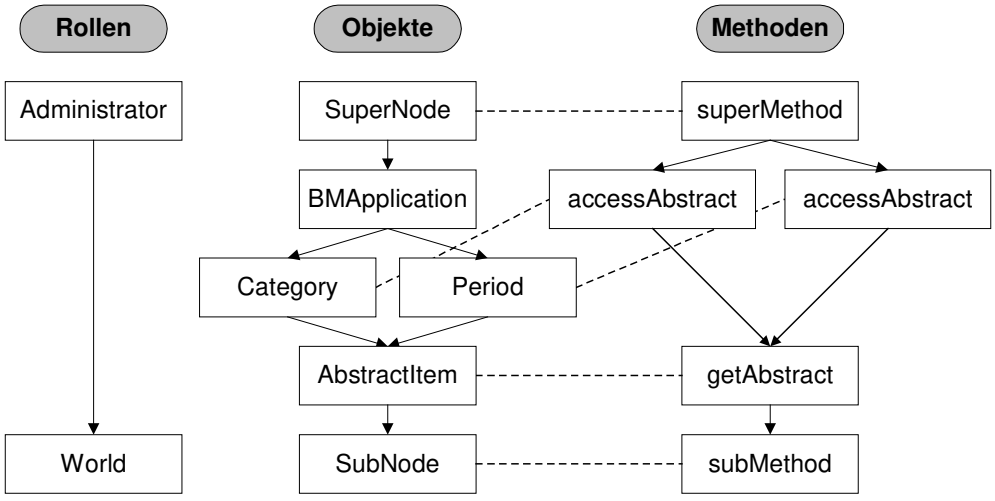

Abbildung 8.4: Konfiguration der Graphen

In die Autorisierungsbasen werden während der Konfiguration die folgenden beiden Autorisierungen eingetragen:

(Administrator, SuperNode, subMethod, -, w, false)

(Administrator, SuperNode, superMethod, +, s, selectable)

Die schwache Autorisierung sichert die Einhaltung der Vollständigkeitsbedingung, indem sie eine negative Autorisierung spezifiziert, aus der sich alle Zugriffsanforde-

rungen ableiten lassen. Sie kann von weiteren schwachen oder starken Autorisierungen überschrieben werden. Die starke Autorisierung gewährt dem Administrator den unbeschränkten Zugriff auf die *BMT Abstract Database* und ist nicht überschreibbar.

Die grundlegende Konfiguration des Autorisierungsdienstes ist hiermit abgeschlossen und kann ausgehend von Inhalten der Datenbank erweitert werden, wie nachfolgendes Beispiel zeigt.

8.4.4 Beispiel einer Autorisierung

Aufbauend auf der vorkonfigurierten Datenbasis müssen entsprechend dem Inhalt der Datenbank zusätzliche Rollen und Objekte spezifiziert werden.

Für alle in der WAIS-Datenbank gespeicherten *Abstracts* müssen Instanzen der Klasse `AbstractItem` mit dem Parameter `ID` instanziiert werden. Die `ID` kann dem WAIS-Datensatz entnommen werden. Für dieses Beispiel wird eine Instanz mit der ID *98052201*[69] erzeugt.

Die Autorisierung von Zugriffen soll auf der Basis von Veröffentlichungsperioden erfolgen. Die Instanzen von `AbstractItem` stehen deshalb in einer Implikationsbeziehung zu Instanzen der Klasse `Period`. Der Veröffentlichungszeitpunkt des obigen Abstracts fällt in das zweite Quartal 1998; es wird daher eine Instanz der Klasse `Period` mit dem Namen *period_2Q1998* und entsprechenden Datumswerten erzeugt.

Jeder Periode wird eine Rolle zugeordnet, die den Zugriff auf alle *Abstracts* des zugehörigen Veröffentlichungszeitraumes gestattet. In den Rollengraphen wird daher die Rolle *Role_2Q1998* eingefügt.

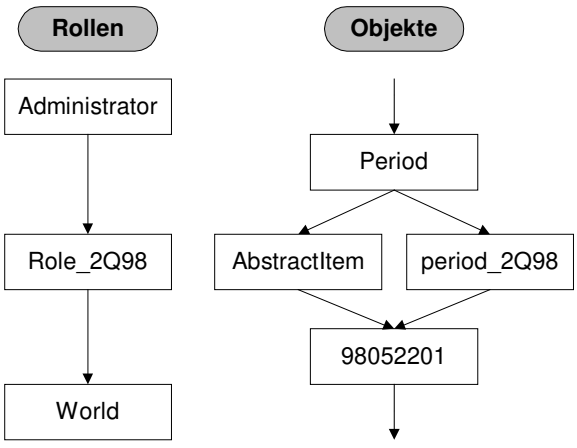

Abbildung 8.5: Graphen des Beispiels

[69] Eine ID besteht aus den letzten beiden Stellen der Jahreszahl der Veröffentlichung, gefolgt vom Monat, dem Tag und einer laufenden Nummer.

Für den gezielten Zugriff auf *Abstracts* des 2. Quartals 1998 wird folgende Autorisierung spezifiziert:

(Role_2Q98, period_2Q98, accessAbstract, +, w, false)

Die Auswertung dieser Autorisierung für einen Zugriff auf den *Abstract 98052201* durch einen Benutzer mit der zugeteilten Rolle *Role_2Q98* verläuft folgendermaßen:

1. **Objekt**: Das Objekt der Autorisierung ist *period2Q98* und impliziert den angeforderten *Abstract 9805501*.

2. **Subjekt**: Die Subjekte der Autorisierung und der Zugriffsanforderung sind identisch.

3. **Zugriffsmodus**: Die Methode `accessAbstract` impliziert die zugegriffene Methode `getAbstract` (vgl. Abbildung 8.4).

4. **Ergebnis**: Das Subjekt konnte direkt autorisiert werden und das Objekt und der Zugriffsmodus über Implikationen. Aus dem Vorzeichen der Autorisierung ergeben sich die erforderlichen Richtungen der Implikationsbeziehungen. Da das Vorzeichen positiv ist und die Implikationen in Richtung Subknoten verlaufen, ist die Autorisierung erfolgreich und der Zugriff wird gestattet.

8.5 Erweiterungsmöglichkeiten

Die durchgeführten Modifikationen an der *BMT Abstract Database* bilden die Grundlage zur Integration einer Abrechnungskomponente (vgl. Abbildung 8.6). Im Beispiel erfolgte die Abrechnung und Zugriffskontrolle auf der Basis von Veröffentlichungszeiträumen. Durch Zuordnung von *Abstracts* zu `AbstractContainern` können beliebige Gruppierungen erzielt und daher unterschiedliche Abrechnungsmodelle unterstützt werden.

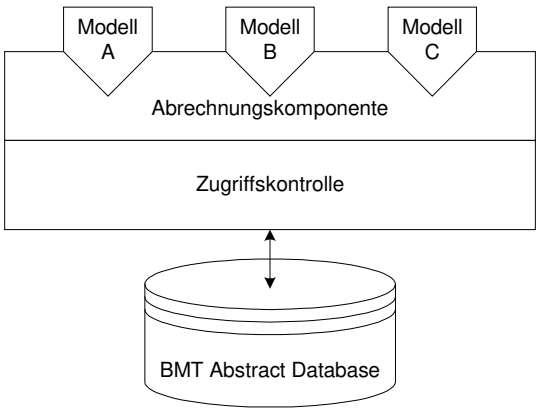

Abbildung 8.6: Abrechnungskomponente und Modelle

Die Administration des Autorisierungsdienstes kann teilweise automatisiert werden, da Autorisierungen direkt von den Geschäftsprozessen der Abrechnungsmodelle abhängen. Wenn ein Benutzer beispielsweise die Zugriffsberechtigung für ein Quartal erworben hat, können die erforderlichen Eintragungen im Autorisierungsdienst von der Abrechnungskomponente vorgenommen werden.

Kapitel 9

Zusammenfassung

Dieses Kapitel bietet eine Zusammenfassung der durchgeführten Realisierung einer Zugriffskontrolle für Internet-Informationssysteme. Dabei wird auf den Stand der prototypischen Implementierung eingegangen und ein Ausblick auf Erweiterungsmöglichkeiten und zukünftige Entwicklungen gegeben.

In der Aufgabenstellung wurde beschrieben, daß die Zugriffskontrolle für Internet-Informationssysteme kein isolierter Sicherheitsmechanismus ist, sondern sich nur als Kombination mehrerer Mechanismen umsetzen läßt. Zur Gewährleistung von Vertraulichkeit und Integrität wurden die Konzepte der Authentifizierung, Verschlüsselung und Autorisierung vorgestellt.

Zur Realisierung der Authentifizierungs- und Verschlüsselungsmechanismen erwies sich die Erweiterung des Protokollstapels um eine Sicherheitsschicht am geeignetsten. Die Funktionalität zur Autorisierung wurde in Form eines Autorisierungsdienstes realisiert, der Zugriffe auf geschützte Dienste des Informationssystems kontrolliert.

Authentifizierung und Verschlüsselung

Der Vergleich von Sicherheitsprotokollen führte zur Auswahl des SSL-Protokolls, da es die benötigten Sicherheitsmechanismen enthält und den Anforderungen an Standardisierung, Verbreitung und Unabhängigkeit genügt. Das SSL-Protokoll ist unabhängig von Protokollen der Anwendungs- und Transportschicht. Deshalb kann es als Grundlage für Internet-Informationssysteme verwendet werden, die Dienste auf der Basis unterschiedlicher Protokolle anbieten. Alle gängigen Webbrowser und Webserver unterstützen das SSL-Protokoll in den Versionen 2 und 3. In zukünftigen Versionen wird zusätzlich das TLS-Protokoll berücksichtigt werden.

In Kapitel 2 wurde detailliert auf die Spezifikation des SSL-Protokolls eingegangen. Die Beurteilung der Sicherheit des SSL-Protokolls hat gezeigt, daß die rechtliche Einschränkung des Protokolls auf schwache Algorithmen starke Auswirkungen auf die gewährleistete Sicherheit hat. Das Kapitel drei umfaßt die Beschreibung einer unterstützenden Infrastruktur, die bei der Verwendung von Zertifikaten im Internet

benötigt wird. Das vorgestellte Konzept der verketteten Zertifikatsautoritäten basiert auf Vertrauen. Den Benutzern des Systems müssen daher Mechanismen zur Verfügung stehen, die sie bei der Bewertung der Vertrauenswürdigkeit von Zertifikaten unterstützen. Wie die Beurteilung der Zertifikatskonzeptes gezeigt hat, ist gerade dieser wichtige Bereich bei der Standardisierung bisher vernachlässigt worden.

Zur Integration des SSL-Protokolls in das Informationsystem wurde der Tycoon-2-Webserver ausgewählt, da er von Diensten des Marinfo-Projektes verwendet wird. Im vierten Kapitel wurde zunächst die Struktur und die Funktionsweise des Tycoon-2-Webservers vorgestellt. Anschließend erfolgte die Beschreibung der C-Bibliothek SSLeay, die das SSL-Protokoll in den Versionen 2 und 3 implementiert. Die Einbindung der SSLeay-Bibliothek in den Tycoon-2-Webserver erforderte die Modifikation einiger Klassen. Das Kapitel endet mit der Darstellung der geänderten Struktur und Funktionsweise des Servers. Der SSL-Webserver stellt die Funktionalität der SSLeay-Bibliothek zur Verfügung und ermöglicht Tycoon-2-Anwendungen über STML-Dokumente auf Daten der Klientenzertifikate zuzugreifen.

Autorisierung

Es gibt zahlreiche Autorisierungsmodelle, die sich in Modellklassen einteilen lassen. Zur Einführung in die Thematik wurden im fünften Kapitel drei Modellklassen vorgestellt und einige Autorisierungsmodelle beschrieben.

Das sechste Kapitel umfaßt die Konzeption des Autorisierungsdienstes und ist in die Abschnitte Anforderungsanalyse, Spezifikation und Entwurf untergliedert. Die Ergebnisse der Anforderungsanalyse führten zur Auswahl der diskreten, objektorientierten Modellklasse, auf die der Autorisierungsdienst basieren soll. In der Spezifikation wurden die Eigenschaften der objektorientierten Autorisierungsmodelle *ORION*, *IRIS* und *Data-Hiding* mit eigenen Erweiterungen kombiniert. Es entstand ein mächtiges Modell, das mit objektorientierten Konzepten wie positiven / negativen, starken / schwachen und impliziten / expliziten Autorisierungen arbeitet.

Im Entwurf wurde die Spezifikation in ein objektorientiertes Klassenmodell umgesetzt. Entsprechend den Ergebnissen der Anforderungsanalyse sollte das Klassenmodell erweiterbar, flexibel einsetzbar und unabhängig von den Diensten und Protokollen des Informationssystems sein. Diese Anforderungen wurden erfüllt, da generische Datenstrukturen den Einsatz des Autorisierungsdienstes in unterschiedlichen Informationssystemen gestatten und ihn unabhängig von konkreten Diensten machen.

Überdies unterstützen in Klassen gekapselte Algorithmen die Flexibilität und Erweiterbarkeit des Dienstes. Die Kapselung der Algorithmen ermöglicht die Spezialisierung und Optimierung des Verhaltens, ohne Änderungen an Schnittstellen vornehmen zu müssen. Der Autorisierungsdienst kann zur Laufzeit geeignete Algorithmen auswählen und instanziieren.

BMT Abstract Database

Zur exemplarischen Integration der realisierten Zugriffskontrolle wurde der Marinfo-Dienst *BMT Abstract Database* ausgewählt. Er soll um Mechanismen zur Abrechnung von Datenbankzugriffen erweitert werden und benötigt daher als Grundlage eine Zu-

griffskontrolle. Für die *BMT Abstract Database* wurde eine Zugriffskontrolle entworfen, die auf dem SSL-Webserver und dem Autorisierungsdienst basiert und unabhängig von konkreten Abrechnungsmodellen ist. Die *BMT Abstract Database* unterstützt daher beliebige Abrechnungsmodelle, die auf der Mächtigkeit des objektorientierten Autorisierungsdienstes aufsetzen können.

9.1 Stand der Implementierung

Die prototypische Implementierung des SSL-Webservers und des Autorisierungsdienstes wurde abgeschlossen. Die Mechanismen wurden in die *BMT Abstract Database* integriert und ermöglichen deren Erweiterung um Abrechnungsmodelle.

Die vorhandene Funktionalität der SSL-Bibliothek wurde in den Tycoon-2-Webserver integriert. Die Funktionalität des Validierungsalgorithmus reicht für Demonstrationszwecke aus und kann entsprechend den individuellen Anforderungen von Internet-Informationssystemen erweitert werden.

Der Autorisierungsdienst kann als globales Objekt instanziiert werden und ist von Tycoon-2-Anwendungen aus über eine Schnittstellenklasse ansprechbar. Zur Initialisierung und Konfiguration des Dienstes wurden einige Hilfsklassen implementiert, die Parameter aus Textdateien auslesen. Falls weitere Unterstützung bei der Administration benötigt wird, kann zusätzlich die Schablonenverwaltung aus Kapitel 6.3.7 implementiert werden. Sie benutzt die Schnittstelle des Autorisierungsdienstes und erfordert daher keine weiteren Modifikationen. Für die starke und schwache Autorisierungsbasis wurde jeweils ein Autorisierungs- und ein Verwaltungsalgorithmus implementiert. Falls zusätzliche Algorithmen verwendet werden sollen, muß Logik zur Auswahl von Algorithmen implementiert werden. Die wichtigsten Funktionen des Autorisierungsdienstes können über eine graphische Benutzeroberfläche, die auf STML-Dokumenten basiert, aufgerufen werden.

9.2 Ausblick

Der Tycoon-2-SSL-Webserver und der Autorisierungsdienst wurden unabhängig voneinander implementiert und können deshalb getrennt an neue Entwicklungen angepaßt werden.

Die Verbreitung des SSL-Protokolls wird weiter zunehmen, da zur Zeit kein Konkurrenzprodukt mit vergleichbaren Eigenschaften existiert. Der SSL-Webserver des Tycoon-2-Systems kann mit zukünftigen Versionen des SSL- oder TLS-Protokolls zusammenarbeiten, solange sich keine Änderungen an der Schnittstelle zur Funktionsbibliothek (DLL) ergeben.

Die zunehmende Verbreitung objektorientierter Software erfordert die Implementierung von Autorisierungsmodellen, die den Anforderungen objektorientierter Systeme genügen. Wie die Beschreibung von Autorisierungsmodellen und der Entwurf des Autorisierungsdienstes gezeigt hat, müssen objektorientierte Autorisierungsmodelle

zahlreiche Konzepte unterstützen, damit die Beziehungen zwischen Objekten zur Spezifikation von Autorisierungen verwendet werden können.

Die Verwendung generischer Datenstrukturen und die Kapselung der Algorithmen ermöglichen die Anpassung des Autorisierungsdienstes an geänderte Anforderungen und die Erweiterung um zusätzliche Modellierungskonzepte.

Aufgrund der Komplexität der Beziehungen zwischen Subjekten, Objekten, Zugriffsmodi und Autorisierungen eines objektorientierten Autorisierungsdienstes benötigen die Benutzer des Systems eine grafische Benutzeroberfläche, die sie bei der Durchführung von administrativen Aufgaben unterstützt. Die realisierte STML-Oberfläche eignet sich zur Interaktion mit dem Autorisierungsdienst, sie muß aber um Werkzeuge zur Unterstützung der Administration erweitert werden. Zur Modifikation von Graphen und Implikationsbeziehungen sind zum Beispiel visuelle Editoren unerläßlich. Insbesondere die Auswertung der Bedingungen der Autorisierungsbasen sind für Benutzer nicht immer nachvollziehbar und erfordern Werkzeuge zur Unterstützuung bei der Fehlerauswertung. Die Komplexität der Verwaltung von objektorientierten Autorisierungsdiensten ist deshalb nur in Verbindung mit Benutzeroberflächen zu bewältigen, die objektorientierte Datenstrukturen und Beziehungen graphisch darstellen können.

Zur Erhöhung der Unabhängigkeit des Autorisierungsdienstes von geschützten Anwendungen kann ein Codegenerator realisiert werden. Die Eintragung der Aufrufe des Autorisierungsdienstes wird dann nicht mehr vom Anwendungsprogrammierer vorgenommen, sondern erfolgt vom Codegenerator durch Modifikation der Quelltexte vor der Tycoon-2-Bytecodegenerierung. Dazu muß der Anwendungsprogrammierer dem Codegenerator lediglich eine Liste der geschützten Klassen und Methoden seiner Anwendung übergeben.

Anhang A

Abkürzungsverzeichnis

ASP	Authorization Space
BMT	British Maritime Technology
CA	Certificate Authority
CRL	Certificate Revocation List
DES	Data Encryption Standard
DLL	Dynamic Link Library
DSS	Digital Signature Standard
DTD	Document Type Definition
EFF	Electronic Frontier Foundation
EU	Europäische Union / European Union
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDEA	International Data Encryption Algorithm
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Standards Organization
JDK	Java Development Kit
JVM	Java Virtuelle Maschine / Java Virtual Machine
KEA	Key Exchange Algorithm
KEA	Key Escrow Agency
MAC	Message Authentication Code
MD5	Message Digest 5
OSI	Open Standards Interconnection
PKIX	Public Key Infrastructure
RTS	Runtime Support
SAB	Strong Authorization Base
SHA	Secret Hash Algorithm
SSL	Secure Socket Layer
STML	Standard Tycoon Markup Language
TCP	Transmission Control Protocol
TL-2	Tycoon Language 2
TLS	Transport Layer Security
Tycoon	Typed Communicating Objects in Open Environments
UML	Unified Modeling Language
W3C	World Wide Web Consortium
WAB	Weak Authorization Base
WWW	World Wide Web

Anhang B

Beschreibung ausgewählter Schnittstellen

In diesem Kapitel werden ausgewählte Schnittstellen des Autorisierungsdienstes beschrieben. Für jede Klasse wird der Klassenname, der Name der Superklasse und die Signaturen ausgewählter Variablen und Methoden einschließlich Kommentaren angegeben. Die Auflistung der Variablen und Methoden ist in öffentliche *(public)* und private *(private)* untergliedert. Die Reihenfolge der Klassen ergibt sich aus der alphabetischen Ordnung.

Class	**AuthorizationAlgorithm (abstract)**
Super	**Object**
Public	
Methods	
(implementation is strong/weak-dependent *)* authorize() :Int *(* checks if object is included in AB and checks implications *)* authorizeObject() :Int *(* checks if subject is included in AuthorizationObject and checks implications *)* authorizeSubject(authorizationObject :AuthorizationObject) :Int *(* checks if mode is included in AuthorizationSubject and checks implications *)* authorizeAccessMode(authorizationSubject : AuthorizationSubject) :Int	
Private	
Variables	
_role :Role *(* role of access request *)* _object :Object *(* object of access request *)* _mode :Method *(* mode of access request *)* _latticeManager :LatticeManager *(* LatticeManager of AuthorizationService*)* _base :AuthorizationBase *(* AuthorizationBase to work on *)* _objectImplication :Int *(* result of object implication check *)* _subjectImplication :Int *(* result of subject implication check *)* _accessModeImplication :Int *(* result of accessMode implication check *)*	
Methods	
(checks if directions of implications are possible for given pos/neg authorization *)* _checkImplicationCombination(sign :Int) :Bool *(* implementation is strong/weak-dependent *)* _checkReturnValue(ret :Int) :Int	

Class	**AuthorizationBase (abstract)**
Super	**Object**
Public	
Methods	
(insert new AuthorizationObject *)* insertObject(object :Object) :AuthorizationObject *(*remove AuthorizationObject; only if no subjects are associated*)* removeObject(object :Object) :Void *(* checks if object is included *)* includesObject(object :Object) :Bool *(* returns AuthorizationObject for object*)* getAuthorizationObject(object :Object) :AuthorizationObject *(* returns all object names for implication check *)* getObjects :Reader(Object) *(* delete AuthorizationObjects; called if deleted in Lattices*)* deleteObject(object :Object) :Void *(* delete AuthorizationSubjects; called if deleted in Lattices*)* deleteSubject(role :Role) :Void *(* delete AuthorizationAccessModes; called if deleted in Lattices*)* deleteAccessMode(accessMode :Method) :Void	
Private	
Variables	
(* contains AuthorizationObjects with Object as key *) _objects :Dictionary(Object, AuthorizationObject)	

Class	**AuthorizationManager**
Super	**Object**
Public	
Methods	
(checks AccessMode for object and user's roles by calling authorize of algorithm *)* authorize(user :User, object :Object, mode :Method) :Void *(* AuthorizationBase management *)* insertStrongAccessRight(role :Role, object :Object, mode :Method, sign :Int) :Void removeStrongAccessRight(role :Role, object :Object, mode :Method, sign :Int) :Void insertWeakAccessRight(role :Role, object :Object, mode :Method, sign :Int) :Void removeWeakAccessRight(role :Role, object :Object, mode :Method, sign :Int) :Void deleteObject(object :Object) :Void deleteSubject(role :Role) :Void deleteAccessMode(accessMode :Method) :Void	
Private	
Variables	
_latticeManager :LatticeManager *(* passed through to Algorithm *)* _strongAuthorizationBase :AuthorizationBase _weakAuthorizationBase :AuthorizationBase _rolesToCheck :Set(Role) *(* contains the users roles *)*	

Class	**AuthorizationObject**
Super	**Object**
Public	
Methods	
(insert new AuthorizationSubject for subject *)* insertSubject(subject :Role) :AuthorizationSubject	

*(*remove AuthorizationSubject; only if no AccessModes are associated*)* removeSubject(subject :Role) :Void *(* checks if subject is included *)* includesSubject(subject :Role) :Bool *(* returns AuthorizationSubject for subject *)* getAuthorizationSubject(subject :Role) :AuthorizationSubject
Private
Variables
(contains AuthorizationSubjects with Role as key *)* _subjects :Dictionary(Role, AuthorizationSubject)

Class	**AuthorizationSubject**
Super	**Object**
Public	
Methods	
(insert new accessMode for subject *)* insertAccessMode(accessMode :Method, sign :Int) :Void *(* remove AccessMode *)* removeAccessMode(accessMode :Method, sign :Int) :Void *(* checks if accessMode is included *)* includesAccessMode(accessMode :Method) :Bool *(* returns sign of accessMode *)* getAccessMode(accessMode :Method) :Int	
Private	
Variables	
(contains signs of authorizations with Methods as key *)* _accessModes :Dictionary(Method, Int)	

Class	**Lattice(E <: Object)**
Super	**Object**
Public	
Methods	
(check if node is included *)* includesNode(node :E) :Bool *(* check implication relationship between two nodes *)* implies(node1 :E, node2 :E) :Int *(* inserts implication between two nodes *)* insertImplication(node1 :E, node2 :E) :Void *(* removes implication between two nodes *)* removeImplication(node1 :E, node2 :E) :Void *(* inserts node with implications superNode>node>subNode *)* insertNode(node :E, name :String) :Void *(* delete node and implications *)* deleteNode(node :E) :Void *(* return node by name *)* getNode(name :String) :E	
Private	
Variables	
_nodes :Dictionary(E, LatticeNode(E)) *(* contains LatticeNodes with node as key *)* _nodeNames :Dictionary(String, E) *(* used by management methods *)* _superNode :LatticeNode(E) *(* superNode of Lattice *)* _subNode :LatticeNode(E) *(* subNode of Lattice *)*	

Class	**LatticeManager**
Super	**Object**
Public	
Methods	
(User management *)* addUser(user :User) :Void deleteUser(user :User) getUser(userName :String) :User *(* wrapper-methods for each lattice not listed *)*	
Private	
Variables	
_users :Dictionary(String, User) *(* contains users with name as key *)* _roleLattice :Lattice(Role) _objectLattice :Lattice(Object) _accessModeLattice :Lattice(Method) _authorizationManager :AuthorizationManager	

Class	**LatticeNode(E <: Object)**
Super	**Object**
Public	
Methods	
(add Succ of node *)* addSucc(succ :LatticeNode(E)) :Void *(* remove Succ of node *)* removeSucc(succ :LatticeNode(E)) :Void *(* add Pred of node *)* addPred(pred :LatticeNode(E)) :Void *(* remove Pred of node *)* removePred(pred :LatticeNode(E)) :Void *(* check if node has succs *)* hasSucc(succ :LatticeNode(E)) :Bool *(* check if node has preds *)* hasPred(pred :LatticeNode(E)) :Bool *(* return node *)* getNode :E *(* check if this node implies lNode1 in up-direction *)* impliesUp(lNode1 :LatticeNode(E)) :Bool *(* check if this node implies lNode1 in down-direction *)* impliesDown(lNode1 :LatticeNode(E)) :Bool	
Private	
Variables	
_name :String _node :E _succ :Set(LatticeNode(E)) *(* contains successors of node *)* _pred :Set(LatticeNode(E)) *(* contains predecessors of node *)*	

Class	**ManagementAlgorithm (abstract)**
Super	**AuthorizationAlgorithm**
Public	
Methods	
(insert authorization into AuthorizationBase *)* insertAccessRight() :Void *(* remove authorization from AuthorizationBase *)* removeAccessRight() :Void	

(check consistency of AB after lattice-modifications *)* checkAuthorizationBaseConsistency() :Bool *(* returns set with AccessRights as Strings *)* printAccessRights() :Set(String)
Private
Variables
_sign :Int *(* sign of authorization *)* _copyMarker :Int *(* copyMarker of Authorization *)*
Methods
(calls consistency properties; overridden in subclasses *)* _checkAuthorizationConsistency :Bool *(* checks consistency property of weak/strong authorization base *)* _consistencyProperty() :Bool

Class	**Role**
Super	**Object**
Public	
Methods	
getName() :String	
Private	
Variables	
_name :String	

Class	**SabManagementAlgorithm**
Super	**ManagementAlgorithm**
Private	
Methods	
_checkConsistency :Bool *(* overriding super *)* *(* checks non-redundancy-property of SAB *)* _nonRedundancyProperty :Bool	

Class	**StrongAuthorizationAlgorithm**
Super	**AuthorizationAlgorithm**
Public	
Methods	
authorize :Int *(* overriding super *)*	
Private	
Methods	
_checkReturnValue(ret :Int) :Int *(* overriding super *)*	

Class	**User**
Super	**Object**
Public	
Methods	
(returns name of user *)* getName() :String *(* add role to user *)* addRole(role :Role) :Void *(* remove role from user *)* removeRole(role :Role) :Void	
(return all roles *)* getRoles() :Set(Role)	

Private
Variables
_name :String _roles :Set(Role) *(* contains roles of user *)*

Class	**WeakAuthorizationAlgorithm**
Super	**AuthorizationAlgorithm**
Public	
Methods	
(weak authorization algorithm evaluates scope of authorizations *)* authorize :Int	
Private	
Methods	
(store successful authorization in _tempAuthorizations and return 0 *)* _checkReturnValue(ret :Int) :Int *(* check if authorization a1 implies a2 *)* _checkImplicity(a1 :AuthorizationContainer, a2 :AuthorizationContainer) :Bool	

Literaturverzeichnis

[Ahad92] R. Ahad et. al. *Supporting Access Control in an Object-Oriented Database Language*. In: Proceedings of the 3rd International Conference on Extending Database Technology (EDBT), Vienna, Springer-Verlag, 1992.

[Atki89] M. Atkinson et. al. *The Object-Oriented Database System Manifesto*. In: Proceedings of the First International Conference on Deductive and Object-Oriented Databases. Elsevier Science Publishers, 1989.

[Atki95] R. Atkinson. *Security Architecture for the Internet Protocol*. Request for Comments 1825, Internet Activities Board, 1995.

[BeLa73] D.E. Bell, L.J. LaPadula. *Secure Computer Systems: Mathematical Foundations.* MTR-2547, Vol. 1-2, MITRE Corp., Bedford, MA, 1974.

[BeMa91] E. Bertino, L. Martino. *Object-Oriented Database Management Systems: Concepts and Issues.* IEEE Computer, 24(4), 1991.

[Bert92] E. Bertino. *Data Hiding and Security in an Object-Oriented Database System*. In: Proceedings of the 8th IEEE International Conference on Data Engineering, Phoenix, Arizona, Feb. 1992.

[BOS94] E. Bertino, F. Origgi and P. Samarati. *A New Authorization Model for Object-Oriented Databases.* In: J. Biskup et al. (eds.), Database Security VIII: Status and Prospects. North-Holland, 1994

[BrDa96] Jeremy Bradley, Neil Davies. *The SSLP Reference Implementation Project.* University of Bristol, Department of Computer Science, 1996.

[Brüg92] Hans H. Brüggemann. *Rights in an Object-Oriented Environment*. In: C.E. Landwehr and S. Jajodia (eds.), Database Security V: Status and Prospects. North-Holland, 1992.

[CFM+95] Silvana Castano, Mariagraza Fugini, Giancarlo Martella, Pierangela Samarati. *Database Security*. Addison-Wesley, 1995.

[ChBe94] William R. Cheswick, Steven M. Bellovin. *Firewalls and Internet Security.* AT&T Bell Laboratories, Inc., 1994.

[Denn82] Dorothy E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[EFF98] Electronic Frontier Foundation. *EFF DES Cracker Machine brings Honesty to Crypto Debate.* Press Release, Electronic Frontier Foundation, <http://www.eff.org>, Jul. 1998.

[EvKe90] M. Evered and J.L. Keedy. *A Model of Protection in Persistent Object-Oriented Systems.* In: J. Rosenberg and J.L. Keedy (eds.), Security and Persistence, Springer-Verlag, 1990.

[Fech98] Peter Fecht. *Internet-Authentifizierung und Verschlüsselung*. Studienarbeit, Technische Universität Hamburg-Harburg, Arbeitsbereich Softwaresysteme, Jan. 1998.

[FLG94] E.B. Fernández, M.M. Larrondo-Petrie, E. Gudes. *A Method-Based Authorization Model for Object-Oriented Databases.* In: B. Thuraisingham et al. (Eds.), Security for Object-Oriented Systems. Proceedings of the OOPSLA-93 Conference Workshop, Washington DC, Sept. 1993. Springer-Verlag, 1994.

[FoBa97] Warwick Ford, Michael S. Baum. *Secure Electronic Commerce - Building the Infrastructure for Digital Signatures and Encryption*. Prentice-Hall, 1997.

[GaWi97] Andreas Gawecki, Axel Wienberg. *STML Developer's Guide*. Higher-Order Informations- und Kommunikationssysteme GmbH, Apr. 1997

[GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns.* Addison-Wesley, 1995.

[GILC98] GILC. *Cryptography and Liberty- An International Survey of Encryption Policy*. Global Internet Liberty Campaign <http://www.gilc.org>, 1998.

[GMPS97] Li Gong, Marianne Mueller, Hemma Prafullchandra, Roland Schemers. *Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2*. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California, Dec. 1997.

[GMSS97] Andreas Gawecki, Florian Matthes, Joachim W. Schmidt, Sören Stamer. *Persistent Object Systems: From Technology to Market*. Higher-Order Informations- und Kommunikationssysteme GmbH, Forschungsbericht, 1997.

[Gong98] Li Gong. *Java Security* Architecture. <http://java.sun.com./security>, Sun Microsystems, 1998.

[HaAt94] N. Haller and R. Atkinson. *On Internet Authentication*. Network Working Group, Request for Comments 1704 (Informational), Oct. 1994.

[Hart81] H. R. Hartson. *Database Security-System Architectures*. In: Information Systems, Vol. 6, 1981.

[HFPS97] Housley, Ford, Polk, Solo. *Internet Public Key Infrastructure- Part I: X.509 Certificate and CRL Profile*. IETF PKIX Working Group, Internet-Draft, July 1997.

[HFPS98] R. Housley, W. Ford, W. Polk, D. Solo. *Internet X.509 Public Key Infrastructure: Certificate and CRL Profile.* IETF PKIX Working Group, Internet-Draft, Jul. 1998.

[Hick95] Kipp Hickman. *The SSL Protocol (Version 2).* Netscape Communications Corporation, Feb. 1995.

[Hing98] Wolf-Christian Hingst. *Die deutsche Krypto-Kontroverse*. In: Telepolis 03/1998. Heise-Verlag.

[Hirs97] F.J. Hirsch. *Introducing SSL and Certificates using SSLeay*. In: Web Security: A Matter of Trust, World Wide Web Journal Vol.2 (3), O'Reilly, Aug. 1997.

[Hof98] Simone van der Hof. *Digital Signature Law Survey*. <http://cwis.kub.nl/~frw/people/hof/simone.htm>, Version 2.7, Jun. 1998.

[HuYo97] T.J. Hudson, E.A. Young. *SSLeay Programmers Reference. Version 0.8.1*. <http://www.cryptsoft.com >, 1997.

[ITU88] ITU. *Recommendation X.509: The Directory- Authentication Framework*. ITU Telecommunication, 1988. (von der ISO/IEC unter 9594-8 geführt)

[ITU95] ITU. *Recommendation X.509: The Directory- Authentication Framework, Amendment 1: Certificate Extensions*. ITU Telecommunication, 1995.

[ITU97] ITU. *Recommendation X.500: The Directory- Overview of concepts, models, and services*. ITU Telecommunication, 1997. (von der ISO/IEC unter 9594-1 geführt)

[JLS76] A.K. Jones, R.J. Lipton, L. Snyder. *A linear time algorithm for deciding security.* In: Proceedings 17th Annual Symposium on Foundations of Computer Science, 1976.

[Kaß95] Thomas Kaß. *Anwendungsspezifische Autorisierungsstrategien in polymorphen persistenten Programmierumgebungen: Ein Bibliotheksansatz.* Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Dez. 1995.

[Kern92] Helmut Kerner. *Rechnernetze nach OSI*. Addison-Wesley, 1992.

[Kers95] Heinrich Kersten. *Sicherheit in der Informationstechnik*. Oldenbourg Verlag, 1995.

[Knap98] Klaus D. Knap, *Kryptographie- Großbritannien plant Kryptogesetz*, In: Spiegel Online 18/1998.

[Koop98] Bert-Jaap Koops. *Crypto Law Survey: Overview per Country*. <cwis.kub.nl/~frw/people/Koops/lawsurvy.htm>, Version 13.0, Jun. 1998.

[Lamp71] B.W. Lampson. *Protection*. In: Proceedings 5th Symposium on Information Sciences and Systems, Princeton University, Mar. 1971.

[LoSc87] P. Lockemann, J.W. Schmidt. *Datenbankhandbuch.* Springer-Verlag, 1987.

[MBCC+90] B. Morrison, A.L. Brown, R.C.H. Connor, Q.I. Cutts and G. Kirby. *Protection in Persistent Object Systems.* In: J. Rosenberg and J.L. Keedy (eds.), Security and Persistence, Springer-Verlag, 1990

[Nets96a] Netscape. *Secure Socket Layer Protocol Version 3 – Specification*. Netscape Communications Corporation, 1996.

[Nets96b] Netscape. *Using RSA Public Key Cryptography for Internet Security*. Netscape Communications Corporation, 1996.

[Nied97] Claudia Niederée. *Einführung in STML.* Folien zum Projektseminar „Objektorientierte Realisierung eines Internet-Informationssystems", TU Hamburg-Harburg, Arbeitsbereich Softwaresysteme, Nov. 1997.

[Nuss98] Stefan Nusser. *Sicherheitskonzepte im World Wide Web*. Springer-Verlag, 1998.

[NyOs94] M. Nyanchama and S.L. Osborn. *Access Rights Administration in Role-Based Security Systems.* In: J. Biskup et al. (eds.), Database Security VIII: Status and Prospects. North-Holland, 1994.

[Oppl97] Rolf Oppliger. *IT-Sicherheit*. Vieweg Verlag, Apr. 1997.

[RBKW91] F. Rabitti, E. Bertino, W. Kim, D. Woelk. *A Model of Authorization for Next-Generation Database Systems*. In: ACM Transactions on Database Systems, Mar. 1991.

[ReSc98] E. Rescorla, A. Schiffman. *The Secure HyperText Transfer Protocol*. Internet Draft, <draft-ietf-wts-shttp-06.txt>, Jun. 1998.

[Rich97a] Ingo Richtsmeier. *Tycoon AdBase, Server mit STML, Programmdokumentation*. Higher-Order Informations- und Kommunikationssysteme GmbH, Sep. 1997.

[Rich97b] Ingo Richtsmeier. *Tycoon AdBase-Parser, Programmdokumentation*. Higher-Order Informations- und Kommunikationssysteme GmbH, Jul. 1997.

[RSA93] RSA. *RSA Certificate Services, An RSA White Paper*. RSA Data Security, Inc., Juli 1993.

[Rudl98] Andreas Rudloff. *Nutzungsrechte in offenen Dienstinfrastrukturen.* Dissertation, Technische Universität Hamburg-Harburg, Arbeitsbereich Softwaresysteme, 1998.

[Salt74] J.H. Saltzer. *Protection and the Control of Information Sharing in Multics.* Communications of the ACM, 17(7), Jul. 1974.

[Schn94] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. John Wiley & Sons, 1994.

[Spoo89] David L. Spooner. *The impact of Inheritance on Security in Object-Oriented Database Systems*. In: C.E. Landwehr (eds.), Database Security II: Status and Prospects. North-Holland, 1989.

[Sun98] Sun Microsystems. *Secure Computing with Java: Now and the Future*. White Paper, <http://java.sun.com/security>, Sun Microsystems, 1998.

[VoBr94] K. Vosseberg, P. Brössler. *A Base for Secure Object Oriented Environments*. In: B. Thuraisingham et al. (Eds.), Security for Object-Oriented Systems. Proceedings of the OOPSLA-93 Conference Workshop, Washington DC, Sept. 1993. Springer-Verlag, 1994.

[Wahl98] Jens Wahlen. *Entwurf einer objektorientierten Sprache mit statischer Typisierung unter Beachtung kommerzieller Anforderungen*. Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 1998.

[Wegn98] Holm Wegner. *Objektorientierter Entwurf und Realisierung eines Agentensystems für kooperative Internet-Informationssysteme*. Diplomarbeit, Technische Universität Hamburg-Harburg, Arbeitsbereich Softwaresysteme, Mai 1998.

[Weik98] Marc Weikard. *Entwurf und Implementierung einer portablen multiprozessorfähigen virtuellen Maschine für eine persistente, objektorientierte Programmiersprache*. Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 1998.

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Diplomarbeit selbständig durchgeführt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamfelde,

_____

(Peter Fecht)