Sicherheitsarchitektur für ein Managementsystem auf der Basis Mobiler Agenten

Dissertation

an der

Fakultät für Mathematik und Informatik
der
Ludwig-Maximilians-Universität München

vorgelegt von

Helmut Reiser

Tag der Einreichung: 27. November 2001
Tag der mündlichen Prüfung: 18. Dezember 2001

1. Berichterstatter: **Professor Dr. Heinz-Gerd Hegering**, Universität München
2. Berichterstatter: **Professor Dr. Martin Wirsing**, Universität München

Dank

Diese Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Kommunikationssysteme und Systemprogrammierung der Universität München und im Rahmen einer Forschungskooperation mit den IBM Research Laboratories Rüschlikon, Schweiz.

Mein ganz besonderer Dank gilt meinem Doktorvater Prof. Dr. Heinz-Gerd Hegering für die hervorragende und vorbildliche Betreuung, seine Förderung und Unterstützung in wissenschaftlichen und auch in persönlichen Fragen und dafür, dass ich sehr viel von ihm lernen durfte. Herzlich bedanken möchte ich mich auch bei Prof. Dr. Martin Wirsing, der trotz hoher Belastung immer bereit war frühere Versionen dieser Arbeit schnell zu lesen, um mir konstruktive Anmerkungen und Anregungen zu geben.

Auf Seiten der IBM Research Labs möchte ich mich besonders bei Dr. Metin Feridun und Dr. Christian Hörtnagel bedanken. Metin dafür, dass er mir den Einstieg in die Kooperation erleichtert hat und Christian für die konstruktive und fruchtbare Zusammenarbeit im Rahmen unserer Veröffentlichung und unserer gemeinsam betreuten Diplomarbeit.

Herzlichen Dank auch den aktiven und ehemaligen Kollegen und Freunden innerhalb des Münchner Netzmanagement Teams, die ein solch hervorragendes Arbeitsklima geschaffen haben, wie ich es bisher noch nicht erleben durfte. Insbesondere möchte ich Prof. Dr. Claudia Linnhoff-Popien, Christian Ensel, Markus Garschhammer, Dr. Boris Gruschke, Dr. Rainer Hauck, Dr. Stephen Heilbronner, Dr. Alexander Keller, Bernhard Kempter, Annette Kostelezky, Dr. Axel Küpper, Dr. Bernhard Neumair, Igor Radisic, Harald Rölle, Dr. Holger Schmidt und Norbert Wienold danken. Auch den zahlreichen Studenten, die mich im Rahmen von Diplomarbeiten, Fortgeschrittenenpraktika und Systementwicklungsprojekten begleitet und unterstützt haben, sei an dieser Stelle gedankt.

Dank auch allen meinen Freunden, die mich unterstützt haben; insbesondere Hedwig Hagl dafür, dass sie für mich den „Fehlerteufel“ durch diese Arbeit „gejagt“ hat.

Auch meiner Familie gebührt aufrichtiger Dank. Ganz besonders danke ich Mathilde; ohne ihre Hilfe, Motivation und dem vorbehaltlosen Rückhalt, dem ich mir immer gewiss sein konnte, wäre es mir sicher nicht möglich gewesen, diesen Weg zu gehen.

München, im November 2001

Kurzfassung

Mobile Agenten werden im IT–Management eingesetzt, um eine statische Funktionszuweisung an Komponenten durch eine dynamische Funktionsdelegierung zu ergänzen bzw. abzulösen. Mobile Agenten sind auch ein adäquates Mittel, um organisationsübergreifende, flexible und dynamische Managementsysteme aufzubauen. Die Sicherheitseigenschaften dieser Managementsysteme spielen die entscheidende und kritische Rolle für deren Akzeptanz. Nur wenn ein Managementsystem auf der Basis Mobiler Agenten geeignet ist, die Sicherheitspolitik eines jeden, am interorganisationalen Managementsystem beteiligten, Unternehmens auch durchzusetzen, wird das System im produktiven Betrieb eingesetzt werden.

In der Arbeit wird eine umfassende und modulare Sicherheitsarchitektur für ein domänenübergreifendes Managementsystem entwickelt. Dazu wird ein abstraktes Systemmodell von Managementsystemen basierend auf Mobilen Agenten abgeleitet, um auf dieser Basis eine prospektive Risikoanalyse — unabhängig von konkreten Implementierungen — durchzuführen. Ergebnis dieser Untersuchung sind Sicherheitsanforderungen, die das gesamte Managementsystem betreffen, d.h. es werden die Sicherheit des Mobilen Agenten, des Agentensystems als Laufzeitumgebung für Mobile Agenten sowie die Sicherheit des Endsystems betrachtet. Für die Realisierung der Sicherheitsanforderungen werden geeignete Mechanismen, Sicherheitskonzepte und Bausteine der modularen Architektur entwickelt und auf Basis der Mobile Agent System Architecture (MASA) implementiert.

INHALT

Kapitel 1

Einführung

Inhaltsverzeichnis

Service Provider sind heutzutage wegen der Kundenorientierung und der Konkurrenz auf den Märkten mit einer sehr hohen Änderungsdynamik konfrontiert. Die verwendeten bzw. vorhandenen IT–Infrastrukturen zeichnen sich durch sehr große Heterogenität und zum Teil auch durch Kurzlebigkeit bei den Hard– und Software–Komponenten aus. Die vom Provider realisierten Dienste müssen auf dieser, sich ständig ändernden, technischen Basis erbracht und administriert werden. Aber auch die Kunden, die mit diesen Diensten ihre eigenen Geschäfts– bzw. Produktionsprozesse vereinfachen oder aus Basisdiensten eigene Mehrwertdienste realisieren, sind dieser hohen Änderungsdynamik ausgesetzt. Damit die Kunden sich auf ihr Kerngeschäft und ihre Kernkompetenzen konzentrieren können, wird das Management von Basis– und Mehrwertdiensten ganz oder zum Teil, im Rahmen von Outsourcing, an zuliefernde Provider übertragen. Für die Managementsysteme, die in solchen Szenarien eingesetzt werden, bedeutet dies die Notwendigkeit, schnell auf geänderte Anforderungen und Umgebungen reagieren zu können. Erschwerend wirkt sich dabei aus, dass sowohl im Falle des Outsourcing als auch im Rahmen der normalen Kunden–Provider Beziehung Managementfunktionalität über organisatorische Grenzen hinweg erbracht werden muss.

hohe Änderungsdynamik im Management

Management überschreitet organisatorische Grenzen

Um eine schnelle Anpassungsfähigkeit an geänderte Anforderungen durch das Managementsystem zu ermöglichen, wurde versucht, die statischen Managementsysteme durch dynamische Konzepte zu erweitern. Management by Delegation (MbD) und das Konzept der flexiblen Agenten [Moun 97] heben die Statik der Funktionszuweisung durch dynamische Funktionsdelegierung auf. Ein flexibler Agent kann zur Laufzeit um zusätzliche Funktionalität erweitert werden. In den letzten Jahren hat man dieses Konzept — in der noch allgemeineren Form des Mobilen Agenten — im Management eingesetzt. Ein **Mobiler Agent (MA)** ist ein flexibler Agent, der um die Fähigkeit der Migration bzw. der Mobilität erweitert wird. Ein **Multi–Hop MA** kann eine Menge von (heterogenen) Systemen besuchen und dort Management–Funktionen ausführen. Im Gegensatz dazu ist ein **Single–Hop MA** nur in der Lage, eine einzige Migration vom Quell– zum Zielsystem auszuführen. Die Entscheidung über die Migration kann vom Mobilen Agenten autonom getroffen oder aber von „außen“ veranlasst werden.

1.1 Fragestellung

Sicherheit entscheidend für die Akzeptanz

Bei der Verwendung von flexiblen, verteilten Managementsystemen spielen Sicherheitseigenschaften im Hinblick auf die Akzeptanz und Anwendbarkeit der Mobilen Agenten Technologie eine entscheidende und kritische Rolle. Das Management von IT–Systemen setzt die Kontrolle über Ressourcen und den vollen Zugriff auf die zu administrierenden Systeme voraus. Die Verfügbarkeit und Durchsetzbarkeit von strengen Sicherheitseigenschaften ist für die Akzeptanz eines Managementsystems basierend auf Mobilen Agenten von entscheidender Bedeutung; das Fehlen solcher Eigenschaften führt zur völligen Ablehnung. Ein Managementsystem, das der Sicherheitspolicy eines Unternehmens widerspricht oder nicht geeignet ist diese Policy auch durchzusetzen, wird im produktiven Bereich nicht eingesetzt werden.

Das Ziel dieser Arbeit ist, die wesentlichen Beiträge zur Entwicklung von sicheren Systemen Mobiler Agenten in einer Top–Down–Analyse zu erarbeiten und in einer Sicherheitsarchitektur zusammenzufassen. Die wichtigsten Fragestellungen bzw. Teilprobleme auf dem Weg zu dieser Sicherheitsarchitektur lauten:

- Strukturierung des Problembereiches durch Modellbildung von Systemen Mobiler Agenten
- Ableitung eines Entitäten– bzw. Relationenmodells für eine modellbasierte und prospektive Risikoanalyse
- Ableitung notwendiger Sicherheitsanforderungen anhand dieser abstrakten Modelle und der Risikoanalyse
- Analyse bestehender Konzepte, inwieweit diese in der Lage sind die Sicherheitsanforderungen zu erfüllen
- Ermittlung von Sicherheitsmechanismen zur Durchsetzung der notwendigen Sicherheitsanforderungen item Spezifikation und prototypische Implementierung der Komponenten einer Sicherheitsarchitektur

1.2 Defizite existierender Lösungen

Bei existierenden MA–Systemen stehen die Realisierbarkeit und Implementierungsfragen der Basisfunktionen Mobiler Agenten im Vordergrund. Die Entwicklung ist getrieben von technischen Fragestellungen. Dies führt dazu, dass jeder Hersteller eines MA–Systems eigene, proprietäre Konzepte implementiert und versucht diese auf dem Markt durchzusetzen. Es gibt keine oder nur marginale Spezifikations– oder Modellierungsversuche. Die verschiedenen existierenden Systeme sind inkompatibel. Es gibt auch kaum Standardisierungsbemühungen. Wegen der fehlenden Modellbildung gibt es bisher keine allgemeine Risikoanalyse für Systeme Mobiler Agenten, sondern nur Einzel– und Insellösungen für bestimmte Implementierungsklassen und nur für bestimmte Sicherheitsanforderungen (z.B. die Verschlüsselung von Mobilen Agenten während ihrer Übertragung). Wenn überhaupt Sicherheitsfragestellungen betrachtet werden, so betreffen diese Spezialgebiete.

nur Einzel– oder Insellösungen

Das am häufigsten zugrunde gelegte Szenario ist E–Commerce mit Hilfe von Mobilen Agenten. Dabei besucht der Mobile Agent stellvertretend für seinen Benutzer verschiedene Agentensysteme, um dort das billigste Produkt zu suchen und ggf. gleich zu erwerben. In diesem Anwendungsfall liegt der Fokus der wissenschaftlichen Untersuchungen naturgemäß auf dem Schutz des Mobilen Agenten vor einem böswilligen Agentensystem (vgl. z.B. [SaTs 98, Yee 97, Vign 98a, Hohl 98, SaTs 97]).

Fokussierung auf offene Szenarien, z.B. E–Commerce

Szenarien wie organisationsübergreifendes IT–Management mit Hilfe Mobiler Agenten wurden bisher kaum untersucht (außer am Rande in [BPW 98]). Die spezifischen Sicherheitsprobleme im IT–Management mit Mobilen Agenten wurden überhaupt nicht betrachtet. Der Hauptunterschied zu offenen Szenarien (wie z.B. dem E–Commerce) ist zum einen, dass die beteiligten Organisationen auch in vertraglichen Beziehungen zueinander stehen (z.B. in einer Kunden Provider Beziehung, vgl. Abschnitt 2.2.1). Zum anderen werden die Mobilen Agenten verwendet, um aktiv Systeme, Anwendungen oder Dienste zu steuern, zu konfigurieren, zu überwachen und auch abzurechnen. Für diese Aufgaben sind entsprechend sensible Rechte erforderlich.

Isolierte (Sicherheits–) Lösungen für Spezialprobleme sind für Managementsysteme nicht ausreichend. Es bedarf eines umfassenden und grundsätzlichen Sicherheitskonzeptes für Systeme, die auf Mobilen Agenten basieren oder diese einsetzen. Dabei reicht es auch nicht aus nur die Sicherheit des Mobilen Agenten zu betrachten, sondern es muss auch die Sicherheit der Laufzeitumgebung des Mobilen Agenten (Endsystem, Agentensystem) sowie der verschiedenen Organisationseinheiten betrachtet werden.

Sicherheit des Mobilen Agenten steht im Vordergrund

1.3 Vorgehensmodell und Ergebnisse

Das Vorgehensmodell und der Aufbau dieser Arbeit wird in Abb. 1.1 dargestellt.

Kapitel 2 definiert verwendete Begriffe und beschreibt einige Szenarien für den Einsatz von Mobilen Agenten im Anwendungsgebiet IT–Management. Um eine Risikoanalyse unabhängig von konkreten Implementierungen

Kapitel 2: Begriffs– und Modellbildung, Anforderungs-analyse

durchführen zu können, wird ein allgemeines Modell für Managementsysteme, die auf Mobilen Agenten basieren, vorgestellt. Es handelt sich dabei um ein generisches Modell, sodass die daraus abgeleiteten grundlegenden Anforderungen für alle Arten von Systemen Mobiler Agenten gelten. Aus einer allgemeinen, strategischen Sicherheitspolicy, dem vorgestellten Modell, dem Lebenszyklus Mobiler Agenten und den Anforderungen der Anwendungsdomäne IT–Management lassen sich Sicherheitsanforderungen ableiten.

Kapitel 3: State–of–the–Art

Kapitel 3 untersucht mit dem CORBA Security Service einen Sicherheitsstandard der für Systeme Mobiler Agenten, die auf der Kommunikationsplattform CORBA basieren, Anwendung finden könnte. Außerdem werden aktuelle und relevante Forschungsansätze analysiert.

Kapitel 4: Konzepte und Mechanismen

Im vierten Kapitel werden Sicherheitskonzepte erarbeitet und geeignete Sicherheitsmechanismen vorgestellt. Der Schutz eines Mobilen Agenten vor einem feindlichen Agentensystem ist ein bisher nicht gelöstes Problem. Für eine allgemeingültige Lösung dieses Problems kann man keine Annahmen über eine bestimmte Ablaufumgebung für Mobile Agenten machen (Stichwort: offene Systeme). In einem Managementsystem hingegen muss und kann davon ausgegangen werden, dass zwischen den beteiligten Organisationseinheiten vertragliche Beziehungen und gewisse Vertrauensverhältnisse existieren.
Es wird gezeigt, dass ein Mobiler Agent, sobald er auf ein Agentensystem migriert, diesem vollständig ausgeliefert ist. Aus dieser Tatsache und dem Einsatzszenario IT–Management wird der Ansatz des „Vertrauens durch Einbettungsbeziehung“ abgeleitet. Dieses Konzept besagt, dass ein Agentensystem bei gewissen sicherheitsrelevanten Aktionen als Stellvertreter („Prokurist“) des Mobilen Agenten auftreten muss, um die Sicherheitsanforderungen des Agenten durchzusetzen. Dieses Konzept ist aber nur anwendbar, wenn ein Vertrauensverhältnis zwischen Agentensystem und Mobilem Agenten aufgebaut werden kann. Das Trust Level Management beschäftigt sich damit, wie diese Vertrauensverhältnisse aufzubauen sind.

Im Folgenden werden die verschiedenen Sichten auf das allgemeine Modell aus Abschnitt 2.4 wieder aufgegriffen. Das Entitätenmodell wird verfeinert; es wird unterschieden zwischen statischen Teilen (Code, Programm; bezeichnet als Gattung) und dynamischen Teilen (in Ausführung befindlicher Code; bezeichnet als Instanz). Diese Unterscheidung und Explizitmachung des Gattungs– und Instanzbegriffs wurde bisher in der Literatur nicht getroffen.

Entitätenmodell

Um Entitäten (sowohl Gattung als auch Instanzen) und Rollen eindeutig identifizieren zu können, wird ein Namensschema entwickelt. Die Identifikation bezeichnet dann die Abbildung dieses Namens auf einen bestimmten Gattungstyp, eine Instanz oder eine Rolle. Die eigentliche Authentisierung ist die zweifelsfreie Feststellung, dass ein bestimmter Name zu einer bestimmten Entität gehört, d.h. es wird ein Mechanismus vorgestellt, mit dem der Name zweifelsfrei an eine Entität gebunden werden kann, und es wird gezeigt, wie diese Bindung verifiziert werden kann. Die Authentisierung ist Basis für die Zugriffskontrolle, daneben dient sie aber auch dazu, Aktionen zurechenbar zu machen. Dazu muss es möglich sein, diejenige Rolle, oder die natürliche Person, die in dieser Rolle agiert, und damit für eine bestimmte Gattung oder Instanz verantwortlich ist, zu bestimmen. Auch für das Zurechenbarkeitsproblem wird eine Lösung vorgeschlagen.

Relationenmodell

Während das Entitätenmodell sich hauptsächlich mit den statischen Aspekten eines Systems Mobiler Agenten beschäftigt, betrachtet das Relationenmodell das dynamische Verhalten und die Interaktionen zwischen Entitäten.

Ausführungsrelation

Die Ausführungsrelation beschäftigt sich mit der Ausführung einer Entität durch eine andere. Mobile Agenten werden beispielsweise von Agentensystemen ausgeführt, d.h. sie sind vollständig unter Kontrolle des ausführenden Agentensystems. Da auf einem Agentensystem auch mehrere Agenten und auch Agenten, für die verschiedene organisatorische Einheiten verantwortlich sind, ablaufen können, muss die Sicherstellung der Integrität und Vertraulichkeit zwischen Instanzen während deren Ausführung betrachtet werden. Dazu werden Fragen der Sichtbarkeit, Konzepte zur Trennung von Namensräumen sowie Sandboxing untersucht.

Aufrufrelation

Die Problemkreise, die unter dem Blickwinkel der Aufrufrelation betrachtet werden, sind Rechtekonzepte (Autorisierung) und die Delegation von Rechten sowie die damit zusammenhängende Durchsetzung von Rechten (Zugriffskontrolle).

Kommunikationsrelation

Die Kommunikations– und Migrationsrelation beschäftigt sich mit den Kommunikationsbeziehungen der Entitäten untereinander, sowie mit der Migration von Mobilen Agenten als einem „Spezialfall“ der Kommunikation. Es wird gezeigt, wie der Agent mittelbar durch das Agentensystem seine gewünschten Sicherheitseigenschaften und –anforderungen im Hinblick auf Migration und Kommunikation erreichen kann. Daneben werden Verfahren und Mechanismen vorgestellt, um die Vertraulichkeit und Integrität sowohl der Kommunikation als auch der Mobilen Agenten während der Migration zu gewährleisten. Dabei wird zwischen privaten und öffentlichen Daten des Mobilen Agenten unterschieden, die veränderbar oder unveränderlich sein können. Für alle Klassen von Daten, die ein Mobiler Agent während seiner „Reise“ transportiert, werden Verfahren zur Sicherung der Integrität und Vertraulichkeit angegeben.

Kapitel 5 + 6: Spezifikation und prototypische Implementierung der Sicherheitsarchitektur

In Kapitel 5 wird untersucht, welche Entität (Endsystem, Agentensystem oder Mobiler Agent) am besten geeignet ist, um die Sicherheitsdienste zu realisieren. Dann werden die Sicherheitskonzepte und –mechanismen zu einer Sicherheitsarchitektur vereinigt und die Komponenten dieser Architektur (CA, Authenticator, Laufzeitumgebung, Migration Manager, Communication Manager, Permission Manager, Integrity Manager und Naming Manager) spezifiziert und modelliert.

Kapitel 6 stellt dann eine prototypische Implementierung der in Kapitel 5 entwickelten Architektur vor. Als Agentensystem wird die Mobile Agent System Architecture (MASA) [GHR 99, KRRV01] verwendet und gezeigt, wie sich die spezifizierten Verfahren mit der Sprache Java und der Middleware CORBA realisieren lassen.

Die Ergebnisse der Arbeit werden in Kapitel 7 zusammengefasst und zukünftige und weiterführende Forschungsfragestellungen angegeben.

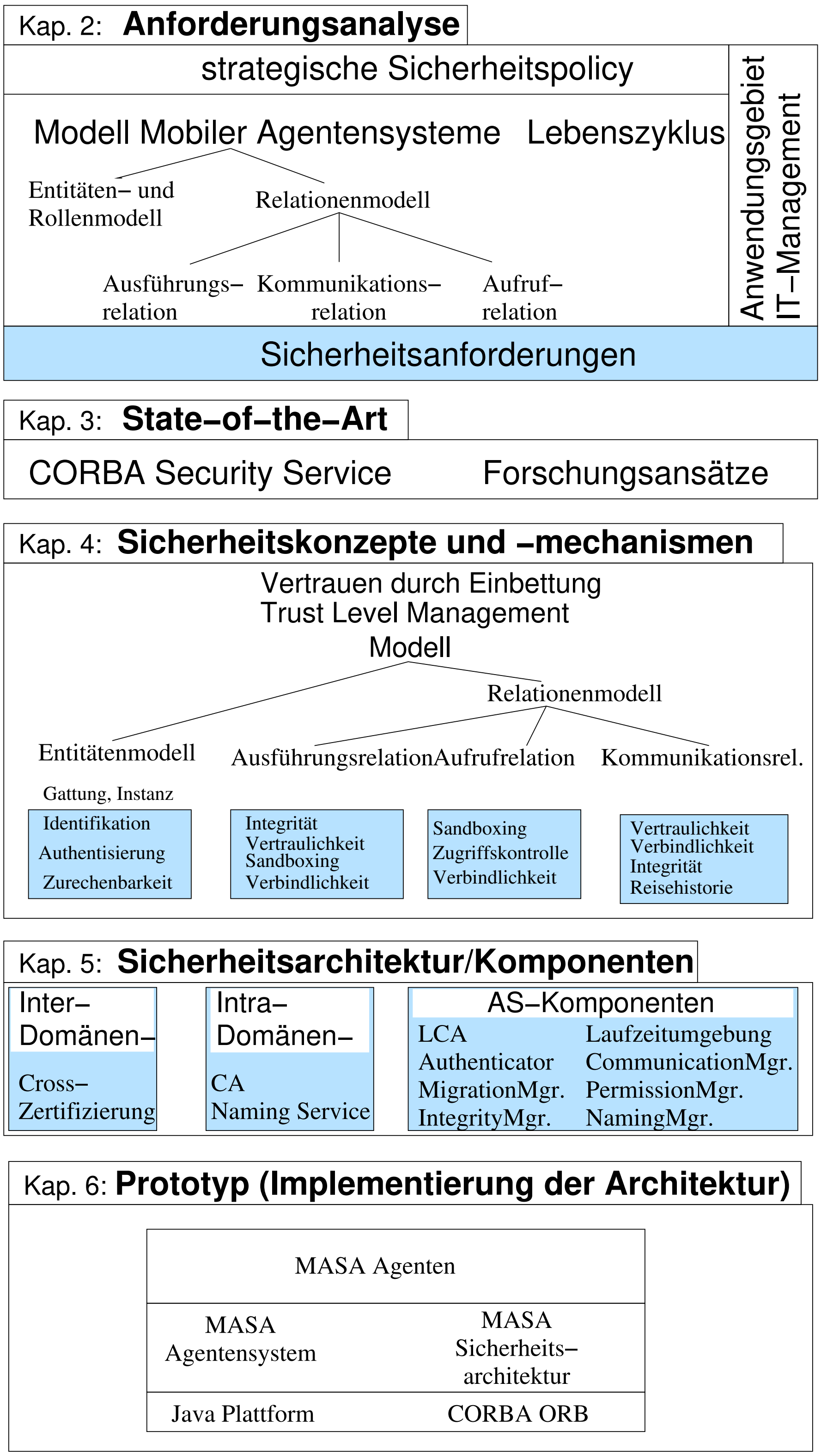


Abbildung 1.1: Vorgehensmodell; Aufbau der Arbeit

Kapitel 2

Problembeschreibung und Anforderungsanalyse: Mobile Agenten im IT–Management

Inhaltsverzeichnis

In den letzten Jahren wurden einige neue Managementkonzepte, –techniken und auch –architekturen entwickelt, um die Defizite klassischer zentralisierter Managementsysteme zu beseitigen. Die Verwendung von Mobilen Agenten im IT–Management ist nur ein Beispiel dafür. Das frühe Stadium dieser Entwicklungen führt dazu, dass noch keine einheitliche Begriffswelt existiert und Begriffe mehrdeutig verwendet werden. Im ersten Abschnitt dieses Kapitels soll insbesondere das Konzept eines erweiterbaren Managementsystems, das den Mobilen Agenten sehr nahe steht, von diesem und den klassischen zentralisierten bzw. statischen Managementsystemen abgegrenzt werden.

Im Folgenden werden Szenarien für den Einsatz Mobiler Agenten im Management vorgestellt. Der Abschnitt 2.3 führt eine einheitliche Nomenklatur für Systeme Mobiler Agenten ein. Um diese Systeme sicherheitstechnisch untersuchen und bewerten zu können, bedarf es eines allgemeinen Modells, das unabhängig von konkreten Implementierungen verwendet werden kann und von diesen abstrahiert. Das Modell muss für alle Implementierungen von Mobilen Agentensystemen verwendet werden können und bildet die Basis für eine allgemeine Risikoanalyse. Ausgehend von einer Entitätensicht werden verschiedene Relationen zwischen den Entitäten abgeleitet. Die Grundidee des Modells ist die Tatsache, dass nur Entitäten eines Systems, bzw. die Relationen zwischen diesen Entitäten, Ziel eines Angriffs sein können. Ein System Mobiler Agenten könnte auch als verteilte Anwendung betrachtet werden und deshalb ein Sicherheitsmodell oder eine Sicherheitsarchitektur einer verteilten Anwendung adaptieren. Diese Betrachtungsweise ist jedoch nicht ausreichend, denn bei verteilten Anwendungen wird Mobilität nicht betrachtet. Im Abschnitt 2.5 wird deshalb der Lebenszyklus von Mobilen Agenten und die Besonderheiten der Mobilität vorgestellt, die Systeme Mobiler Agenten von anderen verteilten Anwendungen unterscheiden.

Ausgehend von der strategischen Policy „Ein Managementsystem muss sicher sein“, dem allgemeinen Modell von Systemen Mobiler Agenten und den Managementszenarios werden am Ende des Kapitels Sicherheitsanforderungen, an ein Managementsystem auf der Basis Mobiler Agenten, abgeleitet.

2.1 Organisationsformen für Managementsysteme

Mit der zunehmenden Verbreitung offener Systeme, leistungsfähiger Kommunikationsinfrastrukturen, neuer Anwendungen und einer Fokusverschiebung hin zur Dienstorientierung wird ein domänenübergreifendes IT–Management, das Providerhierarchien und verschiedene Kunden umfasst, immer wichtiger. Mit klassischen Managementsystemen kann ein solches domänenübergreifendes System oftmals überhaupt nicht realisiert werden, oder das resultierende System ist nicht in der Lage mit der geforderten Geschwindigkeit auf Änderungen zu reagieren.

Um den begrifflichen Kontext für diese Arbeit zu legen, die verschiedenen Verwendungen des Agentenbegriffs zu verdeutlichen und auch um die historische Entwicklung zu zeigen, wird in diesem Abschnitt der Übergang von zentralisierten und statischen Managementsystemen hin zu einem Managementsystem basierend auf Mobilen Agenten, mit den entsprechenden Zwischenstufen, vorgestellt.

Integriertes Management in verteilten heterogenen Umgebungen setzt Managementarchitekturen [HAN 99, HAN 99a, Rose 96] voraus, die in herstellerunabhängiger und systemübergreifender Weise spezifiziert sind. Diese Managementarchitekturen definieren vier Teilmodelle:

Management-architektur: Teilmodelle

1. Im **Organisationsmodell** werden zwei Rollen unterschieden. **Agenten** stellen Managementinformation bereit, melden managementrelevante Ereignisse und stellen die Schnittstelle zu den Managementobjekten dar. Ein **Managementobjekt (MO)** stellt eine Abstraktion der Charakteristika einer zu verwaltenden Ressource dar. Die aktiven Komponenten, welche steuernd auf die Agenten zugreifen, um Managementinformation abzufragen oder den Status einer Ressource zu verändern, und die auf Managementereignisse reagieren, werden als **Manager** bezeichnet.
2. Das **Informationsmodell** definiert die Struktur und das Format der Managementinformationen, insbesondere zur Spezifikation von MOs. In der **Management Information Base (MIB)** wird die Gesamtheit der Managementschnittstellen und –informationen, die ein Agent einem Manager zur Verfügung stellt, zusammengefasst.
3. Das **Kommunikationsmodell** spezifiziert die Managementprotokolle zur Kommunikation zwischen Manager und Agenten.
4. Im **Funktionsmodell** werden Managementdienste beschrieben, die bestimmte Managementfunktionalität zur Verfügung stellen.

2.1.1 Zentralisierte statische Managementsysteme

Bei den zentralisierten Managementsystemen übernimmt eine einzelne zentrale Managementplattform wie z.B. *Tivoli TME*, *CA Unicenter*, *HP Open* zentrale Management-plattform

View oder *Cabletron Spectrum*die Managerrolle. Von diesem Manager aus wird, im Falle des klassischen SNMP–basierten Management [Rose 96, CFSD 90, CMRW 93], eine große Zahl von Agenten verwaltet. Agenten sind die Kommunikationspartner des Managers. Sie sind Anwendungen, die auf den überwachten Systemen (z.B. Netzkomponenten, Endsystemen, usw.) laufen und einen Zugriff auf die verwalteten Ressourcen ermöglichen. Deshalb müssen sie für die jeweilige Ressource konfiguriert werden, was üblicherweise durch einen eigenen Übersetzungs– und Installationsvorgang geschieht. Die Agenten besitzen nur sehr einfache Zugriffsschnittstellen (`get, set`) und sie sind gewöhnlich nicht in der Lage, eine Vorverarbeitung von Managementdaten, die sie sammeln, durchzuführen.

statische Agenten

Insbesondere wegen der Agenten, deren Funktionalität zum Zeitpunkt der Übersetzung festgelegt wird und die vordefinierte Managementinformationen voraussetzen, wird dieser Ansatz als statisch bezeichnet. Eine Änderung der Managementanforderungen und eine damit einhergehende Änderung der Agenten–Funktionalität oder der MIB erfordert eine erneute Übersetzung und Installation des bzw. der Agenten.

Der zentralisierte Ansatz lässt sich bei zunehmender Zahl administrierter Systeme nicht mehr sinnvoll betreiben. Es können nicht mehr sämtliche Managementobjekte von einer zentralen Plattform aus administriert werden.

hierarchisches Management

Durch die Einführung einer Hierarchie innerhalb des Managers wird auf das Skalierbarkeitsproblem reagiert. Das hierarchische Management ist eine Organisationsform, bei der logisch tieferliegende Schichten unnötigen Detailgrad für die höherliegenden Schichten verschatten. Der **Top–Level Manager** administriert mehrere **Mid–Level Manager** die ihrerseits die Managementobjekte in einem eng abgegrenzten Bereich verwalten. Der Top–Level Manager wird damit zu einem **Manager of Managers**. Das hierarchische Management wird jedoch nicht von allen Management–Architekturen unterstützt.

2.1.2 Erweiterbare Managementsysteme; Management by Delegation

Eine Skalierbarkeit in Bezug auf die Anzahl der Managementobjekte lässt sich relativ gut durch hierarchische Organisationsformen sicherstellen. Die Problematik der Änderungsdynamik kann jedoch in statischen Systemen nicht befriedigend gelöst werden.

dynamische Delegation von Funktionen an Agenten

Um die Skalierbarkeit von Managementsystemen in Bezug auf eine leichte Anpassbarkeit hin zu verbessern, wurde 1991 der **Management by Delegation (MbD)**–Ansatzvorgeschlagen [YGY 91, GoYe 95, GoYe 98]. Unter diesem Paradigma werden Ansätze zusammengefasst, die eine Delegation von Managementfunktionalität an den Agenten ermöglichen. Diese Funktionen werden in einer Skript– oder einer nativen Programmiersprache spezifiziert, welche dann von einem Laufzeitsystem innerhalb des Agenten ausgeführt wird. Goldszmidt hat dazu in [Gold 96] ein auf SNMP basierendes Rahmenwerk vorgestellt, das ein Protokoll zur Delegation sowie Dienste zur Steuerung der delegierten Programme umfasst.

Im Umfeld des Internet Managements wurde und wird das MbD–Paradigma in der IETF–Working Group „Distributed Management (disman)" aufgegriffen und weiterentwickelt [LeSc 99a, LeSc 99b, Whit 00, KaSt 00].

Mountzia hat auf dem MbD–Paradigma aufbauend eine neue Klasse von Agenten, den **Flexible Agent** eingeführt [Moun 97, Moun 97a]. Der Flexible Agent kann bis zu einem gewissen Grad autonom handeln und — was ihn vom MbD Ansatz unterscheidet — er kann zur Erfüllung seiner Managementaufgabe mit anderen Flexible Agents kooperieren. Innerhalb des Managementsystems können Flexible Agents zu Gruppen zusammengefasst werden, die kooperativ und autonom eine bestimmte Aufgabe erledigen.

Flexible Agent: kooperative Aufgabenerfüllung

2.1.3 Managementsysteme auf Basis von Mobilen Agenten

Die bisher vorgestellten Ansätze gehen alle von einem mehr oder weniger statischen Agentenbegriff aus. Auf dem zu verwaltenden System muss immer ein (statischer oder erweiterbarer) Agent installiert sein, um das System administrieren zu können. Von dieser Prämisse des fest installierten Agenten lösen sich Ansätze, die Mobile Agenten verwenden.

Der Begriff des **Mobilen Agenten (MA)** wurde Mitte der 90er Jahre eingeführt [HCK 95, RoPo 97]. Ein Mobiler Agent ist ein Programm, das sich innerhalb eines Netzes bewegen kann (**Mobilität**), um im Auftrag eines Nutzers eine bestimmte Aufgabe zu erfüllen. Der Mobile Agent muss insbesondere von Mobilem Code, als Oberbegriff der mit der Sprache Java eingeführten Applets, abgegrenzt werden.

Mobiler Agent — Abgrenzung zu Mobilem Code

Bei **Mobilem Code** wird nur der Programmcode der Anwendung von einem Quell– auf ein Zielsystem übertragen oder nachgeladen, es wird jedoch keine Zustandsinformation mit transportiert. Mobiler Code beinhaltet keine variablen Daten und startet nach jeder Übertragung auf ein anderes System im selben Zustand. Im Gegensatz dazu werden bei einem Mobilen Agenten sowohl der Code der Anwendung als auch sein momentaner Zustand übertragen (Migration eines MA ist **zustandserhaltend**). Die Migration eines Mobilen Agenten kann auch **Multi–Hop**, d.h. über mehrere Zielsysteme hinweg, erfolgen, wobei Zustandsinformationen, die auf einem System gewonnen wurden, in die Berechnung auf einem anderen System mit eingehen können. Die Entität „Mobiler Agent" besteht aus zwei Teilen: der **MA–Instanz** sowie der **MA–Gattung** (die Begriffe werden in Abschnitt 4.2.1 eingeführt und erläutert). Eine derartige Aufteilung ist für Mobilen Code nicht erforderlich bzw. gar nicht sinnvoll. Der Mobile Agent stellt damit die Obermenge von Mobilem Code dar.

2.1.4 Vorteile, Einsatzgebiete von Mobilen Agenten

Die wesentlichen Vorteile Mobiler Agenten sind deren Flexibilität sowie die Möglichkeit heterogene Hardware zu verschatten.

Das Agentensystem, als minimale Infrastrukturkomponente, stellt auf unterschiedlichsten Systemen eine homogene Ausführungsplattform für die Mobilen Agenten dar.

hohe Flexibilität

Erweiterbarkeit

Mit den Mobilen Agenten kann sehr schnell auf Änderungen sowohl auf Seite der Infrastruktur als auch der Anforderungen reagiert werden. Mit der Migration eines geänderten oder neuen Mobilen Agenten lassen sich sehr schnell neue Dienste instantiieren (Service Deployment) oder Änderungen der MIB realisieren und abbilden. Auch die Funktionalität sowohl des Managers als auch des Agenten kann zur Laufzeit erweitert werden; eine Neukompilierung des gesamten Agenten, wie bei klassischen Managementsystemen, ist nicht erforderlich. Selbst eine Erweiterung der Schnittstellen ist zur Laufzeit möglich [KRS 99]. Durch das sehr einfache Hinzufügen, Ändern oder Entfernen von Funktionalität ergibt sich ein hohes Maß an Skalierbarkeit. Die Anzahl der Applikationen, die auf einer zentralen Managementstation ausgeführt werden können, ist sehr begrenzt. Auch hier kann durch die Verlagerung von (Manager–)Funktionalität auf Mobile Agenten skaliert werden.

hohes Maß an Skalierbarkeit

Reduzierung der Netzlast

In Managementsystemen können Mobile Agenten zu einer Reduzierung der Netzlast führen. Beim zentralisierten Management holt sich der Manager u.U. riesige Datenmengen von den Agenten, um diese dann zu verarbeiten. Ein Mobiler Agent hingegen kann relativ nahe zu den MOs, im Normalfall auf demselben System, platziert werden und die Daten vorverarbeiten. Der Mobile Agent kann lokale Methodenaufrufe verwenden anstatt über ein Netz mit dem zu administrierendem System zu kommunizieren.

Auch der Signalisierungsverkehr in Mobilfunknetzen kann durch den Einsatz von Mobilen Agenten verringert werden. In [KuPa 98] wurde der Einsatz Mobiler Agenten für das Session Setup und den Location Update in UMTS mit dem Einsatz von klassischen stationären Agenten verglichen. Für hohe Ankunftsraten und eine überdurchschnittliche Mobilität des Benutzers ist die Verwendung eines Mobilen Agenten performanter und reduziert die Netzlast, die durch Signalisierung entsteht.

hohe Fehlertoleranz

Autonomie

Durch die endsystemnahe Platzierung und die Möglichkeit der Autonomie des Mobilen Agenten ergibt sich auch eine höhere Fehlertoleranz. Selbst bei einem Ausfall der Netzverbindung zwischen Manager und Mobilem Agenten kann dieser autonom weiterarbeiten und dann seine Daten zu einem späteren Zeitpunkt an den Manager übermitteln.

weitere Einsatzgebiete Mobiler Agenten

Mobile Agenten werden aber nicht nur im IT–Management, sondern in den verschiedensten Bereichen eingesetzt. Ein vielversprechendes Einsatzgebiet von Mobilen Agenten sind Context Aware Services [KRS 99]. **Kontextsensitive Dienste (Context Aware Services)** passen sich der Situation ihrer gegenwärtigen Benutzung automatisch an. Insbesondere im Bereich der Mobilkommunikation der nächsten Generation (UMTS) hat die Idee der kontextsensitiven Systeme zentrale Bedeutung.

Aber auch im produzierenden Gewerbe, z.B. zur Steuerung und Optimierung von Produktionsprozessen [SuBu 01] oder zur Ferndiagnose und Fernüberwachung von Automatisierungssystemen [PlWe 01] finden Mobile Agenten Verwendung.

Daneben werden sie auch in Zulieferer– oder Händlerketten eingesetzt, um Supply Chain Management zu betreiben. Wertschöpfungsnetzwerke oder –ketten (**Supply Chains**) werden aus Entitäten gebildet, die für die Beschaffung von Rohstoffen, deren Transformation in Halbfertig– und Fertigprodukte sowie die Verteilung dieser Produkte zuständig sind. Das Supply Chain Management umfasst den kompletten Wertschöpfungsprozess [GSWA 01], der sich durch Mobile Agenten abbilden und unterstützen lässt.

2.2 Szenarios für das Management mit Hilfe von Mobilen Agenten

Der Fokus dieser Arbeit soll auf Mobilen Agenten liegen, die im IT–Management eingesetzt werden. Dieser Abschnitt beschreibt einige Szenarios aus der Praxis, die einerseits die Anwendung von Mobilen Agenten im IT–Management aufzeigen, andererseits aber auch Sicherheitsprobleme, die sich durch ihren Einsatz ergeben, verdeutlichen.

2.2.1 Dienst– und QoS–Management in Customer–Provider Hierarchien

Durch die zunehmende Komplexität von Diensten und IT–Infrastrukturen sowie aufgrund deren räumlicher Verteilung ist es für ein Unternehmen unerlässlich, Netzinfrastrukturen oder Dienste von einem Provider zu kaufen oder zu mieten. Dies gilt auch für Unternehmen, die selbst wieder als IT–Dienstleister auftreten und **Mehrwertdienste (value added services)** an ihre Kunden weiterverkaufen. Dadurch entstehen Kunden–Dienstleister Hierarchien (Customer–Provider Hierarchies) bzw. **Multiprovider Hierarchien**. In einer solchen Hierarchie tritt ein Unternehmen in verschiedenen Rollen auf. Einerseits in der Rolle des Kunden, der Dienste von einem Provider einkauft, andererseits aber auch in der Rolle des Providers, der Mehrwertdienste an eigene Kunden weiterverkauft.

Ein Dienstleister geht dabei mit seinem Kunden **Dienstgütevereinbarungen** (**Service Level Agreements (SLA)**)[Schm 01] ein, deren Erfüllung bzw. Einhaltung er vertraglich zusichert. In diesen SLAs werden für jeden Dienst Qualitätseigenschaften festgelegt, die in ihrer Gesamtheit die Dienstgüte, auch

Problem: Überwachung des QoS in Multiprovider Hierarchien

als **Quality of Service (QoS)** bezeichnet, ausmachen.Dazu ist es notwendig Kennzahlen für die QoS–Parameter und unter Umständen auch Messverfahren zu deren Bestimmung in den SLAs festzulegen. Werden SLAs verletzt, weil die vereinbarten QoS–Parameter nicht eingehalten werden, so sind vom Dienstleister i.d.R. Strafen zu bezahlen oder Nachlässe zu gewähren. Es ist klar, dass die Erfüllung der Dienstgütevereinbarung für einen Provider auch unmittelbar von der Dienstgüte der, von eigenen Zulieferern, eingekauften Dienste abhängt.

Neben der vertraglichen Gestaltung und den rechtlichen Problemen beim Abschluss von SLAs besteht auch das Problem der Überwachung der QoS–Parameter und der Beweis– bzw. Nachweispflicht bei der Verletzung der SLA. Um diese Problematik zu verdeutlichen soll im Folgenden ein exemplarisches Szenario vorgestellt werden, das im Rahmen diverser Forschungskooperationen mit IT–Dienstleistern und großen Netzbetreibern (*DeTeSystem, Siemens, Bayerische Motorenwerke AG*) untersucht wurde [HaRe 99, Hojn 99, Knoe 99].

Szenario: Extranet – Anbindung von Partnerunternehmen

Die BMW AG betreibt **Intranets**, d.h. firmeneigene, „interne“, abgeschlossene Netze, die auf den Internetprotokollen basieren. Damit werden Systeme in verschiedenen Standorten oder innerhalb von Standorten und Abteilungen miteinander verbunden und abteilungs– und standortübergreifende Dienste zur Verfügung gestellt (vgl. z.B. [Albe 98]). Um die Vorteile und Marktchancen von E–Commerce nutzen zu können, sollten im Bereich **Business to Business Commerce (B2B)**, d.h. im Bereich der Geschäftsbeziehungen zwischen eigenständigen Unternehmen [Merz 99], Kompetenzen aufgebaut und Lösungen realisiert werden. Inspiriert von der *Automotive Network eXchange (ANX)* Initiative amerikanischer Kfz–Hersteller (genauer der *Automotive Industrie Action Group [AIAG]*) sollten bestimmte Dienste und Netzinfrastrukturen aus den BMW Intranets auch für Händler, Zulieferer und IT–Partner zur Verfügung gestellt werden, um die Geschäftsprozesse mit diesen Partnerunternehmen zu vereinfachen und zu beschleunigen. So sollte bspw. Händlern die Möglichkeit gegeben werden, Autos online zu bestellen und zu konfigurieren. Die Partnerunternehmen bilden ein so genanntes **Extranet**, d.h. ein „externes“ Netzwerk, für eine abgeschlossene Benutzergruppe außerhalb von BMW. Die verschiedenen Intra– und Extranets zusammen bilden das weltweite BMW Unternehmensnetz, das auch als **Corporate Network (CN)** bezeichnet wird.

Abbildung 2.1 zeigt die Realisierung des Händler–Extranets. Die Netzinfrastruktur für die Verbindung der mehr als 1000 Händler wurde (in Deutschland) von der DeTeSystem realisiert. Die Händler werden entweder über Standleitungen oder ISDN–Wählleitungen mit einem *Point–of–Presence (POP)* verbunden. Sie bilden auf der Infrastruktur der DeTeSystem, bzw. der Telekom die auch von anderen Kunden der DeTeSystem und der Telekom genutzt wird, ein **virtuelles privates Netz (VPN)**. Daneben werden auch Dienste wie z.B. DNS, Mail, Authentisierung oder ein Konfigurationsdienst für die Händler von der DeTeSystem realisiert. Dazu wurde eine Service Area eingerichtet, in der diese Dienste erbracht werden können. In dieser Service

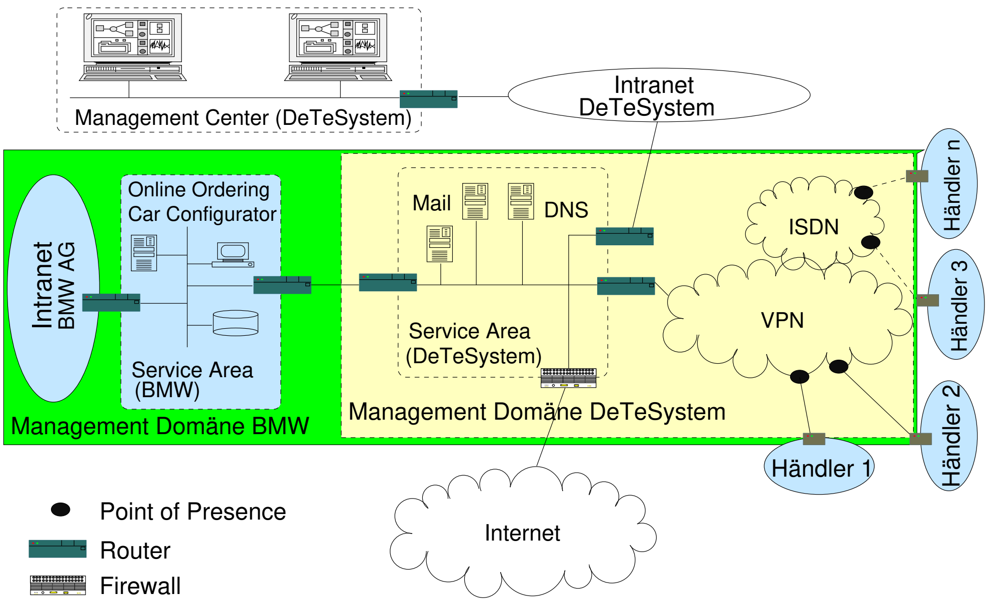


Abbildung 2.1: Extranet Szenario

Area wird dem Händler auch ein wohldefinierter und gesicherter Zugangspunkt zum Internet bereit gestellt. Die beschriebenen Infrastrukturen mit den Diensten müssen von der DeTeSystem, als Extranet–Provider, implementiert werden. Daneben muss sie auch das Management für alle von ihr realisierten Dienste und Komponenten — über die Domänengrenzen hinweg — erbringen.

Die Dienste, die den Händlern von BMW zur Verfügung gestellt werden, werden durch Server in einer eigenen Service Area bei BMW erbracht. Das Management der BMW–Dienste und der BMW Service Area obliegt BMW.

Ein Händler, der die *Online Ordering* Applikation nutzen will, wählt sich bei seinem PoP ein und wird über das VPN und die Service Area der DeTeSystem zu dem Server in der Service Area von BMW geroutet.

Die Händler kaufen alle ihre Dienste bei BMW ein, das bedeutet, dass BMW gegenüber dem einzelnen Händler als Provider, auch für die von der DeTeSystem erbrachten Dienste, auftritt. Die Dienstgüte wird also zwischen dem jeweiligen Händler und BMW vertraglich vereinbart. Können die vereinbarten QoS–Parameter (z.B. Erreichbarkeit, Antwortzeit, Durchsatz u.a.) nicht eingehalten werden, so erhält der Händler von BMW Rabatte auf seine abonnierten Dienste. BMW selbst hat wiederum eine Dienstgütevereinbarung mit der DeTeSystem abgeschlossen, die Konventionalstrafen bei einer Verletzung der SLAs vorsieht. Abbildung 2.2 zeigt einen Ausschnitt aus dieser Multiprovider Hierarchie unter vertraglichen und organisatorischen Gesichtspunkten. Neben der rein vertraglichen Beziehung, d.h. wer schließt mit wem eine Dienstgütevereinbarung, zeigt die Abbildung auch, von wem die Dienste erbracht bzw. administriert werden. Dabei zeigt sich, dass viele Dienste erst durch das Zusammenwirken mehrer Organisationseinheiten erbracht und ver-

SLAs zwischen zwei Vertragspartnern; Diensterbringung über Providergrenzen hinweg waltet werden können. Der DNS–Dienst wird z.B. technisch von der DeTe-System erbracht. BMW muss aber die Informationen, die zur Konfiguration des Dienstes benötigt werden (z.B. Adressschemata), zur Verfügung stellen und laufend aktualisieren. Es wird deutlich, dass ein Kunde, der ein SLA mit seinem Provider schließt, nicht erkennen kann, wer die vereinbarten Dienste tatsächlich erbringt. Je komplexer der Dienst ist, umso mehr „Zuliefer–Dienste“ sind darin enthalten, d.h. ein komplexer Dienst kann oft nur verteilt und über Providergrenzen hinweg, erbracht werden.

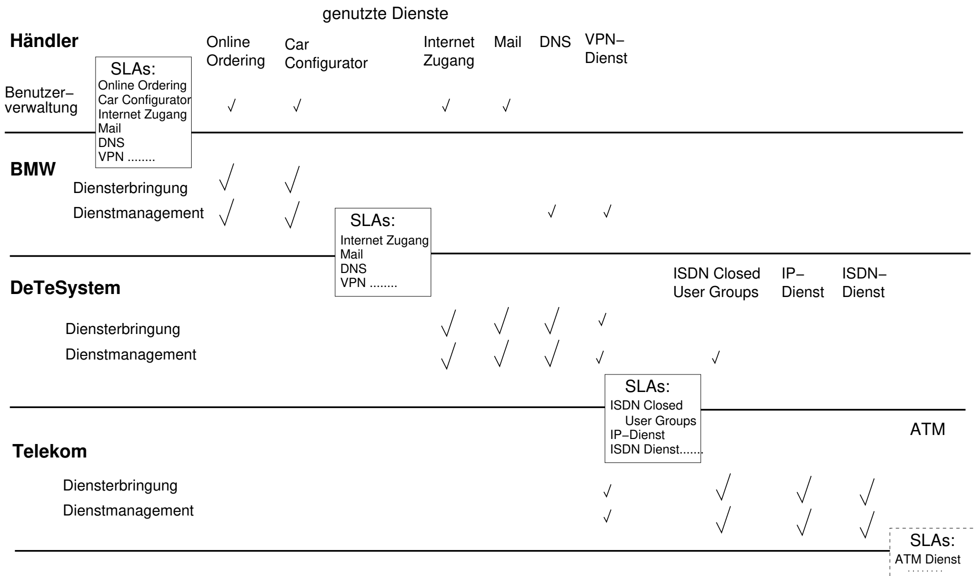


Abbildung 2.2: Multiprovider Hierarchie – Dienstgütevereinbarung und Diensterbringung

Dienstgüte definiert aus Sicht des Kunden

Bei der Überwachung der QoS Parameter in solchen Multiprovider Hierarchien treten sowohl technische als auch organisatorische Probleme auf. Falls eine SLA–Verletzung bzw. die Nichteinhaltung eines QoS–Parameters vorliegt, ist es nicht trivial, den Verantwortlichen dafür zu bestimmen. Im Folgenden sei angenommen, dass mit einem Händler eine SLA vereinbart wurde, in der ihm eine Erreichbarkeit der Online Ordering Applikation von 100 % zwischen 8^{00} und 18^{00} Uhr zugesichert wurde. Falls er den Dienst nutzen will, aber nicht erreichen kann, erhält er bestimmte Rabatte, d.h. der Regressfall tritt nur ein, falls der Händler auch versucht, den Dienst zu nutzen. Falls er den Dienst nicht erreicht, so kann dies viele Ursachen haben. Falls die Server oder Vermittlungsrechner in der Service Area von BMW nicht funktionieren, liegt die Verantwortung bei BMW. Es könnte aber auch sein, dass Komponenten oder Dienste der DeTeSystem ausgefallen sind oder der PoP nicht funktioniert und deshalb nur die Händler, die über diesen PoP angeschlossen sind, den Dienst nicht nutzen können. Das Problem kann aber auch vom Händler selbst zu verantworten sein, wenn es z.B. durch einen Fehler in seinem lokalen Netz verursacht wird.

Der Händler wird bei einer Verletzung der ihm zugesicherten Dienstgüte immer eine Erstattung und unter Umständen sogar Konventionalstrafen von

BMW verlangen. Falls die Ursache der SLA–Verletzung jedoch bei der DeTeSystem liegt wird BMW die DeTeSystem und diese ggf. wiederum die Telekom, in Regress nehmen. Für das Dienst– und QoS–Management bedeutet dies, dass Mechanismen notwendig sind, um die Einhaltung der QoS–Parameter zu überwachen und eine Zuweisung der Verantwortlichkeit bei deren Verletzung zu ermöglichen.

Die Dienstgüte wird im Beispielfall aus Kundensicht definiert, d.h. nur wenn der Kunde versucht einen Dienst zu nutzen, kann eine Verletzung der SLAs auftreten. Fällt ein Dienst aus und versucht keiner der Händler diesen zu nutzen, wird auch kein Regressanspruch entstehen. Viele QoS–Parameter lassen sich also nur „aus der Sicht“ des Händlers bestimmen. Aus diesen Gründen muss der QoS von jedem einzelnen Händler aus überwacht werden. Dies bedeutet, dass Funktionalität des BMW– bzw. des DeTeSystem–Managementsystems auf die Kundenseite delegierbar sein muss, um dort zur Ausführung gebracht zu werden. Für den Fall, dass eine Verletzung von QoS–Parametern vorliegt, ist es denkbar, geeignete Diagnosefunktionen nachzuladen, um den Verursacher zu ermitteln (vgl. auch [HaRe 00]).

QoS nur aus Händlersicht zu bestimmen

In dem angegebenen Szenario kommt erschwerend hinzu, dass individuelle Vereinbarungen mit jedem Händler (selbstständiges Unternehmen) möglich sein müssen. Die zu überwachenden QoS–Parameter unterliegen auch häufigen Änderungen, es genügt also nicht, beim Anschluss eines neuen Händlers, einen Management Agenten (im traditionellen Sinn) auf dem Rechner des Händlers zu installieren, da bei einer Änderung der SLAs dann bei jedem betroffenen Händler ein neuer Agent installiert werden müsste.

Eine Lösung dieser Probleme stellen Mobile Agenten dar. Im angegebenen Szenario muss dazu einmalig eine Ausführungsplattform für Mobile Agenten auf Seiten des Händlers installiert werden. Die Funktionen zur Überwachung der QoS–Parameter können dann individuell für jeden Händler in Form eines Mobilen Agenten implementiert und von BMW und/oder der DeTeSystem auf das System des Händlers migriert werden. Bei einer Veränderung der SLA kann der Mobile Agent entweder sehr einfach durch einen neuen mit geänderter Funktionalität ersetzt werden oder der Mobile Agent kann dynamisch neue Funktionen nachladen, um den geänderten Anforderungen gerecht werden zu können. Der Mobile Agent misst aus der Sicht des jeweiligen Händlers dessen tatsächliche QoS–Parameter. Sollten Verletzungen der SLAs erkannt werden, kann der Agent Diagnosefunktionen ausführen, um den Verursacher für die Verletzung und damit den Regresspflichtigen zu ermitteln.

Mobile Agenten überschreiten Organisationsgrenzen

Da es in Fällen von Verletzungen von SLAs um hohe Schadenssummen geht, ist sowohl die Überwachung der QoS–Parameter als auch die Beweislast von erheblicher wirtschaftlicher Relevanz für alle Beteiligten. Dies bedeutet, dass neben BMW auch die DeTeSystem ein Interesse daran hat, die QoS von Händlerseite aus zu überwachen.

erhebliche wirtschaftliche Relevanz wegen hoher Schadenssummen

Für den Händler bedeutet dies, dass auf seinem System Software von mehreren „fremden“ Firmen, in Form von Mobilen Agenten, ausgeführt wird. Damit der Händler überhaupt bereit ist dies zuzulassen, müssen höchste Sicherheits-

Sicherheitsanforderungen des Händlers

standards eingehalten werden. Der Händler muss beispielsweise kontrollieren können auf welche Informationen und Ressourcen der Mobile Agent zugreifen darf (Zugriffskontrolle). Diese Zugriffe und Aktionen des Mobilen Agenten möchte er aus Gründen der Beweissicherung auch verbindlich protokollieren (Auditing) und später auch justitiabel belegen können (Verbindlichkeit). Er will vermeiden, dass mit Hilfe eines Mobilen Agenten vertrauliche Informationen aus seinem internen Netz ausgespäht werden (Vertraulichkeit) oder die Verfügbarkeit seiner Systeme durch den Mobilen Agenten eingeschränkt wird. Er muss sicherstellen können, dass der Mobile Agent von einer vertrauenswürdigen Quelle auf sein System migriert wurde (Authentisierung der Quelle).

Sicherheitsanforderungen der Provider

Auf der anderen Seite hat auch derjenige, der einen Mobilen Agenten migriert, Anforderungen an die Sicherheit. Im Folgenden seien nur einige dieser Anforderungen exemplarisch angegeben. Grundsätzlich muss sich derjenige, der einen Mobilen Agenten migriert, auf die Daten, die der Mobile Agent zurückliefert, verlassen können. Er muss sicherstellen können, dass der Mobile Agent korrekt ausgeführt wird und dass der Mobile Agent bzw. die Daten, die er liefert, nicht manipuliert wurden (Integrität, Verbindlichkeit).

Im Beispielszenario zahlt es sich potentiell für jeden Beteiligten aus, eine geschickte Manipulation der Daten in seinem Sinne durchzuführen. Der Händler könnte durch geeignete Veränderung des Mobilen Agenten erreichen, dass er unberechtigterweise Rabatte erhält oder Einnahmen durch Konventionalstrafen erzielt. Ein Provider aus der Multiprovider Hierarchie könnte durch geschickte Manipulationen von seiner Verantwortung für eine von ihm verursachte Verletzung der Dienstgütevereinbarung ablenken. Unter Umständen könnte die Manipulation auch dazu führen, dass ein anderer Provider in der Hierarchie als Schuldiger erscheint und sich daher mit Regressforderungen konfrontiert sieht. Alle beteiligten Organisationen wollen und müssen die Nutzung ihrer Hard– und Software–Komponenten kontrollieren und ggf. auch beschränken können (Ressourcen Beschränkung).

Aus diesem Szenario lassen sich im Vorgriff auf Abschnitt 2.6, folgende konkrete Sicherheitsanforderungen an ein Managementsystem ableiten:

- Zugriffskontrolle
- Auditing
- Verfügbarkeit
- Integrität
- Verbindlichkeit
- Vertraulichkeit
- Authentisierung
- Ressourcen Beschränkung

2.2.2 Management und Betrieb von Mobilfunknetzen

Mit der dritten Generation der Mobilfunktechnik — in Europa realisiert durch das **Universal Mobile Telecommunications System (UMTS)** — sollen eine Vielzahl verschiedenster Dienste angeboten und verschiedenste Netztechnologien unterstützt werden. Das Hauptziel von UMTS ist es, einem nomadischen Benutzer einen unbeschränkten Zugang zu einer großen Zahl unterschiedlicher und auch personalisierter Dienste zu gewähren. Dabei sollen sowohl die technischen als auch die administrativen Grenzen für den Benutzer transparent sein.

UMTS soll alle Formen der Mobilität unterstützen. Die **persönliche Mobilität** der Benutzer darf nicht auf die Domäne eines einzelnen Providers beschränkt sein, sondern der Benutzer muss den Provider beliebig wählen und auch kurzfristig wechseln können. Der Benutzer möchte einen möglichst unbeschränkten Zugang zu allen seinen Diensten. Die **Dienstmobilität** soll dies nicht nur für standardisierte Dienste, sondern auch für Dienste, die speziell auf den Benutzer angepasst sind (Tailored Services), gewährleisten. Ein Benutzer muss für Dienste, z.B. den normalen Sprachdienst oder einen Videokonferenzdienst, die Möglichkeit haben die aktuelle Sitzung zu suspendieren, um sie dann auf einem anderen Endgerät und/oder über ein anderes Zugangsnetz fortzusetzen. Im Idealfall erfolgt, z.B. für einen Videokonferenzdienst, ein automatischer Wechsel zwischen dem lokalen Unternehmensnetz und dem UMTS–Netz sowie zwischen dem stationären Rechner und einem mobilen Gerät, sobald der Nutzer sein Büro verlässt. Diese Art der Mobilität wird als **Sitzungsmobilität** bezeichnet [Küp 01].

UMTS soll alle Formen der Mobilität unterstützen

In UMTS wird der Benutzer nicht nur seine Provider und Dienste frei wählen, sondern einen Dienst auch in verschiedenen Ausprägungen nutzen können. Auswahlkriterien hierbei sind u.a. die gewünschte Qualität, die Zuverlässigkeit und die Sicherheit des Dienstes sowie dessen Preis.

Für die technische Abbildung der Personalisierung von Diensten, deren kundenspezifischen Anpassungen und für die Unterstützung der verschiedenen Arten der Mobilität wurde das **Virtual Home Environment (VHE)** spezifiziert und unterliegt im Moment dem Standardisierungsprozess [TS 123 127, TS 22.121]. Das VHE ist damit die Systemkomponente, die personalisierte Dienste ermöglicht, die portabel sowohl über Netzgrenzen als auch über heterogene Endgeräte hinweg, genutzt werden können. Das VHE ist als Hierarchie von Dienstprofilen aufgebaut. Aus Sicht des Kunden bietet das VHE u.a. folgende Funktionalitäten:

Virtual Home Environment (VHE): Dienstprofil eines Benutzers

- Bestellung und Widerruf von Diensten
- Auswahl von verschiedenen Qualitätsstufen eines Dienstes
- Auswahl oder Wechsel auf ein anderes (lokales) Endgerät (Terminal Eigenschaften)
- Auswahl von gesicherten (verschlüsselten) Verbindungen
- Präsentation und Auswahlmöglichkeit für ortsabhängige Dienste (Location based Services)
- Informationen über Abrechnungsdaten
- Präsentation von Fehler– und Zustandsinformationen
- Modifikation von Benutzerdaten

Das VHE des Benutzers muss an jedem Ort und zu jeder Zeit zur Verfügung stehen, da es für ihn die zentrale Konfigurations– und Auswahlschnittstelle für seine Dienste darstellt. Die Standardisierung zum VHE ist noch nicht abgeschlossen. Beispielsweise wird in dem Europäischen ACTS Projekt *Commu-*

nication Agents for Mobility Enhancements in a Logical Environment of Open Networks (CAMELEON) der Einsatz von Mobilen Agenten zur Implementierung des VHE untersucht [cameleon, HGF 98]. Ein Mobiler Agent implementiert dabei sowohl die Funktionalität als auch die benötigte Datenhaltung des Virtual Home Environment. Der Mobile Agent migriert im „Festnetzteil" des UMTS–Netzes — über Providergrenzen hinweg — dem Benutzer nach bzw. voraus. Der Agent repräsentiert das Benutzerprofil für alle Dienste, die der Benutzer abonniert hat.

Mobiler Agent implementiert VHE

UMTS–Geschäftsmodell: Trennung zwischen Dienstlogik und Konnektivität; viele versch. Provider

Das UMTS– Geschäftsmodell kennt auf Seite des Providers drei Rollen: den **Network Operator** als Betreiber der unterschiedlichen Netze und Netztechnologien, den **Service Provider**, der auf dieser Infrastruktur seine Dienste realisiert und den **Value–added Service Provider**, der höhere Dienste oder Inhalte (Content) für Dienste des Service Providers bietet. Das Geschäftsmodell verfolgt eine strenge Trennung zwischen Dienstlogik und Konnektivität. Im Rahmen der Deregulierung wurde auch festgelegt, dass in jeder dieser Rollen die Unternehmen auf dem Markt miteinander konkurrieren können.

MA als integratives Element überwindet Domänengrenzen

Bei diesem Geschäftsmodell wird ein Mobiler Agent, der das VHE implementiert, zum integrativen Element zwischen den verschiedenen Providern und Provider–Rollen. Der Mobile Agent muss deshalb sowohl horizontal, d.h. zwischen den verschiedenen Network Operator Domänen, migrieren als auch die vertikalen Domänengrenzen zwischen den verschiedenen Service und Value–added Service Providern überbrücken können.

Sicherheitsanforderungen

Der Mobile Agent bündelt die Informationen der verschiedenen Provider mit denen des Benutzers. Sowohl der Benutzer als auch jeder der Provider möchte Teile seiner Daten vor anderen Providern geschützt sehen (Vertraulichkeit) oder zumindest festlegen können, wer auf welche Daten zugreifen kann (Zugriffskontrolle). Für Location based Services geht bspw. der geographische Aufenthaltsort eines Benutzers mit in die Dienstcharakteristik ein. Diese Informationen sind aber für einen Sprachdienstprovider weder nötig noch soll er überhaupt Zugriff auf diese Information erhalten.

Eine Änderung an den Dienstprofilen und den Charakteristika eines Dienstes führt i.d.R. auch zu einer Preisveränderung. Nur der Benutzer selbst bzw. für technische Dienstcharakteristika der entsprechende Service Provider dürfen diese Dienstprofile ändern; d.h. sowohl der Benutzer als auch der Provider muss sich zuverlässig beim Mobilen Agenten authentisieren. Änderungen müssen — insbesondere unter Abrechnungsgesichtspunkten — nachvollziehbar (Auditing) und verbindlich (Verbindlichkeit) sein.

2.2.3 Management des Flugverkehrs

Eurocontrol [EC], eine europäische Organisation mit zur Zeit 30 Mitgliedsstaaten [EEC 00], ist verantwortlich für die Sicherheit der Navigation im europäischen Luftraum. Die Aufgabe von Eurocontrol ist die Entwicklung eines kohärenten und koordinierten Luftverkehrskontrollsystems für Europa.

Oberstes Ziel einer koordinierten europäischen Flugverkehrskontrolle muss es sein, Stausituation in bestimmten Bereichen des Luftraums und Verspätungen zu minimieren bzw. möglichst frühzeitig zu erkennen und Gegenmaßnahmen zu ergreifen. Im Idealfall lässt sich die zukünftige Verkehrslast in einem bestimmten Sektor des Luftraumes vorausberechnen und mit diesen Daten eine Stausituation, noch vor ihrem eigentlichen Eintreten, verhindern.

Koordinierte europäische Flugverkehrs-kontrolle zur Minimierung von Verspätungen und Stausituationen

Die Schwierigkeit dabei ist, dass die Flugverkehrskontrolle eine hoheitliche Aufgabe der Mitgliedsstaaten ist und Eurocontrol nur Empfehlungen aussprechen kann. Der Luftraum ist in nationale Flugsicherungsbereiche (Sektoren) aufgeteilt und innerhalb der Grenzen der Mitgliedsstaaten sind diese Bereiche weiter unterteilt. Für jeden Sektor gibt es eine Flugüberwachung, die für die Kontrolle aller den Sektor überfliegenden Flugzeuge zuständig ist. In Abbildung 2.3 [EEC 00] sind diese Sektoren und die Häufigkeit der dort auftretenden Konflikte dargestellt. Bei einem (inner–) europäischen Flug werden in jedem überflogenen Land i.d.R. mehrere nationale Flugsicherungsbereiche überquert. Bei jeder Querung sowohl national als auch zwischen Staaten muss eine Übergabe an die neue Flugsicherung erfolgen.

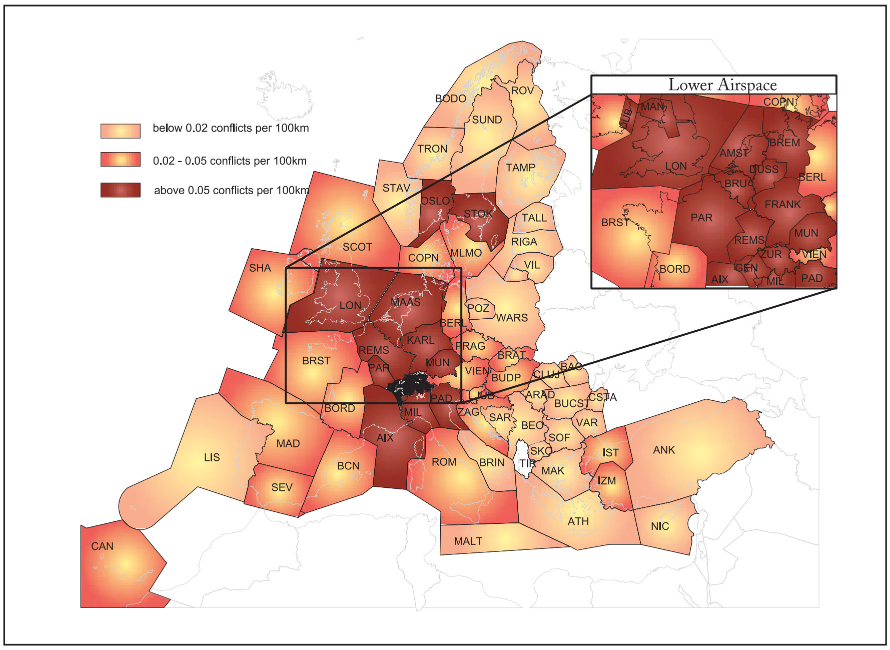


Abbildung 2.3: Europäische Flugsicherungsbereiche aus [EEC 00]

Um den Flugverkehr aus globaler, d.h. europäischer Sicht, optimieren zu können, braucht Eurocontrol Flugdaten. Bisher werden diese Daten nach Beendigung des jeweiligen Fluges von den entsprechenden Fluggesellschaften zur Verfügung gestellt. Mit diesen Daten werden Simulationen durchgeführt, um bspw. bessere Slot–Zuteilungsstrategien zu finden [EEC 01]. In der Regel werden von den Fluggesellschaften lediglich Flugzeiten und Flughöhen mitgeteilt. Der Flugverkehr könnte aber sehr viel besser optimiert werden, wenn möglichst zeitnah zusätzliche Daten über den Flug bekannt gegeben würden. Als Beispiele seien hier nur die momentane Geschwindigkeit, Fluggewicht,

Eurocontrol benötigt aktuelle und detaillierte Daten

Treibstoffmenge, Treibstoffverbrauch, Navigationsdaten sowie die kurz– und mittelfristige Flugplanung des Kapitäns genannt. Einige dieser Daten lassen sich aus den Radardaten der verschiedenen Flugsicherungen extrapolieren, allerdings ist dies fehleranfällig [EEC 01a]. Sind diese Daten bekannt, so lassen sich sehr viel genauere Flugbahnberechnungen und Flugbahnplanungen durchführen.

Auch die Sicherheit des Flugverkehrs ließe sich verbessern, wenn die im Flugzeug ermittelten lokalen Wetterdaten für die Flugsicherung zugänglich wären. Einige Flugzeughersteller bauen bereits Laser basierte Radarsysteme (Lidar) ein, um Turbulenzen, die in Flugrichtung liegen, zu erkennen [Perr 00]. Auch diese Daten wären sehr wertvoll, um den Flugverkehr in dem entsprechenden Sektor an den Turbulenzen vorbeizulenken.

Die Idee ist nun, diese Daten und zusätzliche aktuelle Flugsicherungsdaten direkt während des Fluges zu ermitteln. Dieser Ansatz ist allerdings mit einigen Problemen behaftet. Im Flugverkehr hat der Sprachverkehr absoluten Vorrang vor etwaigem Datenverkehr. Für Datenverkehr steht nur eine niedrige Bandbreite zur Verfügung. Diese Faktoren sowie die häufige Übergabe zwischen verschiedenen Flugsicherungsbereichen führt häufig zum Ausfall der Datenverbindung.

Mobiler Agent ermittelt Daten direkt während des Fluges

Zur Lösung dieser Problematik bietet sich der Einsatz Mobiler Agenten an. Dazu müsste Eurocontrol eine definierte Ausführungsplattform für Mobile Agenten spezifizieren, die in den Flugzeugen implementiert wird. Eurocontrol und jede nationale Flugsicherung kann dann während des Überfluges einen Mobilen Agenten im Flugzeug ausführen, um die gewünschten Daten zu sammeln, zu verarbeiten und zu verdichten. Solange Datenverkehr möglich ist, liefert der Mobile Agent zeitnah die aktuellen Daten. Im anderen Fall, d.h. falls die Datenverbindung, z.B. durch eine Sprachverbindung unterbrochen oder eingeschränkt wird, werden die Daten vom Agenten gepuffert und bei wieder vorhandener Verbindung oder im Rahmen der Übergabe an der Sektorengrenze übermittelt.

Höchste Sicherheitsanforderungen

Diesem Szenario inhärent, müssen höchste Sicherheitsanforderungen eingehalten werden. Der Mobile Agent darf den normalen Flugablauf und die Kommunikation in keinster Weise beeinflussen (Ressourcenbeschränkung). Bereits die oben genannten Daten stellen für die Fluggesellschaften strategisch wichtige Informationen dar. Über ihre individuelle Flugplanung, die sich aus diesen Daten ableiten lässt, unterscheiden sie sich von ihrer Konkurrenz. Durch eine geeignete individuelle Flugplanung können erheblich Kosten eingespart werden. Damit ist auch klar, dass nur berechtigte und authentisierte Agenten diese Daten erhalten dürfen (Authentisierung). Der Agent darf auf die Daten nur lesend zugreifen und auch dieser Zugriff muss auf definierte Daten beschränkt werden können (Zugriffskontrolle). Werden die Daten, neben den Radardaten, auch von der Flugsicherung benutzt, um die Flugbahnberechnung und die Slot–Zuteilung zu bestimmen, müssen die Daten absolut zuverlässig und unverändert sein (Integrität der Daten). Die Verfügbarkeit der Daten ist von untergeordneter Bedeutung. Eine absolute Verfügbarkeit ist auch nicht erforderlich, da traditionelle Verfahren sowie Daten der Luftverkehrskontrolle zur Flugbahnplanung verwendet werden können. Die von Mobilen Agenten gelieferten Daten dienen der Erhöhung der Genauigkeit, zur Optimierung und damit zur Einsparung von Kosten.

2.3 Klassifizierung von Mobilen Agenten Systemen

Mobile Agenten sind ein relativ junger Forschungsbereich, daher ist die Begriffsbildung noch inkonsistent, keineswegs einheitlich und zum Teil sogar widersprüchlich. In diesem Kapitel wird deshalb eine einheitliche Nomenklatur eingeführt, die den begrifflichen Kontext für die folgenden Kapitel bildet. Als Basis für die Begriffsbildung dient dabei die Mobile Agent System Interoperability Facility der **Object Management Group [OMG]**.

2.3.1 Mobile Agent System Interoperability Facility (MASIF)

Bestehende Plattformen für Mobile Agenten (MA) unterscheiden sich in den Konzepten, in ihrer Architektur und in der Implementierung erheblich voneinander. Diese Unterschiede erschweren oder verhindern eine Interoperabilität zwischen verschiedenen MA–Plattformen und damit auch eine rasche Verbreitung der MA–Technologie.

Aus diesen Gründen hat sich die OMG entschlossen, einen Standard für Mobile Agenten Plattformen zu entwickeln. Die **Mobile Agent System Interoperability Facility (MASIF)** [MASIF] soll die Interoperabilität zwischen MA–Plattformen verschiedener Hersteller ermöglichen und eine einheitliche Begriffsbildung etablieren. Dazu wurden ein Basismodell und die grundlegenden Begriffe für eine verteilte MA–Plattform spezifiziert.

Agent

Ein **Agent**, im Sinne von MASIF, ist ein Software–Baustein, der autonom im Auftrag eines Nutzers oder einer Organisation handelt. Diese für den Agenten verantwortliche Person oder Organisation wird als **Agent Authority** bezeichnet. Bei den Agenten werden mobile und stationäre Agenten unterschieden.

Mobiler Agent

Ein **Mobiler Agent** ist nicht an das System gebunden, auf dem er gestartet wurde. Er kann auf eigene Veranlassung, bzw. auch von außen initiiert, auf ein anderes System migrieren.

Stationärer Agent

Im Gegensatz dazu ist ein **Stationärer Agent** nicht in der Lage zu migrieren.

Agentensystem

Als **Agentensystem (Agent System, AS)** wird die Laufzeitumgebung für mobile und stationäre Agenten bezeichnet. Innerhalb eines Agentensystems können Agenten erzeugt, interpretiert, ausgeführt, suspendiert, reaktiviert, transferiert oder terminiert werden. Auch für das Agentensystem existiert ei-

Place

ne verantwortliche **AS–Authority**. Als **Place** wird ein bestimmter Kontext innerhalb eines Agentensystems verstanden, in dem Agenten ausgeführt werden.

Region

Agentensysteme werden über eine Kommunikationsinfrastruktur miteinander verbunden. Die Agentensysteme, die derselben Authority unterstehen, werden zu einer so genannten **Region** zusammengefasst. Das Basismodell von MASIF ist in Abb. 2.4 dargestellt.

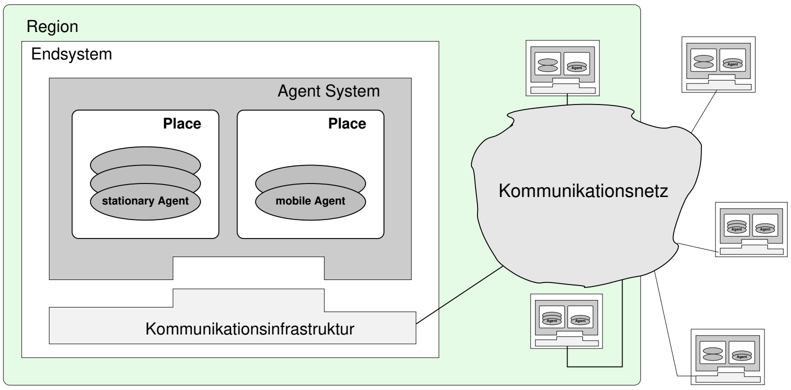


Abbildung 2.4: MASIF–Basismodell

Neben diesen grundlegenden Begriffen wurden in MASIF die Aufrufschnittstellen für folgende Bereiche standardisiert:

Agent Management

1. Beim ***Agent Management*** werden einheitliche Zugriffsschnittstellen definiert, um Agenten erzeugen, suspendieren bzw. reaktivieren oder terminieren zu können. Ziel dabei ist, dass ein Administrator auf die verschiedensten Agentensysteme mit denselben Operationen zugreifen kann.

Agent Transfer

2. ***Agent Transfer*** standardisiert die Art der Übertragung des Mobilen Agenten, d.h. die Aufrufschnittstelle (`migrate_agent`) beim Agentensystem. Es wird ein Life–Cycle für Agenten spezifiziert und festgelegt, dass es eine Möglichkeit der Serialisierung für Daten und Code des Agenten geben muss. Die konkrete Realisierung dieser Verfahren ist nicht Teil der Standardisierung, sondern bleibt der Implementierung überlassen.

Agent Naming

3. ***Agent Naming*** definiert ein Namensschema und die Semantik der Namen sowohl für Agenten als auch für Agentensysteme (vgl. auch Abschnitt 4.2.2).

Agent Tracking

4. ***Agent Tracking*** spezifiziert, wie ein Agent lokalisiert werden kann.

Sicherheitsüberlegungen für Multi–Hop–Agenten, Übersetzung von Agenten in andere Implementierungssprachen und Bridges zwischen verschiedenen AS–Implementierungen, die eine derartige Umsetzung ermöglichen würden, werden explizit als Aspekte genannt, die im Moment nicht standardisiert werden können und sollen.

MASIF macht auch keinerlei Aussagen zu Aspekten der Implementierung. Es werden zwar Schnittstellen in IDL (Interface Definition Language [OMG 01-09-34]) definiert aber Aussagen zu deren technischer Umsetzung werden bewusst nicht gemacht.

2.3.2 Implementierungsklassen

Eine Implementierung eines Systems Mobiler Agenten wird im Wesentlichen bestimmt durch die Implementierung des Agentensystems mit seinen Schnittstellen. Dadurch wird, ein zwar grober, aber doch entscheidender Rahmen auch für die Implementierung der Mobilen Agenten vorgegeben.

Das Agentensystem als Laufzeitumgebung für Mobile Agenten fungiert als Mittler zwischen Agent und dem unterliegenden System. Generell gibt es zwei Möglichkeiten für die Realisierung eines Agentensystems, die Implementierung als spezieller Betriebssystemdienst oder die Realisierung als Anwendungssoftware.

Agentensystem als Teil des Betriebssystems

Im ersten Fall ist das Agentensystem fester Bestandteil des Betriebssystems, wie z.B. bei TACOMA [JRS 95, Joha 98, Tacoma], im zweiten Fall ist das Agentensystem aus Sicht des Betriebssystems ein normaler Anwendungsprozess. Der zweiten Alternative wird in der Mehrzahl der existierenden MA–Plattformen der Vorzug gegeben. Die Implementierung als Teil des Betriebssystems behindert die gewünschte Plattformunabhängigkeit erheblich. Für jedes zu unterstützende System muss bei dieser Architekturvariante ein eigenes Agentensystem implementiert werden.

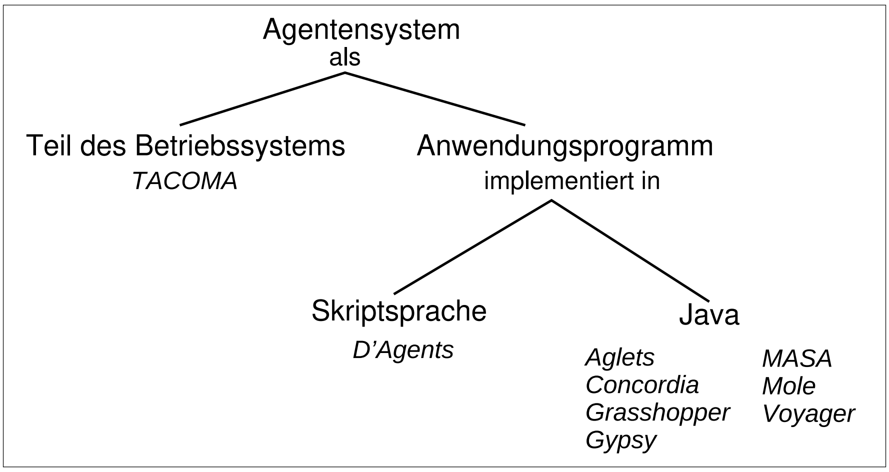


Abbildung 2.5: Implementierungsklassen für Agentensysteme

Agentensystem als Anwendungsprozess

Die zweite Gruppe — Agentensystem als Anwendungsprogramm — lässt sich weiter unterteilen in MA–Plattformen, die Java als Implementierungssprache verwenden, und solche, die Skriptsprachen nutzen. D'Agents [DAgents] (früher als Agent Tcl bezeichnet) [Gray 95] verwenden bspw.

einen veränderten Tcl–Dialekt und damit auch einen erweiterten Tcl–Interpreter. Zukünftig sollen auch Python [Python], Scheme [Scheme] und Java unterstützt werden.

Zu den Java–basierten MA–Plattformen zählen u.a. Aglets [LaOs 98, OKO 98, Aglets], Concordia [Mits 98, Concordia], Grasshopper [BBCM 98, Grasshopper], Gypsy [Gypsy], MASA [KRRV01], Mole [BHR 97, Mole] und Voyager [Voyager].

In Abbildung 2.5 sind die verschiedenen Arten der Implementierung, mit entsprechenden exemplarischen Vertretern, zusammengefasst. Ein Überblick über die verschiedenen Systeme findet sich in [PhKa98, Pets 01].

Da die relative Plattformunabhängigkeit und die einfache Unterstützung heterogener Umgebungen für Managementsysteme ein entscheidender Faktor ist und sich dies mit Agentensystemen, die direkt als Teil des Betriebssystems implementiert sind, nur schwer erreichen lässt, wird diese Klasse im Folgenden nicht mehr betrachtet.

2.4 Modellbildung von Mobilen Agenten Systemen

Ziel: generische Sicherheitsarchitektur

Bisherige Betrachtungen zur Sicherheit in Mobilen Agenten Systemen (vgl. Abschnitt 3) behandeln spezielle Aspekte der Sicherheit, oft aus dem Bereich Electronic Commerce. Bisher gibt es aber kein umfassendes Sicherheitskonzept für Mobile Agenten im Management. Es existieren auch kaum theoretische Modelle, die in der Lage sind Systeme Mobiler Agenten implementierungsunabhängig zu beschreiben. Um aber eine generische Sicherheitsarchitektur entwickeln zu können, die für alle Managementsysteme auf der Basis von Mobilen Agenten verwendet werden kann, muss ein allgemeines Modell unabhängig von konkreten Realisierungen entwickelt und analysiert werden. Die Grundidee ist es, anhand dieses Modells alle möglichen Angriffspunkte auf das System zu finden und diese dann geeignet zu sichern.

Relation zwischen Entitäten

In den folgenden Abschnitten wird das entwickelte Modell, bzw. dessen Teilmodelle, vorgestellt. Die Struktur des Modells basiert auf der Erkenntnis, dass ein Angreifer nur in der Lage ist einen Angriff durchzuführen, wenn er eine Relation zum Angriffsziel herstellen kann. Im Allgemeinen wird der Begriff **Relation** zwischen Partnern (Entitäten) durch die Möglichkeit des Informationsaustausches zwischen den Entitäten definiert. Der Begriff Information muss hierbei noch nicht näher eingeschränkt werden, d.h. wenn Entitäten in der Lage sind, irgend eine Art von Information auszutauschen, besteht zwischen ihnen eine Relation. Der Begriff wird in Abschnitt 2.4.2 näher gefasst werden. Ein Angreifer hat neben dem Angriff auf eine Entität auch die Möglichkeit, eine zwischen Entitäten bestehende Relation anzugreifen. Existiert keine Relation von einer Entität zu einer anderen und kann auch keine Relation zu einer Entität aufgebaut werden, so ist ein Angriff nicht möglich. Um Angriffspunkte und Schwachstellen zu finden, müssen also die Relationen und die an diesen Relationen beteiligten Entitäten ermittelt und näher untersucht werden.

2.4.1 Entitätenmodell

Die Akteure, d.h. die Entitäten eines Managementsystems werden grundsätzlich durch das Organisationsmodell festgelegt. Deshalb kann für ein MA–basiertes Managementsystem ein allgemeines Organisationsmodell als Basis für die Identifikation der Entitäten dienen. Dabei muss es unerheblich sein, ob es sich um ein hierarchisches, zentrales, Multicenter Control oder ein anderes Managementsystem (vgl. [HAN 99a]) handelt, denn in jedem dieser Systeme können Mobile Agenten zum Einsatz kommen.

Als Basis für das im Folgende vorgestellte Entitätenmodell wird das Manager–Agent Modell verwendet, das bereits in Abschnitt 2.1 kurz vorgestellt wurde. Diese Modell wird um die spezifischen Entitäten, die für den Betrieb eines Mobilen Agenten Systems typisch und notwendig sind, erweitert.

Manager–Agent Modell als Basis

Beim **Manager–Agent Modell**, das eine asymmetrische/hierarchische Kooperationsform zugrunde legt, handelt es sich um ein dem Client/Server Modell sehr ähnliches Auftraggeber/Auftragnehmer Verhältnis. Systeme, die steuernd auf andere Systeme einwirken, werden als Manager bzw. Managementsystem bezeichnet. Andererseits werden Komponenten, die vom Manager beauftragt werden, bestimmte Operationen auszuführen oder Informationen bereitzustellen, als Agenten bezeichnet. Nachdem die Akteure Manager und Agent in jedem Managementsystem vorhanden sind und natürlich auch Ziel eines Angriffes sein können, werden sie in das Entitätenmodell übernommen.

Subjekt–Objekt Relation $\triangleq$ Auftraggeber/ Auftragnehmer–Beziehung

Die hierarchische Kooperationsform impliziert auch die sicherheitstechnisch relevante **Subjekt–Objekt Relation**. Ein Subjekt beauftragt eine andere Entität, das Objekt in der Relation. Um kooperatives Management zu ermöglichen, dürfen die Rollen den Entitäten nicht statisch zugeordnet werden, sondern eine Entität kann auch gleichzeitig beide Rollen einnehmen oder zwischen diesen Rollen wechseln (vgl. Abb. 2.6).

Mobile Agenten als Erweiterung des klassischen Agentenbegriffs

Beim Einsatz mobiler Agenten muss dieses „traditionelle“ Entitätenmodell erweitert werden. Neben dem klassischen Agenten, der stellvertretend für eine zu verwaltende Ressource (Managed Ressource) steht, kann ein Mobiler Agent mehrere Ressourcen nacheinander besuchen und ist nicht an eine bestimmte gebunden. Der Mobile Agent stellt damit ein allgemeineres Konzept dar und erweitert den klassischen Agentenbegriff um Migration und Kooperation.

Um einen Mobilen Agenten auf einem System ausführen zu können, wird

Agentensystem als Vermittler zwischen MA und Managed Resource

i.d.R. eine Laufzeitumgebung,das Agentensystem, benötigt. Die beiden Entitäten „Mobiler Agent“ und „Agentensystem“ müssen sich deshalb auch im Entitätenmodell wiederfinden.

Abbildung 2.6 stellt das resultierende Entitätenmodell dar. Es handelt sich dabei um eine verallgemeinerte Darstellung die auf konkrete Managementsysteme abgebildet werden kann. Alle möglichen Kooperationsformen und Subjekt–Objekt Relationen sind dargestellt. Auch ein Manager–of–Managers oder ein System, das Mid–Level–Manager verwendet, kann abgebildet werden.

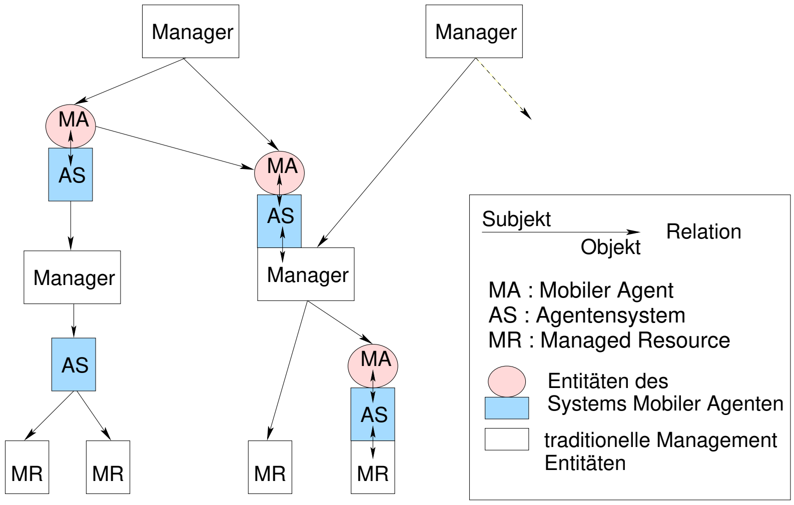


Abbildung 2.6: Entitätenmodell

Anwender = Benutzer, Betreiber oder Verwalter

In einer sicherheitstechnischen Betrachtung dürfen die Menschen, die die bisher aufgeführten Entitäten implementieren, benutzen oder verwalten, nicht außer Acht gelassen werden. Daher wird eine Entität „**Anwender**“ eingeführt, die Benutzer, Betreiber oder Verwalter der anderen Entitäten umfasst. Die Entität Anwender kann ggf. in einem Rollenmodell weiter verfeinert werden, damit funktionale und organisatorische Personalstrukturen auch innerhalb einer Sicherheitsarchitektur abgebildet werden können.

Entitäten des Modells

Die Entitäten eines Managementsystems basierend auf Mobilen Agenten lassen sich wie folgt zusammenfassen:

- **Manager** bzw. **Managementsysteme** als Quelle und als Ziel von Mobilen Agenten sowie als Subjekt und auch als Objekt von Managementoperationen.

- **Managed Resources** als Objekte, auf denen Managementoperationen ausgeführt werden.

- **Agentensysteme** als Ausführungsplattformen für Mobile Agenten. Agentensysteme können in einem Manager eingebettet sein oder auf einer Managed Ressource ablaufen.

- **Mobile Agenten** als Software–Bausteine, die innerhalb eines Netzes migrieren können, die von einem Manager initiiert und von einem Agentensystem zum Zwecke des Managements ausgeführt werden.
- **Anwender** sind Menschen, die in den verschiedensten Rollen, z.B. als Benutzer, Betreiber oder Verwalter der anderen Entitäten auftreten oder die Entitäten anderweitig benutzen.

Neben dem Manager–Agent Modell existiert auch das Peer–to–Peer Modell, bei dem eine vollständig symmetrische Kooperationsform zugrunde liegt. Der Peer–to–Peer Ansatz wurde im Entitätenmodell nicht betrachtet, da der hierarchische Ansatz für Sicherheitsbetrachtungen besser geeignet ist. Durch die implizierte Hierarchie tritt die Subjekt–Objekt Relation, die natürlich auch im Peer–to–Peer Ansatz vorhanden ist, klarer in Erscheinung. Außerdem kann der Peer–to–Peer Ansatz sowohl funktionell als auch strukturell auf das vorliegende Entitätenmodell abgebildet werden.

Aus dem Entitätenmodell wird auch ersichtlich, dass klassische Managementsysteme und Mobile Agenten gleichzeitig bzw. auch nebeneinander verwendet werden können. Agentensysteme können lokal auf einem Managementsystem oder auf einer zu verwaltenden Ressource, aber auch auf einem eigenen und räumlich getrennten System ablaufen. Diese Trennung ist notwendig, da nicht angenommen werden kann, dass alle Managementsysteme und Komponenten in der Lage sind, ein vollständiges Agentensystem auszuführen. Falls beispielsweise eine zu verwaltende Komponente nicht über genügend Rechenleistung oder Speicher verfügt, um darauf ein Agentensystem ablaufen zu lassen, kann ein „naheliegendes“ performanteres System als Stellvertreter für die eigentliche Ressource dienen. Mit Hilfe dieses Proxy–Konzeptes lassen sich Alt–Systeme und –Anwendungen (**Legacy Systeme**) durch Mobile Agenten verwalten.

2.4.2 Relationenmodell: Ausführungs–, Aufruf– und Kommunikationsrelation

Neben der angeführten Subjekt–Objekt Relation zwischen den verschiedenen Entitäten existieren weitere Relationen, die in lokale und entfernte Relationen unterteilt werden. Die **lokalen Relationen** zu einem zu verwaltenden System oder einem Manager werden in den Abbildungen 2.6 und 2.7 durch sich direkt berührende Entitäten repräsentiert. **Entfernte Relationen**, die eine Kommunikation (über ein Netzwerk) mit entfernten Entitäten darstellen, werden in Abbildung 2.6 durch freie Pfeile repräsentiert. Abbildung 2.7 zeigt eine vereinfachte Darstellung von möglichen Interaktionen in einer beispielhaften Konfiguration eines Systems Mobiler Agenten.

Das Agentensystem als Laufzeitumgebung führt die Mobilen Agenten aus und läuft selbst auf einem Host System einer Managed Resource oder auch als integrierter Bestandteil einer Managementplattform oder eines Managementwerkzeugs. In diesem Fall besteht zwischen diesen Entitäten eine loka-

Ausführungsrelation: direkte Interaktion, gemeinsame Ressourcennutzung

le Relation. Diese **Ausführungsrelation** (engl. Execution Relation) besteht zwischen Software–Bausteinen auf demselben Rechner, sie setzt direkte Interaktion sowie die gemeinsame Nutzung von Ressourcen und Betriebsmitteln voraus. Außerdem ist die Ausführungsrelation asymmetrisch, eine Entität (der **Executor**, z.B. das Agentensystem) führt eine andere Entität (den **Executee**, z.B. den Mobilen Agenten) aus und stellt dieser mittelbar oder unmittelbar Ressourcen und Betriebsmittel zur Verfügung. Auf diese Weise entsteht ein hierarchisch geschichtetes System von Executees und Executors. Dabei kann eine Entität wie das Agentensystem auch gleichzeitig Executor und Executee sein; Executee der Managed Resource bzw. deren Betriebssystem und Executor für Mobile Agenten. Der Executor verbirgt die Schicht, auf der er selbst ausgeführt wird, gegenüber seinem Executee.

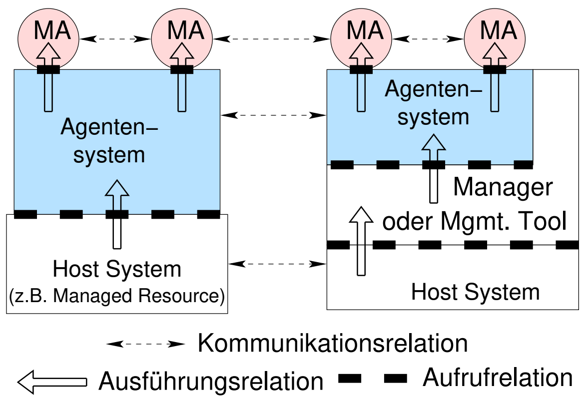


Abbildung 2.7: Relationen zwischen Entitäten

Aufrufrelation: lokaler Informationsfluss an Aufrufschnittstellen

An der Schnittstelle zwischen diesen beiden Entitäten gibt es Aufrufschnittstellen, die der Executor dem Executee und umgekehrt zur Verfügung stellt. Die Relation, die durch den Informationsaustausch an dieser Aufrufschnittstelle definiert ist, wird als **Aufrufrelation** (engl. Calling Relation) bezeichnet. In dieser allgemeinen Definition handelt es sich bei der Aufrufrelation um eine symmetrische Relation, da keiner der Partner ausgezeichnet ist und auch keine „Richtung“ für den Informationsfluss definiert. Die Definition der Aufrufrelation kann aber auch enger gefasst werden. Analog zum Executor und Executee gibt es eine Entität, die einen Dienst und damit einen **Dienstzugangspunkt** (**Service Access Point, SAP**) an der Schnittstelle zur Verfügung stellt (**Diensterbringer**), und eine Entität, die diese Schnittstelle nutzt, d.h. aufruft (**Dienstnutzer**). Aus der allgemeinen Aufrufrelation zwischen zwei Entitäten, z.B. Mobilem Agenten und Agentensystem, werden zwei gerichtete Aufrufrelationen. Diese Unterteilung ist sinnvoll, weil sie äquivalent zur Subjekt–Objekt Relation ist. Ein Subjekt benutzt zur Beauftragung des Objektes eine Schnittstelle, die das Objekt zur Verfügung stellt, d.h. das Subjekt ist der Dienstnutzer, das Objekt der Diensterbringer. Im Folgenden wird der Begriff der Aufrufrelation im symmetrischen Sinne verwendet, d.h. das Vorhandensein einer Aufrufrelation besagt, dass eine Aufrufschnittstelle zwischen zwei Entitäten existiert. Falls in diesem Zusammenhang verdeutlicht werden soll, dass es sich um eine gerichtete Relation handelt, werden die Begriffe Subjekt und Objekt oder die Begriffe Diensterbringer und Dienstnutzer verwendet.

Bei den beiden genannten Relationen findet ein Informationsfluss nur lokal auf einem Rechensystem statt. Für ein verteiltes Managementsystem und, um Kooperation auch zwischen entfernten Agenten zu ermöglichen, ist eine weitere Relation notwendig. Die **Kommunikationsrelation** (engl. Communication Relation) ist definiert durch den Transport von Nachrichten zwischen entfernten Entitäten, die in verschiedenen Umgebungen ausgeführt werden und zwischen denen eine direkte Interaktion, z.B. über eine Aufrufschnittstelle, nicht möglich ist. Die Entitäten, die an einer Kommunikationsrelation teilnehmen, können deshalb auch keine lokalen Betriebsmittel der anderen Seite direkt nutzen. Die einzigen Aktionen in einer Kommunikationsrelation sind das Senden bzw. das Empfangen von Nachrichten. Auch die Migration eines Agenten von einem Quell–Agentensystem auf ein Ziel–Agentensystem ist ein Spezialfall der Kommunikationsrelation. In diesem Fall enthält die Nachricht bzw. die Nachrichten, die zwischen den Entitäten ausgetauscht werden, den gesamten Agenten. Der Empfänger ist in der Lage, den Nachrichtenaustausch vollständig zu kontrollieren, da er entscheiden kann, ob er eine gesendete Nachricht annimmt oder verwirft.

Kommunikationsrelation: Nachrichtentransport

Der Mobile Agent nimmt bezüglich der Kommunikationsrelation zu anderen Mobilen Agenten eine Sonderstellung ein. Für die Kommunikation zwischen zwei Mobilen Agenten gibt es zwei Möglichkeiten. Neben der entfernten Kommunikation über Nachrichten können zwei Agenten, die sich auf demselben Agentensystem befinden, auch mittels Aufrufrelation (d.h. über direkte Methodenaufrufe) lokal kommunizieren.

Sonderstellung von MAs auf gleichem AS

Werden die in Abbildung 2.7 dargestellten Relationen unter dem Gesichtspunkt der Schnittbildung betrachtet, so lässt sich feststellen, dass die Aufrufrelation einen Dienstschnitt und die Kommunikationsrelation einen Protokollschnitt repräsentieren. Auf jeder Schicht des Systems können die Entitäten mittels eines geeigneten Protokolls kommunizieren. Für den tatsächlichen Nachrichtenaustausch müssen die Entitäten der Schicht N die Dienstprimitive der Schicht $N - 1$ an deren Dienstzugangspunkten nutzen.

2.5 Lebenszyklus von Mobilen Agenten

Der dynamische Aspekt der Mobilität ist der entscheidende Punkt, der ein System Mobiler Agenten von einem Client/Server–System bzw. einem statischen verteilten System unterscheidet. Mobile Agenten sind in der Lage sich in einem Netzwerk zu bewegen (vgl. Abschnitt 2.1.3). Um die Besonderheiten, die sich durch die Fähigkeit zur Migration ergeben, untersuchen und geeignet modellieren zu können, wird der Lebenszyklus eines Mobilen Agenten näher betrachtet. Abbildung 2.8 stellt ein vereinfachtes Zustandsübergangsdiagramm eines Mobilen Agenten dar. Der Mobile Agent kann in seinem Lebenszyklus, genauer zwischen zwei Zuständen seines Lebenszyklus, seine Ausführungsumgebung wechseln (migrieren). Dies unterscheidet seinen Lebenszyklus von dem anderer Software–Bausteine in verteilten Systemen.

Zustände und Zustandsübergänge eines MA

Nachdem der Mobile Agent erzeugt (mit der Operation `init`) und gestartet (`start`) wurde, befindet er sich in Ausführung, d.h. im Zustand *running*. Ein Mobiler Agent kann in seiner Ausführung unterbrochen bzw. angehalten werden und befindet sich danach im Zustand *suspended*. Dieser Zustand wird entweder durch Fortsetzung der Ausführung (`resume`) oder durch Migration verlassen.

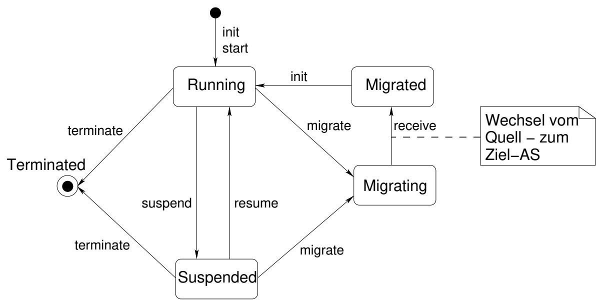


Abbildung 2.8: Zustandsübergangsdiagramm eines Mobilen Agenten

Die Migration eines Mobilen Agenten ist ein zweistufiger Prozess, an dem zwei Agentensysteme beteiligt sind. Ein Mobiler Agent kann sich selbst migrieren oder auch von außen zur Migration veranlasst werden. Die eigentliche Migration, d.h. das Erzeugen und Versenden einer Nachricht, die den Agenten enthält, wird vom Quell–AS ausgeführt. Damit der Agent migriert werden kann, muss das Agentensystem hierfür eine Schnittstelle zur Verfügung stellen. Sobald diese, im Folgenden mit `migrate` bezeichnete Operation aufgerufen wird, befindet sich der Mobile Agent im Zustand *migrating*. Nachdem das Quell–AS den Agenten als Nutzdaten einer Protokoll–Dateneinheit (PDE; engl. Protocol Data Unit, PDU) eingebettet hat, muss die entsprechende PDE auf das Ziel–AS übermittelt werden. Nachdem das Ziel–AS die Nachricht angenommen hat befindet sich der Mobile Agent im Zustand *migrated*. Das Ziel–AS muss den Agenten dann noch initialisieren und ihn wieder in den Zustand *running* versetzen. Ein Agent kann nur aus den Zuständen *running* und *suspended* heraus und nicht während der Migration terminiert werden.

Die interessanteste Fragestellung bei der Betrachtung eines Systems Mobiler Agenten ist die nach der Konsistenzerhaltung des Gesamtsystems. Um diese Frage beantworten zu können, muss sowohl der lokale als auch der globale Zustand eines Mobilen Agenten betrachtet werden. Die lokale Konsistenzerhaltung befasst sich damit, wie der momentane Ausführungszustand eines Mobilen Agenten erhalten und auf das Zielsystem übertragen werden kann (Art der Migration). Die Konsistenzerhaltung des globalen Zustandes wird mit der Semantik der Migration festgelegt.

Die verschiedenen Semantiken der Migration lassen sich durch genauere Betrachtung der Zustände *migrating* und *migrated* sowie möglicher Fehler– bzw. Problemfälle ableiten. Dazu sei angenommen, dass ein Mobiler Agent A von einem Quell–Agentensystem auf das Zielsystem migriert. Um die Migration durchführen zu können, muss eine (Migrations–) Kopie A' von A erzeugt werden. Fragestellungen, die in diesem Zusammenhang betrachtet werden müssen, umfassen:

- Ist es erlaubt, dass sich zu einem Zeitpunkt t der Agent A auf dem Quell–

AS und die Migrationskopie A' auf dem Ziel–AS befinden, d.h. darf es zum Zeitpunkt t zwei Instanzen von A geben (evtl. in verschiedenen Zuständen)?
- Wie kann das (ungewollte) Duplizieren von Agenten, z.B. durch Duplikation von Nachrichten, verhindert werden?
- Wie kann der Verlust eines Agenten bei der Migration verhindert oder zumindest erkannt werden?

Um diese Problemkreise zu behandeln, werden drei Migrationssemantiken unterschieden:

Globale Konsistenz des Systems: Semantik der Migration

1. **At least once**: Der zu migrierende Agent läuft auf dem Quell–AS weiter oder wird dort suspendiert, aber nicht terminiert und gelöscht. Das Quell–AS erzeugt eine Migrationskopie A', versetzt diese in den Zustand *migrating* und überträgt A' auf das Zielsystem. Wenn A' dort (erfolgreich) gestartet werden konnte, wird der Mobile Agent A auf dem Quell–AS terminiert und gelöscht. Ein Fehlerfall kann dazu führen, dass sowohl A als auch A' unabhängig voneinander weiterlaufen.

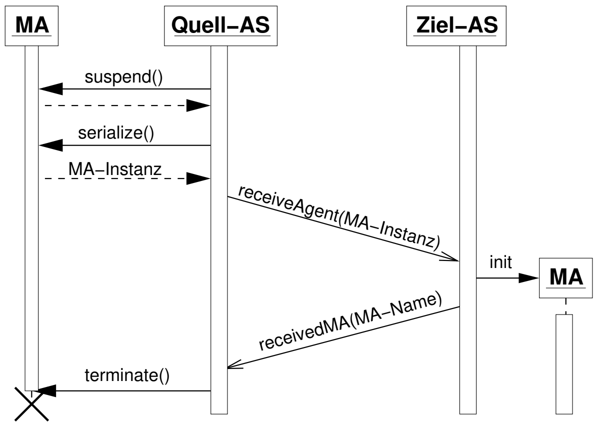


Abbildung 2.9: Migration: At least once

2. **At most once**: Der zu migrierende Agent wird im Quell–AS in den Zustand migrating versetzt, d.h. es wird eine PDE mit einer Migrationskopie A' erzeugt. Der Agent A wird vor dem Versenden der Nachricht terminiert und gelöscht. In diesem Fall führt ein Fehler dazu, dass weder A noch A' weiterlaufen, d.h. der Agent A „stirbt“.

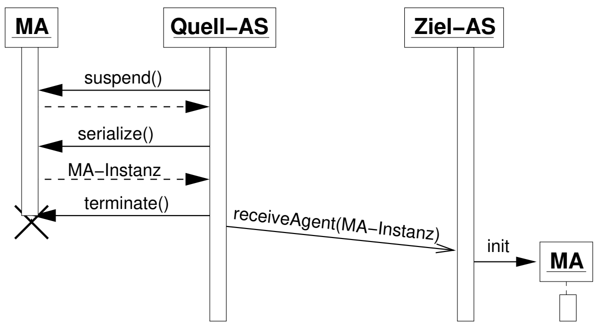


Abbildung 2.10: Migration: At most once

3. **Exactly once:** Es wird ein strenges Transaktionskonzept für die Migration eingeführt, das Teile der aus dem Datenbankbereich bekannten ACID–Prinzipien erfüllt. Damit wird die Migration zu einer atomaren Operation (*Atomicity*) und durch Rollback–Mechanismen bleibt der „globale“ Zustand der beteiligten Agentensysteme und damit auch der entsprechenden Managed Resources gewahrt (*Consistency*). Die Forderung nach logischem Einbenutzerbetrieb (*Isolation*) und nach *Durability*,

d.h. der Dauerhaftigkeit im Sinne von Datenbanken, muss nicht notwendigerweise erfüllt werden. Bei der exactly once Semantik werden Fehlerfälle durch geeignete Mechanismen abgefangen; es gibt zu keinem Zeitpunkt mehr als einen Agenten A.

lokale Konsistenz des MA: Arten der Migration

Im Lebenszyklus eines Mobilen Agenten werden neben den Zuständen und der Semantik der Migration auch zwei Arten der Migration selbst unterschieden: die transparente sowie die schwache Migration. Bei der Migration eines Agenten muss dessen (Ausführungs–) Zustand sowie die Belegung seiner internen Variablen gesichert und auf das Zielsystem übertragen werden.

schwache Migration

Die **schwache Migration** zeichnet sich dadurch aus, dass der Agent nur zu bestimmten Zeitpunkten oder an bestimmten Stellen seines Instruktionspfades migrieren kann. Der Mobile Agent muss sich gegebenenfalls erst in einen Zustand versetzen, in dem er migrieren kann. Auf dem Zielsystem angekommen wird der Agent initialisiert. Dabei wird die auszuführende Operation festgelegt, mit der der Agent auf dem Zielsystem startet.

starke oder transparente Migration

Die **starke** oder **transparente Migration** soll möglichst exakt die idealisierte Vorstellung von Migration implementieren; der Mobile Agent kann an jeder Stelle seines Instruktionspfades angehalten und auch migriert werden. Auf dem Zielsystem wird der Mobile Agent genau im selben Zustand und an der exakt gleichen Stelle innerhalb seines Instruktionspfades fortgesetzt. Um starke Migration möglich zu machen und überhaupt implementieren zu können, muss der Aufruf–Stack des Agenten mit dem Agenten migriert werden können [Funf 98]. Auf dem Zielsystem muss dieser Aufruf–Stack wieder aufgebaut werden, damit der Agent an genau derselben Stelle weiterarbeiten kann. Diese Vorgehensweise kann relativ kompliziert, aufwendig und auch teuer werden. Für alle Konzepte zur starken Migration lassen sich Anwendungsfälle finden, bei denen die vollständig transparente Migration nicht mehr funktionieren kann. Als Beispiele hierfür seien nur lokale Referenzen des Mobilen Agenten, wie z.B. geöffnete Dateien oder Netzwerkverbindungen genannt. Ein Mobiler Agent, der Referenzen auf lokale Systemressourcen hält, kann nicht mehr transparent migriert werden, da diese Referenzen auf dem Zielsystem keine Gültigkeit mehr haben.

2.6 Sicherheitsanforderungen an das Managementsystem basierend auf Mobilen Agenten

Das Ziel dieser Arbeit ist eine Sicherheitsarchitektur für auf Mobilen Agenten basierende Managementsysteme. Um die Sicherheitsanforderungen an ein derartiges System zu ermitteln, wird ausgehend von einer strategischen Sicherheitspolicy eine Risikoanalyse der verschiedenen, in diesem Kapitel vorgestellten Modelle vor dem Hintergrund der beschriebenen Szenarien durchzuführen sein. Dabei sind die Besonderheiten, die sich durch Mobile Agenten und insbesondere durch deren Lebenszyklus ergeben, zu berücksichtigen.

Als Basis für diese Untersuchung soll die Sicherheitsarchitektur für OSI verwendet werden. Obwohl diese Sicherheitsarchitektur für Datenkommunikation und nicht für verteilte Anwendungen spezifiziert wurde, können die darin definierten Begriffe und Konzepte sinnvoll für die Anforderungsanalyse verwendet werden.

2.6.1 Allgemeine Vorgehensweise bei der retrospektiven Risikoanalyse

Das Ziel der Risikoanalyse ist die Ermittlung von Sicherheitsanforderungen, die das zu untersuchende und zu bewertende IT–System erfüllen muss. Dabei müssen die zum Teil konkurrierenden Ziele des effektiven Schutzes des IT–Systems und des effizienten Mitteleinsatzes (in Form von Geld, Personal und Zeit) in Einklang gebracht werden. Alle Risiken sind daher möglichst vollständig zu erfassen und nach den möglichen Schäden, die bei Eintritt einer Gefahr entstehen können, zu priorisieren. Diese Vorgehensweise spiegelt sich auch in der Definition des quantitativen Risikos wieder.

Definition Risiko

$$R(X) = S(X) * E(X)$$

Danach ist das Risiko das Produkt aus der Schadenshöhe S, die bei Eintritt einer bestimmten Gefährdung X entsteht, und dem Erwartungswert, mit dem diese Gefährdung eintritt. Dadurch wird das Risiko operationalisier– und priorisierbar.

Um zu dieser Risikobewertung zu kommen, ist die in Abbildung 2.11 dargestellte Vorgehensweise bei der Risikoanalyse üblich [Raep 98]. Zuerst werden in einer **Bestandsaufnahme** alle Schutzobjekte wie z.B. Rechensysteme, Dienste, Datenbestände usw. ermittelt. In der **Bedrohungsanalyse** wird für jedes Objekt aus dem Schritt 1 untersucht und dokumentiert, welchen potentiellen Bedrohungen und Gefahren das Objekt ausgesetzt ist. Daran anschließend werden die Eintrittswahrscheinlichkeiten, genauer die **Erwartungswerte** für den Eintritt eines Schadens und die entsprechenden **Schadenshöhen** ermittelt, um das quantitative Risiko berechnen und die **Risiken priorisieren** zu können. Die schwierigsten Schritte dabei sind die Ermittlung der Eintrittswahrscheinlichkeiten und der Schadenshöhen.

Vorgehensweise bei der Risikoanalyse

Aus den priorisierten Risiken werden dann Sicherheitsanforderungen abgeleitet. Die Realisierung, bzw. die Durchsetzung der Sicherheitsanforderungen erfolgt durch Sicherheitsdienste, die durch konkrete Sicherheitsmechanismen [Pain 99] implementiert werden. Das Sicherheitskonzept fasst die Risikoanalyse und insbesondere die einzelnen Sicherheitsanforderungen mit den entsprechenden Sicherheitsdiensten zusammen.

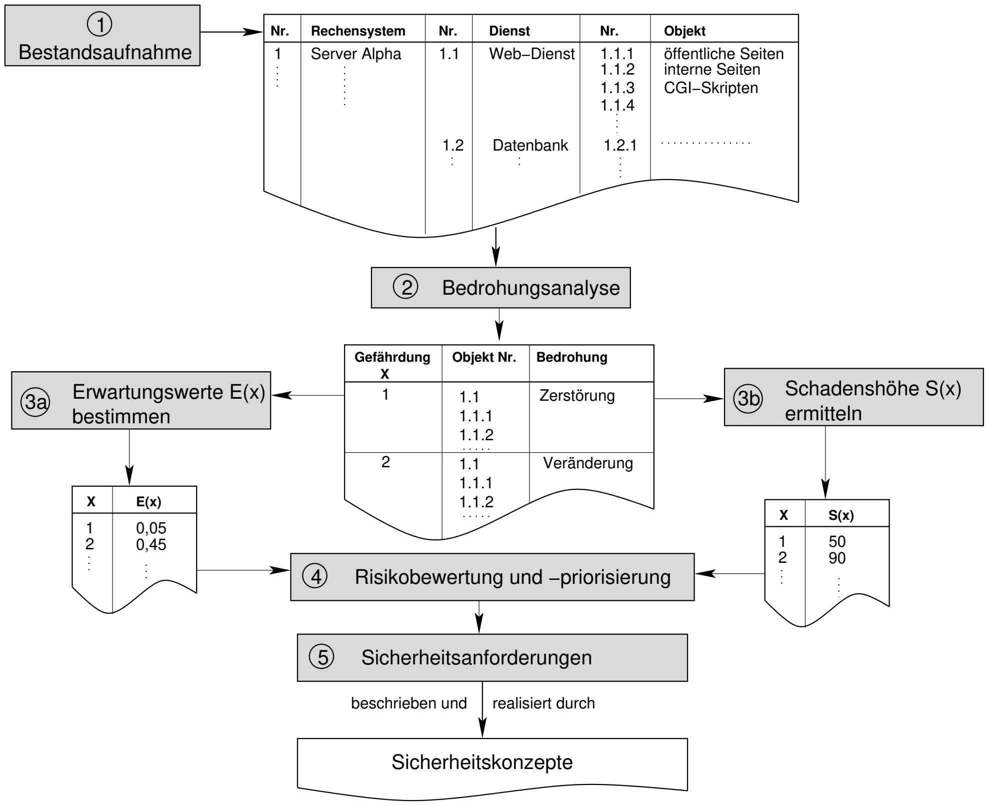


Abbildung 2.11: Risikoanalyse nach [Raep 98]

Nachteil: retrospektive Betrachtung

Bei dieser Art der Risikoanalyse wird ein bestehendes und i.d.R. im Betrieb befindliches IT–System untersucht. Die größten Nachteile dieser Vorgehensweise bestehen darin, dass die Risikoanalyse zum großen Teil retrospektiv erfolgt und sehr große Kenntnisse über das zu bewertende System, Wahrscheinlichkeiten und Schadenshöhen bekannt sein müssen. Die Risikoanalyse ist in diesem Fall immer auf eine konkrete Ausprägung eines IT–Systems zugeschnitten. Retrospektive Ansätze erfordern eine kontinuierliche Anpassung an neu auftretende Angriffe und Gefahren.

Nachteil: erschwerte Wartung und Anpassung

Auch die hierbei entstehenden Sicherheitskonzepte zeichnen sich durch die im Vordergrund stehenden, technischen Gesichtspunkte konkreter Sicherheitsmechanismen aus. Durch diese stark technische Betrachtungsweise wird eine Erweiterung, Wartung oder Anpassung an andere Umgebungen oder neue Anforderungen sehr schwierig. Oftmals wird auch für jedes erkannte Risiko ein eigener Sicherheitsmechanismus entwickelt und implementiert, anstatt abstraktere Sicherheitsanforderungen und Sicherheitsdienste aufzustellen, die für ganze Klassen von Risiken verwendet werden können.

2.6.2 Prospektive Risikoanalyse: Bestandsaufnahme und Bedrohungsanalyse

In dieser Arbeit soll eine möglichst prospektive Sicherheitsarchitektur entwickelt werden, die explizit nicht die konkrete Instanz eines Systems Mobiler Agenten betrachtet, sondern die ganze Klasse von Managementsystemen, die Mobile Agenten in den verschiedensten konkreten Instantiierungen und Ausprägungen, verwenden. Deshalb kann die im vorhergehenden Abschnitt angegebene Vorgehensweise nicht unverändert übernommen und eingesetzt werden.

Risikoanalyse anhand eines abstrakten Systemmodells

Die Bestandsaufnahme und die Bedrohungsanalyse werden nicht rückwirkend und an Hand einer konkreten Instantiierung eines IT–Systems durchgeführt, sondern es wird ein abstraktes Systemmodell zugrundegelegt. Dieses Systemmodell ist das Ergebnis der Bestandsaufnahme, die als erster Schritt der Risikoanalyse in Abschnitt 2.4 bereits durchgeführt wurde. Dort wurden die Entitäten und Relationen als Objekte im Sinne der Bestandsaufnahme identifiziert. Im Folgenden wird nun eine Risikoanalyse auf der Basis des Systemmodells aus dem Abschnitt 2.4 und dem Lebenszyklus Mobiler Agenten durchzuführen sein. Dazu müssen alle Komponenten des Modells (Entitäten und Relationen) auf mögliche Angriffspunkte hin untersucht werden. Aufbauend auf diese Analyse sind Sicherheitsanforderungen abzuleiten, ohne jedoch konkrete Risiken zu quantifizieren und zu priorisieren. In der Realisierung der Sicherheitsarchitektur (vgl. Abschnitte 4 und 5) werden Sicherheitsdienste, Mechanismen und Bausteine spezifiziert, die Sicherheitsanforderungen erfüllen können. Erst wenn ein konkretes System Mobiler Agenten aufgebaut und instantiiert wird, muss der Administrator, unter Berücksichtigung der lokalen Gegebenheiten und Randbedingungen des Einsatzszenarios, eine quantitative Risikoanalyse durchführen, die ermittelten Risiken priorisieren und daraufhin festlegen, welche Sicherheitsdienste, Mechanismen und Bausteine der Sicherheitsarchitektur in welchen Ausprägungen einzusetzen sind, um für sein Einsatzgebiet die optimale Kombination aus effektivem Schutz und effizientem Mitteleinsatz zu finden.

Klassifikation von Angriffen nach [Ecke 98, Stal 98]

Verschiedenste Angriffe und Bedrohungen können die Sicherheit eines Managementsystems gefährden. In der Literatur gibt es unterschiedliche Ansätze, um Bedrohungen und Angriffe zu definieren und einzuteilen. Unter einem **Angriff** (engl. attack) wird bei [Ecke 98] ein nicht autorisierter Zugriff auf ein System verstanden. Grundsätzlich lassen sich Angriffe in aktive und passive Angriffe einteilen [Stal 98]. Falls ein Angreifer die Möglichkeit besitzt Daten oder Bestandteile des Managementsystems zu verändern, wird von einem **aktiven Angriff** gesprochen. Im Gegensatz dazu hat der Angreifer bei einem **passiven Angriff** lediglich die Möglichkeit, Informationen zu erlangen, ohne das System aktiv beeinflussen zu können.

Einheitliche Taxonomie des CERT für sicherheitsrelevante Vorfälle

Für Angriffe auf Endsysteme hat das Computer Emergency Response Team [CERT] versucht eine einheitliche Taxonomie für sicherheitsrelevante Vorfälle zu entwickeln [HoLo 98]. Dabei wird streng zwischen Angriff und Event unterschieden. Ein Event ist eine Zustandsänderung, verursacht durch eine Aktion, die gegen ein bestimmtes Ziel gerichtet ist. Ein Event oder eine Menge von Events ist nur ein kleiner Teil eines Angriffs, der Informationen über verwendete Werkzeuge, die ausgenutzten Schwachstellen sowie das (unautorisierte) Resultat beinhalten muss. Wird zusätzlich noch der Angreifer selbst und dessen Motivation betrachtet, spricht das CERT von einem Vorfall (engl. incident; vgl. Abbildung 2.12). Da die Taxonomie des CERT auch

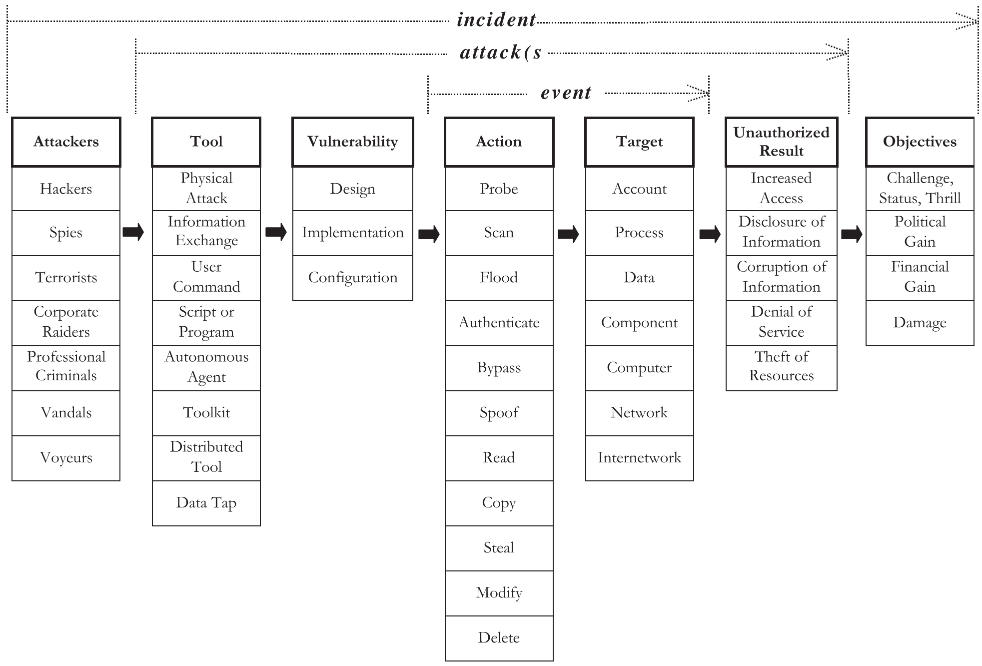


Abbildung 2.12: Angriffsklassifikation des CERT [HoLo 98]

nur retrospektiver Art ist und nur Endsysteme, aber keine verteilten Anwendungen oder gar Mobile Agenten betrachtet werden, kann diese Klassifikation nicht uneingeschränkt übernommen werden. Auch die Definition des Angriffs erscheint zu weit und zu technisch gefasst. Informationen über Werkzeuge, die für einen Angriff verwendet wurden, sollten nicht in die Definition eines Angriffs mit eingehen. Als Definition von Angriff wird deshalb die von [Ecke 98] vorgeschlagene verwendet. Wichtig erscheint hingegen das Konzept des Events. Da der Begriff Event im Management jedoch anders belegt ist, wird er nicht im Sinne des CERT verwendet, sondern es wird das dahinter liegende Konzept der Aktion, die gegen ein bestimmtes Ziel gerichtet ist, angewandt.

Information als Ziel jeden Angriffs

Diese potentiellen Angriffsziele sind bei der Bedrohungsanalyse von entscheidender Bedeutung. Das Ergebnis der Bedrohungsanalyse muss es sein, für alle (potentiellen) Angriffsziele alle möglichen Angriffsarten zu ermitteln. Als Grundannahme kann man davon ausgehen, dass ein Angreifer entweder unberechtigterweise an Informationen gelangen (passiver Angriff) oder in den Ablauf des Systems eingreifen bzw. das System manipulieren (aktiver Angriff) möchte. In beiden Fällen ist Information, im weiteren Sinn, Ziel des Angriffs. Um einen Angriff durchzuführen, bieten sich dem Angreifer in einem Managementsystem zwei Klassen von möglichen Zielen. Er kann entweder versuchen, eine Entität des Managementsystems (als Träger von Information bzw. als Akteur im System) oder aber eine Relation zwischen Entitäten anzugreifen. Als **Angreifer** wird jedes Subjekt bezeichnet, das versucht, unberechtigt auf das System zuzugreifen. Alle anderen Begriffe für Subjekte innerhalb und außerhalb des Managementsystems (z.B. Manager, MA, Nutzer, usw.) bezeichnen berechtigte Subjekte.

Angriff auf Entitäten

Angriffe, die sich gegen die Entitäten des in Abschnitt 2.4.1 eingeführten Entitätenmodells richten, werden als **Entitäten–Angriffe** (engl. Entity Attack) bezeichnet. Jede dieser Entitäten kann sowohl in der Subjekt– als auch in der Objektrolle auftreten. Eine Entität in der Subjektrolle tritt als Akteur im System auf. Ein Subjekt beauftragt Objekte und bestimmt damit aktiv den weiteren Ablauf im Managementsystem. Ein Angreifer kann nun versuchen als Subjekt im System aufzutreten, in dem er sich als berechtigte Entität ausgibt. Diese Angriffsart wird als **Maskerade** (engl. Masquerade) bezeichnet.

Daneben kann der Angreifer auch versuchen, die Entität als Informationsträger (d.h. in seiner Objektrolle) direkt anzugreifen, oder auch versuchen, die Entität in der Ausführung ihrer Aufgaben zu behindern. Dazu muss der Angreifer aber eine Relation zum Angriffsziel etablieren oder eine bestehende Relation attackieren. Diese Art von Angriffen wird im folgenden Abschnitt näher behandelt. Die einzige Möglichkeit, eine Entität direkt anzugreifen, bleibt damit der Versuch die Identität der Entität zu übernehmen.

Angriff auf Relationen

Eine Relation zwischen Entitäten liegt dann vor, wenn zwischen den Entitäten ein Informationsfluss stattfindet. Ein Angreifer kann versuchen, die Informationen, die über eine bestehende Relation ausgetauscht werden, zu nutzen oder zu manipulieren. Daneben kann er aber auch versuchen, aktiv eine Relation, zum Zwecke des Angriffs, zu einer Entität aufzubauen.

<table>
<tr><th>Entitätenangriffe</th><th colspan="3">Relationenangriffe</th></tr>
<tr><td rowspan="3">Maskerade</td><td colspan="3">Abhören
Rechtediebstahl
Leugnung
Vervielfältigung
Veränderung
Denial–of–service
Rechtemissbrauch
Man–in–the–middle</td></tr>
<tr><th>Aufrufrelation</th><th>Kommunikations–relation</th><th>Ausführungsrelation</th></tr>
<tr><td>Umgehung der Schnittstellen</td><td>Wiedereinspielung
Umleitung</td><td>Manipulation des Ausführungspfads
Denial–of–execution</td></tr>
</table>

Abbildung 2.13: Klassifikation von Angriffen

In Abschnitt 2.4.2 wird das Relationenmodell weiter verfeinert in Kommunikations–, Ausführungs– und Aufrufrelation. Auch die Angriffe werden analog in Aufrufrelationen–, Ausführungsrelationen– und Kommunikationsrelationen–Angriffe eingeteilt. Daneben existiert auch eine Menge von Angriffen, die gegen alle drei Klassen von Relation möglich ist. Diese Angriffe werden entsprechend als **Relationen–Angriffe** (engl. Relation Attack) bezeichnet. Die Klassifizierung, die im Folgenden näher erläutert wird, ist in Abbildung 2.13 zusammengefasst. Zuerst wird die allgemeine Klasse von Relationen–Angriffen vorgestellt.

Relationen–Angriffe

Ein klassischer passiver Angriff ist das **Abhören** (engl. Eavesdropping) von Informationen. Der Angreifer versucht dabei, den Informationsfluss zwischen zwei Entitäten mitzulesen. Dieser Angriff kann auf jede Art von Informationsfluss gerichtet sein. Werden Rechte über eine Kommunikationsverbindung übertragen oder an einer Schnittstelle übergeben, kann der Angreifer versuchen, diese Rechte für sich selbst zu nutzen. Daneben kann er auch versuchen, eine Relation zu einer Entität aufzubauen, um einen **Rechtediebstahl** (engl. Theft of Rights) durchzuführen.

Falls eine Entität eine Aktion durchführt, dies aber später leugnet, so muss diese **Leugnung** (engl. Repudiation) als Angriff gewertet werden. Falls beispielsweise die Benutzung eines Dienstes mit Kosten verbunden ist, die der Dienstnutzer zu tragen hat und dieser gegenüber dem Diensterbringer die tatsächlich erfolgte Nutzung des Dienstes bestreiten kann, liegt ein erfolgreicher Angriff vor. Dem Diensterbringer kann dadurch erheblicher Schaden entstehen.

Die **Vervielfältigung** (engl. Replication) und die **Veränderung** (engl. Alteration) von Information sind ebenfalls Angriffe, die grundsätzlich auf jeder Art von Relation ausgeführt werden können. Als Beispiele seien hier nur die Vervielfältigung eines Mobilen Agenten während seiner Migration (Kommunikationsrelation), bzw. die Veränderung der von ihm transportierten Daten durch ein feindliches Agentensystem (Ausführungsrelation) genannt.

Wird ein Subjekt an der Ausführung eines Dienstes gehindert oder gelingt es einem Angreifer den QoS des Dienstes zu verschlechtern, so wird dieser Angriff als **Denial of service** (**DoS**) bezeichnet. Um sein Ziel zu erreichen, kann der Angreifer für jede Relationenklasse einen Angriffsversuch unternehmen. Er kann versuchen die direkte Interaktion zwischen den Entitäten (Ausführungsrelation) und die Aufrufrelation zu behindern, indem er bspw. eine der beiden Entitäten massiv mit Anfragen „überflutet“.

Durch nicht sachgemäße Nutzung seiner Rechte bzw. durch **Rechtemissbrauch** (engl. Resource misuse) wird aus einem berechtigten Subjekt ein Angreifer. Auch die missbräuchliche Nutzung von Ressourcen bzw. Komponenten wird in dieser Angriffsklasse eingeordnet. Als Beispiel sei hier nur die Überschreitung von Kontingenten wie z.B. Rechenzeit– und Speicherplatzkontingente, usw. aufgeführt.

Bei einem **Man in the middle** Angriff steht der Angreifer zwischen den Endpunkten der Relation und kann alle ausgetauschten Informationen mitlesen, verändern und sogar eigene Informationen einfügen, ohne dass die beiden an der Relation beteiligten Partner dies erkennen können. In Abbildung 2.14 wird ein Man in the middle Angriff auf die Kom-

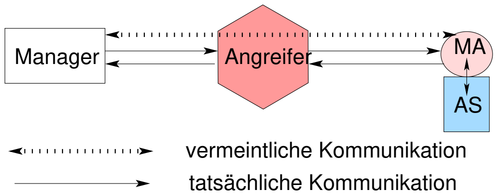


Abbildung 2.14: Man in the middle Angriff (Bsp.)

munikationsrelation zwischen einem Manager und einem Mobilen Agenten dargestellt. Der Angreifer ist in der Lage die Kommunikationsrelation zu unterbrechen und sich gegenüber dem Mobilen Agenten als Manager und gegenüber dem Manager als Mobiler Agent auszugeben. Damit hat er volle Kontrolle über den Informationsfluss erlangt.

Aufrufrelationen–Angriffe

Neben den allgemeinen Relationen–Angriffen gibt es auch einen speziellen Angriff gegen die Aufrufrelation (engl. Calling Relation Attack). Die Aufrufrelation ist durch wohldefinierte Dienstzugangspunkte charakterisiert. Ein Angreifer kann versuchen, durch **Umgehung der Schnittstellen** (engl. Circumvention Attack) direkten und unkontrollierten Zugang auf Ressourcen des Diensterbringers zu erhalten.

Ein Spezialfall des Man in the middle Angriffs bezüglich der Aufrufrelation ist ein böswilliger Interceptor (Abfänger, Abfangjäger). Ein Interceptor ist ein Software–Baustein, ein Objekt oder eine Methode, die zwischen, bzw. vor die eigentliche Aufrufschnittstelle geschaltet wird. Der Interceptor wird bspw. im CORBA Security Service [OMG 2001-03-08] verwendet, um Sicherheitsüberprüfungen auf Ebene des ORB, d.h. unabhängig von den aufgerufenen Objekten durchführen zu können. Bevor der Methodenaufruf vom ORB an das eigentliche Objekt weitergegeben wird, werden im Interceptor diverse Sicherheitsüberprüfungen durchgeführt. Im Erfolgsfall bleibt diese Aufrufumlenkung für den Aufrufer transparent. Ein ähnliches Vorgehen existiert auch innerhalb der Java–Sicherheitsarchitektur [Gong 98] in Form von so genannten `check`–Methoden, die vor der Ausführung der eigentlich aufgerufenen Methode durchlaufen werden. Falls ein Angreifer in der Lage ist, einen Interceptor für seine Zwecke zu etablieren, kann er die Aufrufe mitlesen, manipulieren und ggf. auch umlenken.

Kommunikationsrelations–Angriffe

Bei der **Wiedereinspielung** (engl. Replay) und der **Umleitung** (engl. Redirection) von Nachrichten handelt es sich um **Kommunikationsrelations–Angriffe** (engl. Communication relation attack). Nur wenn die Information als Nachricht mit einem adressierten Empfänger verschickt wird, sind diese Angriffe möglich. Ein Aufruf an einer Dienstschnittstelle oder die Ausführungsbeziehung zwischen zwei Entitäten kann weder umgeleitet noch wiedereingespielt werden.

Ausführungsrelations–Angriffe

Auch die Ausführungsrelation, die sich durch eine sehr enge Bindung zwischen Executor und Executee auszeichnet, bietet für den Executor spezielle Angriffsmöglichkeiten. Da der Executee unter der vollkommenen Kontrolle des Executors ausgeführt wird, kann dieser die Ausführung des Executee beeinflussen, indem er dessen Ausführungspfad manipuliert (**Manipulation des Ausführungspfades**, engl. Execution trace manipulation). Damit kann er den Ablauf des Systems aktiv beeinflussen und ein Agentensystem könnte bspw. durch die Manipulation des Ausführungspfades eines Mobilen Agenten dafür sorgen, dass dieser falsche Ergebnisse an seinen Manager zurückliefert.

Ein Executor hat aber auch die Möglichkeit die Ausführung eines Executee vollständig zu blockieren oder zu verhindern. Dieser Angriff wird als **Denial of execution** (**DoE**) bezeichnet.

2.6.3 Resultierende Sicherheitsanforderungen

Die Sicherheitsanforderungen, die ein Managementsystem erfüllen soll, werden aus der Risikoanalyse abgeleitet. Werden nur die Ergebnisse der Risikoanalyse als Quelle für die Sicherheitsanforderungen verwendet, besteht die Gefahr, dass für jeden potentiellen Angriff eine eigene Abwehrstrategie entwickelt, bzw. eine spezifische Sicherheitsanforderung definiert wird. Um die resultierende Sicherheitsarchitektur flexibel, wartbar und möglichst generisch zu halten, sollten jedoch allgemeinere Sicherheitsanforderungen ermittelt werden, die Schutz vor ganzen Klassen von Angriffen bieten können. Als Grundlage und als weitere Quelle zur Ermittlung der Sicherheitsanforderungen kann die OSI–Sicherheitsarchitektur verwendet werden.

OSI–Sicherheitsarchitektur

Die OSI–Sicherheitsarchitektur schützt die Kommunikation zwischen heterogenen Rechensystemen, nicht jedoch die Rechensysteme selbst. Obwohl die OSI–Sicherheitsarchitektur spezifiziert wurde, um Kommunikationssysteme und nicht verteilte Systeme oder gar Systeme Mobiler Agenten zu sichern, sind die angegebenen Konzepte so allgemeingültig, dass sie auch auf das dieser Arbeit zugrunde liegende Szenario angewendet werden können.

Sicherheitsdienste werden durch Sicherheitsmechanismen realisiert

Die OSI–Sicherheitsarchitektur [X.800, X.810, ISO 10181-1] ist sowohl als X– als auch als ISO–Standard veröffentlicht. Alle X.81y Standards werden bei der ISO unter ISO 10181-x geführt. Die OSI–Sicherheitsarchitektur erweitert das ISO–OSI Referenzmodell (OSI–RM) [ISO 7498] und beschreibt Sicherheitsdienste und –mechanismen. Dienste, die in der Lage sind, eine Sicherheitspolicy durchzusetzen bzw. zu erfüllen, werden als **Sicherheitsdienste** bezeichnet. Ein Sicherheitsdienst wird durch **Sicherheitsmechanismen** realisiert. Aus einfachen Basisdiensten lassen sich dabei auch komplexere Sicherheitsdienste kombinieren. Ein Sicherheitsdienst der OSI–Sicherheitsarchitektur ist immer an eine bestimmte Schicht des OSI–RM gebunden.

Folgende Sicherheitsdienste werden gefordert:

Authentisierung

- Die **Authentisierung (Authentication)** liefert die Gewissheit über die Identität einer Entität. Die Authentisierung ist nur im Kontext einer Relation zwischen einem so genannten **Principal**, d.h. der Entität, die authentisiert werden soll, und einem **Verifier**, der Entität, die die Authentisierung durchführt, sinnvoll [X.811].

Zugriffskontrolle

- Das primäre Ziel der **Zugriffskontrolle (Access Control)** ist die Verhinderung von nicht autorisierten Operationen auf Rechen– oder Kommunikationssystemen [X.812].

Verbindlichkeit

- Der Sicherheitsdienst, der **Verbindlichkeit (Non Repudiation)** gewährleistet, umfasst die Erzeugung, die Verifikation und die Speicherung von Belegen, sowie die spätere Wiederherstellung und wiederholte

Verifikation dieser Belege mit dem Ziel, Kontroversen über das Auftreten von Ereignissen oder Aktionen aufzulösen [X.812].Ein stattgefundenes Ereignis oder eine durchgeführte Aktion kann später nicht geleugnet werden. Beispiele für solche Aktionen sind das Verschicken von Nachrichten, der Aufruf einer entfernten oder lokalen Operation, u.ä.

- **Vertraulichkeit (Confidentiality)** bezeichnet die Eigenschaft von Information, nicht autorisierten Personen, Entitäten oder Prozessen nicht zugänglich zu sein und von diesen auch nicht enthüllt werden zu können [X.814]. Vertraulichkeit

- Der Dienst, der die **Integrität (Integrity)** von Daten und Attributen sichert, kann unautorisierte Veränderung, Löschung, Einfügung, Erzeugung und das Wiedereinspielen erkennen oder verhindern [X.815]. Integrität

- Daneben wird auch ein Dienst für **Sicherheitsaudits (Security Audit)** und **–alarme (Security Alarms)** spezifiziert. Ein Sicherheitsaudit ist eine unabhängige Revision und Prüfung von Systemdatensätzen und Aktivitäten. Der Sicherheitsauditdienst unterstützt einen **Auditverantwortlichen (Audit Authority)** durch die Möglichkeit, Ereignisse und Aktionen zu spezifizieren, auszuwählen und zu verwalten, die in einem **Sicherheitsaudit–Pfad (Security Audit Trail)** aufgezeichnet werden. Sicherheitsaudit und –alarme
  Ein Sicherheitsalarm ist eine Warnung an eine Person oder einen Prozess, die anzeigt, dass eine Situation eingetreten ist, die eine rechtzeitige Aktion erfordert. Der Zweck eines solchen Alarms umfasst u.a. die Meldung von Sicherheitsverletzungen, das Überschreiten von Schwellwerten oder andere sicherheitsrelevante Ereignisse [X.816].

Die Sicherheitsdienste nach OSI bilden eine Hierarchie (vgl. Abbildung 2.15). Die Authentisierung dient als Basisdienst für die Zugriffskontrolle, die Ver-

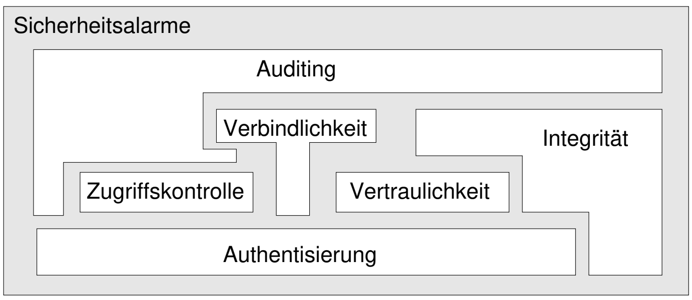


Abbildung 2.15: Hierarchie der OSI–Sicherheitsdienste

traulichkeit sowie die Verbindlichkeit. Ohne eine sichere und zweifelsfreie Authentisierung von Kommunikationspartner kann keine Zugriffskontrolle erfolgen. Rechte werden immer an Entitäten vergeben, die Ihre Identität über den Authentisierungsdienst belegen müssen. Analog muss auch für eine vertrauliche Kommunikation die Identität des Kommunikationspartners zweifelsfrei feststehen.

Verbindlichkeit lässt sich auf unterschiedliche Arten realisieren. Auf jeden Fall muss aber die Identität des Akteurs bestimmt werden. Diese kann dann mit der durchgeführten Aktion in eine (manipulations–) sicheren „Log–Datei" geschrieben werden. In diesem Fall verwendet der Verbindlichkeitsdienst den Authentisierungs– und den Auditingdienst als Basisdienste. Falls die Verbindlichkeit der Aktion durch eine digitale Signatur des Akteurs sichergestellt wird, dienen der Vertraulichkeits– und der Authentisierungsdienst als Basis. Entsprechend gibt es auch verschiedene Möglichkeiten, einen Integritätsdienst zu realisieren, der entweder völlig ohne Basisdienste auskommt oder sich auf den Authentisierungsdienst oder den Vertraulichkeits– bzw. den Verbindlichkeitsdienst abstützt.

Der Dienst, der Sicherheitsalarme erzeugt, ist eine Querschnittsfunktionalität, die bei Sicherheitsverletzungen in allen anderen Diensten zur Verfügung stehen muss. Gleiches gilt für den Auditing–Dienst, der kritische und sicherheitsrelevante Aktionen in einem Log speichert. Der Auditing–Dienst kann aber selbst wieder Basisdienst sein, z.B. für den Verbindlichkeitsdienst.

Sicherheitsanforderungen aus dem Einsatzszenario

Sicherheit über Organisationsgrenzen hinweg

Mobile Agenten eignen sich wegen ihrer Flexibilität, ihrer dynamischen Erweiterbarkeit und der die konkrete Hardware verschattenden Schicht des Agentensystems gut für das Management in Multiprovider Hierarchien. Für das Management dieser, in Abschnitt 2.2.1 beschriebenen, Szenarien ist ein sicheres Managementsystem auch über Organisationsgrenzen hinweg essentiell. Die Managementagenten der beteiligten Organisationen müssen die Organisationsgrenzen überschreiten können, damit bspw. ein Provider den QoS auf Kundenseite messen kann. In diesem Fall wird Software des Providers, in Form eines Mobilen Agenten, auf der Infrastruktur des Kunden ausgeführt [HaRe 00]. Der Kunde stellt dafür die Ausführungsumgebung (Agentensystem) zur Verfügung. Alle beteiligten Organisationen müssen in die Lage versetzt werden können, ihre eigene Infrastruktur zu schützen und ggf. die Rechte der Partnerorganisation einzuschränken. Der Provider will seinen Mobilen Agenten geschützt sehen und dessen Ergebnissen auch trauen können, der Kunde will seine Infrastruktur und seine Daten vor einem feindlichen Mobilen Agenten schützen. Er muss sichergehen können, dass der Mobile Agent nur die Aktionen durchführt und die Daten sehen kann, die er für seine konkrete Aufgabe benötigt.

Im Extremfall können der Besitzer der Hardware, der Benutzer und der Autor (Entwickler) des Agentensystems und der Benutzer sowie der Autor des Mobilen Agenten völlig verschiedene Personen oder Entitäten sein, für die verschiedene Sicherheitspolitiken gelten und die verschiedene Vertrauensbeziehungen untereinander haben [Tsch 99, Klus 99]. Die Sicherheitsarchitektur muss daher mit unterschiedlichen Verantwortungsbereichen umgehen können und Mechanismen bieten, diese auch abzubilden. Jeder Betreiber eines Agentensystems will seinen Sicherheitslevel entsprechend einer eigenen Risikoanalyse selbst festlegen und konfigurieren können. Die Sicherheitsarchitektur muss in der Lage sein, verschiedene Sicherheitsdomänen mit unterschiedlichen Stufen der Sicherheit realisieren zu können. Insbesondere bei den Rechten und bei der Zugriffskontrolle muss es für ein „Gastsystem“ möglich sein, die Rechte des Besuchers (MA) einzuschränken. Für Konfliktfälle muss eine klare Hierarchie von Rechten definiert werden.

verschiedene Sicherheitsdomänen

Rechtehierarchie

Das über mehrere organisatorische Domänen bzw. Domänen mit unterschiedlichen Sicherheitspolitiken verteilte Managementsystem kann niemals vollständig zentral kontrolliert werden. Es kann auch kein „globaler Zustand“ des Systems ermittelt werden. Die Sicherheitsarchitektur muss dementsprechend dezentral einsetzbar und konfigurierbar sein.

dezentral einsetzbar und konfigurierbar

Eine weitere Anforderung kommt aus dem Entwicklungsprozess sicherheitsrelevanter Software. In allen sicherheitsrelevanten Projekten sollen möglichst wenige und möglichst klare Konzepte eingesetzt werden. Es hat sich gezeigt, dass der Einsatz vieler unterschiedlicher und schwer durchschaubarer Verfahren sich wegen ihrer Komplexität, ihrer gegenseitiger Beeinflussung und Abhängigkeit oft selbst als Sicherheitsrisiko erweisen.

wenige und klare Sicherheitskonzepte

Sicherheitsanforderungen aus dem Entitäten- und Relationenmodell

Die Anforderungen, die in der OSI–Sicherheitsarchitektur gefordert werden, sind von einem Managementsystem ebenfalls zu erfüllen. Für ein System, das auf Mobilen Agenten basiert, ergeben sich jedoch zusätzliche Anforderungen. Die OSI–Sicherheitsarchitektur betrachtet als zu authentisierende Entitäten nur Kommunikationssysteme. Dabei wird unterschieden zwischen der Authentisierung des Datenursprungs (engl. data origin authentication) und der Authentisierung der Partnerinstanz (engl. peer entity authentication). Eine strenge Unterscheidung zwischen Identifikation und der eigentlichen Authentisierung erfolgt nicht; auch der Begriff der Entität ist nur unzureichend spezifiziert.

In einer Sicherheitsarchitektur für Systeme Mobiler Agenten ist diese Betrachtungsweise nicht ausreichend. Die Authentisierung muss auf der Basis der in Abschnitt 2.4.1 aufgeführten Entitäten erfolgen. Das bedeutet, dass auch reine Software–Komponenten, wie z.B. Mobile Agenten authentisiert werden müssen. Das primäre Ziel der Authentisierung ist es, für jede Entität mindestens einen Verantwortlichen bzw. den Berechtigten zu ermitteln. Im Folgenden wird zwischen Identifikation und Authentisierung unterschieden. **Identifikation** (engl. Identification) umfasst die Ermittlung eines für die Entität Verantwortlichen (Principal). Um Entitäten identifizieren zu können, ist einerseits ein eindeutiges Namenskonzept nötig. Andererseits muss es ein Konzept geben, um die Entität mit Identifikationsinformationen über den Principal zu verknüpfen. Unter der Authentisierung wird dann nur noch die Verifikation der Identitätsinformation verstanden.

Identifikation von Verantwortlichen

Da das Managementsystem organisatorische Grenzen überschreitet und Entitäten, für die verschiedene Organisationen verantwortlich sind, Teile des verteilten Managementsystems sind, muss jede dieser Entitäten sowohl als Principal als auch als Verifier auftreten können. Ein Agentensystem der Organisation A muss sich gegenüber einem Mobilen Agenten der Organisation B „ausweisen" und als Principal auftreten können, gleichzeitig aber auch als Verifier die Identität des Mobilen Agenten verifizieren.

Verfügbarkeit

Beschränkung der Ressourcennutzung

Auf einem Agentensystem können gleichzeitig Mobile Agenten von verschiedenen Organisationen ablaufen. Für das Agentensystem ist die **Verfügbarkeit** seiner Dienste wichtig. Ein Mobiler Agent sollte nicht alle Ressourcen des Agentensystems belegen können und damit die Dienstnutzung für andere Agenten unmöglich machen. Eng mit dieser Problematik hängt auch die Beschränkung der Ressourcennutzung (z.B. Rechenzeit, Quotas u.ä.) für einzelne Agenten zusammen.

Schutz der MAs voreinander

Sandboxing

Die Agenten müssen auch voreinander geschützt werden können. Ein Mobiler Agent darf nicht (unberechtigt) auf einen anderen Mobilen Agenten zugreifen. Auch ein indirekter Zugriff über Schnittstellen des Agentensystems muss verhindert werden. Es muss möglich sein, den Mobilen Agenten in einer wohldefinierten Ablaufumgebung „einzusperren", die er nur über dezidierte Schnittstellen des Agentensystems verlassen darf. Derartige Mechanismen werden auch als Sandboxing bezeichnet. Für einen Mobilen Agenten muss bei Bedarf eine solche **Sandbox** aufgebaut werden können.

Autorisierung und Delegation

Die Zugriffskontrolle nach OSI muss außerdem um ein Konzept für die Vergabe und Anpassung von Rechten erweitert werden. Diese Zuweisung von Rechten an Entitäten und deren Anpassung wird als **Autorisierung** (engl. Authorization) bezeichnet. Im Hinblick auf kooperatives Management muss, mit Hilfe des Autorisierungskonzeptes, eine Delegation von Rechten zwischen Entitäten möglich sein. Die Zugriffskontrolle nach OSI, die lediglich Rechen– und Kommunikationssysteme betrachtet, muss erweitert werden, um den Zugriff auf Entitäten kontrollieren zu können. Alle im System angebotenen Dienstschnittstellen müssen gesichert werden. Dazu kann es erforderlich sein, eine Entität in die Lage zu versetzen, den Zugriff auf die von ihr angebotenen Schnittstellen, auch in einer fremden Umgebung, beschränken zu können.

Integrität von MA und AS

OSI betrachtet die Integrität von Daten und Attributen bei Kommunikationsbeziehungen. Mobile Agenten und auch Agentensysteme als Software–Komponenten sind Entitäten, deren Integrität sichergestellt werden muss. Ein Agentensystem oder derjenige, der einen Mobilen Agenten beauftragt hat, muss erkennen können, ob der Mobile Agent auf seiner Reise verändert wurde. Andererseits sollte es auch möglich sein, die Integrität des Agentensystems verifizieren zu können.

Sicherheitsanforderungen aus dem Lebenszyklus

Die Besonderheit eines Mobilen Agenten ist dessen Fähigkeit, seine Ausführungsumgebung zu verlassen und auf ein anderes Agentensystem zu migrieren. Die Entscheidung zur Migration kann „aus eigenem Ermessen" des Mobilen Agenten erfolgen oder er kann von einer anderen Entität dazu aufgefordert werden. Auf jeden Fall muss der Verantwortliche für eine Migration ermittelt werden können. Die Verbindlichkeit der Migration muss sichergestellt werden und auch die Aktionen, die ein Mobiler Agent auf einem System ausführt, müssen einem Verantwortlichen zurechenbar sein.

Verbindlichkeit der Migration und der MA–Aktionen

Der Wechsel eines Mobilen Agenten von einem Agentensystem zu einem anderen kann in einer ersten Näherung als normaler Nachrichtenaustausch betrachtet werden, allerdings muss die Identifikationsinformation des Mobilen Agenten länger gespeichert werden, als dies bei einer Quelle–Senken–Kommunikation der Fall wäre. Für die Migration eines Mobilen Agenten müssen daher alle Sicherheitsanforderungen, die die OSI–Sicherheitsarchitektur für den Austausch von Nachrichten vorsieht, erfüllt werden. So muss bspw. die Integrität der Nachricht, die den Mobilen Agenten enthält, sichergestellt werden. Daneben muss aber auch die Integrität und die Vertraulichkeit von Daten, die der Mobile Agent (als Attribute) transportiert, gewährleistet werden. Nur der Empfänger von vertraulichen Daten darf diese auch lesen können und nicht etwa ein beliebiges Agentensystem auf dem Weg des Mobilen Agenten. Daneben müssen aber Attribute, die für die Verwaltung oder Ausführung des Mobilen Agenten notwendig sind, von allen Agentensystemen auch genutzt werden können. Bei einem Mobilen Agenten, der nach einer „Reise“ zurückkehrt, sollte es möglich sein zu erkennen, ob und ggf. auf welchem Agentensystem Daten, die der Mobile Agent transportiert, in unberechtigter Weise verändert wurden. Dazu ist es zwingend notwendig auf eine Historie des Mobilen Agenten zugreifen zu können.

Integrität und Vertraulichkeit der Daten des MA

Reisehistorie des MA

OSI	Entitäten– und Rollenmodell	Lebenszyklus	Einsatzszenario
Authentisierung	Namensschema Identifikation Authentisierung		Sicherheit über Organisationsgrenzen hinweg Verschiedene Sicherheitsdomänen Rechtehierarchie Dezentral einsetzbar und konfigurierbar Trust Level Konzept
Zugriffskontrolle	Autorisierung Zugriffskontrolle Delegation		
Verbindlichkeit		Verbindlichkeit der MA–Aktionen Verbindlichkeit der Migration	
Vertraulichkeit	Vertraulichkeit während der Ausführung	Vertraulichkeit während der Migration	
Integrität	Datenintegrität Entitätenintegrität Ausführungsintegrität		
	Sandboxing		
	Verfügbarkeit Ressourcen–beschränkung		
		Reisehistorie	

Abbildung 2.16: Sicherheitsanforderungen an ein Managementsystem basierend auf Mobilen Agenten

Zusammenfassung der Sicherheitsanforderungen

In den vorhergehenden Abschnitten wurden, von OSI ausgehend, Sicherheitsanforderungen an ein Managementsystem basierend auf Mobilen Agenten ermittelt. Abbildung 2.16 fasst diese Anforderungen noch einmal zusammen. Dabei wird auch verdeutlicht, welche neuen oder erweiterten Anforderungen durch die Modellbildung (s. Abschnitt 2.4), den Lebenszyklus (s. Abschnitt 2.5) und das Anwendungsszenario „organisationsübergreifendes IT–Management" hinzugekommen sind.

Werden in der Risikoanalyse nur konkrete Angriffe betrachtet und für die Ableitung der Sicherheitsanforderungen verwendet, besteht die Gefahr, dass die resultierenden Anforderungen immer jeweils auf einen speziellen Angriff ausgerichtet sind. Dies kann dazu führen, dass der Sicherheitsadministrator den Angreifern „hinterher läuft" und beim Bekanntwerden neuer Angriffe auch seine Sicherheitsanforderungen anpassen muss. Allgemeinere Sicherheitsanforderungen, die vor ganzen Klassen von Angriffen schützen, werden als geeignet betrachtet, um dieses Problem weitestgehend zu vermeiden. Nachdem das erklärte Ziel dieser Generalisierung die Aufhebung der strengen Eins zu Eins Abbildung zwischen Angriff und Sicherheitsanforderung ist, gibt es auch Angriffe, die nur durch eine Kombination von mehreren Sicherheitsdiensten verhindert werden können. Oder es gibt Sicherheitsdienste, die selbst wieder Basis für andere Sicherheitsdienste sind. Die Identifikation und Authentisierung ist beispielsweise eine Grundvoraussetzung für viele andere Sicherheitsanforderungen (vgl. auch Abbildung 2.15). Nur wenn die Entität identifiziert und authentisiert werden kann, können dieser Entität Rechte erteilt (Autorisierung) und diese Rechte auch durchgesetzt werden (Zugriffskontrolle). Tabelle 2.1 fasst die Sicherheitsanforderungen und die in Abschnitt 2.6.2 vorgestellten Angriffe nochmals zusammen.

Sicherheitsanforderung	**Schutz vor**
Identifizierung und Authentisierung	Maskerade, Rechtediebstahl, Leugnung, Vervielfältigung, Wiedereinspielung, Umleitung, DoE, DoS
Autorisierung and Zugriffskontrolle	Rechtediebstahl, Rechtemissbrauch, Umgehung der Schnittstellen, DoS
Vertraulichkeit	Abhören, Rechtediebstahl
Integrität	Veränderung, Vervielfältigung, Wiedereinspielen, Umleitung, Manipulation des Ausführungspfads
Verbindlichkeit	Leugnung
Ressourcenbeschränkung	DoS, Ressourcenmissbrauch
Sandboxing	Umgehung der Schnittstellen

Tabelle 2.1: Gegenüberstellung: Sicherheitsanforderungen, Angriffe

Kapitel 3

Sicherheit in Mobilen Agentensystemen: Status Quo

Inhaltsverzeichnis

Dieses Kapitel analysiert die aktuellen Forschungsaktivitäten die Sicherheit von Mobilen Agenten betreffend. Als Basis für die Untersuchungen zu allgemeinen Sicherheitsarchitekturen wird im folgenden Abschnitt der CORBA Security Service kurz vorgestellt. Diese Spezifikation stellt ein sehr umfangreiches und fortschrittliches Konzept dar, um Sicherheit auf Objektebene zu realisieren. In Abschnitt 3.2 werden relevante Forschungsarbeiten analysiert.

3.1 CORBA Security Service Specification

Dieser Arbeit liegt in großen Teilen die Begriffsbildung des MASIF Standards der OMG zugrunde (vgl. Abschnitt 2.3.1). Außerdem wird die in dieser Arbeit entwickelte Sicherheitsarchitektur prototypisch in einem System Mobiler Agenten implementiert, das sich auf CORBA abstützt (vgl. Abschnitt 6.1). Nachdem die OMG innerhalb ihrer Object Management Architecture (OMA) [Sole 95, OMG 00-06-41] auch einen Security Service spezifiziert hat [OMG 2001-03-08], wird dieser Querschnittsdienst hier näher betrachtet.

Objektbasierte Sicherheit

Der Security Service ist die Grundlage für den so genannten „Security Ready ORB". Dieser ORB muss die im „Conformance" Abschnitt der Spezifikation beschriebene Funktionalität implementieren. Ziel ist es, einen Schutz auf Ebene der Objekte zu realisieren, der Authentisierung, Zugriffskontrolle, Vertraulichkeit, Non–Repudiation und Auditing umfasst. Der Security Service soll möglichst flexibel sein, um alle Sicherheitspolitiken der verschiedenen Anwender zu unterstützen.

Verschiedene Security Level

Die Spezifikation unterscheidet zwei verschiedene Security Level:

- **Level 1:** Der erste Level bietet Sicherheitsdienste für Anwendungen, die selbst keine eigenen Sicherheitsanforderungen bzw. keinen eigenen Sicherheitsbegriff haben („unaware of security") oder die nur geringe Anforderungen in Bezug auf Zugriffskontrolle und Auditing haben.
- **Level 2:** Im Level 2 haben die Anwendungen die Möglichkeit, die Sicherheitsdienste nach eigenen Policies zu konfigurieren und zu administrieren. Innerhalb des Level 2 werden drei so genannte **Common Secure Interoperability (CSI) Features** unterschieden:
  1. *Identity based policies without delegation (CSI level 0)*: Innerhalb dieses Levels kann nur Identitätsinformation von der Quelle zum Ziel übertragen werden. Diese Information kann nicht an weitere Objekte delegiert werden.
  2. *Identity based policies with unrestricted delegation (CSI level 1):* Auch hier kann nur Identitätsinformation für sicherheitsrelevante Entscheidungen verwendet werden. Die Identitätsinformation kann an Objekte delegiert werden, die diese wiederum nutzen können. Die Delegation kann nicht eingeschränkt werden, sodass die Objekte mit den gleichen Rechten wie der ursprüngliche Principal handeln können.
  3. *Identity and privilege based policies with controlled delegation (CSI level 2):* In diesem Fall können zusätzliche Attribute des Principals (z.B. Gruppen oder Rolleninformation) übertragen werden. Es können auch nur einzelne Attribute delegiert werden und der Principal kann damit die Delegation einschränken.

Neben den verschiedenen Security und CSI Levels wird eine sichere Variante des Internet Inter ORB Protokolls (IIOP), das so genannte **SECIOP**, und ein

Non–Repudiation Service spezifiziert. SECIOP kennt den Begriff der Security Association, um damit verschlüsselte Verbindungen, Reihenfolgesicherung, Schutz vor Replay Angriffen usw. zu realisieren. Der Non–Repudiation Service macht die eindeutige Zurechnung von Aktionen zu Benutzern möglich.

Sicheres Protokoll: SECIOP

In Abbildung 3.1 aus [OMG 2001-03-08] wird die grundsätzliche Funktionsweise des Security Service am Beispiel der Zugriffskontrolle dargestellt. Für

Beispiel: Zugriffskontrolle

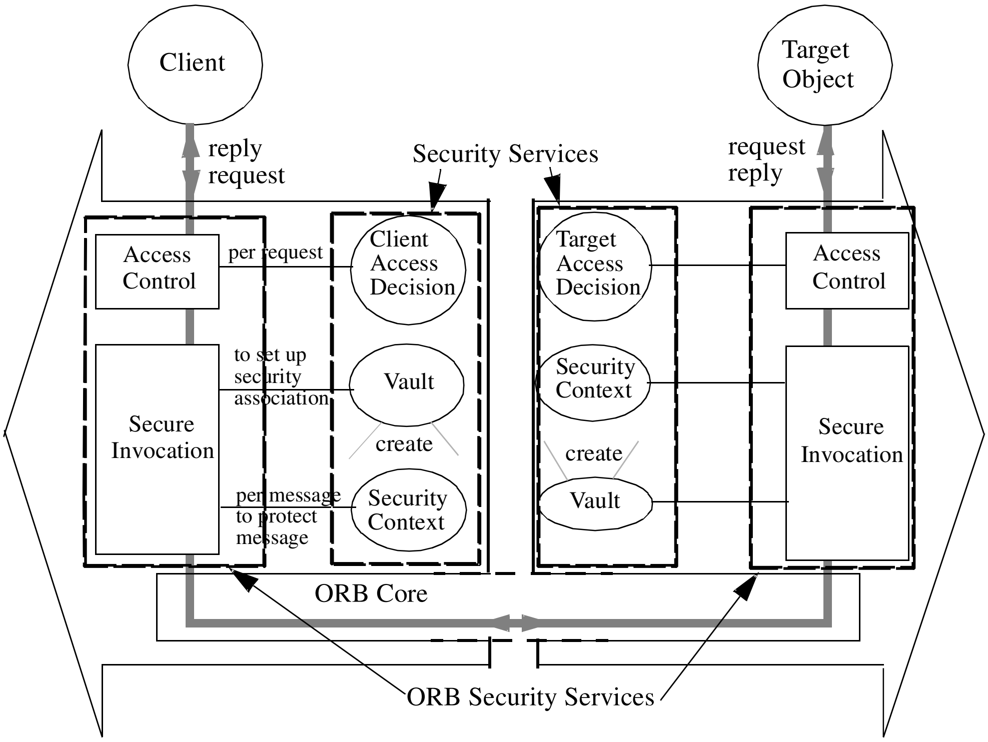


Abbildung 3.1: Security Service innerhalb des ORB [OMG 2001-03-08]

jedes Objekt werden vom Security Service Access Decision Objekte erzeugt, die entsprechende Credentials des Principals enthalten, d.h. innerhalb des Access Decision Objektes sind alle Informationen zusammengefasst, die für die Zugriffskontrolle erforderlich sind. Abhängig von diesem Objekt, der Operation, die ausgeführt werden soll, und der Policy für die Zugriffskontrolle entscheidet die Access Control Komponente, ob der Zugriff erlaubt wird. Aus der Abbildung wird ersichtlich, dass diese Entscheidung immer lokal getroffen wird und zwar sowohl auf Seite des Clients als auch auf Seite des Targets. Das Vault Objekt ist für den Aufbau einer Security Association zwischen Client und Target verantwortlich. Es erzeugt ein entsprechendes Security Context Objekt, das alle Informationen bezüglich der Verbindung, wie z.B. Schlüsselmaterial u.a., enthält. Der Security Service ist dem ORB „vorgeschaltet" und kann jeden Request und jeden Reply unterbrechen, um sicherheitsrelevante Entscheidungen zu treffen.

Für die Implementierung des Security Service werden zwei verschiedene Möglichkeiten innerhalb der ORB vorgeschlagen: das *ORB Service replaceability package* und das *Security Service replaceability package*. Im ersten Fall

Zwei Implementierungsvarianten:

1. Interceptor

wird der Security Service vom ORB nicht unterstützt, sondern mit Hilfe des Interceptor Konzeptes implementiert. Im oben angegebenen Beispiel würde die Access Control sowie die Secure Invocation Komponenten implementiert und über die Interceptor Schnittstelle in den ORB eingebunden (vgl. Abbildung 3.2 und Diskussion in Abschnitt 5.4.7 auf S. 167).

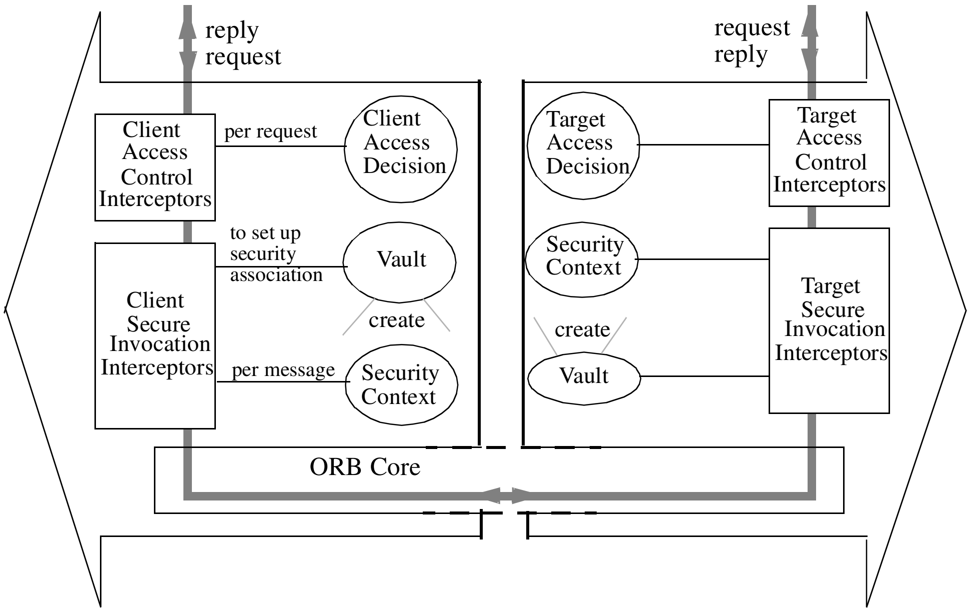


Abbildung 3.2: Security Service implementiert durch Interceptor [OMG 2001-03-08]

2. Replaceable Interfaces

Im zweiten Fall muss der ORB alle Schnittstellen des Security Service über so genannte *replaceablity interfaces* definieren, aber nicht notwendigerweise auch implementieren. Die Objekte, welche die Funktionalität tatsächlich implementieren — diese können z.B. auch durch Dienste des unterliegenden Betriebssystems erbracht werden, können dann über diese replaceability interfaces in den ORB eingebunden und auch angesprochen werden.

Der Security Service bietet mächtige Konzepte und wäre grundsätzlich sehr gut geeignet, um eine Sicherheitsarchitektur zu implementieren. Der entsprechende ORB müsste einen Level 2 Security Service implementieren, damit eigene Policies spezifiziert werden können. Auch das System Mobiler Agenten müsste einige Voraussetzungen erfüllen. Die ganze Infrastruktur müsste natürlich CORBA als Kommunikationsplattform verwenden und alle Entitäten müssten als CORBA Objekte implementiert werden.

Allerdings gibt es wegen des erheblichen Umfangs der Spezifikation und der zum Teil sehr aufwendig zu implementierenden Konzepte bisher keinen ORB, der den Security Service im Level 2 voll unterstützt. In Abschnitt 5.4.7 wird die Anwendbarkeit des Interceptor–Konzeptes für die Rechtedurchsetzung diskutiert.

3.2 Forschungsansätze

Dieser Abschnitt befasst sich mit aktuellen Forschungsfragestellungen. Im ersten Teil werden die Arbeiten bezüglich der Anforderungen an die Sicherheit Mobiler Agenten vorgestellt. Daran anschließend werden Arbeiten präsentiert, die den Schutz des Agentensystems vor feindlichen Mobilen Agenten und das wohl am häufigsten untersuchte Problem des Schutzes des Mobilen Agenten vor feindlichen Agentensystemen fokussieren.

3.2.1 Anforderungen an MA Sicherheit

Die MITRE Corporation [MITRE] — eine öffentlich geförderte Forschungseinrichtung in den Vereinigten Staaten — hat sich relativ früh mit Fragen der Sicherheit Mobiler Agenten beschäftigt. Farmer, Guttman und Swarup geben in [FGS 96] drei Klassen von Sicherheitsanforderungen an: solche, die leicht erfüllbar, solche, die überhaupt nicht durchsetzbar, und solche, die vielleicht erfüllbar sind. Dabei werden die einzelnen Anforderungen kurz vorgestellt und eine sehr kurze Begründung mit angegeben, wieso eine Sicherheitsanforderung leicht, unmöglich oder vielleicht durchsetzbar ist. Die Autoren wollen dabei so generisch wie möglich bleiben, d.h. sie treffen keinerlei Annahmen über Agentensystem, Hardware, Betriebssystem oder gar organisatorische Rahmenbedingungen. Im Folgenden werden die Klassen kurz vorgestellt:

1. Einfach zu erfüllende Sicherheitsanforderungen
   - Der Programmierer und der Sender von Mobilen Agenten können authentisiert werden.
   - Veränderungen am Programm–Code des Agenten können erkannt werden.
   - Während der Übertragung kann das Agentensystem die Vertraulichkeit des Mobilen Agenten sicherstellen.
   - Das Agentensystem kann sich vor böswilligen Agenten schützen.
2. Nicht durchsetzbare Sicherheitsanforderungen
   - Es gibt kein zuverlässiges Verfahren für den Mobilen Agenten, um Veränderungen am Programm–Code des Agentensystems zu erkennen.
   - Es kann nicht sichergestellt werden, dass der Agent in korrekter Art und Weise sowie vollständig ausgeführt wird.
   - Ob der Agent in der erwarteten Weise migriert wird, d.h. ob er zum gewünschten Zeitpunkt zum gewünschten Ziel–Agentensystem transportiert wird, kann nicht gewährleistet werden.
   - Daten und Programm–Code des Mobilen Agenten können vor dem Agentensystem nicht geheim gehalten werden.

- Der Agent kann keinen — vor dem Agentensystem geheimen — Schlüssel besitzen.
- Eine vertrauliche Kommunikation zwischen Agenten ist nicht möglich.

3. Sicherheitsanforderungen bzw. –fragen, die vielleicht lösbar sind:
    - Gibt es eine Sprache, in der alle Programme sicher sind?
    - Kann das Agentensystem sicherstellen, dass sich der Agent in einem sicheren Zustand befindet?
    - Kann der Sender eines Mobilen Agenten kontrollieren, welche Agentensysteme seinen Mobilen Agenten ausführen dürfen?

Die angegebenen Fragestellungen sind relevant und werden im Rahmen dieser Arbeit untersucht werden. Allerdings ist keine klare Systematik erkennbar, wie diese Anforderungen ermittelt wurden, und es wird auch nicht klar, ob die Vollständigkeit der Anforderungen ein angestrebtes Ziel der Autoren war. Die Klassifizierung ist fragwürdig und insbesondere bei den nicht lösbaren Problemen zum Teil überholt. Sobald gewisse einschränkende Annahmen getroffen werden, sind fast alle dieser Probleme, zumindest für Spezialfälle, lösbar.

3.2.2 Trust Level Management

Das **Trust Level Management** beschäftigt sich mit dem Aufbau einer Vertrauensbeziehung zwischen Entitäten.

Erste Definition von Vertrauen (Trust)

Der Begriff Trust bzw. Vertrauen wird in der Literatur unterschiedlich gesehen und der Aufbau einer Vertrauensbeziehung auch unterschiedlich realisiert. Vertrauen wird immer zwischen zwei Entitäten A und B aufgebaut, genauer A ermittelt seinen Trust Level für B und umgekehrt. Dabei ist für [Essi 97] Vertrauen eine Funktion des Kontextes, der Identität, der Reputation, der Fähigkeit und des Risikos. Will A seinen Trust Level für B ermitteln, so wird er in seine Überlegungen den Kontext, in dem sich beide befinden, miteinbeziehen und A muss die Identität von B bestimmen können. A betrachtet auch den „Ruf“ und die Kenntnisse und Fertigkeiten des B. Und zuletzt bestimmt auch das Risiko, das mit der Gewährung bzw. dem Missbrauch von Vertrauen verbunden ist, mit über die Höhe des Level of Trust. Diese Definition kommt aus dem Bereich der Soziologie und lässt sich nur bedingt auf Systeme Mobiler Agenten übertragen.

Für Managementsysteme basierend auf Mobilen Agenten ist der **Level of Trust** eine Klassifikation des Vertrauens, die eine Entität einer anderen entgegenzubringen bereit ist. Der bestimmende Faktor bei der Berechnung des Level of Trust, in diesem Szenario, ist die Identität der Partner–Instanz. Das Kontextwissen wird i.d.R. implizit mit der Identität verknüpft. Daneben können auch weitere Parameter in die Berechnung mit einfließen (weitere Informationen hierzu vgl. Abschnitt 4.1.2).

Prinzip der schützenden Zusicherungen

Kassab und Voas versuchen Vertrauen durch Beobachtbarkeit des Mobilen Agenten zu realisieren [KaVo 98]. Die Beobachtbarkeit wird durch so genannte **schützende Zusicherungen** implementiert, deren Prinzip auf Hoare zurückgehen [Hoar 69]. Der Code des Mobilen Agenten wird dazu mit Funktionen instrumentiert, die diese schützenden Zusicherungen (z.B. Vorbedingung, Nachbedingung, Invariante, Systemprofile) realisieren. Auf der Basis des so instrumentierten Agenten wird ein Dämonprozess, der als Oracle bezeichnet wird, generiert. Der Mobile Agent schickt die Ergebnisse der Zusicherungen an den Oracle Prozess, der diese sammelt und analysiert. Das Oracle soll dann böswillige Agentensysteme identifizieren können. Das Konzept der schützenden Zusicherung kann neben der Bereitstellung eines bestimmten Vertrauenslevels auch als Schutz vor feindlichen Agentensystemen (vgl. Abschnitt 3.2.4) eingesetzt werden.

Instrumentierung des MA mit schützenden Zusicherungen

Dieser Ansatz ist allerdings nur unter sehr strikten Voraussetzungen anwendbar, die für ein Managementsystem und i.d.R. auch für Systeme Mobiler Agenten nicht gelten. So wird bspw. davon ausgegangen, dass die schützenden Zusicherungen die Ausführungsumgebung testen können. Wie in Abschnitt 4.1 gezeigt wird, kann jedes böswillige Agentensystem dem Agenten eine „falsche“ Ausführungsumgebung vorspielen. Der Ansatz ist auch wegen der großen Komplexität und wegen des hohen Aufwands der Instrumentierung nicht geeignet.

Pessimistischer Ansatz nach Wilhelm et al.

In der Arbeit von Wilhelm et al., die sich mit E–Commerce Anwendungen und dem Aufbau von Vertrauen beschäftigt [WSB 00], besteht der Begriff Trust aus zwei Teilen; der Angemessenheit der Policy und der Vertrauenswürdigkeit des Ausstellers der Policy. Damit ein Mobiler Agent dem Agentensystem vertrauen kann, muss das Agentensystem eine Policy besitzen und diese verifiziert werden. Der Mobile Agent muss entscheiden, ob diese Policy angemessen seine Interessen repräsentiert. Im zweiten Schritt muss der Mobile Agent ein Vertrauen aufbauen, dass sich das Agentensystem entsprechend an seine Policy hält. Für den Aufbau dieses Vertrauens bieten die Autoren zwei Ansätze.

AS muss Policy besitzen, die Verhalten des AS beschreibt

Aufbau des Vertrauensverhältnisses

1. **Optimistischer Ansatz:** Der Mobile Agent glaubt, dass sich das Agentensystem entsprechend seiner Policy verhält, da das Agentensystem durch eine Verletzung der Policy wenig zu gewinnen und viel zu verlieren hat. In diesem Fall basiert das Vertrauen auf expliziter Bestrafung von Policy–Verletzungen. Dem Einsatzszenario muss also eine gewisse „Rechtsordnung“ zugrunde liegen.
2. Beim **Pessimistischen Ansatz** wird davon ausgegangen, dass das, die Policy betreffende, Verhalten des Agentensystems total überwacht werden kann.

Die Autoren verfolgen den Pessimistischen Ansatz. Die Implementierung dieses Ansatzes basiert auf vertrauenswürdiger und manipulationssicherer Spezialhardware (Tamper Proof Hardware) für das Agentensystem und einem eigenen Protokoll, mit dem alle Policy–relevanten Aktionen der beteiligten Entitäten überwacht werden können.

Diese Lösung von Wilhelm et al. ist für ein E–Commerce Szenario entwickelt worden und für ein Managementszenario so nicht anwendbar. Das Hauptproblem stellt die Abstützung auf vertrauenswürdige Spezialhardware dar. Durch diese wird die universelle Einsetzbarkeit von Agentensystemen auf unterschiedlichsten heterogenen Systemen zunichte gemacht. Würde dieser Ansatz verfolgt, müsste für jedes zu verwaltende Gerät, bzw. für eine Gruppe von zu administrierenden Systemen, Spezialhardware eingesetzt werden. Dies widerspricht klar dem Universalcharakter von Systemen Mobiler Agenten. Daneben bekennen die Autoren selbst, dass mit genügend Zeit und Ressourcen auch jede Spezialhardware manipuliert werden kann.

In [AnKu 97] werden die Prinzipien gezeigt, mit denen so genannte **Tamper Proof Hardware** manipuliert werden kann. Dieselben Autoren beschreiben in [AnKu 96] die erfolgreiche Manipulation eines „manipulationssicheren“ Spezialprozessors, der in Pay–TV Systemen und Terminals für Finanztransaktionen eingesetzt wird.

3.2.3 Schutz des Agentensystems vor feindlichen Agenten

Der Schutz des Agentensystems vor böswilligen Agenten ist sehr eng mit der allgemeinen Frage einer objektbasierten Zugriffskontrolle verbunden. Auf die allgemeine Access–Control Theorie wird an dieser Stelle nicht näher eingegangen. Es werden jedoch zwei Ansätze vorgestellt, die Zugriffskontrolle bzw. Schutz des Agentensystems auf eher unkonventionelle Weise zu lösen versuchen: *History–based Access Control* und *Proof Carrying Code*.

Zugriffskontrolle basierend auf der Historie

Klassische Zugriffskontrollverfahren machen die Entscheidung, ob ein Zugriff gestattet wird, an der Identität des Aufrufers oder einer Art von Ticket fest, das an die aufrufende Software–Komponente gebunden sein muss.

Zugriffskontrolle anhand der bisherigen „Geschichte“ des Agenten

Einen völlig anderen Ansatz verfolgt **History–based Access Control** [EAC 98]. Hier wird keinerlei Benutzerinformation oder Rechteinformation verwaltet. Die Zugriffskontrolle wird abhängig gemacht von der bisherigen „Geschichte“ des aufrufenden Mobilen Agenten. Anhand der History des Agenten wird dieser in eine Äquivalenzklasse eingeordnet. Für alle Agenten in einer Äquivalenzklasse gelten dieselben Restriktionen.

Um eine agentenspezifische Historie zu bilden, müssen die Agenten, und nicht deren Benutzer oder Besitzer, zweifelsfrei identifiziert werden können. Dies wird durch die Berechnung eines Hash–Wertes über den Code des Agenten erreicht. Um an dieser Stelle Eindeutigkeit zu erreichen, verlangen die Autoren, dass der Code des Agenten statisch gelinkt wird. Für (sicherheitsrelevante) Operationen können Events und Event–Handler spezifiziert werden. Sobald ein Agent eine derartige Operation aufruft, entscheidet der Event–Handler anhand einer oder mehrerer Policies sowie der History des Agenten, ob die Operation erlaubt wird. Außerdem wird die Operation mit in die History des Agenten aufgenommen.

Agenten und nicht Benutzer werden identifiziert

Aufgerufene Operationen bilden die History

Die Autoren machen keinerlei Angaben, wie die History implementiert werden soll. Sie sind sich zwar bewusst, dass diese sehr groß werden kann, überlassen aber die Entscheidung, wie die Histories verwaltet werden sollen, den konkreten Policies.

Der Ansatz könnte in Szenarien Sinn machen, in denen der Absender des Agenten entweder nicht ermittelbar oder nicht von Interesse ist. Hier könnte man den Agenten in einer restriktiven Umgebung beobachten, um dann sukzessive „bei guter Führung“ seine Rechte zu erweitern. Für ein Managementszenario erscheint dieser Ansatz nicht geeignet. In organisationsübergreifenden Szenarien ist eine Zurechenbarkeit zu einer Organisation oder zu einem Mitarbeiter zwingend erforderlich. Es ist nicht ausreichend „anonyme“ Agenten über deren Hash–Werte zu identifizieren.

Proof–Carrying Code

Beim Konzept des **Proof–Carrying Code** trägt der Agent die Kodierung eines Beweises mit sich, der belegt, dass der Agent eine bestimmte Sicherheitspolitik nicht verletzt [NeLe 98b]. Diese Sicherheitspolitik wird einmal vom Betreiber des Agentensystems (code consumer) spezifiziert. Der Agentenprogrammierer (code producer) muss den Code seiner Agenten um Invarianten und andere Annotationen erweitern.

Agent beinhaltet Beweis, dass er eine vorher festgelegte Sicherheitspolitik nicht verletzt

Wenn ein Mobiler Agent auf ein Agentensystem migriert, extrahiert dieses aus den Annotationen des Mobilen Agenten ein Sicherheitsprädikat. Dieses Prädikat kann nur bewiesen werden, wenn die Ausführung des Mobilen Agenten die Sicherheitspolitik des entsprechenden Agentensystems nicht verletzt. Das Agentensystem schickt dieses Prädikat an einen externen Beweiser, der versucht den Beweis zu erbringen und das Ergebnis zum Agentensystem zurückschickt. In einem weiteren Schritt muss das Agentensystem die Gültigkeit des Beweises mit einem einfachen und schnellen *Proof Checker* testen und, falls der Beweis des Sicherheitsprädikats gültig ist, kann der Mobile Agent ausgeführt werden.

Aus der Beschreibung des zweistufigen Prozesses wird bereits deutlich, dass diese Vorgehensweise bei der Ausführung sehr aufwendig ist. Dies wird auch durch die experimentellen Untersuchungen der Autoren deutlich. Aber auch bei der Erstellung des Agenten muss der Programmierer Schleifeninvarianten finden und den gesamten Code instrumentieren. Dieser Prozess lässt sich zwar durch einen so genannten Certification Compiler unterstützen und auto-

Problem: erheblicher Aufwand

matisieren, der resultierende Beweis ist aber in allen in [NeLe 98b] untersuchten Fällen um den Faktor drei bis acht größer als der eigentliche Agent. Für ein flexibles und hochdynamisches Managementsystem, bei dem die Agenten möglichst schlank bleiben sollen, um nicht unnötig Ressourcen zu belegen, erscheint dieser Aufwand insgesamt als zu hoch.

3.2.4 Schutz des Mobilen Agenten vor feindlichen Agentensystemen

Das wohl am meisten untersuchte und zugleich schwierigste Problem ist der Schutz eines Mobilen Agenten vor einem böswilligen Agentensystem. In [Hohl 98] werden die Möglichkeiten, die ein Agentensystem besitzt, um einen Mobilen Agenten anzugreifen, zusammengefasst:

Angriffsmöglichkeiten des AS auf MAs

1. **Ausspähen** des Programm–Codes, des Kontrollflusses, der Daten des Mobilen Agenten oder dessen Kommunikation mit anderen Entitäten
2. **Manipulation** des Programm–Codes, des Kontrollflusses, der Daten des Mobilen Agenten oder dessen Kommunikation mit anderen Entitäten
3. **Veränderung** des Ausführungspfades eines Mobilen Agenten
4. **Denial of execution**
5. Rückgabe von **falschen Ergebnissen** beim Aufruf von Systemfunktionen durch den Mobilen Agenten

Fragen die in Verbindung damit untersucht werden, lauten:

- Können böswillige Agentensysteme erkannt werden?
- Kann eine Veränderung oder Manipulation eines Mobilen Agenten erkannt oder verhindert werden?

In diesem Abschnitt werden einige grundlegende Arbeiten aus diesem Forschungsbereich vorgestellt und bewertet.

Detection Objects

Catherine Meadows [Mead 97] versucht in ihrem Ansatz, Veränderungen an Mobilen Agenten nachträglich zu erkennen. Dabei geht sie davon aus, dass der Mobile Agent keine Daten vor dem Agentensystem geheim halten kann. Allerdings kann das Quell–Agentensystem dem Mobilen Agenten „zusätzliche“ Datensätze mitgeben und aus deren Veränderung durch das Agentensystem auf böswillige Aktionen schließen.

Test Datensätze zur nachträglichen Erkennung von Manipulationen

Der Ansatz kommt ursprünglich aus Datenbanksystemen, in denen so genannte **Detection Objects** verwendet werden, um nicht autorisierte Veränderungen an Datensätzen zu erkennen. Bei einem „normalen“ Gebrauch der Datenbank werden die Detection Objects nicht verändert. Meadows versucht diesen Ansatz auf Mobile Agenten und E–Commerce Szenarien zu übertragen. Als Beispiel gibt sie einen Mobilen Agenten an, der Flugpreise ermitteln und sortieren soll. Als Detection Objects werden ein oder mehrere

Flugpreise bereits beim Start des Agenten mitgegeben. Nach der Rückkehr wird überprüft, ob diese verändert oder gelöscht wurden, um den Preis einer anderen Gesellschaft als günstiger erscheinen zu lassen.

Die Autorin gibt auch gleich die Probleme, die mit diesem Ansatz verbunden sind, in ihrer Arbeit an:

- Der Ansatz ist sehr anwendungsspezifisch und nicht allgemein einsetzbar.
- Detection Objects müssen plausibel sein, d.h. es ist großes Wissen über die zu erwartenden Ergebnisse erforderlich.

Meadows selbst sieht das Einsatzgebiet auf Spezialfälle beschränkt, in denen stereotype Abfragen sehr häufig durchgeführt werden und für die deshalb der Aufwand lohnt, Detection Objects zu entwickeln.

nur für Spezialfälle geeignet

Ausführungspfade und kryptographische Ausführungspfade

Bei diesen Verfahren wird der **Ausführungspfad (Trace)** des Mobilen Agenten überprüft [Yee 97, Vign 98a]. Ein Ausführungspfad ist ein Protokoll aller Operationen des Mobilen Agenten während dessen gesamtem Lebenszyklus. Das heißt, das Quell–Agentensystem kann theoretisch jeden Berechnungsschritt, den der Mobile Agent ausgeführt hat, nachvollziehen. Dazu muss der protokollierte Ausführungspfad an das Quell–Agentensystem übermittelt werden. Da dieser Trace sehr leicht größer wird als das eigentliche Ergebnis der Berechnung des Mobilen Agenten, wird versucht, die zu übertragende Datenmenge zu reduzieren.

Bei Vigna [Vign 98a] ist ein Trace eine Sequenz aus Paaren `<ID, sig>`. `ID` ist ein eindeutiger Bezeichner, `sig` die Signatur der vom Mobilen Agenten aufgerufenen Methode. Um den Trace möglichst klein zu halten, werden „weiße“ und „schwarze“ Methoden unterschieden. Weiße Methoden verwenden nur interne Variablen des Mobilen Agenten, wohingegen schwarze Methoden Input von der Ausführungsumgebung nutzen. Nur für schwarze Methoden wird die Belegung der Attribute mit externen Werten mit in der Signatur `sig` gespeichert. Vigna gibt ein kryptographisch gesichertes Protokoll für den Austausch von Agenten und deren Ausführungspfaden an.

Im Normalfall migriert der Mobile Agent über verschiedene Agentensysteme AS_i und jedes gibt dem Agenten einen gesicherten Hash–Wert (vgl. auch Seite 88) des Ausführungspfades $\big(H(T_{AS_i}^{MA})\big)$ mit. Nach der Rückkehr des Mobilen Agenten und bei einem Verdacht der Manipulation muss das Quell–Agentensystem der Reihe nach alle Ausführungspfade anfordern. Der Mobile Agent wird dann noch einmal Schritt für Schritt ausgeführt und die externen Daten werden aus den Signaturen im Trace übernommen. Jede Diskrepanz wird gefunden; entweder weicht das simulierte Ergebnis vom zurückgebrachten Ergebnis ab, dann wurde der Mobile Agent manipuliert, oder der Hash–Wert des Ausführungspfades stimmt nicht mit dem vom Mobilen Agenten gelieferten überein, dann wurde der Ausführungspfad manipuliert. Für Implementierungen, die interpretierte Sprachen verwenden, kann

Bei Manipulationsverdacht simuliert Quell–AS die Ausführung des MA der Ansatz versagen, wenn die Sprache Methoden bietet, die Zeichenfolgen als Parameter erhalten und als Befehlsfolge interpretieren. Werden mit Hilfe solcher Methoden externe Programme gestartet, kann für diesen Teil der Ausführung kein Ausführungspfad erzeugt werden, da externe Programme dies in der Regel nicht unterstützen.

Probleme: Größe des Trace; Test nur bei konkretem Verdacht

Die Hauptprobleme des Ansatzes sind jedoch die Größe der Ausführungspfade und die Tatsache, dass Verletzungen der Integrität nur bei Verdacht und nach expliziter Anforderung des gesamten Ausführungspfades erkannt werden.

Mobile Kryptographie; Verschlüsselte Funktionen

Funktionalität wird verschlüsselt

Sander und Tschudin [SaTs 97, SaTs 98, SaTs 98a] versuchen die Manipulation des Mobilen Agenten konzeptionell zu verhindern. Die Grundidee dabei ist, die Methoden des Mobilen Agenten zu verschlüsseln. Das Agentensystem führt also verschlüsselte Funktionen aus, ohne diese zu „verstehen“. Die Verschlüsselung liefert Vertraulichkeit und Integrität. Die verschlüsselten Methoden können nicht mehr bewusst, sondern nur noch zufällig verändert werden (vgl. auch die Diskussion über „blinde“ Manipulationen in Abschnitt 4.5.4).

Die Autoren unterscheiden zwischen Funktion und Programm. Ein Programm implementiert in ihrem Verständnis eine Funktion. Was sie erreichen wollen, ist die Verschlüsselung der Funktion, sodass deren Transformation in ein Programm (bzw. in einen Mobilen Agenten), zwar von einem Agentensystem interpretiert und ausgeführt, aber nicht „verstanden“ werden kann. Dies wird von ihnen als **Computing with encrypted Functions (CEF)** bezeichnet.

Rechnen mit verschlüsselten Funktionen

Alice möchte, dass Bob ihre Funktion f mit seinem Input x berechnet, ohne dabei verstehen zu können, wie f arbeitet und was damit berechnet wird. Alice verschlüsselt ihre Funktion in $E(f)$ und erzeugt daraus ein ausführbares Programm $P(E(f))$, das $E(f)$ implementiert. Dieses Programm (bzw. den MA, der $P(E(f))$ implementiert) schickt sie an Bob, der $P(E(f))(x)$ berechnet und an Alice zurückschickt, die durch eine Entschlüsselung das Ergebnis $f(x)$ erhält (vgl. Abbildung 3.3). Um diesen Ablauf realisieren zu können, bedarf es Verfahren, welche die Verschlüsselung allgemeiner Funktionen f ermöglichen. Als erster Versuch werden von den Autoren Verschlüsselungsverfahren (auf Basis homomorpher Einwegfunktionen) für Polynome und rationale Funktionen angegeben.

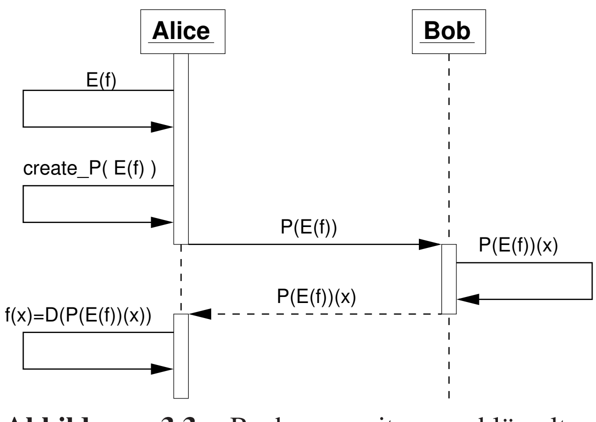


Abbildung 3.3: Rechnen mit verschlüsselten Funktionen

Allerdings sagen die Autoren selbst in [SaTs 98a], dass es sich um theoretische Ergebnisse handelt und dass die Verfahren im Moment in Bezug auf ihre Durchführbarkeit und die erforderliche Interaktivität für den praktischen Einsatz nicht geeignet seien. Daneben besteht das Problem, dass das Agentensystem auch bei verschlüsselten Funktionen deren In– und Output überwachen und damit u.U. die Funktionalität ableiten bzw. dem Mobilen Agenten falsche Input–Werte geben kann.

Umgebungsabhängige Schlüsselgenerierung

Eine weitere Möglichkeit, um Teile des Mobilen Agenten zu verschlüsseln wird in [RiSc 98] vorgeschlagen. Ein so genannter **Clueless Agent (Ahnungsloser Agent)** beinhaltet verschlüsselte Teile (Daten oder Instruktionen). Außerdem besitzt er eine Funktion, um den Schlüssel für diese Daten zu berechnen. Diese Funktion zur Berechnung des Schlüssels ist abhängig von einer bestimmten Ablaufumgebung. Erst innerhalb dieser Umgebung und mit Daten, die in dieser Umgebung zugänglich sind, kann der Agent den gültigen Schlüssel berechnen. Ohne den umgebungsabhängigen Input kann der Agent seine Daten nicht entschlüsseln; er bleibt „ahnungslos“.

Idee: Agent kann Daten nur in einer bestimmten Umgebung entschlüsseln

1. if $H(N) = M$ then $K := N$
2. if $H(H(N)) = M$ then $K := H(N)$
3. if $H(N_i) = M_i$ then $K := H(N_1 \bullet \cdots \bullet N_i)$
4. if $H(N) = M$ then $K := H(r_i \bullet N) \oplus r_2$

Abbildung 3.4: Umgebungsabhängige Schlüsselgenerierung; Algorithmen nach [RiSc 98]

Die umgebungsabhängige Schlüsselerzeugung wird auch als **Environmental Key Generation** bezeichnet. Einwegfunktionen lassen sich nutzen, um die umgebungsabhängige Schlüsselgenerierung zu realisieren. In [RiSc 98] werden die in Abbildung 3.4 dargestellten Verfahren vorgeschlagen. Dabei bezeichnet N die umgebungsabhängige Information, r_i eine Zufallszahl, H die Einwegfunktion, z.B. eine kryptographische Hash–Funktion (vgl. auch Seite 88) und K den Schlüssel. Die Konkatenation wird durch $\bullet$ symbolisiert und $\oplus$ bezeichnet den Exklusiv–Oder Operator. Im ersten Fall wird das umgebungsabhängige Datum N als Schlüssel verwendet, falls dessen Hash–Wert mit dem vorgegebenen Wert M übereinstimmt. Das heißt nur in der Umgebung, die das „richtige“ N zur Verfügung stellen kann, ist der Agent in der Lage, seine Daten zu entschlüsseln.

Dieser Ansatz erscheint eher realisierbar als die im vorhergehenden Abschnitt beschriebene Mobile Kryptographie. Allerdings wird der Code zur umgebungsabhängigen Schlüsselgenerierung selbst wieder im Klartext transportiert. Ein feindliches Agentensystem hat damit die Möglichkeit durch wiederholtes Testen und ggf. Analyse des Programm–Codes N zu ermitteln und dem Agenten die geforderte Ausführungsumgebung vorzutäuschen (Maskerade), um an die verschlüsselten Daten zu gelangen.

Time Limited Blackbox Security

Blackbox: AS kann MA weder lesen noch verändern

Der Ansatz von Fritz Hohl [Hohl 98] ist dem CEF–Ansatz ähnlich. Der Mobile Agent wird in einen neuen Agenten, die so genannte **Blackbox**, transformiert. Ein Agent erfüllt die Eigenschaften einer Blackbox, wenn seine Daten und sein Programmcode vom Agentensystem weder gelesen noch modifiziert werden können. Und er erfüllt die Eigenschaften einer zeitlich begrenzten Blackbox (**Time Limited Blackbox**), wenn der Agent für ein bestimmtes und vorher bekanntes Zeitintervall die Blackbox Eigenschaft erfüllt und nach diesem Zeitraum Angriffe zwar möglich sind, aber keine Auswirkungen mehr haben. Dem Agenten wird von Beginn an eine begrenzte Lebenszeit zugewiesen, nach deren Ablauf er weder migrieren noch mit anderen Entitäten interagieren kann.

Original Programmcode:

```
if (a(b)<c) {
              b = s(d(e)+f);
}
```

Obfuscated Code:

```
z=0
DO
  if(z==0) then t1 = a(b); z=1; continue;
  if(z==1)then t2 = t1<c; z=2; continue;
  if(t2) then t3 = d(e); z=3; continue;
  if(z==3) then t4 = t3+f; z=4; continue;
  if(z==4) then b = t4; z=5; continue;
  if(z==5) then break;
LOOP
```

Abbildung 3.5: Beispiel für Obfuscating nach [Hohl 98]

Verschleierung des Kontrollflusses durch Obfuscating

Hohl sagt in seiner Arbeit, dass ein Agentensystem immer den Inhalt jeder Variablen kennt, jedes Bit im Speicherbereich eines Agenten und jede Zeile Programmcode auslesen kann. Allerdings gilt diese Aussage, nach Hohl, nur für die Semantik „im Kleinen“ und nicht für die Semantik „im Großen“. Ein Angreifer kann zwar einzelne Bits und Code–Zeilen der Blackbox auslesen, aber daraus nicht das „Mentale Modell“ des Programmierers, d.h. den eigentlichen Algorithmus, ermitteln. Um die Blackbox zu erzeugen, werden **obfuscating** oder **mess–up Algorithmen** verwendet, die den Programmcode und die Daten „durcheinander würfeln“. Beim Programmcode wird die Art des Statements und der Kontrollfluss verborgen. Dies geschieht im Wesentlichen durch die Umwandlung des Ausgangs–Codes in schwer verständlichen

„Spaghetti–Code“ mit DO–WHILE Schleifen und konditionalen Sprüngen, die von Werten abhängen, die erst zur Laufzeit berechnet werden können (Ein Beispiel ist in Abbildung 3.5 angegeben). Bei den Variablen werden durch eine Restrukturierung der Variablen Datentyp und Dateninhalt verschleiert (vgl. Abbildung 3.6).

Datentyp und Dateninhalt: Verschleierung durch Restrukturierung

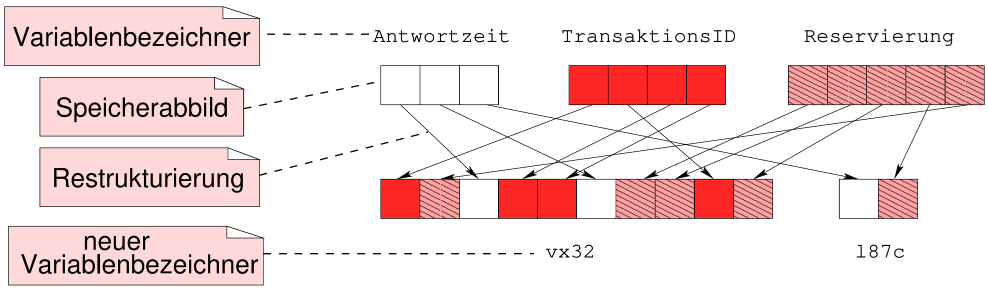


Abbildung 3.6: Rekonfiguration von Daten nach [Hohl 98]

Der Ansatz von Hohl ist mit folgenden Problemen behaftet:

Problem: Sicherheit des Obfuscating Algorithmus

- Eine sinnvolle und sichere Festlegung der Blackbox–Lebenszeit ist schwierig, da der Wert abhängig ist von der Aufgabe, die der Agent zu erledigen hat, und der Sicherheit des Obfuscating Algorithmus. Der Agent muss seine Aufgabe innerhalb seiner Lebenszeit abschließen können.
- Bisher existieren keine formalen Modelle für Obfuscating Algorithmen. Eine Abschätzung des Aufwands, der zum Brechen der Blackbox benötigt wird, ist deshalb nur schwer möglich.
- Auch die Blackbox kann die in Abschnitt 4.5.4 diskutierte blinde Manipulation des Agenten nicht verhindern. Hohl bezeichnet diese Art der Manipulation der Blackbox als Sabotage.
- Ein Angreifer kann die Blackbox durch systematisches und systemunterstütztes Testen brechen. Dabei wird versucht, das Innere der Blackbox zu erschließen, indem verschiedenste Input–Werte dem entsprechenden Output gegenüberstellt werden, um daraus den Algorithmus oder die Funktionalität der Blackbox abzuleiten.

Sichere Hardware

In [BoKa 01] wird ein Verfahren angegeben, um die vom Agenten auf seiner Reise transportierten Daten vor unberechtigtem Zugriff zu schützen. Dabei wird ein Szenario aus dem Bereich des E–Commerce zugrundegelegt, bei dem ein Agent verschiedene Händlerplattformen besucht, um Preise zu ermitteln oder Produkte zu kaufen.

Bohn und Karjoth spezifizieren grundlegende Schnittstellen zur Bearbeitung (Lesen, Hinzufügen, Auswerten und Aktualisieren) der entsprechenden Preisdaten. Über die Preisdaten werden so genannte Hash–Chains gebildet, die eine nachträgliche Manipulation erkennen lassen. Die Implementierungen für diese Schnittstellen und die Hash–Chains werden durch Container–Klassen erbracht. Das Besondere dabei ist, dass sich diese Container auf sichere Hardware in Form von Chipkarten (JavaCard [Chen 00]) abstützen. Während der Berechnung können die kritischen Elemente des Stacks sowie Kontrollinformationen nur innerhalb der sicheren Hardware bearbeitet werden. Eine Manipulation durch den Betreiber des Agentensystems wird dadurch ausgeschlossen.

sicherheitskritische Operationen nur auf sicherer Hardware

Die sicherheitskritischen Datenstrukturen werden dabei so klein gehalten, dass sie komplett in den Speicher der Smart–Card passen. Die restlichen, zum Teil sehr umfangreichen Daten können im konventionellen ungesicherten Speicher verbleiben.

Das Hauptproblem bei diesem Ansatz, falls er in einem Managementszenario angewendet werden soll, ist die Nachrüstung aller Geräte mit Chipkarten und den entsprechenden Lesegeräten. Außerdem liefert die Verwendung von manipulationssicherer Hardware, wie auf Seite 56 bereits diskutiert, nur relativen Schutz. Alle Karten und Lesegeräte müssten regelmäßig physisch auf Manipulationen hin untersucht werden. Bei Anwendungsszenarien, wie z.B. der in Abschnitt 2.2.1 beschriebenen europaweiten Händlervernetzung eines Automobilherstellers, dürfte dies kaum möglich sein.

Allerdings ist das Prinzip der Hash–Chains durchaus tragfähig. Auch in dieser Arbeit finden Hash–Chains, allerdings ohne die Abstützung auf sichere Hardware, Verwendung (vgl. Abschnitt 4.5.2).

Secret Sharing Schemes, Threshold Schemes; Replikation von Agenten

Fred Schneider [Schn 97] gibt ein Verfahren an, um zwei Arten von Angriffen auf die Daten eines Mobilen Agenten zu verhindern:

1. Ein böswilliges Agentensystem kann geheime Daten missbrauchen.
2. Ein böswilliges Agentensystem kann Teile der Daten vernichten und damit die erfolgreiche Berechnung eines Gesamtergebnisses verhindern.

Als Prämisse wird ein bestimmter Anteil böswilliger Agentensysteme zugrunde gelegt, die ein Mobiler Agent auf seiner Reise besucht. Das vorgeschlagene Verfahren basiert auf dem Konzept des Teilens von Geheimnissen (**Secret Sharing**). Dabei wird ein Geheimnis, das nur Quelle und Ziel kennen sollen, in mehrere Teile aufgeteilt und jeder Teil wird von einem anderen Mobilen Agenten auf einem, von den anderen Mobilen Agenten unabhängigen Weg, zum Ziel transportiert.

Threshold Scheme: Verfahren zum Aufteilen von Geheimnissen

Die Verfahren zur Aufteilung des Geheimnisses werden als Threshold Schemes bezeichnet, die erstmals in [Sham 79] eingeführt wurden. Ein (n, k) **Threshold Schemes** teilt ein Geheimnis in n Teile und nur der Besitz von mindestens k dieser Teile ermöglicht eine Wiederherstellung des Geheimnisses [DeFr 89, DeFr 91, GPSN 96, ShGe 98].

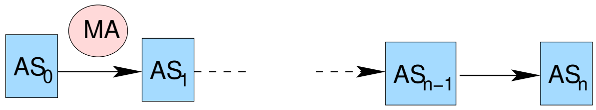


Abbildung 3.7: Ausgangsszenario für Secret Sharing Scheme

Als Ausgangsszenario wird ein Mobiler Agent verwendet, der ein Geheimnis von einem Quell– zu einem Ziel–Agentensystem transportiert. Dabei besucht er die Agentensysteme AS_1 bis AS_{n-1} (vgl. Abbildung 3.7); jedes besuchte Agentensystem wird dabei als Stufe i bezeichnet. Der Agent bzw. das transportierte Geheimnis wird aufgeteilt und repliziert, d.h. es wird nicht mehr ein Mobiler Agent von Stufe i zu Stufe $i+1$ übertragen, sondern $2k-1$ Agenten auf $2k-1$ verschiedene Agentensysteme. Jeder dieser Agenten trägt dabei einen Teil des Geheimnisses, das mit einem $(2k-1, k)$ Threshold Scheme aufgeteilt wurde (vgl. Abbildung 3.8). Diesem Ansatz liegt die Annahme zugrunde, dass in keiner Stufe i die Mehrheit k der Agentensysteme bösartig ist. Bei diesem Schema können also weniger als k böswillige Agentensys-

Idee: Daten werden mit Threshold Scheme aufgeteilt und über getrennte Wege übertragen

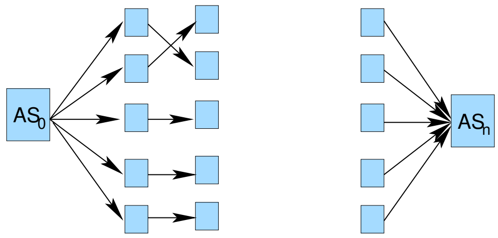


Abbildung 3.8: Secret Sharing mit einem $(2k-1, k)$ Threshold Scheme

teme pro Stufe das Geheimnis weder enthüllen noch zerstören. Dies gilt nicht mehr, sobald k böswillige Agentensysteme, die über mehrere Stufen verteilt sind, zusammenarbeiten.

Diese Schwäche wird beseitigt durch eine wiederholte Anwendung des $(2k-1, k)$ Threshold Schemes auf jeder Stufe. Das heißt, das Quell–Agentensystem teilt das Geheimnis in $2k-1$ Teile auf. Jedes Agentensystem der Stufe i nimmt alle Geheimnisse, die es über Agenten von der Stufe $i-1$ erhalten hat, konkateniert diese nach einer bestimmten Ordnung und wendet auf die resultierenden Daten erneut ein $(2k-1, k)$ Threshold Scheme an. Die dabei erhaltenen $2k-1$ Daten werden mit Mobilen Agenten an $2k-1$ verschiedene Agentensysteme der nächsten Stufe übertragen (vgl. Abbildung 3.9) Das Ziel–Agentensystem muss dann dieses Threshold Scheme $n-1$ mal „rückwärts" auf mindestens k Agenten anwenden, um das Geheimnis wieder zu erhalten.

Obwohl sich ein modifiziertes Verfahren nach Schneider auch auf Zwischenergebnisse, die der Mobile Agent auf seiner Reise berechnet, anwenden ließe, zeigen sich sehr schnell die Schwächen dieser Verfahrensklasse.

Probleme dieses Ansatzes

Der Datenverkehr und die Anzahl der übertragenen Mobilen Agenten steigt sehr stark an. Bei einer Reise der Länge n, d.h. von AS_0 zu AS_n, und ei-

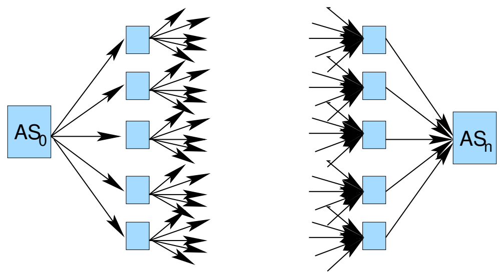


Abbildung 3.9: Secret Sharing mit wiederholtem $(2k - 1, k)$ Threshold Scheme

nem $(2k - 1, k)$ Threshold Scheme werden $4k - 2 + (n - 2)(2k - 1)^2$ „Agenten–Hops" benötigt. Ein Mobiler Agent, der von einem AS_i zu einem AS_{i+1} übertragen wird, repräsentiert einen Agenten–Hop. Im Fall einer normalen Reise der Länge n würden n Agenten–Hops benötigt. Das Hauptproblem dieses Ansatzes ist jedoch, dass der zugrunde liegende Anwendungsfall — ein Mobiler Agent transportiert ein Geheimnis über mehrere Agentensysteme zum Ziel–Agentensystem — in Managementanwendungen sehr selten vorkommt und diese Daten i.d.R. besser und einfacher durch Verschlüsselungstechniken als durch Secret Sharing Schemes geschützt werden können.

Kooperierende Agenten

Idee: Agenten teilen sich regelmäßig die geplante Reiseroute mit

Volker Roth versucht in seinem Ansatz der kooperierenden Agenten, die Reiseroute von Agenten zu sichern. Für Mobile Agenten, denen beim Start eine lose Reiseroute zugewiesen wurde, besteht die Gefahr, dass ein feindliches Agentensystem die Route des Agenten verändert, verkürzt oder beendet, ohne dass dies vom Sender des Agenten erkannt werden kann. Bei einer **losen Reiseroute** kann der Agent sein nächstes Ziel in Abhängigkeit bereits besuchter Systeme oder bisheriger Berechnungen selbst festlegen.

Kooperierende Agenten sind in der Lage, diese Manipulationen auch für lose Reiserouten zu erkennen. Roth geht in seinem Ansatz [Roth 98], davon aus, dass es drei Klassen von Agentensystemen gibt, denen er verschiedene Farben zuweist:

1. Weißen Agentensystemen kann der Mobile Agent vollkommen vertrauen. Allerdings gibt es davon sehr wenige.
2. Graue Agentensysteme sind nur teilweise vertrauenswürdig. Es kann nicht ausgeschlossen werden, dass ein bestimmter Prozentsatz versucht den Agenten anzugreifen. Dabei arbeiten die Agentensysteme aber nicht zusammen.
3. Rote Agentensysteme sind böswillig und können zusammenarbeiten, um einen Agenten anzugreifen.

Die Grundidee des Ansatzes von Roth ist es, Daten auf zwei kooperierende Agenten zu verteilen. Die Agenten besuchen zwei völlig disjunkte Mengen von Agentensystemen, für die gilt, dass kein rotes Agentensystem der einen Domäne mit einem roten Agentensystem der anderen Domäne zusammenarbeitet. Innerhalb der Domäne kann der Agent aber weiterhin von einzelnen grauen oder Gruppen von roten Agentensystemen attackiert werden.

In [Roth 98] wird ein Protokoll angegeben, bei dem sich die Agenten vor jeder Migration jeweils gegenseitig die Identität des vorher besuchten und des als nächstes zu besuchenden Agentensystems mitteilen. Damit kennt jeder Agent die geplante und tatsächliche Reiseroute des anderen. Über das Protokoll können dann die oben genannten Angriffe erkannt werden.

Allerdings ist dies nur möglich, wenn folgende Prämissen gelten:

1. Die Agentensysteme stellen den Agenten authentisierte Kommunikationskanäle zur Verfügung. Der Agent A kann sich sicher sein, mit dem Agenten B zu kommunizieren und umgekehrt.
2. Authentisierte Identitätsinformationen über das Quell–Agentensystem, das nächste Ziel–Agentensystem und das gerade besuchte Agentensystem werden vom besuchten Agentensystem zur Verfügung gestellt.

Dieser Ansatz funktioniert also nur für ein sehr eng beschränktes Anwendungsgebiet und die zugrunde liegenden Annahmen sind sehr restriktiv.

3.2.5 Zusammenfassung

Die in diesem Abschnitt vorgestellten Konzepte können durchaus Anregungen für die Sicherheitsarchitektur, deren Dienste und Komponenten liefern. Allerdings ist eine einfache unveränderte Übernahme der einzelnen Konzepte in der Regel nicht möglich.

Die Ansätze haben oft ein sehr beschränktes Anwendungsszenario, meist aus dem Bereich E–Commerce. Oder aber die Ansätze sind nur unter sehr restriktiven Voraussetzungen oder nur für Spezialfälle anwendbar.

Einige der Konzepte sind schwierig zu implementieren, nicht skalierbar oder nicht performant genug, um in großen heterogenen Managementszenarien eingesetzt zu werden. Auch die in Managementsystemen erforderliche Dynamik wird nicht von allen Ansätzen unterstützt.

Kapitel 4

Sicherheitskonzepte und Sicherheitsmechanismen

Inhaltsverzeichnis

Traditionelle Sicherheitsmodelle [BeLa 73, LaPa 96a, LaPa 96b, BrNa 89] basieren auf einem Referenzmonitorkonzept, das durch Hardware unterstützt wird und administrativ definierte Sicherheitspolitiken durchsetzt. Die meisten dieser Modelle stammen aus dem militärischen Bereich und aus einer Zeit zu der es relativ wenige Großrechner gab, die teuer und nicht oder nur wenig vernetzt waren, sowie von spezialisierten Teams bedient und gewartet wurden. Die Hardware selbst wurde innerhalb von besonders gesicherten Rechenzentren physisch geschützt. Die Modelle betrachten Computer als „Informations–Festungen". Der Rechner selbst stellt die äußere Mauer dieser **Trusted Computer Base (TCB)** dar. Die Sicherheitsgrenze kann nur durch berechtigte und vertrauenswürdige Benutzer über dezidierte Zugangssysteme und mittels eines geeigneten „Passwortes" überwunden werden. Die Trusted Computer Base wird dadurch organisatorisch um vertrauenswürdige Benutzer „erweitert". Im Prinzip wird dieses Konzept noch immer und auch für vernetzte Systeme verwendet. Dabei stellen die vernetzten Systeme eine Menge von einzelnen Trusted Computer Bases dar, die lediglich durch Sicherheitsdienste für die Übertragung von Daten zwischen den einzelnen Systemen erweitert wurden.

Das Prinzip der Trusted Computer Base basiert größtenteils auf dem physischen Schutz der Hardware, Zugangssystemen und besonders gesicherten Betriebssystem–Komponenten, die den Referenzmonitor bilden. Der Referenzmonitor überwacht den Zugriff auf Daten sowie Ressourcen. Implizit wird bei der Trusted Computer Base davon ausgegangen, dass keine der Bestandteile der TCB manipuliert werden kann, ohne dass diese Manipulation erkannt wird.

Für große verteilte und heterogene Systeme hat jedoch keine dieser Prämissen Gültigkeit [Blak 96]. Der physische Schutz tausender Rechner sowie mobiler Computer Systeme, die überall genutzt werden können, ist nicht zu gewährleisten. Obwohl es für das Management verteilter Systeme i.d.R. ein spezialisiertes Team gibt, fällt die Administration aller Rechner nicht in den Aufgabenbereich nur eines Teams, sondern es existieren Systeme, die dezentralen oder sogar externen Organisationseinheiten zugeordnet sind und von diesen auch selbständig und eigenverantwortlich administriert werden. Insbesondere kann in einem derartigen System nicht von *einer* Trusted Computer Base ausgegangen werden.

Sicherheitsdienst	Entitäten–modell	Relationenmodell		
		Ausführungs–relation	Aufruf–relation	Kommunikations– u. Migrationsrelation
Identifikation	✔ (4.2.2)	—	—	—
Authentisierung	✔ (4.2.4)	—	—	—
Autorisierung	✔ (4.4.2)	—	—	—
Zugriffskontrolle	—	✔ (4.1)	✔ (4.4.1)	✔ (4.4.1)
Verbindlichkeit	—	✔ (4.1)	✔ (4.4.3)	✔ (4.5.4)
Vertraulichkeit	(Anonymität) (4.5.2)	✔ (4.3.2)	—	✔ (4.5.1)
Integrität	—	✔ (4.3.1)	—	✔ (4.5.2)
Sandboxing	—	✔ (4.3.3)	—	—
Reisehistorie	✔ (4.5.3)	—	—	✔ (4.5.3)

Tabelle 4.1: Entitäten– und Rollenmodell: Sicherheitsdienste

Besonders für Systeme Mobiler Agenten ist der TCB–Ansatz nicht mehr haltbar. Mobile Agenten sind in keiner Weise von irgendwelcher „vertrauenswürdiger“ Hardware abhängig. Mobile Agenten als Software–Komponenten können sich autonom bewegen und organisatorische Grenzen überschreiten. Es reicht nicht mehr aus, den „physischen Zugriff“ auf Rechenanlagen zu schützen, sondern es müssen autonome Software–Bausteine in Umgebungen, die sowohl technisch als auch organisatorisch heterogen sind, geschützt werden.

In Kapitel 2 wurde ausgehend von einem allgemeinen Modell für Systeme Mobiler Agenten eine Risikoanalyse durchgeführt und es wurden Sicherheitsanforderungen abgeleitet. Diese Anforderungen werden durch Sicherheitsdienste, die in diesem Kapitel betrachtet werden, realisiert.

Nicht alle Sicherheitsdienste sind auf alle Entitäten bzw. Relationen anwendbar. Beispielsweise macht der Authentisierungsdienst nur für Entitäten und nicht für Relationen Sinn. Nur an Entitäten kann Identitätsinformation geknüpft werden, um diese später zu verifizieren. Die Authentisierung dient aber als Basis für die Zugriffskontrolle auf Relationen. Tabelle 4.1 fasst eine grobe Klassifikation der Sicherheitsdienste mit den Entitäten bzw. Relationenmodellen zusammen. In der Tabelle ist angegeben, ob der Sicherheitsdienst für die entsprechende Klasse des Modells zur Verfügung gestellt werden muss. Die entsprechenden Abschnitte dieses Kapitels, in denen der Dienst untersucht wird, sind in Klammern angegeben.

4.1 Vertrauen durch Einbettungsbeziehung

Wird die Ausführungsrelation näher betrachtet, so lässt sich feststellen, dass es sich dabei um eine gerichtete Relation handelt. Das Endsystem führt das Agentensystem aus und dieses wiederum die Mobilen Agenten. Eine Umkehrung der Relationsrichtung ist i. Allg. nicht möglich. So kann bspw. auf einem Agentensystem nicht das Betriebssystem eines Endsystems zur Ausführung gebracht werden. Die gerichtete Relation verdeutlicht auch Kontroll– und Manipulationsmöglichkeiten. Der Executee wird unter der vollständigen Kontrolle des Executors ausgeführt.

vollständige Kontrolle des Executors über den Executee

Der Executor stellt die Ablaufumgebung für den Executee bereit, er kontrolliert und überwacht dessen Ausführung. Der Executor ist grundsätzlich in der Lage, den Code, die Daten und den Ausführungspfad des Executee einzusehen und sogar zu manipulieren. Das heißt, alle Informationen des Executees sind für den Executor zugänglich [Gong 97, CGHL 95]. Der Executee kann keine Geheimnisse vor dem Executor haben. Das bedeutet auch, dass der Executee z.B. keine Schlüssel vor dem Executor verbergen kann, da dieser sie spätestens bei ihrer Verwendung „mitlesen“ kann.

Zur Verdeutlichung sei ein böswilliges Agentensystem als Executor eines Mobilen Agenten angenommen. Wenn der Mobile Agent auf das Agentensystem migriert, kann das Agentensystem die von ihm transportierten Daten auslesen. Führt der Mobile Agent Berechnungen durch, in die lokale Daten eingehen, kann ihm das Agentensystem beliebig „gefälschte“ Daten als Input liefern. Es könnte dem Mobilen Agenten z.B. vorspiegeln, dass er sich auf einem anderen Agentensystem befindet. Das Agentensystem kann auch durch Manipulation des Ausführungspfades (z.B. Veränderung von Sprungadressen) oder des Codes des Mobilen Agenten (z.B. Entfernen oder Überschreiben bestimmter

Code–Teile) dafür sorgen, dass bestimmte Operationen des Mobilen Agenten überhaupt nicht ausgeführt werden. Mit Hilfe dieser Techniken kann das Agentensystem auch die Ergebnisse von Operationen manipulieren. Der Executee selbst ist dabei nicht einmal in der Lage, Manipulationen durch den Executor zu erkennen, solange er sich auf dem Agentensystem befindet. Angenommen der Mobile Agent hätte eine boolsche Funktion `integrity`, um seine eigene Code–Integrität zu testen. Wenn der Mobile Agent diese Funktion ausführt, kann das Agentensystem durch geeignete Manipulationen dafür sorgen, dass die Funktion immer `true` zurückliefert, selbst wenn der Mobile Agent verändert wurde.

Das bedeutet, dass die Executees AS bzw. MA folgende Fragestellungen (ohne die Hilfe Dritter) nicht beantworten kann.

- Wird oder wurde der MA/das AS vom AS/Endsystem korrekt ausgeführt?
- Wurde der MA/das AS vom AS/Endsystem verändert?
- Wird oder wurde ein MA auf das gewünschte AS migriert oder aber umgeleitet?

Es gibt zwar Ansätze, um Teil– bzw. Spezialprobleme aus diesem Komplex zu lösen (vgl. z.B. Abschnitt 3 oder [SaTs 98, Hohl 98, Vign 98a]), aber in Allgemeinheit wird sich das Problem der vollständigen Kontrolle des Executors über den Executee nicht lösen lassen.

Diese Problematik dürfte auch der Grund sein, warum sich Mobile Agenten in einem offenen Szenario, wie es z.B. im E–, oder M–Commerce gegeben ist, bisher nicht durchgesetzt haben. In einem offenen Szenario stellen verschiedene Anbieter Agentensysteme zur Verfügung, um z.B. bestimmte Produkte anzubieten und zu verkaufen. Die Mobilen Agenten, die von beliebigen Quellen stammen können, besuchen diese Agentensysteme, um z.B. das preiswerteste Produkt einer bestimmten Klasse zu kaufen. In diesem Szenario bestehen per se keinerlei Beziehungen zwischen den Betreibern der Agentensysteme und denjenigen, die die Mobilen Agenten beauftragen. Es können auch keine allgemeinen Aussagen oder Annahmen über Vertrauensbeziehungen getroffen werden. Offene Szenarien werden daher in dieser Arbeit nicht betrachtet.

In den Managementszenarien, wie sie hier zugrunde liegen, muss und kann jedoch vom Grundsatz des „**erzwungenen Vertrauens** durch **Einbettungsbeziehung**“ ausgegangen werden. Wie die angegebenen Beispiele zeigen, hat der Executee im allgemeinen Fall nicht die Möglichkeit, sich vollständig vor einem böswilligen Executor zu schützen. Der Grundsatz des erzwungenen Vertrauens besagt, dass der Executee zu einem gewissen Grad seinem Executor vertrauen muss. Für das Sicherheitsmodell impliziert diese Erkenntnis, dass es Mechanismen geben muss, um ein **Vertrauensverhältnis** (Level of Trust) zwischen Executor und Executee aufzubauen.

Executee muss seinem Executor vertrauen

Das erzwungene Vertrauen durch Einbettungsbeziehungen darf für ein Managementsystem vorausgesetzt werden. Die Organisationen, die in einer Geschäftsbeziehung stehen und ein gemeinsames organisationsübergreifendes Managementsystem betreiben oder nutzen, sind gegenseitig bekannt, können bestimmte organisatorische und technische Vereinbarungen vertraglich fixieren und auch durchsetzen, bevor sie Mobile Agenten austauschen. Dies unterscheidet das Managementszenario klar von offenen Szenarien, in denen diese Annahmen nicht gelten. In den folgenden Abschnitten wird sich zeigen, dass der Grundsatz des erzwungenen Vertrauens die Realisierung von Sicherheitsdiensten in vielen Punkten vereinfacht. Erst durch diesen Grundsatz wird es möglich, eine umfassende Sicherheitsarchitektur überhaupt anzustreben.

4.1.1 Agentensystem als „Prokurist" des Mobilen Agenten

Durch das erzwungene Vertrauen durch Einbettungsbeziehung ergeben sich zwangsweise folgende uneingeschränkten Vertrauensbeziehungen:

- Das Agentensystem vertraut dem Endsystem, von dem es ausgeführt wird.
- der Mobile Agent vertraut dem Agentensystem, von dem er ausgeführt wird.

Handlungsvollmacht des MA für das AS

Diese erzwungene Vertrauensbeziehung kann und muss der Mobile Agent nutzen, um seine eigenen Sicherheitsanforderungen überhaupt durchsetzen zu können. Der Mobile Agent nutzt das Agentensystem als seinen „Prokuristen". Mit seiner Ankunft auf einem Agentensystem erteilt der Mobile Agent diesem implizit Handlungsvollmacht, d.h. das Agentensystem darf stellvertretend für den Mobilen Agenten handeln.

Wie im vorhergehenden Abschnitt gezeigt, kann ein Mobiler Agent nach einer Migration zu einem Agentensystem A nicht zweifelsfrei entscheiden, ob er sich auf dem gewünschten Agentensystem befindet. Falls er auf ein böswilliges Agentensystem M umgeleitet worden wäre, könnte M ihm die Identität von A vorspielen. Wegen der Einbettungsbeziehung ist der Mobile Agent nicht in der Lage dies zu erkennen. Diese Problematik lässt sich nur mit Hilfe des Quell–Agentensystems lösen. Der Mobile Agent darf das Quell–Agentensystem erst verlassen, wenn zum Ziel–Agentensystem ein Vertrauensverhältnis aufgebaut werden konnte. Dazu muss der Mobile Agent das **AS als Stellvertreter** benutzen, um dieses Vertrauen aufzubauen. D.h. er kann das Agentensystem, auf dem er sich befindet und dem er (zu einem gewissen Grad) vertraut, nutzen, um den Vertrauenslevel, dem er dem Ziel–AS entgegenzubringen bereit ist, zu berechnen. Danach muss der Mobile Agent sich darauf verlassen, dass das Quell–Agentensystem ihn sicher, d.h. ohne die Möglichkeit für einen Angreifer den Mobilen Agenten umzuleiten, zum Ziel–Agentensystem überträgt. Er bewegt sich damit nur auf Agentensysteme, die einen bestimmten Vertrauenslevel besitzen.

4.1.2 Trust Level Management

Wie im vorhergehenden Abschnitt gezeigt, ist für den Mobilen Agenten der Trust Level eines zu besuchenden Agentensystems von entscheidender Bedeutung. In diesem Abschnitt wird das Trust Level Management, dass sich mit der Klassifizierung und dem Aufbau einer Vertrauensbeziehung befasst, näher betrachtet.

Definition von Vertrauen

Trust bzw. **Vertrauen** ist, wie bereits in Abschnitt 3.2.2 erwähnt, eine nicht symmetrische (d.h. unidirektionale) Beziehung zwischen genau zwei Entitäten [AbHa 97]. D.h. das Vertrauen, das Alice Bob entgegenbringt, muss nicht dasselbe sein, das Bob Alice gewährt. Der **Vertrauenslevel (Level of Trust)**, d.h. die „Maßzahl“ des Vertrauens, wird bestimmt durch einen (numerischen) Vertrauenswert, der angibt, wie stark das Vertrauen zur entsprechenden Entität ist.

Transitivität des Vertrauens

Bei allen Überlegungen zum Trust Level Management muss die Transitivität betrachtet werden. Bei uneingeschränktem transitivem Vertrauen gilt, wenn Alice Bob vertraut und Bob wiederum vertraut Carl, dann vertraut auch Alice Carl. Uneingeschränkte Transitivität ist nur sehr schwer zu kontrollieren und für domänenübergreifendes Management nicht einsetzbar. Aus diesem Grund wird die bedingte Transitivität favorisiert. Bei der **bedingten Transitivität** muss der Vertrauenslevel explizit kommuniziert werden. Falls Alice Vertrauen zu Carl aufbauen wollte, würde dies im angegebenen Beispiel folgendermaßen geschehen:

Bedingte Transitivität mit Hilfe von Bürgen

1. Bob muss seinen Vertrauenslevel für Carl als „Empfehlung“ an Alice kommunizieren.
2. Alice vertraut Bob als **Bürge**. Sie kann die Qualität von Bob's Empfehlungen, abhängig von seinem Vertrauenslevel und Alice's eigener Policy, bewerten.
3. Abhängig von der Bewertung der Empfehlung kann Alice Carl einen Vertrauenslevel zuordnen.

Kategorien des Vertrauens

Vertrauenswerte

Beim Trust Level Management gibt es folglich zwei Kategorien von Vertrauen: **unmittelbares** und **mittelbares Vertrauen**, für die, analog zu [AbHa 97], sechs Vertrauenswerte von -1 bis 4 definiert sind (vgl. Tabelle 4.2).

Wert	Bedeutung	Erläuterung
-1	Misstrauen	Nicht vertrauenswürdig
0	Unkenntnis	Entscheidung über Vertrauenslevel nicht möglich
1	Minimal	Niedrigster Vertrauenslevel
2	Durchschnittlich	Durchschnittlicher Vertrauenslevel
3	Hoch	Höherer Vertrauenslevel als die meisten Entitäten
4	Vollständig	Höchster Vertrauenslevel

Tabelle 4.2: Diskrete Vertrauenswerte

Jeder Entität wird damit eine **Reputation** zugewiesen, die wie folgt definiert ist:

$$\mathit{Reputation} := \{\text{Identität, Kategorie, Vertrauenslevel}\}$$

Die Reputation beinhaltet die Identität derjenigen Entität, für die der Vertrauenslevel gilt. Unter Kategorie wird angegeben, für welchen Bereich das Vertrauen gilt. Der Vertrauenslevel enthält einen Wert aus Tabelle 4.2.

Berechnung mittelbaren Vertrauens

Um mittelbares Vertrauen aufzubauen, bedarf es eines Protokolls und eines Algorithmus, um dem Vertrauenswert für eine Zielentität zu berechnen. Der Initiator sendet an einen Teil oder an alle Entitäten, die einen bestimmten Vertrauenslevel in der Kategorie „Bürge“ besitzen, eine Anfrage (`RecommendationRequest (RRQ)`) mit der Identität des Ziels und der Kategorie, für die das Vertrauen gelten soll. Jeder Bürge kann den RRQ entweder direkt mit einer Empfehlung (`RecommendationReply (RR)`) beantworten oder selbst wieder auf eigene Bürgen zurückgreifen. Die Empfehlung (RR) besteht aus der ID des Bürgen, der Reputation, die der Bürge der Entität für die entsprechende Kategorie entgegenbringt, und einem Gültigkeitszeitraum.

Der Initiator empfängt u.U. über mehrere Pfade RRs und berechnet für jeden Pfad das Vertrauen des Ziels. Abdul–Rahman und Hailes [AbHa 97] schlagen für diese Berechnung folgende Formel vor:

$$tv_p(Z) = \frac{tv(B_1)}{4} \cdot \frac{tv(B_2)}{4} \cdots \frac{tv(B_N)}{4} \cdot rtv(Z)$$

Dabei bezeichnet $tv_p(Z)$ den Vertrauenslevel für das Ziel Z, der aus Empfehlungen, die über den Pfad p geliefert wurden, berechnet wird. Mit $tv(B_i)$ wird der Trust Level der Entität B_i, mit $rtv(Z)$ wird der mittelbare, d.h. empfohlene Vertrauenslevel für Z (Recommended Trust Level) bezeichnet.

Die Werte $tv(B_i)$ können zwischen -1 und 4 liegen. Empfehlungen von Bürgen mit höchstem Vertrauenslevel (4) werden unverändert übernommen. Die Empfehlungen von Bürgen mit niedrigerem Vertrauenslevel werden durch den Faktor $tv(B_i)/4$ relativiert. Je länger der Pfad und je niedriger der Vertrauenslevel eines Bürgen, umso kleiner wird der berechnete Vertrauenslevel des Ziels. Falls sich im Pfad ein Bürge mit Vertrauenslevel 0 (Unkenntnis) oder ein Bürge, dem misstraut wird (Vertrauenswert -1), befindet, ist der resultierende Vertrauenslevel ebenfalls 0 bzw. negativ. Die Werte, die aus den unterschiedlichen Pfaden berechnet werden, werden gemittelt.

Beispiel zur Berechnung des Vertrauens-levels

Ein Mobiler Agent, der auf einem Agentensystem gestartet wurde und migrieren möchte, muss das Agentensystem nutzen, um den Vertrauenslevel des Ziel–Agentensystems zu ermitteln (vgl. Abb. 4.1). Zur Verdeutlichung des Konzeptes sei im folgenden Beispiel angenommen, dass der Mobile Agent seinem Quell–Agentensystem einen Vertrauenswert von 3, in allen Kategorien, entgegenbringt. Vor der Migration fragt der Mobile Agent sein Agentensystem mit einem `RRQ`, für die Kategorie Migration und die Entität des Ziel–AS, nach einer Empfehlung. Weiterhin sei angenommen, dass das Quell–AS bisher keine Vertrauensbeziehung zum Ziel–AS aufgebaut hat und selbst

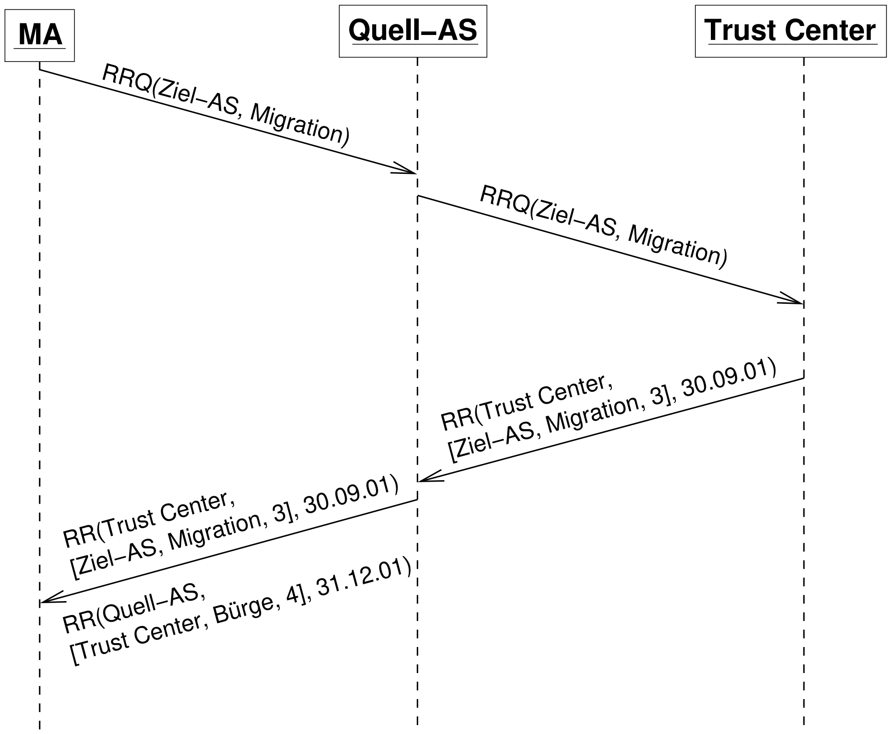


Abbildung 4.1: Berechnung eines Vertrauenslevels (Sequenzdiagramm)

einen weiteren Bürgen, z.B. ein Trust Center, um eine Empfehlung bittet. Diesem Trust Center vertraut das Quell–AS in Bezug auf Empfehlungen vollständig (Trust Level 4). Das Trust Center gibt für das Ziel–AS eine Empfehlung (`RR`) für die Kategorie Migration ab, die den Vertrauenswert 3 enthält. Das Quell–AS gibt seinen Trust Level von 4 für das Trust Center und den `RR` des Trust Centers an den Mobilen Agenten weiter. Daraus berechnet der Mobile Agent einen resultierenden Vertrauenslevel von

$$tv(\text{Ziel–AS}) = \frac{3}{4} \cdot \frac{4}{4} \cdot 3 = 2,25$$

In diesem Beispiel wird bereits die Schwäche der Formel deutlich. Obwohl alle beteiligten Entitäten einen Vertrauenslevel ≥ 3 besitzen, ist der resultierende Vertrauenslevel kleiner als 3. Würde die Formel unverändert übernommen und würde der Mobile Agent vor jeder Migration diese Formel anwenden, würde der Vertrauenswert immer mehr abnehmen und nach einigen Migrationsschritten wäre der Vertrauenswert des nächsten Ziel–AS nahe Null. Um dies zu verhindern, wird die Formel wie folgt um eine obere Gaußklammer erweitert:

$$tv_p(Z) = \left\lceil \frac{tv(B_1)}{4} \cdot \frac{tv(B_2)}{4} \cdots \frac{tv(B_N)}{4} \cdot rtv(Z) \right\rceil$$

Die Veränderung sorgt dafür, dass der Vertrauenslevel nicht kontinuierlich abnimmt.

4.1.3 Schlussfolgerungen für die Sicherheitsarchitektur

Die vollständige Kontrolle des Executor über den Executee ist systemimmanent. Nachdem der Executee sich der Kontrolle nicht entziehen und diese auch nicht verhindern kann, muss versucht werden, diese Abhängigkeit konstruktiv zu nutzen.
Der **Grundsatz des erzwungenen Vertrauens durch Einbettung** besagt, dass der Mobile Agent (als Executee) dem Agentensystem (als Executor), auf dem er abläuft, „ausgeliefert" ist. Insofern muss die Sicherheitsarchitektur Mechanismen zum **Trust Level Management** zur Verfügung stellen.

Der Mobile Agent hat zu seinem Heimat–Agentensystem, d.h. zu dem Agentensystem, auf dem er erstmals gestartet wurde, vollstes Vertrauen und kann dieses als Basis für die Berechnung des Vertrauenslevels zu einem Ziel–Agentensystem verwenden, das er dann wiederum zur Berechnung des Vertrauenslevel für das nächste Ziel verwendet. Das Agentensystem muss stellvertretend für den Mobilen Agenten den Trust Level bestimmen und über eine Schnittstelle dem Mobilen Agenten zur Verfügung stellen. Dabei ist sicherzustellen, dass nur **bedingte Transitivität** bei den Vertrauensbeziehungen gelten darf. Uneingeschränkte Vertrauensbeziehungen sind, insbesondere in domänen– und organisationsübergreifenden Systemen, ein Sicherheitsrisiko.

4.2 Sicherheitskonzepte für das Entitätenmodell

Entitätenmodell muss um Rollen erweitert werden

Das Entitätenmodell aus Abschnitt 2.4.1, das Personen nur als Entität Anwender modelliert, ist für klassische Managementszenarien nicht ausreichend. Es zeigt sich, dass in der Praxis verschiedene, abgestufte Rollen– und Gruppenkonzepte erforderlich sind. Auch im Hinblick auf das abstrakte Entitätenmodell lässt sich die Entität Anwender in verschiedenen Rollen wie z.B. Benutzer, Implementierer und Authority aufteilen. Dabei wird diejenige Person, die für eine andere Entität verantwortlich ist, Authority genannt (vgl. auch Abschnitt 2.3.1). Diejenige Person, die die Software–Komponenten entwickelt und implementiert, wird als Implementierer bezeichnet. Derjenige, der eine Software–Komponente erstmalig instantiiert und startet, heißt **Besitzer**. Oftmals fallen die Entitäten Besitzer und Authority zusammen. Benutzer nennt man Personen, die (Software–) Entitäten für ihre Zwecke nutzen. Die Gruppe der Benutzer lässt sich in einem Managementsystem weiter unterteilen, z.B. in Sicherheits–, Netzkomponenten–, System–, Serviceadministrator, Verantwortlicher für die Software–Verteilung, usw. (vgl. auch rollenbasierte Zugriffskontrollkonzepte in Abschnitt 4.4.1).

Im Folgenden wird das Sicherheitsmodell und die notwendigen Sicherheitsdienste, die sich aus dem Entitäten– und dem erweiterten Rollenmodell ergeben, näher betrachtet.

4.2.1 Trennung zwischen Gattung und Instanz

Für Entitäten, die überwiegend durch Software–Komponenten realisiert werden und die in der Lage sind sich im Netz zu bewegen (MA), ist eine Trennung des Entitätenbegriffes in Gattung und Instanz erforderlich. Diese Unterscheidung wurde zum ersten Mal in [Roel 99a] getroffen. Sie kann in Analogie zu der Trennung von Klassen und Objekten in der Welt der Objektorientierten Programmierung betrachtet werden.

Gattung: statische Teile der Entität

Instanz: im Ablauf befindliche Entität

Unter einer **Gattung**, z.B. unter einer Agentengattung, werden alle statischen Teile des Agenten verstanden, die für alle Agenteninstanzen innerhalb einer Gattung gleich sind. Primär ist dies der Programmcode, der für alle, sich im Ablauf befindlichen, Agenteninstanzen gleich ist. Demgegenüber versteht man unter einer **Instanz**, z.B. unter der Agenteninstanz, die konkrete Ausprägung eines sich im Ablauf befindlichen Agenten mit allen seinen Attributbelegungen, seinen variablen und transportierten Daten, seiner Historie, usw.

Am Beispiel des Agenten A bedeutet dies, dass zum Start des Agenten zuerst die Agentengattung von A, d.h. der Programmcode, der A implementiert, ausgewählt werden muss. Unter Angabe von Startparametern wird eine neue Agenteninstanz erzeugt, die dann auf dem Agentensystem zur Ausführung gebracht wird. Wird mit anderen, oder auch denselben Parametern, eine zweite A Instanz gestartet, so gehören beide Agenten einer gemeinsamen Gattung an, stellen aber unterschiedliche Agenteninstanzen dar.

Die Unterscheidung in Gattung und Instanz ist sowohl im Hinblick auf Rollen, als auch auf Rechte– und sonstige Sicherheitskonzepte sinnvoll. Die Trennung spiegelt sich beispielsweise in der Trennung zwischen Implementierer und Benutzer wider. Der Implementierer der Agentengattung ist i.d.R. nicht identisch mit dem späteren Benutzer der Agenteninstanz. Da beide Rollen unterschiedliche Teile des Lebenszyklus einer Entität berühren, ist die Aufteilung für Sicherheitsbetrachtungen notwendig. Diejenige Entität, die die Agenteninstanz das erste mal instantiiert, wird im Folgenden als Besitzer (Owner) der Agenteninstanz bezeichnet.

Betrachtet man den Zustandsübergang eines Mobilen Agenten von *migrating* nach *migrated* (vgl. Abschnitt 2.5) genauer, insbesondere für Mobile Agenten, die nacheinander mehrere Agentensysteme besuchen und dabei diesen Zustandsübergang mehrmals durchlaufen, wird die Notwendigkeit des Gattungs– und Instanzenbegriffs nochmals verdeutlicht.

Im Zustand *migrating* wird der Mobile Agent mit seinem momentanen Zustand in eine Protokoll–Dateneinheit eingebettet. Aus Effizienzgründen, d.h. um die PDE möglichst klein zu halten, werden hier häufig nur die variablen Daten und ein Bezeichner des Mobilen Agenten, nicht jedoch dessen Pro-

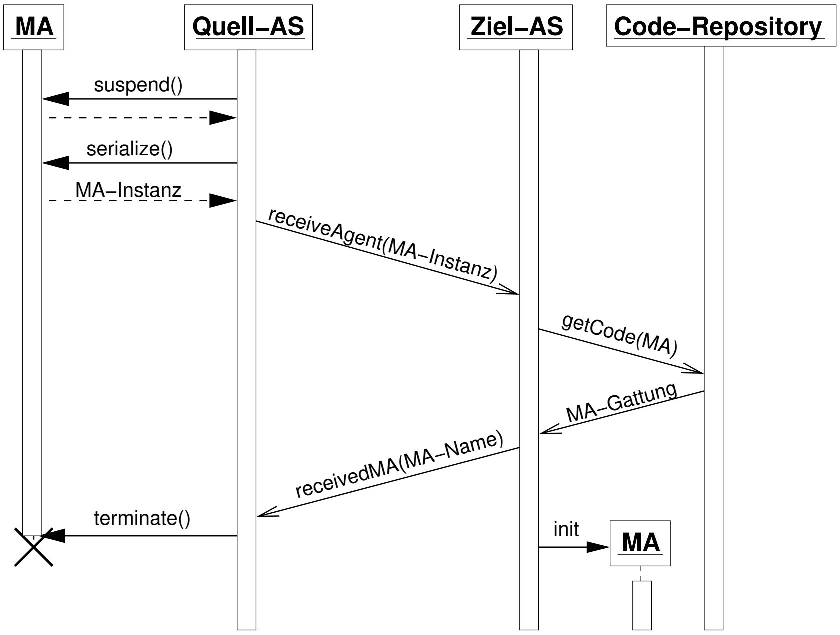


Abbildung 4.2: Migration eines Mobilen Agenten (Sequenzdiagramm)

Serialisierung: Vorbereitung der MA–Instanz zur Übertragung

grammcode in die PDE eingebettet. Dieser Vorgang wird auch als **Serialisierung** bezeichnet. Die resultierende PDE wird auf das Zielsystem übertragen. Dort angekommen, befindet sich der Mobile Agent (logisch) im Zustand *migrated*. Das Ziel–AS muss den Agenten initialisieren und starten. In einem ersten Schritt werden dazu die Daten aus der PDE wiederhergestellt, d.h. **deserialisiert**. Das Agentensystem kann über den Bezeichner erkennen, um welchen Agenten es sich handelt. Da nur die variablen Daten übertragen wurden, muss das Agentensystem dann den ausführbaren Programmcode in einem zweiten Schritt aus einem lokalen oder naheliegenden Code Repository laden, diesen Programmcode mit den deserialisierten Datenwerten belegen und den Mobilen Agenten starten. Der Ablauf der Migration wird im Sequenzdiagramm in Abbildung 4.2 noch einmal verdeutlicht. Auf die Fehlerfälle bzw. die Ausnahmebehandlung wird in dieser Abbildung, aus Gründen der Übersichtlichkeit, verzichtet.

Instanz und Gattung können aus unterschiedlichen Quellen stammen

Da die statischen Teile des Agenten, d.h. die Agentengattung, und die dynamischen Teile, d.h. die Agenteninstanz, über unterschiedliche Wege und von unterschiedlichen Quellen zum Agentensystem kommen können, sind diese Teile auch unabhängig voneinander zu sichern. Zur Verdeutlichung sei hier nur ein möglicher Angriff angegeben. Falls die Integrität und die Authentizität der Agentengattung nicht gewährleistet werden kann, könnte ein Angreifer im Schritt 2 dem Agentensystem ein Trojanisches Pferd anstelle der MA–Gattung unterschieben, das dann mit den gültigen variablen Daten initialisiert wird und diese an den Angreifer übermitteln kann.

Die Unterscheidung in Gattung und Instanz ist für Mobile Agenten zwingend erforderlich und unterscheidet sie von Mobilem Code. Mobiler Code wird nur zwischen zwei Systemen bewegt und beinhaltet keine variablen Daten (vgl. auch Abschnitt 2.1.3). Nur ein Mobiler Agent „transportiert“ Daten, die er auf seiner Reise berechnet hat. Und insbesondere diese Daten müssen auf der Reise geeignet geschützt werden können. Selbst bei einem Single–Hop Agenten, der sehr viel Ähnlichkeiten mit Mobilem Code hat, ist die Trennung in Gattung und Instanz erforderlich. Denn im Gegensatz zu Mobilem Code kann der Mobile Agent auf seinem Quellsystem vor der Migration bereits variable Daten berechnen und diese mit zum Zielsystem nehmen.

4.2.2 Namens– und Identifikationsschema

Für nahezu alle Sicherheitsdienste und für alle Managementoperationen ist es erforderlich, einen Verantwortlichen (Authority) bestimmen zu können. Insofern kann der Authentisierungsdienst als Basisdienst für viele andere Sicherheitsdienste betrachtet werden.

Auch um Angriffe auf Entitäten (vgl. Abschnitt 2.6.2) erkennen bzw. verhindern zu können, ist es erforderlich, die Identität der Entitäten zweifelsfrei zu bestimmen. Dazu ist es notwendig, ein eindeutiges Namensschema zu etablieren, damit alle Entitäten benannt und in einem weiteren Schritt auch identifiziert werden können.

eindeutiges Namensschema erforderlich

In der Tabelle 4.3 werden für alle Entitäten die Namen angegeben, mit deren Hilfe sie sich identifizieren lassen. Alle angegebenen Namen müssen global eindeutig sein. Dies lässt sich durch ein hierarchisches Namenskonzept relativ einfach erreichen.

Entität		Identifizierung durch
Agenten	Gattung	Eindeutige Bezeichnung des Quellcode–Paketes (z.B. Java Package)
	Instanz	Eindeutiger Name, MASIF Namensschema mit Erweiterung
Agentensystem		Eindeutiger Name, MASIF Namensschema
Endsystem		Adresse oder Full Qualified Host Name
Benutzer	Implementierer, Benutzer, Administrator, ...	Eindeutiger Name und/oder Rolle bzw. Gruppenzugehörigkeit (Distinguished Name, z.B. X.500)

Tabelle 4.3: Identifikatoren für Entitäten

Für Agentensysteme und MA–Instanzen kann das in [MASIF] vorgeschlagene Namenskonzept verwendet werden. In MASIF ist ein Name als Struktur definiert, die aus den drei Komponenten Authority, Identity und Agent System Typebesteht. Diese drei Teile bilden den **Compound Name**. Dieser Name wird sowohl für Agentensysteme als auch für Agenten verwendet. Abbildung

MASIF Namensschema für MA und AS

4.3 fasst die formale Definition eines MASIF–Namens, in Form einer **IDL**–Spezifikation (**Interface Definition Language**), zusammen.

```
typedef short AgentSystemType;

typedef sequence<octet> OctetString;
struct ClassName{
   string           name;
   OctetString      discriminator;
};
typedef OctetString Authority;
typedef OctetString Identity;
struct Name{
    Authority        authority;
    Identity         identity;
    AgentSystemType  agent_system_type;
}
```

Abbildung 4.3: MASIF Namensschema (IDL)

Die **Authority** definiert die für das Agentensystem bzw. den MA verantwortliche Person oder Organisation. Die **Identity** bezeichnet den Namen des Agentensystems bzw. des Agenten. Sowohl Authority als auch Identity können selbst wieder strukturierte Namen sein. Der **Agent System Type** beschreibt, welches Agentensystem zugrunde liegt bzw. für welchen Typ Agentensystem die Agenten entwickelt wurden. Für einige Agentensystemimplementierungen wurde von der OMG bereits ein Agent System Type definiert. So könnte bspw. ein Agentensystem mit dem AS–Type „4“, das vom Münchner Netzmanagement Team (mnm) verwaltet wird und auf dem Rechner „hpheger3“ läuft, wie folgt benannt werden: „[mnm!4!hpheger3]“. Ein Agent A, für denselben AS–Type entwickelt, vom Benutzer *user*, der auf diesem Agentensystem läuft, hieße entsprechend: „[mnm!4!hpheger3!user!4!A]“ Ein Mobiler Agent ändert also nach einer Migration seinen Namen. Für die Identity von Agenten wird in MASIF der Bezeichner der (Haupt–) Klasse, in der der Agent implementiert wird, verwendet. Dieser Klassenname besteht aus einer Zeichenkette. Die MASIF Spezifikation sagt explizit, dass keinerlei Konzepte festgelegt wurden, um Klassennamen global eindeutig zu machen. Dieser Problemfall tritt dann auf, wenn mehrere Agenten mit „demselben (Klassen–) Namen“, d.h. derselben Gattung, auf demselben Agentensystem zur Ausführung gebracht werden. Die globale Eindeutigkeit von Instanznamen muss gewährleistet werden, wenn mehrere Instanzen derselben Agentengattung gleichzeitig auf einem Agentensystem ablaufen sollen, denn dann muss deren Name unterscheidbar sein. Dies lässt sich z.B. durch einen globalen Zähler, einen Hash–Wert oder Uhrinformationen, die mit dem (Klassen–) Namen konkateniert werden, erreichen.

MA ändert nach Migration den Namen

Eine Agentengattung wird über den Namen der Implementierungsklasse angesprochen. Bei Java als Implementierungssprache wäre dies bspw. der Bezeichner des Package und der Name der Agenten(haupt)klasse. Der Package–Bezeichner selbst stellt auch einen hierarchischen Namen dar.

Benutzer lassen sich über ihren Namen identifizieren. In vielen Fällen, z.B. bei der Zugriffskontrolle, ist nicht der Name erforderlich, sondern es werden Rechte an eine Gruppen– oder Rollenzugehörigkeit geknüpft. Für die Benennung von Benutzern mit ihren Gruppen– und Rollenzugehörigkeiten eignet sich ein **Distinguished Name (DN)** wie er z.B. in [ITU- 93] vorgegeben wird.

4.2.3 Authentisierung auf Basis von Zertifikaten

Die Authentisierung dient dazu, zweifelsfrei die Identität einer Entität zu bestimmen. Im Grunde sollen der oder die für die Entität Verantwortlichen (Authority) bestimmt werden. Für den eigentlichen Vorgang der Authentisierung sind die authentisierende Entität, auch als Verifier bezeichnet, und die authentisierte Entität, d.h. der Principal, zu unterscheiden. Bei der Authentisierung selbst gibt es die lokale Authentisierung, d.h. Verifier und Principal befinden sich auf demselben Rechensystem, sowie die entfernte Authentisierung, bei der Verifier und Principal nur über eine Kommunikationsrelation miteinander in Beziehung stehen. Um eine erfolgreiche Authentisierung durchführen zu können, muss eine der folgenden Prämissen erfüllt sein:

Authentisierung: lokal oder entfernt

1. Der Principal bzw. dessen Identitätsdaten müssen vom Verifier direkt inspiziert werden können, oder
2. eine andere vertrauenswürdige Instanz, die selbst authentisiert wurde, fungiert als Stellvertreter für den Verifier, um die Authentisierung durchzuführen.

Falls eine Entität stellvertretend für eine Authority handelt, tritt die Entität als Subjekt im System auf, um andere Entitäten zu beauftragen oder bestimmte Managementoperationen auszuführen. Modelliert man diesen Sachverhalt mittels UML [RJB 98] so ergibt sich das in Abbildung 4.4 angegebene Klassendiagramm.

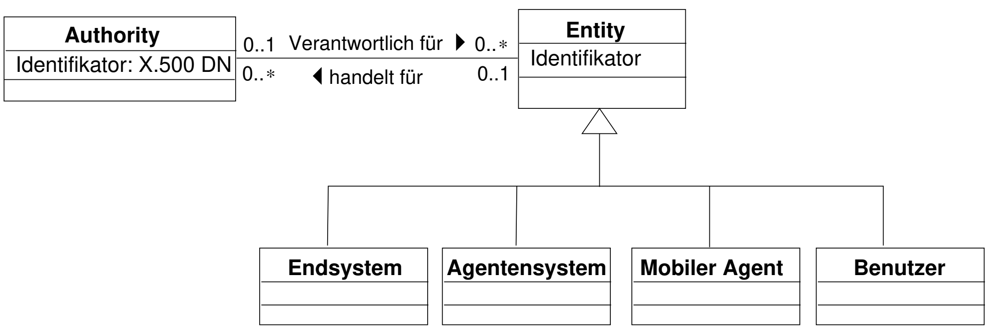


Abbildung 4.4: Principal und Entitäten (UML Klassendiagramm)

Eine Authority ist verantwortlich für mehrere Entitäten und die Entität kann stellvertretend für mehrere Authorities handeln. Eine Entität ist gekennzeichnet durch einen eindeutigen Identifikator (Namen). Jede konkrete Entität (Endsystem, Agentensystem, Mobiler Agent und Benutzer) ist eine Spezialisierung der Superklasse Entity und erbt von dieser Superklasse das Identifikator Attribut. In Tabelle 4.3 sind konkrete Ausprägungen für diese Identifikatoren angegeben.

Die Vergabe von eindeutigen Identifikatoren ist der erste Schritt. In einem weiteren Schritt muss sichergestellt werden, dass für jeden Principal sein Name zweifelsfrei mit der Entität verbunden werden kann. Kann dies nicht sichergestellt werden, so ist der Angriff gegen eine Entität in Form von Maskerade sehr einfach möglich.

zweifelsfreie Bindung des Namens an die Entität

Diese Bindung des Namens lässt sich durch digitale Zertifikate sicherstellen. Der Principal erhält ein Schlüsselpaar für ein asymmetrisches Verschlüsselungsverfahren, d.h. einen öffentlichen und einen privaten Schlüssel. Eine Certification Authority (CA), oft auch als Trust Center bezeichnet, erzeugt eine Datenstruktur mit dem Namen des Principal und mit dessen öffentlichem Schlüssel. Diese Datenstruktur wird dann mit dem privaten Schlüssel der CA digital signiert. Die CA fungiert dabei in der Rolle eines „Notars“. Durch die digitale Signatur wird der Name untrennbar mit dem öffentlichen Schlüssel verbunden. Das Ergebnis dieses Prozesses wird als **Zertifikat** bezeichnet (vgl. Abbildung 4.5). In ein Zertifikat werden auch noch weitere Daten, z.B. der Verwendungszweck des Zertifikates u.ä., mit aufgenommen. Die Verwendung von Zertifikaten ist an folgende Bedingungen geknüpft:

Zertifikat = Name und öffentlicher Schlüssel von einer CA signiert

Bedingungen an Zertifikate

Bedingung 1: Der Zertifizierte ist technisch in der Lage, seinen privaten Schlüssel geheim zu halten, und es ist sichergestellt, dass er den Schlüssel nicht an Dritte weitergibt.

Bedingung 2: Ein Zertifikat kann nur maximal so lange gelten, wie sich die zertifizierten Daten nicht ändern. Ändert sich bspw. der Distinguished Name eines Zertifizierten, muss dieser ein neues Zertifikat erhalten.

Bedingung 3: Es muss sichergestellt sein, dass Zertifikate widerrufen werden können.

Bedingung 4: Widerrufene Zertifikate müssen zu einem späteren Zeitpunkt zugänglich sein. Nur dann lassen sich vor dem Widerruf getätigte digitale Signaturen noch verifizieren.

Das Zertifikat wird nach der festgelegten Zeitspanne ungültig und muss dann entweder verlängert oder neu ausgestellt werden. Dieses Konzept lässt sich im Grundsatz für alle Entitäten anwenden, nicht nur auf Personen.

Wenn die Authentisierung auf digitale Signaturen abgestützt wird, so muss die Sicherheitsarchitektur eine oder mehrere CAs, bzw. Trust Center beinhalten. Im nächsten Abschnitt werden die Aufgaben, die Strukturen einer CA–

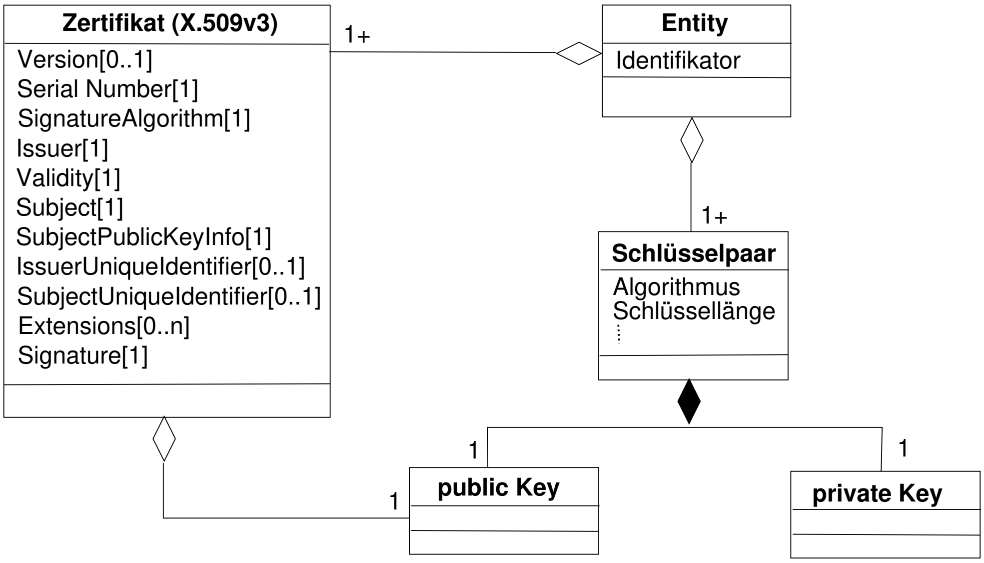


Abbildung 4.5: Zertifikate: Bindung des Namens an die Entität

Infrastruktur und der allgemeine Ablauf der Zertifizierung beschrieben. In den folgenden Abschnitten wird dann evaluiert, wie die einzelnen Entitäten aus dem Entitätenmodell zertifiziert und authentisiert werden können. Dabei wird auch untersucht, ob und inwieweit die verschiedenen Entitäten die Einhaltung der Bedingungen an Zertifikate gewährleisten können.

Aufgaben und Aufbau einer Certification Authority; Ablauf der Zertifizierung

Eine **Certification Authority** stellt einen „**vertrauenswürdigen Dritten**" (**Trusted Third Party**, **TTP**) dar. Das Vertrauensverhältnis, das der CA entgegengebracht wird, gilt für einen bestimmten Benutzerkreis, der auch als **Realm** bezeichnet wird. Bei allen Mitgliedern dieser Realm genießt die CA so hohes Vertrauen, dass die Aussagen der CA von allen Mitgliedern als gültig, richtig und wahr angenommen werden können. Eine mögliche Aussage der CA ist z.B.: „Dieser öffentliche Schlüssel gehört der Entität X."

Aufgaben einer CA

Die Aufgaben der CA bestehen im Wesentlichen aus folgenden Punkten:

- Generierung von Zertifikaten (**Certification Issue**): Die CA muss die Datenstrukturen für die Zertifikate erzeugen und diese mit ihrem privaten Schlüssel digital signieren.
- Speicherung der Zertifikate (**Certification Repository**): Zertifikate sind in einem allgemein zugänglichen Verzeichnisdienst zu speichern und die Zertifikate an beliebige Benutzer zu verteilen, damit diese die Bindungen von öffentlichen Schlüsseln an Zertifizierte überprüfen können.
- Widerruf und Sperrung von Zertifikaten (**Certification Revocation**): Zertifikate sind für einen bestimmten Zeitraum ausgestellt und während dieser Zeit grundsätzlich gültig. Falls jedoch ein privater Schlüssel ge-

brochen wurde oder abhanden gekommen ist, muss das entsprechende Zertifikat widerrufen werden können. Durch einen Certification Revocation Dienst einer CA lässt sich die oben genannte Bedingung 3 erfüllen.
- Schlüssel Update (**Certification Update**): Vor dem Ablauf eines Zertifikates muss die CA dafür sorgen, dass der Zertifizierte neue Schlüssel und ein neues Zertifikat erhält.
- Historienverwaltung (**Certification History**): Um die Bedingung 4 zu erfüllen, muss eine CA abgelaufene und widerrufene Zertifikate auch nach Ablauf der Gültigkeit speichern und allgemein zugänglich halten.
- **Schlüsselgenerierung**: Die CA kann stellvertretend für Entitäten Schlüsselmaterial erzeugen. Falls dieser optionale Dienst zur Verfügung gestellt wird, ist darauf zu achten, dass die privaten Schlüssel auf sichere Art und Weise und nur an den entsprechenden Benutzer übergeben werden.
- Zeitstempeldienst (**Time Stamping**): Mit dem Zeitstempeldienst bindet die CA einen bestimmten Sachverhalt an einen Zeitpunkt. Z.B. wird ein Dokument zu einem bestimmten Zeitpunkt von der CA digital signiert und damit der Zustand des Dokumentes zu diesem Zeitpunkt festgeschrieben.
- Notardienst (**Notarization**): Die CA tritt in einer Notariatsfunktion auf, wenn sie bestimmte Vorgänge zwischen Partner (z.B. das Zustandekommen eines SLA) beglaubigt. Diese Beglaubigung muss dabei auch von anderen als den unmittelbar am Vorgang beteiligten Entitäten verifizierbar sein.
- Domänen–übergreifende Zertifizierung (**Cross Certification**): Eine CA ist grundsätzlich für eine bestimmte Realm verantwortlich. Soll ein Vertrauensverhältnis zu einer anderen Domäne aufgebaut werden können, so muss die eigene CA das Zertifikat der fremden CA digital signieren; d.h. die eigene CA stellt ein Zertifikat für die fremde CA aus. Durch dieses Cross Zertifikat wird das Vertrauensverhältnis auf die fremde Realm ausgedehnt.
- Attribut Zertifikate (**Attribute Certificate**): Eine CA kann zusätzlich zu den normalen Zertifikaten, bei denen ein öffentlicher Schlüssel mit einer Identität verknüpft wird, auch andere Attribute (z.B. Rechte, Vollmachten, usw.) mit einem Schlüssel und/oder mit einer Identität verknüpfen.

Die Hauptaufgabe einer CA ist die digitale Signatur von bestimmten Datenstrukturen (z.B. Zertifikaten, Sperrlisten, Zeitstempel, u.ä), um deren Richtigkeit und Gültigkeit zu zertifizieren. Dazu werden kryptographische Verfahren verwendet. Um Verschlüsselungs– und Signaturverfahren präzise beschreiben zu können, wird die in Tabelle 4.4 angegebene Notation verwendet.

Ablauf der Zertifizierung

Im Folgenden wird exemplarisch der Ablauf einer Benutzerzertifizierung dargestellt.

1. Schlüsselgenerierung

Bei einem Benutzerzertifikat wird i.d.R. ein Schlüsselpaar für ein asymmetrisches Verschlüsselungsverfahren benötigt. Die Generierung der Schlüssel kann lokal beim Benutzer oder zentral bei der CA erfolgen. Es muss

A_p	öffentlicher Schlüssel von A (public key of A)
A_s	privater Schlüssel von A (secret key of A)
d, m, n	Nachrichten, Daten
$m \bullet n$	Konkatenation von m und n
$A_p[m]$	Verschlüsselung der Nachricht m mit dem öffentlichen Schlüssel von A. Es gilt $A_s[A_p[m]] = A_p[A_s[m]] = m$
$A_s[m]$ bzw. $A\{m\}$ sig_m^A	von A erstellte (digitale) Signatur der Nachricht m $sig_m^A = A\{m\}$
$H_m = H(m)$ $MAC_s(m) = H(m \bullet S)$	Hash–Wert H der Nachricht m MAC, HMAC (vgl. S. 120) der Nachricht m, gesichert mit dem Schlüssel S
Z_A	Datenstruktur für das Zertifikat von A
C_A^{CA}	von CA ausgestelltes Zertifikat für A
S	Schlüssel für ein symmetrisches Verschlüsselungsverfahren (secret key)
$S[m]$	Verschlüsselung der Nachricht m mit dem Schlüssel S
Alice $\longrightarrow$ *Bob* : m	Alice schickt die Nachricht m an Bob

Tabelle 4.4: Verschlüsselung und Signaturen: Notation

gewährleistet sein, dass ausreichend sichere Schlüssel erzeugt werden und dass der private Schlüssel nur dem Zertifizierten zugänglich gemacht wird. Die zweite Prämisse lässt sich am einfachsten erfüllen, wenn jeder Benutzer sein Schlüsselpaar selbst erzeugt und der private Schlüssel nie das System des Benutzers verlässt. Für die zentrale Erzeugung der Schlüssel durch die CA spricht, dass nicht für jeden Benutzer sichergestellt werden kann, dass er in der Lage ist, sichere Schlüssel zu erzeugen. Im Normalfall kann beim Benutzer z.B. kein Wissen über Schlüssellängen oder sichere Zufallsgeneratoren vorausgesetzt werden; dieses Wissen ist aber für die Generierung sicherer Schlüssel erforderlich.

2. Personalisierung

Nachdem der Benutzer A im Besitz eines öffentlichen Schlüssels ist, muss eine Personalisierung durchgeführt werden. Zweck der Personalisierung ist es, sicherzustellen, dass der öffentliche Schlüssel auch wirklich dem Zertifizierten A gehört. Gelingt es einem Angreifer M seinen eigenen öffentlichen Schlüssel unter dem Namen von A zertifizieren zu lassen, so kann M die Identität von A übernehmen (Maskerade).

Bei einer Person A wird zur Personalisierung i.d.R. verlangt, dass sie persönlich bei der CA erscheint, sich ausweist und belegt, dass sie im Besitz des zum öffentlichen Schlüssel A_p (public key of A) gehörenden privaten Schlüssels A_s (secret key of A) ist.

Dieser Nachweis kann durch ein dreistufiges Challenge–Response Verfahren erbracht werden. Die CA verschlüsselt eine Nachricht m mit dem zu zertifizierenden öffentlichen Schlüssel ($A_p[m]$) und schickt diese Nachricht an A ($CA \longrightarrow A : A_p[m]$). Im zweiten Schritt muss die verschlüsselte Nachricht vom Benutzer entschlüsselt ($A_s[A_p[m]] = m$) und in einer vorher vereinbarten Art und Weise verändert werden (z.B. $n = m + 17$). Im dritten Schritt wird die veränderte Nachricht mit dem öffentlichen Schlüssel der CA verschlüsselt ($CA_p[n]$) und an diese zurückgegeben ($A \longrightarrow CA : CA_p[n]$). Die CA entschlüsselt die Nachricht mit ihrem privaten Schlüssel ($CA_s[CA_p[n]]$) und überprüft, ob die ursprüngliche Nachricht in der erwarteten Art und Weise verändert wurde ($n \stackrel{?}{=} m + 17$). Denn nur derjenige, der im Besitz von A_s ist, kann die mit A_p verschlüsselte Nachricht entschlüsseln und damit auch verändern.

3. Generierung der Datenstruktur (X.509v3)

Die CA muss anschließend die Datenstruktur für das Zertifikat erzeugen und die Attribute mit den entsprechenden Werten von A belegen. Tabelle 4.5 fasst die Attribute eines **X.509 Zertifikates** zusammen [X.509].

version	1 bis 3 (falls dieses Attribut nicht gesetzt ist, wird Version 1 als Default angenommen)
serialNumber	für die CA eindeutige Seriennummer des Zertifikates
signatureAlgorithm	Bezeichnung des Algorithmus, der für die digitale Signatur verwendet wurde
issuer	Name (DN) der ausstellenden CA
validity	Gültigkeitsdauer des Zertifikates; angegeben in zwei Zeitpunkten für den Beginn (notBefore) und das Ende (notAfter) der Gültigkeitsdauer
subject	Gegenstand des Zertifikates, z.B. DN des Zertifizierten
subjectPublicKeyInfo	Algorithmus, Parameter und der öffentliche Schlüssel
issuerUniqueIdentifier	Eindeutiger Bezeichner der CA (ab Version 2; optional)
subjectUniqueIdentifier	Zusätzliche Informationen über den Gegenstand des Zertifikates (ab Version 2; optional)
extensions	Seit Version 3 können Zertifikate mit Einschränkungen, Bedingungen und Erweiterungen versehen werden.

Tabelle 4.5: Attribute eines X.509 Zertifikates

4. Digitale Signatur des Zertifikates

Im letzten Schritt muss die CA die mit dem Werten des Benutzers A belegte Datenstruktur Z_A digital signieren. Dazu wird ein kryptographischer Hash–Wert H (z.B. mit Hilfe des MD5 Algorithmus [Rive 92]) über Z_A berechnet ($H_{Z_A} = MD5(Z_A)$). Eine **kryptographische Hash–Funktion** bildet eine beliebig lange Zeichenkette Z_A auf eine Zeichenkette H mit fester Länge ab,

die in der Regel sehr viel kürzer ist als Z_A. Die Hash–Funktion wird als **kollisionsfrei** angenommen, d.h. falls $Z_A \neq Z'_A$ gilt immer $H(Z_A) \neq H(Z'_A)$ (weitere Informationen zu kryptographischen Hash–Funktionen finden sich z.B. in [Schn 96]).
Dieser Hash–Wert wird dann mit dem privaten Schlüssel der CA verschlüsselt ($sig_{Z_A} = CA_s[H_{Z_A}]$). Das Ergebnis dieser Verschlüsselung stellt die **digitale Signatur** dar. Das Zertifikat C_A^{CA} des Benutzers A erhält man durch die Konkatenation von Z_A mit sig_{Z_A} ($C_A^{CA} = Z_A \bullet sig_{Z_A}$ oder in verkürzter Schreibweise $C_A^{CA} = Z_A \bullet sig$) (vgl. auch Abbildung 4.6).

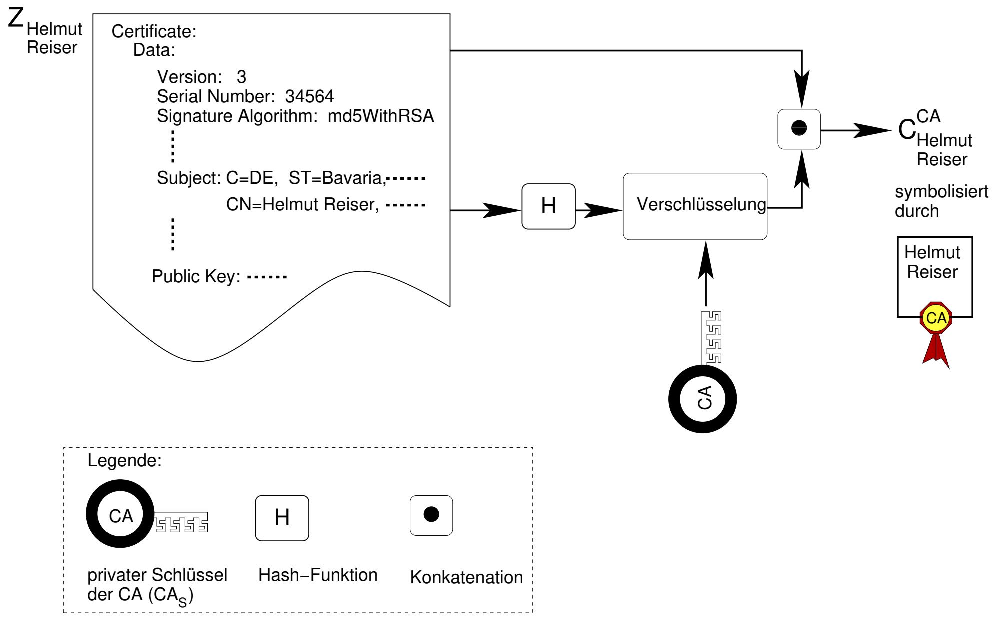


Abbildung 4.6: Erzeugung eines digitalen Zertifikates

Das folgende Beispiel zeigt ein Benutzerzertifikat für Helmut Reiser (`Subject`), dass von der MNM CA (`Issuer`), der Organisation (`O`) Munich Network–Management Team, in München (`L`=Location) ausgestellt wurde und eine Gültigkeit (`Validity`) von 2 Jahren hat. Im `Subject Public Key Info` Feld ist der öffentliche Schlüssel des Zertifizierten angegeben. Das Zertifikat verwendet die seit Version 3 eingeführten Erweiterungen (`extensions`). Damit wird z.B. festgelegt, für welche Zwecke die Schlüssel benutzt werden dürfen. Das Schlüsselpaar kann zur digitalen Signatur, für Non–Repudation und zur Verschlüsselung anderer (z.B. symmetrischer) Schlüssel verwendet werden. Der Gebrauch des Schlüsselpaares zur Zertifizierung anderer Benutzer wird explizit ausgeschlossen (CA:FALSE). Am Ende des Zertifikates ist die digitale Signatur (`Signature`) angegeben. Für die Berechnung wurde als Hash–Funktion der MD5 Algorithmus und für die digitale Signatur des Hash der RSA Algorithmus verwendet (`md5WithRSAEncryption`).

X.509v3 Zertifikat; Beispiel

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 34 (0x22)
    Signature Algorithm: md5WithRSAEncryption
    Issuer: C=DE, ST=Bavaria, L=Munich,
      O=Munich Network-Management Team, CN=MNM CA,
      Email=certmaster@nm.informatik.uni-muenchen.de
    Validity
        Not Before: Feb  8 12:44:40 2000 GMT
        Not After : Feb  7 12:44:40 2002 GMT
    Subject: C=DE, ST=Bavaria, L=Munich, O=Munich
      Network-Management Team, CN=Helmut Reiser,
      Email=reiser@nm.informatik.uni-muenchen.de
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        RSA Public Key: (1024 bit)
            Modulus (1024 bit):
                00:a3:88:f3:15:2c:5e:c7:7c:16:d5:e7:a0:f4:15:
                be:8f:c3:e4:3c:61:6a:26:1e:9d:3a:40:d4:55:b8:
                ef:23:11:5c:1a:63:1c:31:a0:b8:66:4a:fc:e7:fa:
                d4:ee:f1:e6:23:04:4f:c5:e4:fa:2e:ce:cd:dc:97:
                59:08:42:3f:d4:e9:33:76:1b:7e:bd:b3:21:3e:a4:
                3d:7b:f8:5c:aa:ae:fc:2d:2d:19:9d:8a:03:77:72:
                7c:a3:2b:01:e4:b1:96:20:7a:60:8b:b8:e4:15:40:
                de:8a:a7:da:21:fc:02:c4:a8:a6:fd:1e:4f:03:89:
                db:21:17:86:b0:87:c7:42:4d
           Exponent: 65537 (0x10001)
    X509v3 extensions:
       X509v3 Basic Constraints:
           CA:FALSE
       X509v3 Key Usage:
            Digital Signature, Non Repudiation, Key Encipherment
       X509v3 Subject Key Identifier:
            B5:EF:FA:3F:75:B3:8F:46:47:B2:14:9F:80:C2:5B:
            1B:37:31:AF:90
       X509v3 Authority Key Identifier:
            DirName: C=DE, ST=Bavaria, L=Munich,
              O=Munich Network-Management Team, CN=MNM CA,
              Email=certmaster@nm.informatik.uni-muenchen.de
            serial:00

    Signature Algorithm: md5WithRSAEncryption
        a7:99:64:9d:1d:22:1e:6a:d8:f3:ef:5a:92:ac:71:61:d4:b8:
        80:3b:f3:0d:c3:90:97:84:e7:b3:48:4d:03:2e:bb:d2:bc:72:
        ae:79:3a:81:e0:54:70:4e:4e:b9:43:5b:ea:2a:f4:6a:59:22:
        c0:59:ab:3b:d7:d4:17:c5:4d:1c:6b:2e:92:23:46:25:1f:f2:
        a6:7f:5b:37:6d:64:9a:32:22:fe:79:30:8e:ef:f2:fd:0f:b2:
        c9:52:3e:98:43:a3:51:02:27:06:07:bf:e6:14:f6:97:a2:de:
        2d:d8:eb:70:4a:1b:9a:2d:89:9d:c6:76:27:80:de:cc:07:ba:
        26:bf
```

4.2.4 Authentisierung der Entitäten

Die Authentisierung mittels digitaler Zertifikate erfolgt im lokalen und im entfernten Fall grundsätzlich auf dieselbe Art und Weise. Der Principal legt dem Verifier sein Zertifikat vor. Der Verifier kennt den öffentlichen Schlüssel der CA bzw. der Local Certification Authority (LCA) und kann damit den Hash–Wert entschlüsseln. Dann berechnet der Verifier selbst den Hash–Wert über die Daten des Zertifikates und vergleicht diesen mit dem entschlüsselten Wert. Sind beide Werte gleich, ist das Zertifikat korrekt und der Verifier kann sicher sein, dass der im Zertifikat enthaltene öffentliche Schlüssel zu der im Zertifikat angegebenen Entität gehört (vgl. Abb 4.7). Nun muss die Entität noch belegen, dass sie auch im Besitz des dazugehörigen privaten Schlüssels ist, sonst könnte jeder Angreifer ein kopiertes Zertifikat zur Authentisierung verwenden. Dazu wird i.d.R. ein Challenge–Response Protokoll zwischen Verifier und Principal abgewickelt, ähnlich dem zwischen CA und Benutzer bei der Personalisierung (weitere Informationen dazu, vgl. z.B. [Schn 96]).

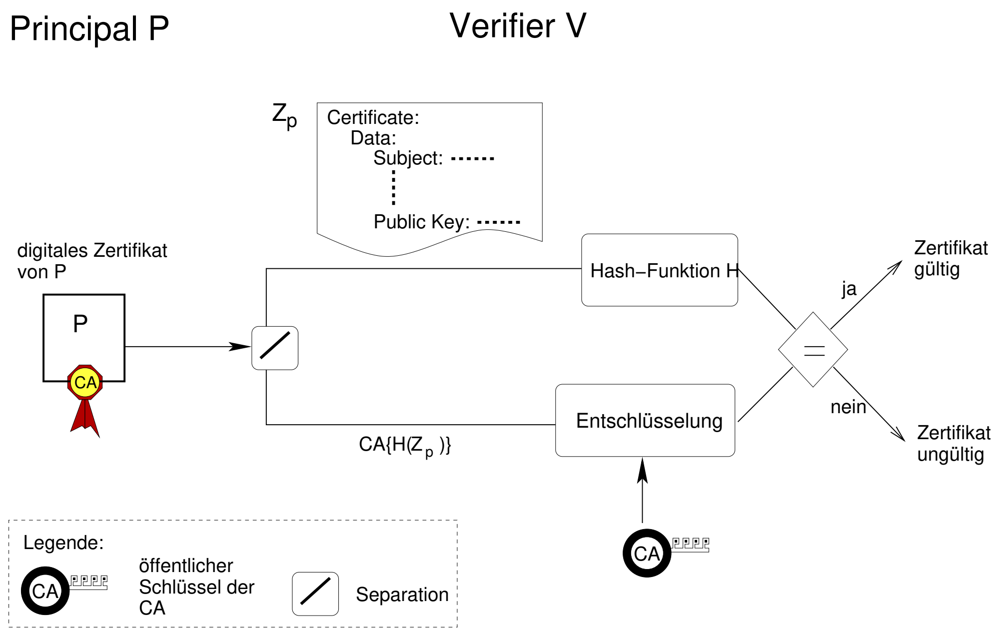


Abbildung 4.7: Verifikation eines Zertifikates

Der Principal muss das Zertifikat nicht notwendigerweise direkt an den Verifier übermitteln. Bei einer funktionierenden CA Infrastruktur reicht es aus, dass der Principal seinen Distinguished Name dem Verifier bekanntgibt. Dieser kann dann das entsprechende Zertifikat direkt von der CA oder aus einem Verzeichnisdienst abrufen. Für den Fall, dass der Verifier den öffentlichen Schlüssel des Principals schon kennt, kann sofort mit dem Challenge–Response Protokoll begonnen werden.

Neben der Authentisierung mit Hilfe einer CA und den von der CA ausgestellten Zertifikaten sind auch andere Verfahren bekannt und unter Umständen anwendbar (z.B. Passwörter zur Authentisierung von Benutzern). In diesem

Abschnitt werden für jede Entität des Entitätenmodells die am besten geeigneten Verfahren ermittelt. Falls zertifikatsbasierte Verfahren verwendet werden, ist für jede Entität eines allgemeinen MA–Szenario zu untersuchen, ob die auf Seite 84 angegebenen Bedingungen 1 und 2 erfüllt werden können. Bedingungen 3 und 4 werden durch eine geeignete CA–Infrastruktur erfüllt.

Authentisierung von Anwendern

Bei den Authentisierungsverfahren für Anwender werden zwei Verfahrensklassen unterschieden. Entweder basiert die Authentisierung auf „gemeinsamem Wissen“ oder auf „Besitz“.
Der bekannteste Vertreter für die Klasse „gemeinsames Wissen“ ist das Passwort–Verfahren. Principal und Verifier kennen beide das (verschlüsselte) Passwort. Der Principal gibt zur Authentisierung seinen Namen und sein Passwort an; der Verifier überprüft die angegebenen Werte. Derartige Verfahren sind den **schwachen Authentisierungsverfahren** zuzuordnen, die nur bedingt Schutz bieten. Zur Klasse der „Besitz“–basierten Verfahren gehören Zertifikate, Smart–Cards, Biometrische Verfahren, u.ä. Daneben existieren auch mehrstufige und Mischverfahren. „Besitz“–basierte Verfahren, die starke kryptographische Algorithmen verwenden, werden als **starke Authentisierungsverfahren** klassifiziert [X.509].

Anwender: Zertifikate der CA

Da für die Sicherheitsarchitektur nur starke Verfahren als Basis für die anderen Sicherheitsdienste dienen können, wird für Anwender eine auf Zertifikaten basierende Authentisierung verwendet, d.h. die Anwender und deren öffentlicher Schlüssel werden von der CA zertifiziert. Über das Attribut `subjectUniqueIdentifier` und die `extensions` Attribute lassen sich ggf. Rollenmodelle, die im Management erforderlich sind, abbilden. Ein Anwender erhält ein auf seinen Namen (DN) ausgestelltes Zertifikat. Seine Rollen, in denen er tätig wird, werden im `subjectUniqueIdentifier` Attribut aufgenommen oder als „Tätigkeitsfelder“ in den `extensions` angegeben.

Rollen als Bestandteil des Zertifikates

Bedingungen an Zertifikate erfüllbar

Ein Anwender in einem Managementsystem ist in der Lage, die Bedingungen, die an Zertifikate gestellt werden, zu erfüllen. Es existieren ausreichend viele technische Möglichkeiten, den privaten Schlüssel geheim zu halten; z.B. die verschlüsselte Speicherung auf einem Read–only Medium oder einer Smart Card, bzw. einer biometrisch gesicherten Smart–Card. Für die Gruppe der Anwender kann auch durch organisatorische Regeln und Vereinbarungen sichergestellt werden, dass ein Anwender seinen privaten Schlüssel nicht weitergibt, d.h. Bedingung 1 auf Seite 84 kann für Anwenderzertifikate eingehalten werden.

Auch Bedingung 2 ist unkritisch. Die (zertifizierten) Anwenderdaten sind relativ langlebig und statisch. Ein Zertifikat kann daher eine relativ lange Lebensdauer haben (mehrere Jahre). Für Anwender, deren DN sich ändert, da sie beispielsweise innerhalb eines Unternehmens die Abteilung wechseln, muss ein neues Zertifikat mit dem geänderten DN ausgestellt werden. Für den Fall, dass der Anwender eine weitere Rolle einnehmen muss, kann ihm ein neues zusätzliches „Rollen–Zertifikat“ ausgestellt werden, ohne sein bestehendes Zertifikat ändern zu müssen. Falls er eine Rolle verliert, muss das Zertifikat, das diese Rolle enthält, widerrufen und ein neues Zertifikat, ohne das entsprechende Rollenattribut ausgestellt werden.

Authentisierung von Endsystemen

Bei den Endsystemen besteht die Möglichkeit, sich auf Authentisierungsverfahren abzustützen, die vom Betriebssystem zur Verfügung gestellt werden. Es gibt auch dort Verfahren, um die Adresse bzw. den Namen eines Endsystems zweifelsfrei mit dem System zu verbinden bzw. den Hostnamen eindeutig an die IP-Adresse zu knüpfen. Bei **IPSec** [KeAt 98a], den Sicherheitserweiterungen für das IP–Protokoll, werden dazu die beiden Protokolle Authentication Header (AH) [KeAt 98b] und Encapsulation Security Payload (ESP) [KeAt 98c] verwendet. Ein geheimer Schlüssel, den nur die zwei Kommunikationspartner kennen, stellt die Bindung der Adresse an das Endsystem sicher. Nur die beiden Partner können die Nachrichten des jeweils anderen lesen und sich damit sicher sein, dass die im IP–Paket angegebene Absenderadresse richtig ist. Ein weiterer Ansatz sind die Sicherheitserweiterungen (**DNSSec**) für das Domain Name System (DNS) [EaKa 97]. Dabei wird der Directory Dienst, der die Abbildung von Host–Namen auf IP–Adressen und umgekehrt durchführt, durch digitale Signaturen erweitert. Jeder Datensatz in der DNS–Datenbank (engl. Resource Record, RR) wird dabei vom DNS–Server digital signiert. Damit wird sichergestellt, dass der Host–Name zweifelsfrei an die entsprechende IP–Adresse gebunden wird. Auch die **Secure Shell** (**ssh**) kann zur Authentisierung verwendet werden. Bei ssh–fähigen Endsystemen besitzt jedes System ein Schlüsselpaar. Der öffentliche Schlüssel wird dabei in einem Challenge Response Protokoll zur Authentisierung verwendet (näheres siehe [YKSR 01a, YKSR 01b, YKSR 01c, YKSR 01d]).

Endsystem: IPSec, DNSSec, ssh oder Zertifikate

Auch für Endsysteme können digitale Zertifikate ausgestellt werden, die die IP–Adresse und den *Full Qualified Host Name (FQHN)* an einen öffentlichen Schlüssel binden. Die Geheimhaltung des privaten Schlüssels (Bedingung 1) eines Endsystems kann durch gesicherte Speicherbereiche Spezialhardware oder die Auslagerung auf ein Read–Only Wechselmedium sichergestellt werden. Auch bei der IP–Adresse und dem FQHN handelt es sich um relativ langlebige Daten. Lediglich bei einer Umbenennung des Systems vor dem Ablauf des Zertifikates muss ein neues Zertifikat ausgestellt werden, d.h. auch Bedingung 2 kann sehr einfach eingehalten werden.

Bedingungen an Zertifikate erfüllbar

Authentisierung von Agentensystemen

Bei Agentensystemen handelt es sich um reine Software–Bausteine. Eine Abstützung auf gesicherte Hardware oder Smart–Cards, wie bei Endsystemen oder Anwendern, ist deshalb nicht möglich. Trotzdem können für die

AS: Zertifikat der CA

Authentisierung Zertifikate verwendet werden. Jedes Agentensystem erhält ein Schlüsselpaar und der öffentliche Schlüssel und der Name des Agentensystems werden von der CA digital signiert.

Einbettungsbeziehung: Endsystem sichert privaten Schlüssel

Das Agentensystem muss, um Bedingung 1 erfüllen zu können, in der Lage sein, den privaten Schlüssel geheim zu halten. Wegen der Einbettungsbeziehung (vgl. Abschnitt 4.1) kann es sich dazu jedoch auf Methoden des Endsystems abstützen. Der gespeicherte Schlüssel kann und muss vom Betriebssystem vor dem Zugriff anderer Prozesse geschützt werden. Dazu können die Rechte– und Zugriffskontrollkonzepte auf Dateisystemebene verwendet werden. Für den Fall, dass das Endsystem über die Dateisystemrechte keinen ausreichenden Schutz bietet, kann der Schlüssel auch mit einer „Pass–Phrase“ verschlüsselt gespeichert werden. Beim Start des Agentensystems muss der Besitzer dann diese Pass–Phrase einmal eingeben, um den Schlüssel entschlüsseln zu können. Aber auch während des Betriebs des Agentensystems muss der private Schlüssel in einem gesicherten Speicherbereich des Arbeitsspeichers vor dem Zugriff anderer Prozesse geschützt werden. Auch dafür bieten die Betriebssysteme der jeweiligen Endsysteme Speicherschutzmechanismen. Der Name und die Attribute des Agentensystems, die zertifiziert werden, sind relativ statisch. Agentensysteme werden auf einem Endsystem installiert und dann nicht mehr verschoben. AS–Zertifikate können daher auch eine relativ lange Lebensdauer haben (Bedingung 2).

Authentisierung von Mobilen Agenten

Mobile Agenten stellen bezüglich der Authentisierung einen Sonderfall dar. Die Entität Mobiler Agent muss, wie in Abschnitt 4.2.1 beschrieben, in einen statischen und einen dynamischen Teil aufgespalten werden. Diese beiden Teile, d.h. die MA–Gattung und auch die MA–Instanz, müssen getrennt voneinander authentisiert werden.

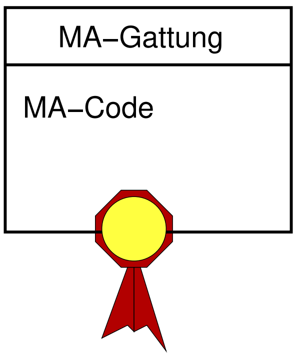


Abbildung 4.8: Authentisierung der MA–Gattung

MA–Gattung und –Instanz können von verschiedenen Quellen stammen, und es können unterschiedliche Authorities für sie verantwortlich sein. Auch im Lebenszyklus und in der Änderungsdynamik unterscheiden sich beide erheblich.

MA–Gattung: digitale Signatur durch Authority

Bei MA–Gattungen geht es bei der Identifikation darum, den gesamten statischen Programmcode an eine Authority zu binden. Hierfür eignet sich eine direkte digitale Signatur der gesamten MA–Gattung besser als die Erteilung eines Zertifikates. Ein Zertifikat verbindet Informationen innerhalb des Zertifikates (vgl. Tabelle 4.5) mit dem öffentlichen Schlüssel. Bei einer MA–Gattung geht es aber primär darum, den gesamten Programmtext eindeutig, z.B. einem Implementierer, zuzuordnen. Dazu wird über den Quellcode und den Gattungsnamen ein Hash–Wert H berechnet und dieser wird dann von der Authority (z.B. dem Implementierer) digital signiert (vgl. Abbildung 4.8). Der Hash–Wert stellt sicher, dass jede Veränderung, z.B. die Veränderung des Gattungsnamens oder auch Veränderungen am Programmtext, erkannt werden können, und die digitale Signatur bindet den Gattungsnamen als Identifikator zweifelsfrei an die MA-Gattung. Für die Signatur muss die signierende Entität im Besitz eines asymmetrischen Schlüsselpaares und damit eines gültigen Zertifikates sein.

MA–Instanz: Besitzer signiert Namen

MA–Instanzen bestehen aus den variablen Daten des Mobilen Agenten und sind daher extrem dynamisch. Würde man die MA–Instanz zur Authentisierung, analog der Gattung, digital signieren, würde die Signatur mit jeder Änderung der Daten der MA–Instanz ungültig. Aus diesem Grund kann das Konzept zur Authentisierung der Gattung nicht direkt für die Instanz übernommen werden. Das Ziel der Authentisierung ist es, die MA–Instanz eindeutig zu identifizieren und den Verantwortlichen (Authority) zu bestimmen. Für einen Mobilen Agenten ist der Besitzer, d.h. derjenige, der den Mobilen Agenten erstmalig startet, verantwortlich. Wenn der Besitzer den Namen der MA–Instanz (vgl. Abschnitt 4.2.2) digital signiert, wird sein Benutzername untrennbar mit dem Namen des Mobilen Agenten verbunden. Die entstehende Datenstruktur wird dem Mobilen Agenten im Feld `Authority` mitgegeben. Damit lässt sich immer ermitteln, wer für den Mobilen Agenten verantwortlich ist. Nach der Migration ändert der Mobile Agent jedoch seinen Namen, da der Name des Agentensystems mit in seinen eigenen Namen eingeht. Daraus folgt unmittelbar, dass das jeweilige Quell–AS die MA–Instanz vor der Migration digital signieren muss, damit das Ziel–Agentensystem die Instanz authentisieren kann (vgl. Abbildung 4.9).

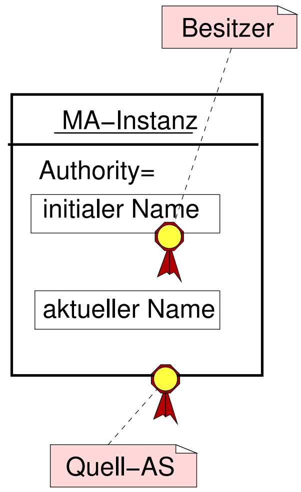


Abbildung 4.9: Authentisierung der MA–Instanz

AS signiert Instanz vor jeder Migration

Insbesondere bei der Authentisierung von MA–Gattung bzw. Instanz handelt es sich um lokale Authentisierung. Der Verifier (das AS) verifiziert dabei die digitale Signatur der Entität. Dazu wird mit dem öffentlichen Schlüssel des Signierenden der Hash–Wert entschlüsselt, der über die gesamte Gattung, bzw. den Namen der Instanz, berechnet wurde. Dann muss der Verifier selbst den Hash–Wert berechnen und die beiden Werte vergleichen. Stimmen die beiden Werte überein, kann er sich einerseits sicher sein, dass die Entität von der Authority unterzeichnet wurde, mit deren öffentlichem Schlüssel der Verifier die digitale Signatur entschlüsseln konnte. Andererseits kann er wegen der Art der Hash–Funktion auch sicher sein, dass die signierten Daten nach der digitalen Signatur nicht mehr verändert wurden. Durch die direkte digitale Signatur wird die Identität von Gattung und Instanz zertifiziert, d.h. es ist zu prüfen, inwieweit die Bedingungen, die auf Seite 84 an Zertifikate gestellt wurden, auch für digital signierte Gattungen und Instanzen eingehalten werden müssen.

Bedingungen an digitale Signatur

Bedingungen 3 und 4 sind nur für digitale Zertifikate erforderlich. Eine digitale Signatur von Gattung und Instanzname muss nicht zurückgenommen werden können. Mit der digitalen Signatur wird die Identität bestätigt, bzw. ein neuer Principal „geschaffen". Dieser Vorgang muss und darf nicht zurückgenommen werden.

Bedingung 2 muss auch für digitale Signaturen eingehalten werden. Sobald sich die signierten Daten ändern, muss eine neue Signatur erzeugt werden. Für MA–Gattungen ist dies der Fall, wenn sich bspw. die Implementierung eines Mobilen Agenten ändert. Der Implementierer, der die Änderung vornimmt, muss die MA–Gattung erneut signieren. Bei MA–Instanzen muss, wie oben erläutert, vor jeder Migration eine digitale Signatur des Namens erzeugt werden. Insbesondere durch das fortlaufende Signieren der Instanz vor jeder Migration entsteht mit der „Signaturkette" eine Historie der Reise des Mobilen Agenten (vgl. auch Abschnitte 4.5.3 und 5.4.4).

Da mit der digitalen Signatur einer Entität kein privater Schlüssel dieser Entität verknüpft ist, sondern nur ein privater Schlüssel der signierenden Entität verwendet wird, muss Bedingung 1 auch nur für den Zertifizierenden erfüllt werden. Der Zertifizierende muss im Besitz eines privaten Schlüssels sein. Als zertifizierende Entität kann jede andere Entität auftreten. Für Anwender, Endsysteme und Agentensysteme kann sichergestellt werden, dass diese ihren privaten Schlüssel geheim halten können. Für Mobile Agenten wird dieses Problem im Folgenden untersucht.

MA benötigt zusätzlich Schlüsselpaar und Zertifikat

Durch die digitale Signatur von Gattung und Instanz können diese beiden Entitäten authentisiert werden. Um aber aktiv im System zu handeln, d.h. um als Subjekt auftreten zu können, muss auch ein Mobiler Agent im Besitz eines privaten Schlüssels und damit eines Zertifikates sein. Das Problem bei der Ausstellung von Zertifikaten an Mobile Agenten ist die extrem hohe Dynamik des gesamten Managementsystems. Ein Mobiler Agent ändert nach jeder Migration seinen Namen, da i.d.R. der Place oder das Agentensystem, auf dem der Agent ausgeführt wird, Teil seines Namens sind. Aus diesem Grund muss der MA–Instanz spätestens nach jeder Migration ein neues Zertifikat ausgestellt werden (Bedingung 2). Klassische zentrale CAs sind nicht in der Lage, mit dieser hohen Dynamik umzugehen. Sie sind nicht dafür ausgelegt, in kürzesten Abständen für eine Entität neue Zertifikate auszustellen. Aus sicherheitstechnischen oder rechtlichen Gründen müssen die Systemkomponenten, die die digitale Signatur des Zertifikates durchführen, oftmals im Offline–Modus betrieben werden [Gres 99], d.h. MA–Zertifikate können nicht von der zentralen CA ausgestellt werden.

MA kann privaten Schlüssel nicht vor AS geheim halten

Ein weiteres Problem ist die Generierung des Schlüsselpaares und die Geheimhaltung des privaten Schlüssels für den Mobilen Agenten (Bedingung 1). Ein Mobiler Agent, der unter der vollständigen Kontrolle des Agentensystems ausgeführt wird, kann seinen privaten Schlüssel nicht vor dem Agentensystem geheim halten. Im Allgemeinen kann auch nicht davon ausgegangen werden, dass der Mobile Agent technisch in der Lage ist, seine Schlüssel selbst zu erzeugen. Wollte man dies realisieren, müsste jeder Mobile Agent den Code für die Schlüsselgenerierung mitführen. Dies widerspricht aber dem Grundsatz, den Agenten möglichst „schlank“ zu halten.

Um die beiden Problemgruppen zu lösen, und damit die Bedingungen an Zertifikate einhalten zu können, kann die Einbettungsbeziehung genutzt werden. Der Mobile Agent muss vor der Migration einen bestimmten Vertrauenslevel zum Ziel–AS aufbauen; er muss dem Agentensystem bis zu diesem Grad vertrauen. Der Mobile Agent kann deshalb auch das Agentensystem nutzen, um sich von diesem Schlüssel generieren zu lassen. Das Agentensystem kann zweifelsfrei identifiziert und authentisiert werden, es besitzt ein eigenes von der CA ausgestelltes Zertifikat und ein entsprechendes Schlüsselpaar. Mit seinem privaten Schlüssel kann das Agentensystem auch als **Local Certification Authority** (LCA), d.h. als Stellvertreter der globalen CA, „ihren“ MA–Instanzen Schlüsselpaare erzeugen und Zertifikate ausstellen. Das Agentensystem muss sicherstellen, dass der private Schlüssel eines Agenten für die anderen Agenten nicht zugänglich ist.

Principal / Verifier	Endsystem	Agentensystem	Mobiler Agent: Gattung	Mobiler Agent: Instanz	Anwender
Endsystem	Betriebssystem–spezifisch (entfernt)	Prozeßverwaltung des Betriebssystems (lokal)	i.d.R. nicht möglich (MA durch AS verschattet)		Betriebssystem–spezifisch (lokal)
Agentensystem	nicht möglich (Einbettungsbeziehung)	Zertifikat (entfernt)	digitale Signatur (lokal)	digitale Signatur (lokal)	Zertifikat (lokal und entfernt)
MA Gattung	stellvertretend durch Agentensystem; Einbettungsbeziehung (lokal und entfernt)	stellvertretend durch Quell–Agentensystem (entfernt; Einbettungs–beziehung) (lokal nicht möglich; Einbettungsbeziehung)	nicht anwendbar	nicht anwendbar	Zertifikat (lokal und entfernt)
MA Instanz			stellvertretend durch Agentensystem; Einbettungsbeziehung (lokal oder entfernt)	Zertifikat (lokal und entfernt)	
Anwender	Betriebssystem–spezifisch (lokal und entfernt)	Zertifikat (lokal und entfernt)	Zertifikat (lokal und entfernt)		

Tabelle 4.6: Authentisierungsarten und –möglichkeiten

4.2.5 Schlussfolgerungen für die Sicherheitsarchitektur

Bei der Authentisierung überprüft der Verifier die Identität eines Principals entweder lokal oder entfernt. Dabei verifiziert er entweder selbst oder aber der Verifier stützt sich zur Authentisierung auf einen vertrauenswürdigen Stellvertreter. Damit die Identität eines Principals auch eindeutig ist, muss ein eineindeutiges **Namens– bzw. Identifikationsschema** definiert sein. Tabelle 4.6 fasst alle möglichen Principal/Verifier Paare und damit alle Authentisierungsfälle, die in einem System Mobiler Agenten auftreten können, zusammen.

Um in einem Managementsystem basierend auf Mobilen Agenten eine **Authentisierung** für alle Entitäten **mit digitalen Zertifikaten** durchführen zu können, muss ein **Trust Center bzw. globale CA** aufgebaut werden, die für alle Entitäten des Systems (bis auf die Mobilen Agenten) Zertifikate ausstellt.

Die Sonderstellung der Mobilen Agenten äußert sich durch die Notwendigkeit, sowohl **MA–Gattung** als auch **MA–Instanz** zu authentisieren. Im Gegensatz zu den anderen Entitäten werden Gattung und Instanz dazu direkt digital signiert. Die MA–Gattung wird z.B. vom Implementierer oder von einem Code Repository signiert. Die MA–Instanz wird vom Besitzer und vor jeder Migration vom Quell–AS signiert. Um selbst aktiv im System zu handeln, bedarf der Mobile Agent eines Schlüsselpaares und eines Zertifikates. Wegen des Vertrauens durch Einbettungsbeziehung wird hierzu das **AS als Local Certification Authority** für den Mobilen Agenten die Schlüssel erzeugen und auch die Zertifikate ausstellen.

Abbildung 4.10 fasst den Zusammenhang zwischen Zertifikaten, digitalen Signaturen und den Entitäten noch einmal in einem UML–Diagramm zusammen.

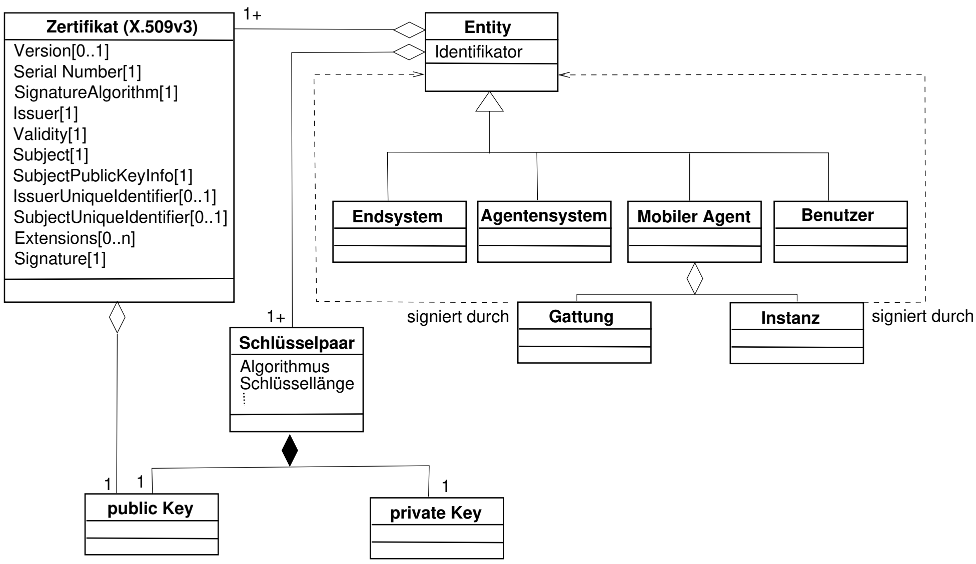


Abbildung 4.10: Zertifikate und digitale Signatur von Entitäten

4.3 Sicherheitskonzepte für das Relationenmodell: Ausführungsrelation

In einem System Mobiler Agenten kann ein Angreifer nur Entitäten direkt oder Relationen zwischen Entitäten angreifen (vgl. Abschnitt 2.4). Die Ausführungsrelation besteht zwischen Software–Bausteinen auf demselben System. Sie zeichnet sich durch eine direkte Interaktion und eine gemeinsame Ressourcennutzung aus. Die Ausführungsrelation ist asymmetrisch, da der Executor den Executee ausführt und damit kontrollieren kann. Neben den Angriffen, die auf jede Relation möglich sind, gibt es mit der Manipulation des Ausführungspfades und der Denial of execution spezielle Angriffe auf die Ausführungsrelation. In diesem Abschnitt wird untersucht, inwieweit Angriffe, welche die Integrität des Executees während der Ausführung betreffen, verhindert werden können. Auch die Gefahren, die sich durch die gemeinsame Ressourcennutzung der verschiedenen Entitäten ergeben, werden betrachtet.

4.3.1 Ausführungsintegrität

Bei der Ausführungsintegrität, d.h. der Integrität des Executees, ist insbesondere die Entität Mobiler Agent von Interesse. Daneben gibt es auch noch die Integrität der Kommunikations– bzw. der Migrationsrelation (diese Fragen werden in Abschnitt 4.5.2 untersucht). Betrachtet man die Integrität des Mobilen Agenten während seiner Ausführung auf einem Agentensystem, so sind drei Fälle zu unterscheiden:

1. Die Integrität bezüglich Veränderungen durch das **Gast–Agentensystem**. Die Frage, die sich hier stellt, ist: Wie kann eine Manipulation am Mobilen Agenten erkannt oder verhindert werden?
2. Die Integrität bezüglich der Ausführung selbst. Wurde der Agent überhaupt ausgeführt? Wurde er vom Agentensystem bis zum Ende seiner Berechnungen ausgeführt oder vorzeitig angehalten (**Denial of execution**)?
3. Integrität bezüglich **anderer Mobiler Agenten**, die ebenfalls als Executees desselben Agentensystems ausgeführt werden und durch Wechselwirkungen den Mobilen Agenten manipulieren können.

Punkt 1 ist im Grunde äquivalent zum Problem des Schutzes vor böswilligen Agentensystemen. Dazu gibt es relativ viele Untersuchungen (vgl. Abschnitt 3.2.4), die aber alle nur für, zum Teil exotische, Spezialfälle eine Lösung versprechen. In Allgemeinheit scheint dieses Problem nicht lösbar.

Ausführungsintegrität in Allgemeinheit nicht lösbar

Da das Agentensystem die vollständige Kontrolle über den Mobilen Agenten besitzt (vgl. Abschnitt 4.1), kann eine Veränderung nicht verhindert werden. Auch die nachträgliche Erkennung ist nicht immer durch ein automatisches Verfahren möglich. Es gibt bspw. Verfahren, um die Integrität der Daten zu sichern, die von einem Agenten transportiert werden, der mehrere Agentensysteme besucht (siehe Ausführungen in Abschnitt 4.5.2). Für einen Mobilen Agenten, der ein Agentensystem besucht, um dann wieder zu seiner Quelle zurückzukehren, sind diese Verfahren nicht anwendbar. Auch für die Klasse der Denial of execution Angriffe gilt, dass sie in Allgemeinheit weder verhindert noch erkannt werden können.

Auch in diesem Bereich kann das Trust Level Management (vgl. Abschnitt 4.1.2) Anwendung finden. Die Wahrscheinlichkeit, mit der ein Agentensystem einen Mobilen Agenten manipuliert, kann mit in die Berechnung des Vertrauenslevels einbezogen werden. Dies verhindert zwar die Manipulation nicht, aber dem Besitzer des Mobilen Agenten wird damit zumindest eine Möglichkeit gegeben, den Grad der Zuverlässigkeit der Daten einordnen zu können.

organisatorische Lösung: Sanktionssystem

Eine andere Lösung des Integritätsproblems kann nur auf organisatorischem Wege gefunden werden. Zum einen kann im Management, im Gegensatz zu einem offenen Szenario, auf die organisatorischen und rechtlichen Beziehungen und Abhängigkeiten zwischen den beteiligten Organisationseinheiten zurückgegriffen werden. Die Verletzung der Integrität eines Mobilen Agenten muss für einen Beteiligten so „teuer“ sein, dass sich die Manipulation für den Angreifer nicht „rechnet“. Das heißt, die beteiligten Firmen und Organisationseinheiten müssen ein Sanktionssystem festlegen, das die Verletzung der Integrität unter Strafe stellt.

Damit ein Sanktionssystem wirksam werden kann, muss die Möglichkeit gegeben sein, Manipulationen zumindest stichprobenartig erkennen zu können. Werden die Spezifika sowie Kontextinformationen des konkreten Anwendungsfalls mit in die Betrachtungen einbezogen, so gibt es folgende Möglichkeiten:

- **Bereichsverifikation** und **Datenplausibilität**: Bei vielen Anwendungen ist bekannt, in welchen Bereichen die Daten liegen müssen. Stärkere Abweichungen von den erwarteten Werten lassen sich erkennen.
- **Statistische Verfahren**: Da im Management sehr viele gleichartige Daten über einen längeren Zeitraum von verschiedenen Quellen anfallen, können statistische Verfahren (Mittelwert, Varianz, χ^2–Test, usw.) verwendet werden, um „Ausreißer“ zu erkennen.
- **Zeitrestriktionen**: Analog zur Datenplausibilität gibt es auch Anwendungen, für die zeitliche Plausibilität besteht, d.h. es ist bekannt, in welchen Bereichen eine Antwort normalerweise zu erwarten ist.
- **Detection Objects** sind Testdatensätze, die dem Mobilen Agenten mitgegeben werden und bei einer sachgemäßen Ausführung des Mobilen Agenten nicht verändert werden. Sind die Detection Objects nach der Rückkehr verändert, so ist die Integrität des Mobilen Agenten nicht mehr gewahrt (vgl. Abschnitt 3.2.4 und [Mead 97]).
- **Tracing:** Eine sehr aufwendige Methode ist das Tracing. Dabei schreibt das Agentensystem alle Operationen des Agenten in ein Protokoll und signiert dieses. Dieser Trace kann dann Schritt für Schritt nachvollzogen werden (vgl. Abschnitt 3.2.4, Seite 59).

Alle diese Ansätze erfordern entweder Wissen über die Anwendung oder sind nicht für alle Anwendungsfälle einsetzbar.

Die Sicherung der Integrität eines Mobilen Agenten bezüglich anderer Agenten kann garantiert werden, wenn die Mobilen Agenten vollständig voreinander abgeschottet werden. Die Abschottung ist sehr eng mit der Sicherheitsanforderung nach Vertraulichkeit verknüpft, die im nächsten Abschnitt untersucht wird.

4.3.2 Vertraulichkeit während der Ausführung

Vertraulichkeit bezüglich der Ausführungsrelation bedeutet, dass Entitäten bestimmte Daten während ihres Ablaufes vor anderen Entitäten geheim halten können. Klar ist auch, dass Entitäten, die in einer Einbettungsbeziehung zueinander stehen, nur für bestimmte Grenzfälle diese Vertraulichkeit sicherstellen können (vgl. z.B. Abschnitt 3.2.4). Ein Agentensystem kann seine internen Daten nur sehr bedingt vor dem Endsystem verbergen, auf dem es ausgeführt wird. Gleiches gilt für den Mobilen Agenten und das ausführende Agentensystem (vgl. dazu die Beispiele in Abschnitt 4.1).

Im Management, insbesondere dann, wenn organisatorische Grenzen überschritten werden, ist die Vertraulichkeit der Agenten von besonderem Interesse. Als Beispiel sei hier die Kunden–Provider Beziehung genannt. Der Kunde ermöglicht es seinen verschiedenen Providern, Agenten auf seinem Agentensystem zur Ausführung zu bringen. Auf einem Agentensystem laufen also neben den Agenten des AS–Betreibers (in diesem Fall der Kunde) auch die Agenten der verschiedenen Provider ab. Da die Provider auf dem Markt miteinander konkurrieren, muss sichergestellt werden, dass die Daten der Mobilen Agenten vor „fremden“ Mobilen Agenten geschützt werden können.

Das Agentensystem stellt eine allgemeine Laufzeitumgebung für Mobile Agenten dar. Diese könnten versuchen, das Agentensystem zu missbrauchen, um an Daten anderer Agenten zu gelangen oder andere Agenten auf dem Agentensystem zu manipulieren. Daher muss das Agentensystem auch Eigenschaften aufweisen, wie sie normalerweise zur Trennung von Prozessen auf Betriebssystemen verwendet werden, wie:

1. Trennung der Speicherbereiche verschiedener Agenten (**Speicherschutz**)
2. Trennung der Laufzeitumgebung der Agenten (**Laufzeitschutz**)
3. **Trennung der Namensräume**

Speicherschutz

Das Agentensystem muss sicherstellen, dass die Speicherbereiche, die von einem Agenten genutzt werden, von keinem anderen Agenten gelesen oder beschrieben werden können. Andernfalls könnte ein böswilliger Mobiler Agent Daten von anderen Mobilen Agenten auslesen oder, falls er schreibend auf den Speicher zugreifen könnte, sogar die MA–Instanz verändern.

Speicherschutzmechanismen des Endsystems nicht anwendbar

Speicherschutzmechanismen des Endsystems bzw. des darauf ablaufenden Betriebssystems können nicht direkt verwendet werden, da sie im Allgemeinen prozessorientiert arbeiten. Die kleinste zu sichernde Einheit ist dabei der Speicher, der von einem Prozess belegt wird. Aus Sicht des Betriebssystems erscheint ein Agentensystem, mit allen darauf ablaufenden Agenten, als ein Prozess (vgl. auch Abschnitt 5.1.1). Das Betriebssystem kann die Speicherbereiche der einzelnen Agenten nicht unterscheiden und deshalb auch nicht schützen.

Zur Realisierung des Speicherschutzes könnten jedoch Betriebssystemkonzepte verwendet werden [Tane 92, Stal 97]. Diese müssten allerdings im Agentensystem erneut implementiert werden, d.h. das Agentensystem müsste ein komplettes Speicherverwaltungs– und Speicherschutzsystem für Mobile Agenten implementieren. Das Problem bei klassischen Speicherschutzsystemen sind so genannte **Buffer Overflows**. Diese Angriffe nutzen eine fehlende Bereichsüberprüfung bei dynamisch erzeugten Daten (auf dem Benutzer–Stack oder dem Heap) aus, um beispielsweise die Rücksprungadresse eines Prozeduraufrufs innerhalb eines Prozesses zu ändern und dadurch einen anderen Prozess zu starten und auszuführen (detailliertere Informationen sind bspw. in [Smit 97] zu finden). Für die fehlerhafte Bereichsüberprüfung ist nicht das Betriebssystem verantwortlich, sondern entweder der Implementierer des Programms oder Schwächen in der Implementierungssprache. D.h. das Speicherschutzkonzept eines Betriebssystems kann durch fehlerhaft programmierte Anwendungen verletzt werden. Damit könnte ein fehlerhaft programmierter Mobiler Agent das Speicherschutzkonzept des Agentensystems unwirksam machen. Aus diesem Grund und da die Implementierung eines Speicherschutzkonzeptes im Agentensystem sehr aufwendig wäre, wird eine Lösung favorisiert, die Speicherschutzverletzungen konzeptionell verhindert, indem direkte Speicherzugriffe verboten werden.

Verhinderung direkter Speicherzugriffe

Werden Programmiersprachen verwendet, in denen es keine Pointer und keine Methoden zur direkten Speichermanipulation gibt, kann kein Mobiler Agent direkt auf den Speicher und damit auf Speicherbereiche anderer Mobiler Agenten zugreifen. Java ist eine Programmiersprache, die diesen Ansatz konsequent verfolgt [LiYe 99, GJSB 00]. Alle Systeme Mobiler Agenten, die in Java implementiert sind, können sich bezüglich Speicherschutz auf die virtuelle Maschine abstützen.

Java Virtual Machine realisiert Speicherschutz

Laufzeitschutz

Neben den Methoden, die ein Mobiler Agent selbst anbietet, existieren auch Methoden des Agentensystems (z.B. `suspend, stop`) oder des allgemeinen Laufzeitsystems (z.B. `setPriority`), um den Status eines Mobilen Agenten zu ändern. **Laufzeitschutz** ist dann gegeben, wenn ein Mobiler Agent nicht durch Missbrauch des Agentensystems oder des Laufzeitsystems andere Mobile Agenten beeinflussen kann. Das Agentensystem muss deshalb den Zugriff auf eigene Schnittstellen, mit denen Mobile Agenten manipuliert werden können, schützen. Dies ist eine Aufgabe, die vom Zugriffskontrollsystem realisiert wird (vgl. Abschnitt 4.4.1).

Zugriffskontrolle auf Methoden des AS

Daneben muss aber auch beim Design des Agentensystems darauf geachtet werden, dass Methoden des Laufzeit– bzw. Interpretersystems nicht missbraucht werden können. Bei Agentensystemen, die in Java implementiert sind, werden Mobile Agenten häufig als Threads realisiert.

Ein **Thread** ist ein eigener Kontrollfluss innerhalb eines größeren Anwendungskontextes (z.B. Prozess im Betriebssystem, Applet oder Application in Java). Threads besitzen einen Vater und nutzen gemeinsam dieselben Ressourcen. Standardmäßig können sich Threads auch gegenseitig beeinflussen (z.B. stoppen, unterbrechen, Prioritäten ändern, usw.). Werden Mobile Agenten als Threads implementiert, muss deshalb auch in diesem Bereich ein Laufzeitschutz realisiert werden. In Java ist dies möglich, indem man mit Hilfe der Java Sicherheitsarchitektur (Policy Manager und Access Controller) den Zugriff auf Threads und Gruppen von Threads beschränkt ([OaWo 99] vgl. auch Abschnitt 5.4.3 und 6.2.2).

Methoden zur Thread Manipulation sichern

Trennung der Namensräume, Sichtbarkeit

Eine Trennung der Namensräume ist insbesondere für Systeme, die eine dynamische Bindung von Laufzeitbibliotheken oder dynamisches Nachladen von Klassen ermöglichen, unabdingbar. Die Kollision von Agentennamen ist zwar durch das hierarchische Namenskonzept (vgl. Abschnitt 4.2.2) ausgeschlossen, allerdings können die Mobilen Agenten Bibliotheksfunktionen des Agentensystem nutzen oder selbst Bibliotheksfunktionen oder Klassen „mitbringen“. Es sei bspw. angenommen, dass sich die Instanzen zweier Agenten A und B auf einem Agentensystem befinden, um Accounting Daten bezüglich des Netzverkehrs zu sammeln. Beide Agenten nutzen dazu verschiedene `Accounting`–Objekte und daraus eine Methode gleichen Namens, `NetworkUsage`. Im Fall des Agenten A soll die Methode der Klasse `Accounting` die übertragene Datenmenge in MegaByte, im Fall B in übertragenen ATM–Zellen zurückliefern.

Bindung dynamisch geladener Laufzeitbibliotheken

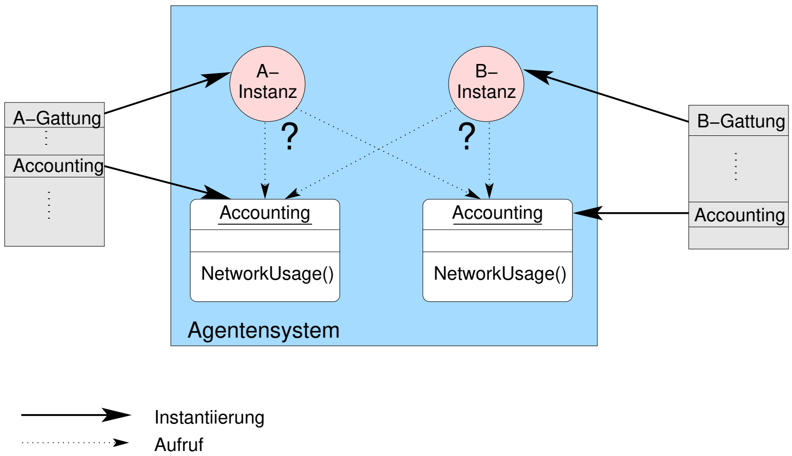


Abbildung 4.11: Verwendung von dynamisch geladener Klassen ohne Trennung der Namensräume

Werden nun die Instanzen von A und B auf dem Agentensystem ausgeführt, so kann ohne die Trennung von Namensräumen keine Aussage darüber getroffen werden, „welche“ Methode `Accounting.NetworkUsage` wirklich verwendet wird (vgl. Abb. 4.11). Die wird im konkreten Fall von der Implementierung der Laufzeitumgebung bestimmt und es können die folgenden zwei Fälle auftreten:

1. Bei Bibliotheken bzw. Klassen gleichen Namens wird immer die zuerst geladene ausgeführt (**early bound Semantik**). Beim zweiten Versuch der Verwendung wird erkannt, dass die Klasse bzw. Funktion bereits bekannt und geladen ist, und diese verwendet.
2. Bei Bibliotheken bzw. Klassen gleichen Namens wird immer die zuletzt geladene ausgeführt (**late bound Semantik**). Wird das Nachladen einer Klasse oder Bibliothek angefordert, so wird diese nachgeladen, auch wenn schon eine gleichen Namens geladen wurde.

In beiden Fällen besteht für Agenten die Möglichkeit, anderen Agenten bösartige Implementierungen „unterzuschieben“. Damit könnten Agenten dazu veranlasst werden, Klassen oder Methoden zu benutzen, die Schadensfunktionen enthalten, Daten ausspähen oder falsche Ergebnisse zurückliefern. Für den Mobilen Agenten ist es nicht erkennbar, dass er nicht die „richtige“ Funktion (`NetworkUsage`) verwendet.

Java verwendet bspw. die **lazy loading**, aber early bound Strategie [LiYe 99, Gong 98, Oaks 98], d.h. das Nachladen von Klassen wird so lange wie möglich hinausgezögert; aber sobald eine Klasse geladen wurde, wird beim zweiten Versuch des Nachladens die bereits geladene Klasse verwendet.

Mit einer Trennung der Namensräume kann sichergestellt werden, dass derartige Probleme nicht auftreten. D.h. ein Agent oder eine Agentengruppe, die denselben Satz von Methoden verwenden, erhalten einen eigenen abgeschlossenen Namens– und Speicherraum (vgl. Abb. 4.12). In diesem Fall ist der

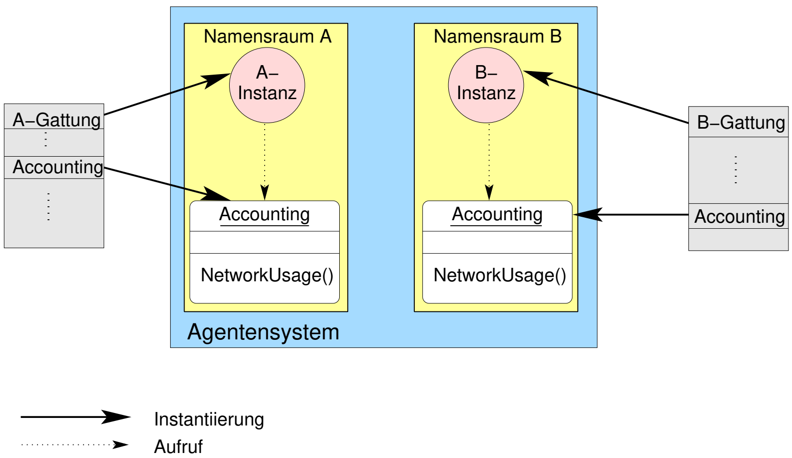


Abbildung 4.12: Trennung von Namensräumen

Agent B gar nicht mehr in der Lage, Methoden aus Bibliotheken des Agenten A „zu sehen“. Er kann sie also weder bewusst noch unbewusst nutzen. Allerdings kann dieses Konzept bei ungünstiger Administration der Namensräume auch dazu führen, dass dieselben Bibliotheken bzw. Klassen mehrfach in verschiedenen Namens– und Speicherbereichen geladen werden und dadurch Ressourcen verschwendet werden.

4.3.3 Sandboxing

Das von Applets bekannte Sandbox–Konzept kann auch zum Schutz des Agentensystems und als Ergänzung zum Zugriffskontrollkonzept (vgl. Abschnitt 4.4.1) eingesetzt werden. Ein unbekannter oder nicht vertrauenswürdiger Mobiler Agent, der auf ein Agentensystem migriert, wird in einem besonders geschützten Bereich — der als Sandbox bezeichnet wird — ausgeführt. Innerhalb der Sandbox sind keinerlei Zugriffe auf Ressourcen des Endsystems, des Agentensystems oder anderer Mobiler Agenten möglich. Der Agent wird innerhalb der Sandbox ausgeführt, Methodenaufrufe, die andere Entitäten betreffen, werden jedoch nur simuliert und protokolliert, aber nicht tatsächlich ausgeführt. Ein potentiell gefährlicher Agent kann in eine kontrollierbare Ablaufumgebung „eingesperrt“ werden, um zu untersuchen, welche Ressourcen er nutzen würde. Gegebenenfalls kann die Sandbox dann nach und nach geöffnet werden.

4.3.4 Schlussfolgerungen für die Sicherheitsarchitektur

Die Integrität des Mobilen Agenten während dessen Ausführung wird potentiell von zwei Seiten bedroht:

1. vom Agentensystem, das den Mobilen Agenten ausführt und
2. von anderen Agenten, die auf demselben Agentensystem ablaufen.

Der erste Punkt kann in Allgemeinheit weder verhindert noch erkannt werden. Um den Mobilen Agenten vor dem Agentensystem zu schützen, muss eine organisatorische Lösung, z.B. mit einem **Sanktionssystem**, gefunden werden.

Um einen Mobilen Agenten vor anderen Agenten „abzuschotten“, sind Entwicklungsumgebungen und Implementierungssprachen zu verwenden, die Speicherschutzverletzungen dadurch konzeptionell verhindern, dass **direkte Speicherzugriffe verboten** werden. Außerdem muss die Sicherheitsarchitektur Konzepte bieten, um die **Namensräume** von Agenten verschiedener Organisationseinheiten während der Ausführung auf einem Agentensystem zu **trennen**.

Daneben ist auch ein Laufzeitschutz zu realisieren. Der Zugriff auf Methoden des Agentensystems — zur Manipulation Mobiler Agenten — muss abgesichert werden (**Zugriffskontrolle**). Analoges gilt für Methoden des Laufzeit– oder Interpretersystems, soweit diese geeignet sind, direkt den Zustand eines Mobilen Agenten zu verändern oder zu beeinflussen.

Um das Agentensystem und die Ressourcen, die über das Agentensystem erreichbar sind, vor möglicherweise böswilligen Agenten zu schützen, ist eine **Sandbox** für Mobile Agenten zu implementieren, in der Mobile Agenten, die z.B. nicht eindeutig identifizierbar sind, eingesperrt und untersucht werden können. Die Sandbox ersetzt aber nicht das erforderliche Konzept der Zugriffskontrolle (vgl. folgenden Abschnitt).

4.4 Sicherheitskonzepte für das Relationenmodell: Aufruf– und Kommunikationsrelation

Dienstnutzer und Diensterbringer stehen in einer Aufrufrelation zueinander. Der Diensterbringer stellt dem Dienstnutzer einen Dienstzugangspunkt zur Verfügung, an dem dieser den Dienst aufrufen kann. Die Aufrufrelation kann lokal auf einem System bestehen, d.h. es erfolgt ein lokaler Informationsfluss an einer Aufrufschnittstelle, oder aber die Aufrufrelation ist entfernt und damit ein Spezialfall der Kommunikationsrelation. Dies ist der Fall, wenn sich Dienstnutzer und Diensterbringer auf unterschiedlichen Systemen befinden. Die Nutzung einer Dienstschnittstelle muss autorisiert werden. Der für den Diensterbringer verantwortliche Administrator muss festlegen und auch beschränken können, wer in welcher Art und Weise seinen Dienstzugangspunkt nutzen darf.

Rechtekonzept: allgemeine Konstruktionsprinzipien

Dieser Abschnitt befasst sich mit dem Rechtekonzept, das sich aus zwei Teilen zusammensetzt. Die Autorisierung beschäftigt sich mit der Vergabe eines Rechtes, d.h. wie muss ein Recht beschaffen sein und wie kann das Recht an ein Subjekt erteilt bzw. auch wieder entzogen werden. Die Zugriffskontrolle (Access–Control) regelt, wie diese Rechte in einem System durchgesetzt werden können. Bereits 1975 wurden in [SaSc 75] allgemeine Konstruktionsprinzipien zur Sicherung von Information definiert, die auch heute noch Gültigkeit besitzen. Ein Rechtekonzept zum Schutz der Aufruf– und Kommunikationsrelation muss folgende Grundsätze erfüllen:

- **Erlaubnisprinzip (fail–safe default)**: Alles, was nicht explizit erlaubt ist, ist verboten.
- **Vollständigkeit (complete mediation)**: Jeder Zugriff muss auf Zulässigkeit überprüft werden.
- **Minimale Rechte (least privilege, need to know):** Nur genau die Rechte, die zur Erfüllung einer bestimmten Aufgabe erforderlich sind, werden auch erteilt.

Daneben sind auch Anforderungen aus dem Anwendungsszenario des organisationsübergreifenden Managements mit Mobilen Agenten zu beachten. Durch die Beteiligung verschiedener organisatorischer Einheiten, die sich selbst verwalten, ist eine zentrale Administration nicht realisierbar. Für einen Mobilen Agenten, der zwischen diesen Domänen migriert, muss es eine Möglichkeit der lokalen Rechteanpassung oder –vergabe geben. Das Rechtesystem muss in der Lage sein, mit einer sehr hohen Dynamik umzugehen. Die beteiligten Entitäten und auch die Dienstschnittstellen sind nicht notwendigerweise alle im Vorhinein bekannt. Es können dynamisch neue Entitäten entstehen, die Dienstzugangspunkte anbieten.

4.4.1 Rechtekonzept: Zugriffskontrolle

Das Ziel der Zugriffskontrolle ist es, die Aktionen oder Operationen einer handelnden Entität zu beschränken. Die handelnde Entität wird als Subjekt bezeichnet, das Aktionen auf Objekten des Systems ausführt. Zugriffskontrollmodelle beschreiben, wie entschieden wird, ob der Zugriff bzw. die Aktion eines Subjektes auf einem Objekt zugelassen wird. Diese Zugriffskontrollmodelle, auch als Access Control Policies bezeichnet, lassen sich in vier Klassen einteilen [SaSa 94, Ecke 01]:

- **Benutzerbestimmte Zugriffskontrolle (Discretionary Access Control (DAC))**: Beim DAC–Modell darf der Eigentümer eines Objektes die Zugriffsrechte für seine Objekte frei vergeben. Mit dieser benutzerbestimmbaren Strategie können objektbezogene Sicherheitseigenschaften, aber keine globalen Eigenschaften festgelegt werden. Die Rechte können sowohl positiv als auch negativ formuliert werden. Wird beides zugelassen, kann dies zu Konflikten und Widersprüchen führen [BSJ 97]. Das Konzept der Zugriffsmatrix ist der bekannteste Vertreter des MAC–Modells. Die Zugriffsmatrix lässt sich auf unterschiedliche Arten implementieren, z.B. als Access Control List (d.h. objektbasiert) oder in Form von Capabilities (subjektbasiert). DAC
- **Systembestimmte Zugriffskontrolle (Mandatory Access Control (MAC)**: MAC–Schemas spezifizieren über systembestimmte, regelbasierte Festlegungen systemglobale Eigenschaften, d.h. die systembestimmte Regel dominiert ggf. über ein benutzerbestimmtes Recht. Die bekanntesten Vertreter der MAC–Modelle sind das Modell von Bell–LaPadula [LaPa 96a, LaPa 96b, BeLa 73] und das Chinese-Wall Modell [BrNa 89]. Beim Bell–LaPadula Modell werden alle Subjekte und Objekte in Sicherheitsklassen (z.B. vertraulich, geheim, streng geheim) eingeteilt.Die Sicherheitsklasse eines Subjektes wird dabei auch als **Clearance** bezeichnet. Es werden zwei systembestimmte Regeln definiert, die als „Read Down“ bzw. „Write Up“ Regel bezeichnet werden. **Read Down** besagt, dass die Clearance des Subjektes, das lesend auf das Objekt zugreift, größer oder gleich der Sicherheitsklasse des Objektes sein muss. Um der **Write Up** Regel zu genügen, muss die Sicherheitsklasse des Objektes größer oder gleich der Clearance des Subjektes sein, das schreibend auf das Objekt zugreift. MAC
- **Rollenbasierte Zugriffskontrolle (Role based Access Control (RBAC)**: Das rollenbasierte Modell, eingeführt in [FeKu 92], hebt die starke Bindung zwischen Subjekt und Recht auf, die bei den anderen Modellen im Vordergrund steht. Das Recht wird an die konkrete Aktivität bzw. die Aufgabe, die das Subjekt im System ausführt, gebunden. Das Recht wird an die Rolle und nicht mehr direkt an das Subjekt erteilt. Das Subjekt erhält die Rechte mittelbar über seine Rollenmitgliedschaft. RBAC
- **Informationsflussmodelle**: Bei den Informationsflussmodellen werden die Informationsflüsse zwischen Subjekten beschränkt. Es werden also nicht die Zugriffe auf die Objekte beschränkt, sondern der Umgang Informationsflussmodelle

mit der Information, die durch die Objekte repräsentiert wird. Es werden zulässige und unzulässige Informationskanäle spezifiziert. Das bekannteste Informationsflussmodell ist das Verbands–Modell [Denn 76].

Klassifikationskriterien

Diese Modelle werden von Eckert in [Ecke 01] nach folgenden Kriterien klassifiziert:

- Granularität der Subjekt– und Objekt–Modellierung
- Art des Rechtes
- Art der Zugriffsbeschränkung

Granularität der Modellierung

Die Granularität der Subjekt– bzw. Objektmodellierung unterscheidet eine grobe von einer feingranularen Modellierung. Bei einer groben Subjektmodellierung sind Aktivitäten einzelner Entitäten nicht differenziert kontrollierbar. Eine grobe Subjektmodellierung ist daher in einem Managementsystem unbrauchbar. Auch die Objekte müssen möglichst feingranular modelliert werden können, um das Prinzip der minimalen Zugriffsrechte realisieren zu können, d.h. das Zugriffsmodell muss eine anwendungsspezifische Subjekt– und eine feingranulare Objektmodellierung ermöglichen.

Art des Rechtes

Bei der Art des Rechtes werden universelle von objektspezifischen Rechten unterschieden. Ein universelles Recht wird von allgemeinen, nicht jedoch von objektspezifischen Operationen bestimmt. Ein Lese– oder Schreibrecht wäre ein Beispiel für ein universelles Recht, das für ganze Klassen von Objekten definiert ist. Demgegenüber ist das Recht, einen Mobilen Agenten zu migrieren, ein objektspezifisches Recht, da dieses nur für das Objekt Mobiler Agent definiert ist. An diesem Beispiel wird bereits ersichtlich, dass objektspezifische Zugriffsrechte benötigt werden.

Zugriffsbeschränkung

Eine Zugriffsbeschränkung wird als einfach bezeichnet, wenn nur das Zugriffsrecht (universell oder objektspezifisch) erforderlich ist, um eine Aktion auf einem Objekt auszuführen. Entsprechend ist eine Zugriffsbeschränkung komplex, wenn zusätzliche Bedingungen erfüllt sein müssen. Ein Beispiel für eine zusätzliche Bedingung wäre die Reisehistorie eines Mobilen Agenten. Ein Recht p muss an einen Mobilen Agenten in Abhängigkeit davon vergeben werden können, ob der Mobile Agent auf seiner Reise bereits Agentensysteme besucht hat, die als nicht vertrauenswürdig eingestuft werden.

Subjektindividuelle, objektspezifische und feingranulare Kontrolle mit komplexen Zugriffsbeschränkungen erforderlich

Für ein Managementsystem basierend auf Mobilen Agenten muss ein Zugriffskontrollmodell gewählt werden, das subjektindividuelle, objektspezifische und feingranulare Kontrolle mit komplexen Zugriffsbeschränkungen ermöglicht. In Abbildung 4.13 sind diese Kriterien und die entsprechenden Modelle noch einmal zusammengefasst. Werden die Bereiche markiert, die anwendungsspezifische Subjektmodellierung, feingranulare Objekte, objektspezifische Rechte und komplexe Zugriffsbeschränkungen ermöglichen, dann bleiben im Schnittpunkt dieser vier Kriterien nur noch DAC und RBAC Modelle, die sich bspw. mit Hilfe von Zugriffskontrollmatrizen bzw. Rollenmodellen realisieren lassen. Um die geforderten Kriterien erfüllen zu können, muss eines dieser Modelle oder eine Kombination daraus realisiert werden. Im Folgenden wird eine Kombination von Zugriffsmatrix und Rollenmodell favorisiert.

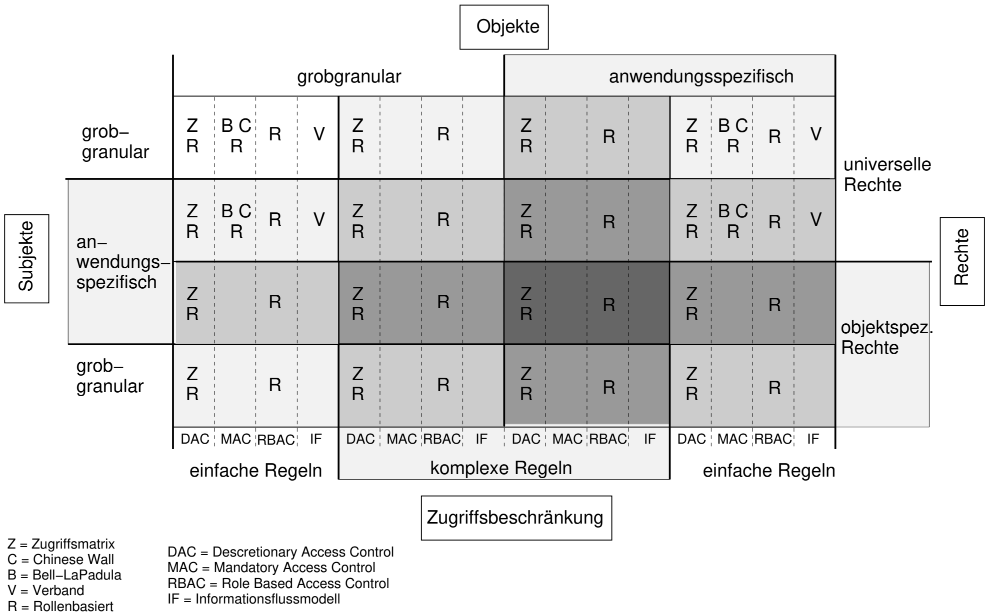


Abbildung 4.13: Klassifikationsschema für Zugriffskontrollmodelle nach Eckert [Ecke 01]

Eine Zugriffsmatrix M wird durch $|S| \times |O|$ modelliert wobei Zugriffsmatrix

- S die Menge der Subjekte ist, die zeilenweise in die Matrix eingetragen werden und
- O die Menge der Objekte, die spaltenweise in der Matrix stehen.
- In den Zellen der Matrix $M(s, o) = \{p_1, \ldots p_n\}$ sind dann die Rechte (Permissions) p_i, die das Subjekt $s \in S$ am Objekt $o \in O$ besitzt.

Das Hauptproblem bei Zugriffsmatrizen in hochdynamischen Systemen ist jedoch die enge Bindung von Rechten an Subjekte. Da häufig neue Subjekte erzeugt und bestehende gelöscht werden, muss das Rechtesystem kontinuierlich mit verändert werden.

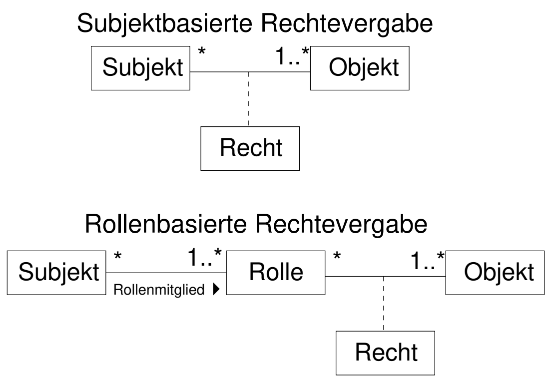


Abbildung 4.14: Vergleich subjektbasierter Rechtevergabe mit RBAC

Eine aufgabenorientierte Rechtevergabe, die durch RBAC ermöglicht wird, bricht diese enge Bindung auf. Das Recht wird nicht mehr direkt an die Subjekt–Objekt Beziehung gebunden, sondern an eine oder mehrere Rollen (vgl. Abbildung 4.14). Eine **Rolle** wird durch eine konkrete Aufgabe definiert, die durch eine Menge von Aktivitäten und Verantwortlichkeiten beschrieben werden kann. Ein Beispiel einer Rolle wäre Rolle *ManageBackup*, die für die Backups in einer bestimmten Domäne verantwortlich ist. Das Subjekt, das in dieser Rolle aktiv ist, darf bspw. (System–) Sicherungen auf bestimmten Medien anlegen, d.h. das Subjekt kann alle Rechte der Rolle ausüben.

Aktivierung

Ein Subjekt muss explizit die Rolle, in der es agieren will, aktivieren. Diese Menge der aktiven Rolle(n) eines Subjektes wird im Folgenden als **Aktivierung** bezeichnet. Ein Subjekt kann in mehreren Rollen aktiv sein und eine Rolle kann gleichzeitig von mehreren verschiedenen Subjekten aktiviert werden.

formale Beschreibung von RBAC

RBAC lässt sich, in Anlehnung an [FeKu 92, SCFY 96, Ecke 01], durch folgendes Tupel beschreiben:

$$\mathit{RBAC} = (S, R, P, O, \mathit{sa}, \mathit{pa}, \mathit{activation})$$

mit

1. S die Menge der Subjekte
2. R die Menge der Rollen
3. P die Menge der Rechte (Permissions)
4. O die Menge der Objekte
5. *sa*, *pa* und *activation* beschreiben Abbildungen bzw. Relationen.
   Die *sa*–Abbildung (Subject Assignment) beschreibt die Rollenmitgliedschaft(en) eines Subjektes $s \in S$:
   $$\mathit{sa} : S \rightarrow 2^R$$
   Die Menge, die durch *sa* festgelegt wird ($\mathit{sa}(s) = \{r_1, \ldots, r_n\}$), beschreibt, in welchen Rollen das Subjekt s aktiv werden kann.

   Die Abbildung *pa* (Permission Assignment) beschreibt die Rechtezuweisung an Rollen $r \in R$:
   $$\mathit{pa} : R \rightarrow 2^P$$
   Alle Rechte, die einer Rolle $r \in R$ zugeordnet sind, werden durch *pa* bestimmt ($\mathit{pa}(r) = \{p_1, \ldots p_n\}$). Diese Rechte können von Recht $p_i \in P$ den Rollenmitgliedern während der Aktivierung der Rolle genutzt werden.

   Die Aktivierung
   $$\mathit{activation} \subseteq S \times 2^R$$
   beschreibt Paare (s, RA). Für ein Subjekt $s \in S$ mit der Aktivierung $(s, RA) \in \mathit{activation}$ ist RA die Menge der Rollen des Subjektes s, in denen s aktiv ist. Damit besitzt s alle Rechte der Rollen $RA \in R$.

Vorteile von RBAC

Durch die konzeptionelle Trennung zwischen Subjekt und Recht wird die Administration vereinfacht. Verlässt z.B. ein Mitarbeiter (d.h. ein Subjekt) das Unternehmen, muss nicht das gesamte Rechtesystem angepasst werden, sondern lediglich die *sa*–Abbildung. RBAC ermöglicht auch eine dezentrale Administration. Jeder Betreiber eines Agentensystems kann sein eigenes Rollenmodell entwickeln, spezifizieren und mit Hilfe von RBAC durchsetzen. Wenn einer Rolle alle für ihre Aufgabe benötigten Rechte zugeteilt sind, ändern sich diese i.d.R. über einen längeren Zeitraum nicht mehr. Die Zuteilung von Subjekten zu Rollen und von Rechten zu Rollen kann unabhängig voneinander erfolgen. Durch die aufgabenorientierte Erteilung der Rechte lässt sich das need to know Prinzip sehr einfach realisieren. Eine Trennung von Subjekt und Recht ermöglicht es, dynamisch erzeugten Subjekten, z.B. in Form von neu ankommenden Mobilen Agenten, Rechte zu erteilen, auch wenn die Subjekte vorher noch nicht (vollständig) bekannt waren.

Zugriffsmatrix mit Rollen statt Subjekten

Zugriffsmatrix und RBAC–Modell werden im Folgenden kombiniert und erweitert, um die Anforderungen an das Zugriffskontrollsystem umzusetzen. Die Zeilen der Zugriffsmatrix werden durch Rollen und nicht durch Subjekte indiziert und beschreiben die *pa* Funktion. Aus der Zeile, die durch eine Rolle indiziert wird, lässt sich die Rechte–Menge der Rolle ablesen. Die Matrix $M' = |R| \times |O|$ wird Agentensystem–lokal definiert. Abhängig von der Organisationsstruktur und den zu erledigenden Aufgaben muss der Betreiber des Agentensystems ein Rollenmodell spezifizieren. Die Zuweisung von Rechten an Rollen erfolgt durch Einträge in die entsprechende Zeile der Matrix M'.

Rechte

Ein **Recht** $p \in P$ hat folgende Form:

$$p := \texttt{grant}\{\texttt{action[, action]}^*\}.$$

Dabei ist `grant` das Schlüsselwort, um ein Recht zu erteilen, und *`action`* bezeichnet die Operation, die ausgeführt werden darf. In der Kurzschreibweise kann das Schlüsselwort `grant` in der Matrix M' auch weggelassen werden und nur die Aktion angegeben werden.

sa Funktion realisiert durch Rollenzertifikate

Die Menge R der Rollen wird lokal auf dem Agentensystem definiert. Interessant ist die Realisierung der *sa* Funktion über Domänengrenzen hinweg. Ein Anwendungsfall dafür ist beispielsweise ein Mobiler Agent eines Providers, der auf dem Agentensystem des Kunden QoS–Messungen durchführen soll. Der Kunde, bzw. der Rollenadministrator des Kunden, hat für diese Aufgabe die Rolle `QoSMeter` spezifiziert. Damit der Mobile Agent des Providers seine Messungen durchführen kann, muss er dieser Rolle zugeordnet werden. Diese Rollenzuweisung (*sa* Funktion) erfolgt durch die Ausstellung eines Rollenzertifikates durch die CA des Kunden (vgl. Abschnitt 4.2.4).

Aktivierung der Rolle

Für die Aktivierung einer Rolle gibt es zwei Möglichkeiten. Im ersten Fall kennt das Subjekt die Rollenbezeichnung und teilt dem Rechtesystem mit, in welcher Rolle es aktiviert werden möchte. Die eigentliche Aktivierung wird vom Agentensystem durchgeführt.
Der zweite Fall tritt ein, wenn ein Mobiler Agent eine fremde Domäne besucht, ohne das dort gültige Rollenmodell zu kennen oder ohne im Besitz eines Rollenzertifikates zu sein. In diesem Fall muss das Agentensystem die Rollenmitgliedschaft für den Mobilen Agenten bilden und aktivieren. Dazu kann es entweder eine Default–Rolle mit minimalen Rechten geben. Oder der Mobile Agent muss im Vorhinein seine geplanten Aktionen und die Objekte, auf denen er die Aktionen durchführen will, bekannt geben. Mit diesen Informationen kann durch einen Abgleich mit den Einträgen in der Matrix das vom

Agenten implizit gewünschte Rollen–Set ermittelt werden. Das Agentensystem kann dann entscheiden, welche Rollen für den Mobilen Agenten aktiviert werden.

Im letzten Fall wird ersichtlich, dass eine statische Definition der *sa*–Abbildung nicht ausreichend ist. Es muss die Möglichkeit geben, ankommenden Mobilen Agenten neue Rollen zuzuweisen.

Beispiel eines RBAC–Modells

In Abbildung 4.15 ist exemplarisch ein Ausschnitt des resultierenden Modells dargestellt. In dem Beispiel sind vier Rollen angegeben. Der `WWWAdmin` ist zuständig für die Verwaltung und Konfiguration eines Web-Servers. Deshalb darf die Rolle die Konfigurationsdatei lesen und editieren, den WWW–Server–Prozess starten, beenden und dessen Priorität verändern (`renice`). Die Rolle, die für Sicherungen zuständig ist (`ManageBackup`), darf die angegebenen Dateien sowie den Inhalt einer Festplatte lesen und auf ein Band schreiben. An diesem Beispiel zeigt sich, dass diese Rolle eine geschriebene Sicherung nicht mehr zurückspielen kann, weil dazu die Schreibrechte auf der Festplatte bzw. den Dateien nötig wären. Dazu wird eine eigene Rolle `ManageRestore` spezifiziert. Die Rolle `QoSManager` hat als Aufgabenbereich die Überwachung der QoS–Parameter (vgl. Szenario in Abschnitt 2.2.1). Dazu werden der Rolle alle Rechte eingeräumt, um mit einem Mobilen Agenten `MessAgent` zu kommunizieren, die Daten des Agenten abzufragen sowie diesen zu suspendieren oder zu migrieren. Die Rolle `QoSMeter` ist für die eigentliche Messung des Netzdurchsatzes verantwortlich und muss deshalb lesenden Zugriff auf die Netz–Schnittstelle bekommen.

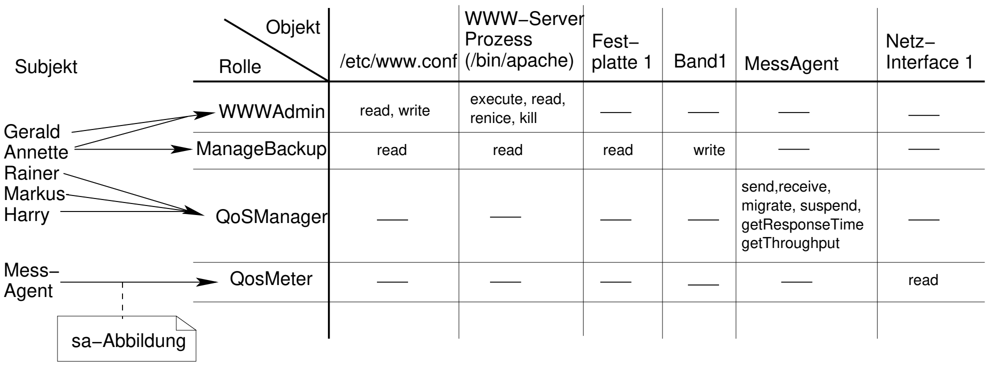


Abbildung 4.15: Beispiel einer RBAC–Zugriffskontrolle

An dem letzten Beispiel lassen sich einige typische Problemstellungen für ein organisationsübergreifendes Management mit Mobilen Agenten zeigen. Legt man das Szenario aus dem Abschnitt 2.2.1 zugrunde, wird die angegebene Matrix auf einem Agentensystem des Händlers verwaltet. Der `MessAgent` wird jedoch im Auftrag eines Providers auf dem Händlersystem zur Ausführung gebracht. Die Rolle `QoSManager` kann sowohl von Entitäten des Händlers (z.B. Mitarbeiter oder Mobiler Agent) als auch von solchen des Providers eingenommen werden. Mobile Agenten können als Objekt und auch als Subjekt auftreten. Im angegebenen Beispiel ist der `MessAgent` sowohl Subjekt als auch Objekt. In der Subjektrolle muss er mit Hilfe der *sa*–Abbildung an die Rolle `QoSMeter` gebunden werden.

Tritt der Mobile Agent des Providers in der Objektrolle auf, stellt er eine Entität mit schützenswerten Ressourcen dar. Der Provider möchte beispielsweise einem Mitbewerber, der ebenfalls einen Mobilen Agenten auf dem Agentensystem des Kunden betreibt, keine Zugriffsrechte auf die Schnittstellen des eigenen Mobilen Agenten einräumen. Das Problem dabei ist jedoch, dass der Mobile Agent des Providers auf dem Agentensystem des Kunden, unter dessen vollständiger Kontrolle, abläuft. Der Mobile Agent hat deshalb weder organisatorisch noch technisch die Möglichkeit eine eigene Zugriffskontrolle zu implementieren. Die Zugriffsrechte auf die Schnittstellen des Mobilen Agenten (z.B. `getThroughput`) müssen aber vom Provider, der für den Mobilen Agenten verantwortlich ist, und nicht nur vom Händler, als Betreiber des Agentensystems, spezifiziert werden können. Auch hier lässt sich die Einbettungsbeziehung (vgl. Abschnitt 4.1) nutzen. Der Mobile Agent verwendet die vom Agentensystem implementierte Zugriffskontrolle für den eigenen Schutz. Dazu muss der Mobile Agent über eine Schnittstelle des Agentensystems die Möglichkeiten haben, seine eigenen Zugriffsrechte mit in die *pa*–Funktion des Agentensystems zu integrieren. Das heißt der Mobile Agent darf die Objektspalte (`MessAgent`) der Matrix editieren.

Problem: Wie schützt sich der MA?

Es zeigt sich auch, dass insbesondere in Multiprovider–Szenarien und im organisationsübergreifenden Management bedingte Rechte erforderlich sind. Im angegebenen Beispiel möchte der Mobile Agent des Providers A dem Kunden, nicht aber einem Provider B, Rechte einräumen. Aus diesem Grund werden die Rechte p, die in den Feldern der Matrix M' eingetragen werden, wie folgt erweitert:

```
p :=  if Bedingung   then grant{action[, action]*}
                     [else grant{action[, action]*}]
      fi
```

Für jede Rolle und jedes Objekt kann damit die Erteilung eines Rechts an eine Bedingung geknüpft werden. Rechte werden nur aktiv gewährt (`grant`). Negative Rechte sind nicht zulässig, da eine Vermischung von positiven und negativen Rechten sehr schnell zu Widersprüchen und Inkonsistenzen führen kann (vgl. z.B. [BSJ 97]). Das Rechtesystem verfolgt damit das Erlaubnisprinzip (fail–safe default), d.h. standardmäßig wird jede Aktion verboten, außer die Aktion wurde explizit autorisiert.

Im Beispiel würde dem `MessAgent` ein bedingtes Recht in der folgenden Form:

```
if (subjekt == MessAgent) then grant{writeMatrix(r,
MessAgent)} fi
```

auf die Matrix eingeräumt. Das heißt, die Matrix M' selbst ist ein Objekt, auf das bedingte Rechte vergeben werden. Damit könnte der `MessAgent` als Subjekt „seine“ Spalte der Matrix beschreiben und damit seine eigenen Schnittstellen autorisieren.

Ein weiteres Beispiel für bedingte Rechtegewährung ist die Einbeziehung der Reisehistorie eines Mobilen Agenten in die Entscheidung über eine Rechtevergabe. Ein Agentensystem, das von einem Mobilen Agenten besucht wird, möchte diesem ein bestimmtes Recht nur gewähren, falls der Mobile Agent vorher kein Agentensystem besucht hat, dessen Vertrauenslevel unter einem bestimmten Wert liegt. Über ein bedingtes Recht lässt sich dies relativ leicht realisieren; die Bedingung wäre:
$\forall AS_i \in \mathit{history}(MA) : \mathit{trustlevel}(AS_i) >$ level
Ohne bedingte Rechte müssten diese Fälle über eine komplizierte Ausnahmebehandlung in der Zugriffskontrollkomponente realisiert werden.

4.4.2 Rechtekonzept: Autorisierung

Grundsatz: benutzerbestimmbare Autorisierung

Autorisierung befasst sich mit der Erteilung und dem Widerruf von Rechten. Die wichtigste Frage in diesem Kontext ist, wer ein Recht erteilen darf. Im beschriebenen Sicherheitskonzept wird grundsätzlich eine benutzerbestimmbare Autorisierung (DAC) verfolgt. Die Entität, die für ein Objekt verantwortlich ist bzw. der ein Objekt gehört, darf Rechte an dem Objekt vergeben.

Für das Endsystem vergibt der Betreiber die Rechte, entsprechend den Möglichkeiten des Betriebssystems. Der Betreiber des Agentensystems spezifiziert ein Rollenmodell und bindet die Rechte an die Rollen. Der Besitzer des Mobilen Agenten spezifiziert die Rechte an den Ressourcen des Mobilen Agenten.

Ausnahme: AS kann Rechte an fremden Objekten vergeben

Das Agentensystem stellt als Mittler zwischen Mobilen Agenten und dem Endsystem eine Ausnahme der benutzerbestimmbaren Autorisierung dar, d.h. das Agentensystem kann Rechte an „fremden" Objekten vergeben oder beschränken. Auf der einen Seite kann es den Zugriff auf Ressourcen des Endsystems beschränken, da diese für Mobile Agenten nur über dezidierte Schnittstellen des Agentensystems zugänglich sind und das Agentensystem die Rechte für diese Schnittstellen vergibt. Auf der anderen Seite hat das Agentensystem auch die Möglichkeit, Rechte an Ressourcen des Mobilen Agenten zu beschränken, auch wenn der Mobile Agent das Recht gewähren würde. Ein Anwendungsfall hierfür ist die Verarbeitung von sensibler Information durch den Mobilen Agenten. Wenn dieser mit vertraulichen Daten arbeitet, die das Agentensystem nicht verlassen dürfen, muss verhindert werden, dass von außen mit diesem Agenten kommuniziert wird. Beim Auftreten derartiger Konflikte hat die Autorisierung durch das Agentensystem das stärkere Gewicht. Dies spiegelt auch die Einbettungsbeziehung wider.

4.4.3 Verbindlichkeit

Die Verbindlichkeit hat das Ziel, Kontroversen über das Auftreten von Ereignissen oder Aktionen aufzulösen [X.812] bzw. justitiabel, d.h. beweisbar zu machen. In diesem Abschnitt wird die Verbindlichkeit der Aufrufrelation näher betrachtet. Verbindlichkeit bezüglich der Kommunikations– und Migrationsrelation wird in Abschnitt 4.5.4 untersucht.

Ein Ziel der Verbindlichkeit ist die Auflösung von Kontroversen über das Auslösen von Aktionen. Bezüglich der Aufrufrelation umfasst dies i.d.R. die Frage: Hat ein Subjekt eine bestimmte Methode eines Objektes genutzt?
Das zweite Ziel der Verbindlichkeit ist die Beweisbarkeit der Aktion, d.h. auch ein unbeteiligter Dritter, z.B. ein Gericht, muss überprüfen können, ob eine Aktion stattgefunden hat.

In Betriebssystemen wird eine Auditing–Funktion genutzt, um vorher definierte Methodenaufrufe der Benutzer in eine Protokoll–Datei zu schreiben. Zu einem späteren Zeitpunkt kann dann nachvollzogen werden, welcher Benutzer eine bestimmte Aktion ausgeführt hat. Bei dieser Vorgehensweise wird das Betriebssystem als Teil der Trusted Computing Base betrachtet, d.h. dem Betriebssystem und den Einträgen in den Protokoll–Dateien wird vertraut.

Auditing durch Agentensystem

Analog dazu kann das Agentensystem alle Methodenaufrufe, die von ihm „vermittelt“ werden, protokollieren. Wenn sichergestellt werden kann, dass alle Methodenaufrufe nur mittelbar über das Agentensystem möglich sind (in Abschnitt 6.2.2 wird gezeigt, wie sich dies realisieren lässt), kann das Agentensystem alle kritischen Aufrufe mitprotokollieren.

Bei dieser Lösung sind die Methodenaufrufe des Agentensystems selbst nicht überwachbar, da das Agentensystem die Protokoll–Datei selbst schreibt und verwaltet. Wegen der Einbettungsbeziehung kann ein Methodenaufruf des Agentensystems bei einem Mobilen Agenten nicht zweifelsfrei bewiesen werden. Dem Agentensystem muss also in Bezug auf eine „saubere“ Protokollierung vertraut werden.

Das größere Problem bei dieser Lösung ist jedoch die fehlende Beweisbarkeit. Die Protokoll–Dateien als solche sind nicht justitiabel, da die Betreiber des Agentensystems die Einträge fälschen könnten.

Beweisbarkeit durch Signatur der Audit–Datensätze

Eine echte Beweisbarkeit ist zu erreichen, wenn die Protokolleinträge vom Verursacher, d.h. dem Subjekt, das die Aktion ausführt, digital signiert werden. Mit dieser Signatur bestätigt das Subjekt, den Aufruf einer bestimmten Methode.

Die digitale Signatur (vgl. auch Seite 88) erfüllt, vergleichbar einer Unterschrift unter einem Dokument, folgende Funktionen:

1. Die Signatur kann nicht gefälscht werden und sie ist dauerhaft (**Perpetuierungsfunktion**). Die **Fälschungssicherheit** wird dadurch gewährt, dass der private Schlüssel verwendet wird, der nur dem Unterzeichner bekannt ist. Nachdem die CA die Zertifikate dauerhaft speichert, kann die Perpetuierungsfunktion gewährleistet werden.
2. Die Signatur ist authentisch (**Echtheitsfunktion**), d.h. die Signatur überzeugt den Empfänger, dass der Unterzeichner das Dokument unterzeichnet hat. Bei der digitalen Signatur wird dies durch das zugrunde liegende kryptographische Verfahren sichergestellt.

3. Eine Signatur kann nicht wiederverwendet oder kopiert werden (**Unikatfunktion**), sie ist Teil der signierten Daten. Entweder werden die Daten als solche mit dem privaten Schlüssel verschlüsselt (signiert), oder es wird der Hash–Wert der Daten signiert. Damit gehen die zu signierenden Daten mit in die Signatur ein.
4. Die signierten Daten können später nicht verändert werden (**Abschlussfunktion**), da sich sonst der Hash–Wert ändern und die Signatur ungültig werden würde.
5. Der Unterzeichner kann die Signatur nicht leugnen (**Beweisfunktion**). Jeder, der den öffentlichen Schlüssel des Unterzeichners kennt, kann die Signatur — auch ohne die Mithilfe des Unterzeichners — verifizieren. Da die Zertifikate und damit die öffentlichen Schlüssel über eine CA für jeden zugänglich sind, kann jeder die Signatur verifizieren.

4.4.4 Schlussfolgerungen für die Sicherheitsarchitektur

Die Ausführungsrelation befasst sich mit den direkten Interaktionen von Software–Bausteinen. Die Sicherheitsanforderungen, die hierbei bestehen, umfassen die Integrität und Vertraulichkeit der beteiligten Entitäten, insbesondere die des Mobilen Agenten als Executee, ein Rechte– und Zugriffskontrollkonzept sowie die Verbindlichkeit der durchgeführten Aktionen.

Um die Ausführungsintegrität zu gewährleisten, müssen die Agenten voreinander abzuschotten sein. Dies wird erreicht durch **Speicherschutz**, **Laufzeitschutz** und **Trennung der Namensräume**. Um einen einzelnen, möglicherweise nicht vertrauenswürdigen Agenten abzuschotten, wird eine **Sandbox** verwendet, in die der Mobile Agent „eingesperrt" werden kann. Den Mobilen Agenten vor einem böswilligen Agentensystem zu schützen ist ein — in Allgemeinheit — nicht lösbares Problem. Im Umfeld IT–Management kann jedoch über vertragliche Beziehungen der beteiligten Unternehmen eine **organisatorische Lösung** in Form eines **Sanktionssystems** etabliert werden.

Die Zugriffskontrolle für alle Objekte wird beim Agentensystem als zentrale Schnittstelle realisiert. Um Subjekt–individuelle, objektspezifische und feingranulare Kontrolle mit komplexen Zugriffsbeschränkungen zu realisieren, wird eine Kombination aus Zugriffsmatrix und **RBAC** verwendet. Dazu muss in jeder Domäne bzw. lokal auf dem Agentensystem ein **Rollenmodell** spezifiziert werden. Die Zuordnung von Subjekten zu Rollen — und damit mittelbar zu Rechten — geschieht über **Rollenzertifikate**. Diese Zertifikate werden von der Domäne ausgestellt, in der die Rollen gültig sind. Damit domänenübergreifend Mobile Agenten ausgetauscht werden können, muss es Verfahren geben, um die Rollenzertifikate ausstellen und austauschen zu können, d.h. eine domänenübergreifende CA–Infrastruktur muss aufgebaut werden. Ein Subjekt, das ein Recht nutzen will, muss in einer Rolle aktiv werden, die das entsprechende Recht besitzt. Grundsätzlich wird eine benutzerbestimmbare Rechtevergabe (**DAC**) zugrunde gelegt. Ein Mobiler Agent vergibt die Rechte an seinen Methoden, die dann stellvertretend vom Agentensystem durchgesetzt werden. Damit der Agent die Rechte vergeben kann, muss er die Möglichkeit haben, die Teile der Zugriffskontrollmatrix, die ihn betreffen, zu verändern. Damit komplexere Zugriffsbeschränkungen formalisiert werden können sind **bedingte Rechte** erforderlich.

Damit kein Mobiler Agent zu einem späteren Zeitpunkt das Ausführen einer kritischen Aktion leugnen kann, muss ein Verfahren zur Sicherstellung der **Verbindlichkeit** etabliert werden. Für Umgebungen mit niedrigeren Sicherheitsanforderungen ist es ausreichend, wenn das Agentensystem über eine **Auditing**–Funktion die Methodenaufrufe protokolliert. In Umgebungen mit höchsten Sicherheitsanforderungen kann es erforderlich sein, dass das Subjekt die Protokolleinträge für kritische Aktionen **digital signiert**, um auch vor unbeteiligten Dritten zu bezeugen, dass die Aktion ausgeführt wurde.

4.5 Sicherheitskonzepte für das Relationenmodell: Kommunikations– und Migrationsrelation

Die Kommunikationsrelation ist durch den Austausch von Informationen bzw. Nachrichten zwischen Entitäten definiert. In einem System Mobiler Agenten gibt es eine ausgezeichnete Kommunikationsrelation zwischen zwei Agentensystemen, bei der die Nachrichten, die übertragen werden, MA–Instanzen beinhalten. Diese Migrationsrelation stellt auch einen entscheidenden und wichtigen Zustandsübergang im Lebenszyklus eines Mobilen Agenten dar (vgl. Abschnitt 2.5). Im nächsten Abschnitt wird die Vertraulichkeit der Migrations– und der Kommunikationsrelation untersucht. Daran anschließend werden die Daten, die ein Mobiler Agent transportiert, klassifiziert und Protokolle zur Integritätssicherung dieser Daten vorgestellt. Diese Protokolle können auch verwendet werden, um eine Reisehistorie des Mobilen Agenten aufzubauen. Im letzten Teilabschnitt wird untersucht, wie die Verbindlichkeit bei der Migration und der Kommunikation gewährleistet werden kann.

4.5.1 Vertraulichkeit

Alle Kommunikationsrelationen zwischen Entitäten des Managementsystems müssen vor dem Ausspähen von vertraulichen Informationen geschützt werden. Jeder Entität muss die Möglichkeit gegeben werden, die Vertraulichkeit ihrer Kommunikation sicherzustellen. Auch die Übertragung von Mobilen Agenten, d.h. das Versenden von MA–Gattung und MA–Instanz hat vertraulich zu erfolgen. Dies lässt sich durch eine Verschlüsselung der Kommunikation sicherstellen.

verschlüsselte Kommunikation

Die an der Kommunikation beteiligten Entitäten müssen dazu gemeinsame Schlüssel besitzen oder auf sicherem Wege austauschen. Für diesen Zweck kann eine CA–Infrastruktur genutzt werden. Falls A mit B kommunizieren möchte, kann A das Zertifikat von B bei der CA abrufen und gelangt damit in den Besitz des öffentlichen Schlüssels von B. Der öffentliche Schlüssel B_P von B könnte direkt zum Verschlüsseln der Nachricht m verwendet werden. Da die Verschlüsselung mit asymmetrischen Verfahren sehr viel weniger performant ist als Verschlüsselung mit symmetrischen Verfahren, wird B_P nur dazu verwendet, einen gemeinsamen Schlüssel S für ein symmetrisches Verfahren zu verschlüsseln $(B_P[S])$, die eigentliche Nachricht m, die sehr viel länger als S sein kann, wird dann mit diesem Schlüssel gesichert $(S[m])$. A konkateniert die verschlüsselte Nachricht und den verschlüsselten symmetrischen Schlüssel und überträgt diese Daten $(S[m] \bullet B_P[S])$ an B. Nur B, der im Besitz des privaten Schlüssels B_S ist, kann S und damit die Nachricht m entschlüsseln (vgl. Abbildung 4.16).

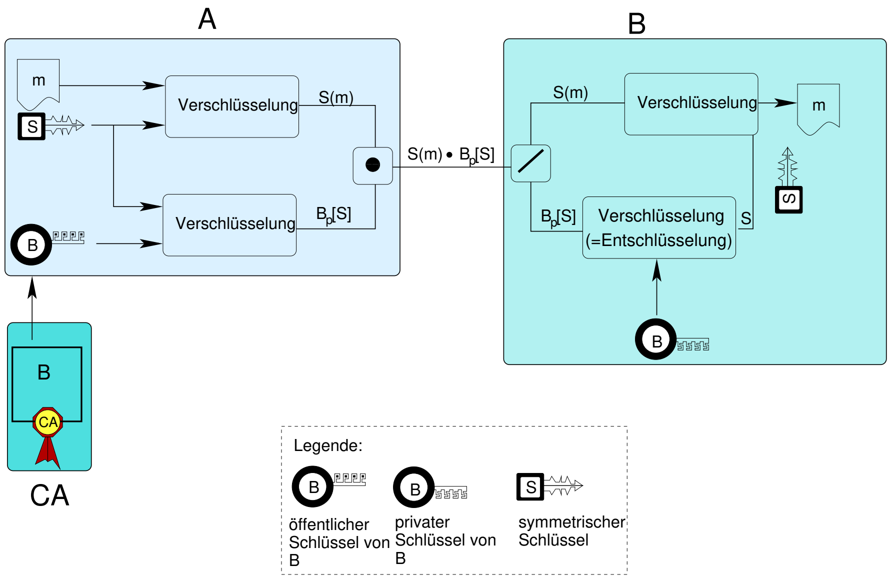


Abbildung 4.16: Vertrauliche Kommunikation zwischen Entitäten

Um die Realisierbarkeit von Vertraulichkeit durch Verschlüsselung im konkreten Szenario bewerten zu können, ist zu untersuchen, welche Entitäten miteinander Kommunikationsrelationen eingehen und ob die entsprechende Kommunikation durch Verschlüsselung gesichert werden kann. In Tabelle 4.7 sind Beispiele für mögliche Kommunikationsbeziehungen zwischen den Entitäten angegeben.

Von besonderem Interesse sind hierbei alle, in der Tabelle schattiert dargestellten Kommunikationsrelationen, an denen ein Mobiler Agent beteiligt ist. Die interessierenden Fragestellungen hierbei sind:

- Ist ein Mobiler Agent überhaupt in der Lage, seine Kommunikation zu sichern?
  Da die Implementierung eines Mobilen Agenten relativ klein gehalten

Initiator \ Partner	Anwender	Mobiler Agent	Agentensystem	Endsystem
Endsystem	Statusmeldung über Start des AS	keine direkte Kommunikation möglich	Terminierung des AS (lokal, entfernt)	"normaler" Netzverkehr
Agentensystem	Informationen über Zustand des AS	Methodenaufruf (lokal oder entfernt)	Migration von MA, Statusinformationen	Methodenaufruf (Betriebssystem spezifisch)
Mobiler Agent	Ereignis– oder Ergebnismeldung	Kooperation der MAs	Methodenaufruf (lokal oder entfernt)	nicht direkt möglich nur mittelbar über das AS
Anwender		Methodenaufruf (Managementfunktion)	Start, stop von MA (lokal und entfernt)	Start, stop des AS (lokal und entfernt)

Tabelle 4.7: Beispiele für Kommunikationsrelationen zwischen Entitäten

werden soll, implementiert der Mobile Agent im Allgemeinen keine Verschlüsselungsfunktionalität.

- Wie erhält der Mobile Agent seine geheimen Schlüssel?
  Der Mobile Agent muss mit jedem Kommunikationspartner ein Verschlüsselungsverfahren vereinbaren und entsprechende Schlüssel austauschen.
- Wie kann der Mobile Agent seine geheimen Schlüssel während seiner Reise vor feindlichen Agentensystem geheim halten?
  Daten, wie z.B. private Schlüssel, die der Mobile Agent transportiert, könnten vom Quell–AS verschlüsselt werden. Damit ist der Schlüssel aber für den Mobilen Agenten auch nicht mehr nutzbar, da er ihn auf dem Ziel–AS nicht mehr entschlüsseln kann. Und selbst wenn er dies könnte und den Schlüssel auf dem Ziel–AS verwenden würde, könnte das Ziel–AS den Schlüssel bei der Verwendung kopieren oder „mitlesen". Der Schlüssel wäre damit kompromittiert und das Ziel–AS könnte z.B. in die Rolle des Quell–AS schlüpfen (Maskerade).

Eine Aushandlung von Schlüsseln für alle Kommunikationspartner vor der Migration und ein Transport aller Schlüssel durch den Mobilen Agenten ist daher nicht möglich.

Auch in diesem Fall kann die Einbettungsbeziehung zwischen Agentensystem und Mobilem Agenten genutzt werden. Der Mobile Agent muss seinem Agentensystem bis zu einem gewissen Grad (Vertrauenslevel) vertrauen. Daher kann der Mobile Agent auch Verschlüsselungsfunktionen, die ihm vom Agentensystem zur Verfügung gestellt werden, nutzen und muss diese nicht selbst implementieren.

Das Agentensystem stellt jedem Mobilen Agenten als LCA ein Schlüsselpaar und ein Zertifikat aus (vgl. Abschnitt 4.2.4). Mit Hilfe dieses Zertifikates kann der Mobile Agent mit jedem anderen Kommunikationspartner einen Sitzungsschlüssel vereinbaren. Durch die Abstützung auf das Agentensystem und die Einbettungsbeziehung folgt, dass das Agentensystem die vertrauliche Kom-

Auch MA kann vertraulich kommunizieren

munikation des Mobilen Agenten mitlesen kann. Dies lässt sich aber wegen der in Abschnitt 4.1 angegeben Gründe prinzipiell nicht verhindern. Ein Mobiler Agent muss das Agentensystem, auf dem er sich gerade befindet, als Trusted Third Party akzeptieren.

Die Sitzungsschlüssel werden vom Mobilen Agenten nie transportiert. Sie, das Schlüsselpaar und das entsprechende Zertifikat werden ungültig, sobald der Mobile Agent das aktuelle Agentensystem verlässt. Das Ziel–AS muss dem Mobilen Agenten ein neues Zertifikat ausstellen und der Mobile Agent dann mit seinen Kommunikationspartnern neue Sitzungsschlüssel aushandeln.

Für alle anderen Entitäten, d.h. für Endsysteme, Agentensysteme und Anwender, ist ein auf Zertifikaten basierter Austausch von Schlüsseln und damit eine vertrauliche Kommunikation problemlos möglich.

4.5.2 Integrität

Die Integrität bezüglich Kommunikations– und Migrationsrelation ist gewahrt, wenn die Veränderung von Daten während der Übertragung verhindert oder erkannt werden können. Daten, die über diese Relationen ausgetauscht werden, sind zum einen Kommunikationsdaten, zum anderen MA–Gattungen und MA–Instanzen.

Integrität: MA–Gattung und Nachrichten

MAC für MA–Gattung und Kommunikation

Die Integrität der Kommunikationsdaten und MA–Gattungen kann leicht erreicht werden. Dazu wird über die zu übertragenden Daten m ein Hash–Wert ($H(m)$) berechnet und dieser Hash–Wert wird mitübertragen ($m \bullet H(m)$). Ein Angreifer, der in der Lage ist, die Daten zu verändern, könnte natürlich auch den Hash–Wert entsprechend verändern. Aus diesem Grund wird der Hash durch ein Passwort p oder einen Schlüssel Sgesichert. Dazu wird die Nachricht vor der Berechnung des Hash–Wertes mit p bzw. S konkateniert und der entsprechend modifizierte Hash–Wert mit übertragen (z.B. $m \bullet H(m \bullet S)$). Derartig modifizierte Hashes werden auch als **Message Authentication Codes (MAC)** bzw. als **Hashed Message Authentication Codes (HMAC)** bezeichnet ($MAC_S(m) = H(m \bullet S)$).

Nur der rechtmäßige Empfänger, der auch das gemeinsame Geheimnis (p bzw. S) kennt, ist in der Lage, den MAC zu berechnen und mit dem übertragenen Wert zu vergleichen. Wurden die Daten auf dem Weg vom Sender zum Empfänger verändert, sind die beiden Werte unterschiedlich.

Eine weitere Möglichkeit, die diskutiert wird, um die Integrität von Daten zu sichern, ist die Verschlüsselung. Die Integrität ist bereits gewährleistet, wenn eine Veränderung der Daten während der Übertragung zuverlässig verhindert werden kann. Wenn alle Informationen, die bei einer Kommunikations– bzw. Migrationsrelation übertragen werden, verschlüsselt sind, kann ein Angreifer die Daten nicht in seinem Sinne manipulieren, da er deren Inhalt nicht im Klartext lesen kann. Allerdings lässt sich eine Manipulation bzw. ein Denial of service Angriff nicht vollständig verhindern, da für den Angreifer die Möglichkeit besteht, die Daten „blind" zu verändern. In diesem Fall müsste die Veränderung für den Empfänger erkennbar sein, um die Integrität sicherstellen zu können. Wie sich eine Manipulation des verschlüsselten Textes (**Ciphertext**) auf den daraus wieder entschlüsselten Klartext (**Plaintext**) auswirkt, hängt vom verwendeten Verschlüsselungsverfahren und dem Betriebsmodi des Verfahrens ab. Wird z.B. der **Data Encryption Standard (DES)** verwendet, breitet sich der Fehler aus (error extension). Im Electronic Codebook Mode (ECB) führt die Veränderung eines Bits im Ciphertext zu einem Fehler im resultierenden Plaintext, der einen ganzen Block (64 Bit) umfasst. Im Cipher Block Chaining Mode (CBC) sind von einem Bit–Fehler im Ciphertext zwei Blöcke des resultierenden Plaintexts betroffen, jedoch keine weiteren Blöcke, d.h. CBC und ECB sind fehlerbehebend (self–recovering),

Integritätssicherung durch Verschlüsselung nur bedingt möglich

Neben den Bit–Fehlern sind auch Synchronisationsfehler (synchronization error), die durch das Einfügen oder das Löschen von Bits im Ciphertext verursacht werden, zu betrachten. Falls im CBC Modus, durch Manipulationen eines Angreifers, ein Synchronisationsfehler auftritt, sind alle Blöcke des Ciphertextes während der Entschlüsselung verschoben und es wird total fehlerhafter Plaintext generiert [Schn 96]. Im Falle eines Synchronisationsfehlers ist dieser für den Empfänger bei der Entschlüsselung erkennbar. Um einen Bitfehler zu erkennen, muss der Plaintext semantisch untersucht werden, d.h. eine automatische Fehleranalyse ist hier nicht möglich.

Da keine allgemeingültige Aussage getroffen werden kann, inwieweit Veränderungen im Ciphertext bei der Entschlüsselung automatisch erkannt werden können, ist Verschlüsselung allein nur bedingt für die Sicherung der Integrität zu verwenden.

Integrität + Vertraulichkeit: Hashes + Verschlüsselung für MA–Gattung und Kommunikation

Um eine zuverlässige Integritätssicherung sowie Vertraulichkeit für die Übertragung der Kommunikationsdaten und MA–Gattungen zu gewährleisten, sind MACs oder auch ungesicherte Hashes in Kombination mit einer Verschlüsselung der Daten zu verwenden. Im letzten Fall wird ein Hash–Wert über die Nachricht berechnet ($H(m)$), mit der Nachricht konkateniert und der resultierende String verschlüsselt ($S[m \bullet H(m)]$).

Integrität: MA–Instanz

Die MA–Instanz ist wegen ihrer Fähigkeit zur Mobilität gesondert zu betrachten. Die Integrität der MA–Instanz muss während ihrer gesamten „Reise" sichergestellt werden können. Das Ziel dabei ist, bei Angriffen den Schuldigen identifizieren zu können. Eine MA–Instanz kann neben der Manipulation während der Migration auch von einem feindlichen Agentensystem während des Aufenthaltes dort verändert werden. Insbesondere der Fall der Manipulation durch das Agentensystem kann wegen der vollständigen Kontrolle des Agentensystems über den Agenten (vgl. Abschnitt 4.1) im Allge-

Integrität der MA–Instanz während der gesamten Reise meinen nicht verhindert und auch nicht erkannt werden. Daher steht bei der Integritätssicherung der MA–Instanz die Erkennung der Manipulation der Daten, die der Mobile Agent schon auf ein böswilliges Agentensystem mitgebracht hat, im Vordergrund. Es muss zumindest nach der Rückkehr des Agenten von seiner Reise erkennbar sein, dass seine Daten verändert wurden und ab welchem Punkt oder für welchen Abschnitt der Reise den Daten nicht mehr vertraut werden kann.

Die Daten, die eine Agenten–Instanz ausmachen, unterliegen unterschiedlich häufigen Änderungen. Dabei stellen sich die beiden Extreme wie folgt dar:

1. Daten, die einmal gesetzt und dann nie wieder verändert werden (**unveränderliche Daten**) und
2. Daten, die laufend verändert werden (**veränderliche Daten**).

Die beiden Klassen lassen sich dann jeweils noch in **öffentliche**, d.h. für jeden zugängliche, und private, d.h. **vertrauliche Daten** unterteilen. Diese Klassi-

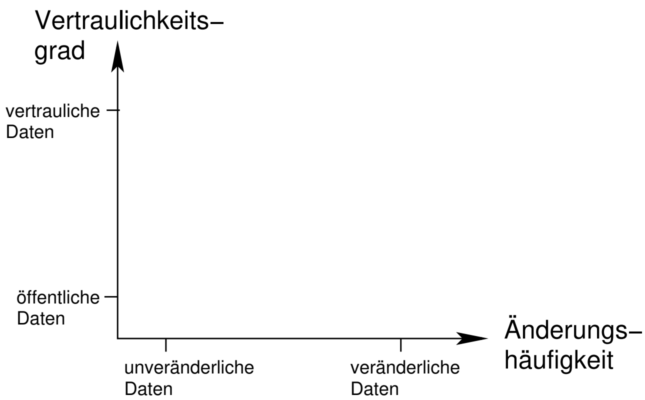


Abbildung 4.17: Klassifikation von Instanz–Daten eines Mobilen Agenten

fikation spannt einen Raum auf (vgl. Abbildung 4.17), in dem es auch Mischformen geben kann, wie z.B. Daten, die zwar nicht völlig unveränderlich sind, die sich aber relativ selten ändern und die nicht vollständig öffentlich, sondern nur für eine bestimmte Gruppe von Entitäten zugänglich sein soll. Sind Verfahren zur Integritätssicherung für die angegeben Datenklassen gefunden, lassen sich diese auch auf Mischformen anpassen.

Integrität unveränderlicher Daten einer MA–Instanz

Unveränderliche Daten werden im Lebenszyklus eines Agenten einmal gesetzt und dann nie wieder verändert. Diese Daten können an einem beliebigen Punkt der Reise des Mobilen Agenten gesetzt werden und der Mobile Agent muss sie nicht seit Beginn der Reise bei sich tragen.

Zur Integritätssicherung kann ein kryptographischer Hash–Wert berechnet werden. Da derjenige, der die Daten später nützt und die Integrität verifizieren will, nicht notwendigerweise bei der Erzeugung der Daten bereits bekannt ist, können keine MACs verwendet werden, da mit einem unbekannten Datennutzer kein geheimer Schlüssel vereinbart werden kann. Aus diesem Grund wird der Hash–Wert (H_m) der Daten m vom Erzeuger E, mit seinem privaten Schlüssel signiert $(E\{H_m\})$. Mit dem Hash kann jede Veränderung der Daten erkannt und durch die digitale Signatur zusätzlich der Erzeuger der Daten eindeutig bestimmt werden (Authentisierung des Datenursprungs und Non–Repudiation Funktion). Bei öffentlich zugänglichen Daten wird dieser signierte Hash mit den Daten konkateniert $(m \bullet E\{H_m\})$. Dann kann jeder die Daten lesen und nutzen, jeder kann die Integrität und den Ursprung der Daten ermitteln. Dazu muss der Datennutzer den entsprechenden Hash–Wert nochmals berechnen, den mitübertragenen Wert mit dem öffentlichen Schlüssel des Erzeugers entschlüsseln und beide Werte vergleichen. Über die Verifikation des Zertifikates kann er sich auch sicher sein, dass die Daten von E erzeugt wurden.

signierte Hashes für MA–Instanz Daten

Sollen die Daten nur einem bestimmten Nutzer zugänglich sein (vertrauliche unveränderliche Daten), wird dasselbe Verfahren zur Integritätssicherung verwendet und die Nachricht wird zusätzlich noch für den Nutzer N verschlüsselt $(S(m \bullet E\{H_m\}) \bullet N_p[S])$. Falls Anwender oder ein Agentensystem die Daten erzeugen oder nutzen sollen, ist eine digitale Signatur bzw. eine Verschlüsselung problemlos möglich, da beide Entitäten im Besitz von längerfristig gültigen Zertifikaten sind bzw. auch die öffentlichen Schlüssel über die CA zugänglich sind.

Verschlüsselung vertraulicher Daten

Der Fall, dass ein Mobiler Agent die Daten erzeugt oder ein anderer Mobiler Agent die (vertraulichen) Daten nutzen soll, ist gesondert zu betrachten. In beiden Fällen besteht das Problem, dass die Zertifikate der Mobilen Agenten maximal solange gültig sind, wie die Mobilen Agenten auf den entsprechenden Agentensystem bleiben (vgl. Abschnitt 4.2.3)

Problem: beschränkte Gültigkeit von MA Zertifikaten

Zur Sicherung der Integrität der von einem Mobilen Agenten erzeugten Daten, gibt es zwei Möglichkeiten:

1. Stellvertretende Signatur durch das Agentensystem
2. Eigene Signatur durch den Mobilen Agenten

Im letzten Fall verwendet der Mobile Agent den vom Agentensystem als LCA ausgestellten privaten Schlüssel für die Signatur. Der Vorgang der Integritätssicherung unterscheidet sich in diesem Fall nicht von der Vorgehensweise bei anderen Entitäten. Der Mobile Agent tritt selbst als Erzeuger der Daten auf.

Integritätssicherung durch MA selbst

Wegen der begrenzten Gültigkeit und wegen des lokalen Charakters des privaten Schlüssels sowie des Zertifikates können, bei der Verifikation der Signatur und damit auch des Hashes, Probleme auftreten. Das Zertifikat des Mobilen Agenten $A.X$ auf dem Agentensystem X ist maximal solange gültig, wie sich A auf X befindet. Bei einer Migration auf das Agentensystem Y ändert sich der Name des Agenten in $A.Y$ und das von X für $A.X$ ausgestellte Zertifikat $C^X_{A.X}$ wird von X widerrufen. Ein Nutzer von Daten, die von $A.X$ signiert

Beim Verlassen des AS werden MA–Instanz–Zertifikate widerrufen wurden, muss, um den Hash und die Signatur verifizieren zu können, in den Besitz von $C^X_{A.X}$ gelangen. Dazu könnte sich der Nutzer an X als ausstellende LCA wenden. Das Zertifikat und der entsprechende öffentliche Schlüssel von $A.S$ müssen aber auch noch zugänglich sein, wenn A das Agentensystem X bereits verlassen hat und $C^X_{A.X}$ bereits widerrufen wurde. Die lokal und zeitlich sehr begrenzt gültigen, von LCAs ausgestellten Zertifikate müssen also über einen längeren Zeitraum für alle Entitäten des Systems zugänglich sein.

Würde der Mobile Agent nur Daten signieren, die er selbst transportiert, ließe sich eine obere Schranke für die Aufbewahrungsfrist der MA–Instanz–Zertifikate angeben. Die Zertifikate könnten spätestens nach der Terminierung des Mobilen Agenten A gelöscht werden, weil damit auch die Daten, die von A signiert wurden, untergehen würden. In der Praxis kann diese obere Schranke weder verwendet noch angegeben werden, da ein Mobiler Agent A Daten erzeugen und signieren kann, die dann von einem Mobilen Agenten B transportiert werden. Die Zertifikate müssen deshalb über die Lebensdauer von A hinaus, zugänglich bleiben. Zur Speicherung könnte entweder das Agentensystem als LCA oder das Trust Center genutzt werden. Sinnvoll erscheint ein hybrider Ansatz.

Aufbewahrungsfrist der Zertifikate viel größer als Lebensdauer der MA–Instanzen

Das Agentensystem als Software–Baustein besitzt eine begrenzte Lebensdauer. Ein Absturz oder Neustart des Endsystems führt zur Beendigung des darauf ablaufenden Agentensystems. Um die Zertifikate sicher speichern zu können, müsste jedes Agentensystem eine eigene Zertifikats–Datenbank besitzen. Um auch die AS–Implementierung schlank zu halten, werden Zertifikate vom Agentensystem nicht dauerhaft gespeichert. Während der Lebensdauer des Agentensystems werden alle von ihm ausgestellten Zertifikate in einem lokalen Cache gespeichert. Alle Zertifikate, die für Signaturen verwendet wurden, werden vom Agentensystem an die CA zur langfristigen Speicherung übertragen. Da die CA auch das Zertifikat des Agentensystems speichert, kann eine Signatur auch nach der Terminierung des Agentensystems noch korrekt verifiziert werden. Ein Datennutzer kann sich damit entweder an das Agentensystem direkt oder an das Trust Center wenden, um ein Zertifikat abzurufen. Dieser Ansatz erfordert eine mächtige und skalierbare CA–Infrastruktur, die in der Lage ist, die doch sehr große Zahl an Zertifikaten zu speichern und den Nutzern effizient und schnell zugänglich zu machen.

Integritätssicherung stellvertretend durch AS

Um die Anzahl der zu speichernden Zertifikate zu reduzieren, kann der 1. Ansatz gewählt werden. Dabei tritt das Agentensystem in einer Notariatsfunktion für den Mobilen Agenten auf. Das Agentensystem X signiert die Daten stellvertretend für den Mobilen Agenten $A.X$. Die Zuordnung der Daten zum Mobilen Agenten als Erzeuger geschieht in diesem Fall über die Verbindung des MA–Identifikators mit den Daten. Das Agentensystem stellt dem Agenten eine **Urkunde** $u = m \bullet DN(A.X)$ aus, die sich aus den zu sichernden Daten, dem Distinguished Name des Erstellers, sowie evtl. zusätzlichen Daten wie z.B. dem Erzeugungsdatum o.ä. zusammensetzen. Diese Urkunde u wird dann von X signiert $(u \bullet X\{u\})$ und damit die Richtigkeit der Daten zugesichert.

Integrität veränderlicher Daten einer MA–Instanz

Die Integrität von Daten, die sich wirklich ständig ändern, lässt sich nicht sicherstellen, da bei jeder Änderung der berechnete Hash–Wert ungültig wird. Bei Daten, die sich laufend ändern, handelt es sich i.d.R. um Hilfsvariablen oder Zwischenergebnisse von Berechnungen auf einem Agentensystem. Deren Integrität kann wegen der Einbettungsbeziehung nicht gesichert werden.

Daneben gibt es noch veränderliche Daten, die während der Reise über mehrere Agentensysteme genutzt und auch verändert werden. Ein typisches Szenario bei dem diese Datenart anfällt, ist ein Mobiler Agent, der von einer Entität auf dem Agentensystem AS_0 instantiiert wird und sich von Agentensystem zu Agentensystem bewegt, um Daten zu ermitteln. Die Entität, die den Mobilen Agenten erstmalig instantiiert wird im Folgenden als Besitzer O (Owner) des Agenten bezeichnet. Am Ende der Reise liefert der Mobile Agent auf dem Quell–AS die Ergebnisse seiner Berechnungen seinem Besitzer O zurück. Der Mobile Agent besucht also der Reihe nach die Agentensysteme $AS_0, AS_1, \ldots, AS_n$, um dort die Daten $d_1, d_2, \ldots, d_n$ zu ermitteln. Dabei gilt $AS_{n+1} = AS_0$ (vgl. Abbildung 4.18). Die Reise eines Agenten ist dabei vorher nicht vollständig bestimmt, da der Agent lokale Entscheidungen, über das als nächstes zu besuchende Agentensystem, treffen kann. Ein böswilliges Agentensystem, das vom Mobilen Agenten besucht wird, hat die Möglichkeit, Daten zu verändern, einzufügen oder zu löschen.

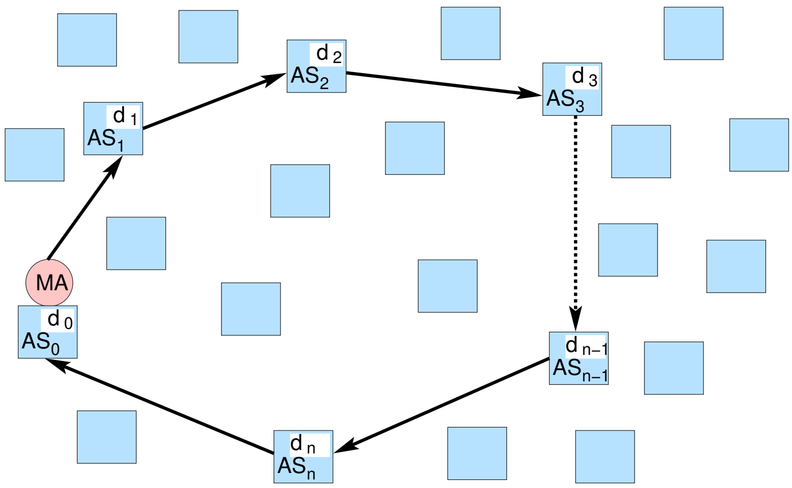


Abbildung 4.18: Reise eines Mobilen Agenten, um Datensätze d_i zu ermitteln

Um die Integrität von veränderlichen Daten zu sichern, müssen sie auf unveränderliche Daten abgebildet werden. Verfahren, die hierzu verwendet werden, sind Listen von elementweise unveränderlichen Daten und Integritätsprotokolle für diese Listen nach Yee, Karjoth, Asokan und Gülcü. Yee hat in Anlehnung an den MAC den Begriff des **PRAC (Partial Result Authentication Code)** zur Integritätssicherung von Teilergebnissen (d.h. von veränderlichen Daten) geprägt und dazu Listen von elementweise unveränderlichen Daten verwendet [Yee 97]. Karjoth et al. haben diesen Ansatz

veränderliche Daten werden auf unveränderliche abgebildet erweitert [KAG 98]. Das Ziel der Protokolle ist, dem Besitzer des Mobilen Agenten die Möglichkeit zu geben, nach der Rückkehr des Mobilen Agenten festzustellen, ob ein böswilliges Agentensystem Daten verändert oder gelöscht hat.

Eigenschaften der KAG Protokolle

Karjoth et al. definieren vier Protokolle (P1 bis P4) mit denen sich folgende Eigenschaften zusichern lassen:

1. **Vertraulichkeit** der Daten (Data Confidentiality): Nur der Besitzer des Mobilen Agenten kann die Listeneinträge lesen.
2. **Non Repudiation**: Der Erzeuger der Daten kann später seine Urheberschaft nicht leugnen.
3. **Strenge Vorwärts–Integrität** (**Strong Forward Integrity**): Die einzelnen Listenelemente können nach dem Festschreiben nicht mehr modifiziert werden, die Liste kann aber weitergeführt werden. Das Agentensystem AS_m kann seine Daten d_m an die Liste anhängen, ohne d_i mit $i < m$ modifizieren zu können. Dies ist die Eigenschaft, die einen PRAC ausmacht und ihn von einem MAC unterscheidet.
4. **Öffentliche Verifizierung der Vorwärts–Integrität** (**Publicly Verifiable Forward Integrity**): Jede Entität kann während der Reise des Mobilen Agenten die Integrität der bisherigen Liste überprüfen. Auch der Mobile Agent selbst kann die Integrität seiner eigenen Liste überprüfen, um dann ggf. seine Reise abzubrechen.
5. **Stabilität gegenüber nachträglicher Einfügung**: An keinem Punkt der Reise können Daten in die Liste eingefügt werden. Daten können immer nur am Ende der Liste angefügt werden. Lediglich der Besitzer des Mobilen Agenten kann nachträglich Daten hinzufügen.
6. **Stabilität gegenüber Verkürzung**: Die Liste kann von einem Agentensystem AS_m nur verkürzt werden, wenn es mit einem anderen böswilligen Agentensystem zusammenarbeitet.
7. **Vorwärts–Anonymität** (**Forward Privacy**): Die Identität des Erzeugers der Daten kann nur vom Besitzer des Mobilen Agenten, nicht jedoch von anderen Entitäten gelesen werden.

Die Protokolle werden im Folgenden mit KAG–P1 bis KAG–P4 bezeichnet. Tabelle 4.8 fasst die verwendete Notation zusammen (vgl. auch Tabelle 4.4). KAG–P1 und KAG–P2 setzen eine Public–Key Infrastruktur und damit eine CA voraus. KAG–P3 und KAG–P4 arbeiten ohne CA; allerdings muss der öffentliche Schlüssel des MA–Besitzers allen besuchten Agentensystemen bekannt sein. Außerdem muss zwischen den verschiedenen Agentensystemen eine vertrauliche Kommunikation möglich sein. Da für die vorgestellte Sicherheitsarchitektur auch an anderen Stellen (vgl. z.B. Abschnitt 4.2.3 oder 4.4) eine CA vorausgesetzt wird, bietet sich die Verwendung von KAG–P1 bzw. KAG–P2 an. Mit KAG–P1 und KAG–P2 lassen sich die Eigenschaften 1 bis 7 realisieren. Die öffentliche Verifizierbarkeit der Liste (Eigenschaft 4) kann nur mit KAG–P1 realisiert werden. Vorwärts–Anonymität (Eigenschaft 7) lässt sich nur mit KAG–P2 erreichen.

$AS_i, 1 \leq i \leq n$	auf der Reise besuchte Agentensysteme
$AS_0 = AS_{n+1}$	Ausgangs–Agentensystem des Mobilen Agenten; Startpunkt der Reise
d_0 $d_i, 1 \leq i \leq n$	vom Besitzer des Agenten erzeugte Daten Daten auf dem Agentensystem AS_i erzeugt
D_i	von AS_i gekapselte (geschützte) Daten
h_i	von AS_i erzeugter Hash–Wert
$D_0, D_1, D_2, \ldots, D_n$	Liste von elementweise unveränderlichen Daten
r_i	Zufallszahl, generiert von AS_i
$O_p[m]$	Nachricht m, verschlüsselt mit dem öffentlichen Schlüssel des Besitzers O des MA
$AS_{i\,s}[m] = AS_i\{m\}$	von AS_i erstellte digitale Signatur von m
Alice $\longrightarrow$ *Bob* : m	Alice schickt die Nachricht m an Bob

Tabelle 4.8: Notation für Integritätsprotokolle nach Karjoth et al.

Integritätsprotokoll KAG–P1 für Listen von elementweise unveränderlichen und vertraulichen Daten

Hash Chains als Verkettungsrelation

Die Grundidee der KAG–Protokolle sind verkettete Hash–Werte, so genannte **Hash Chains**. Die Hash Chains sind Listen von Daten D_i. Diese werden im Protokoll als **gekapselte Daten** bezeichnet. Bei der Berechnung der Hash Chains werden die Daten des soeben besuchten Agentensystems mit dem Identifikator des als nächstes zu besuchenden Agentensystems verknüpft. Dieser Vorgang wird im Protokoll **Verkettungsrelation** genannt. Dadurch wird es einem Agentensystem unmöglich gemacht seine Daten zu einem späteren Zeitpunkt zu modifizieren. Die Hash Chains realisieren damit die PRAC–Eigenschaft der strengen–Vorwärts–Integrität.

Im Folgenden wird der Ablauf von KAG–P1 beschrieben:

Protokollablauf KAG–P1

- gekapselte Daten:

$$D_i = AS_i\{O_p[d_i \bullet r_i] \bullet h_i\},\ 0 \leq i \leq n$$

- Verkettungsrelation:

$$h_0 = H(r_0 \bullet DN(AS_1))$$

$$h_i = H(D_{i-1} \bullet DN(AS_{i+1})),\ 1 \leq i \leq n$$

- Protokoll:

$$AS_i \longrightarrow AS_{i+1} : \prod_{k=0}^{i} D_k, 1\ \leq i \leq n$$

Besitzer erzeugt Anker der Hash–Chain

Der Besitzer erzeugt zu Beginn eine Zufallszahl r_0, mit der der Identifikator des als nächstes zu besuchenden Agentensystems konkateniert wird. Darüber wird der Hash–Wert berechnet. Der resultierende Wert h_0 ist der Ausgangspunkt (Anker) der Verkettungsrelation. Danach erzeugt der Besitzer einen Datensatz d_0 und verschlüsselt diesen sowie r_0 mit seinem eigenen öffentlichen Schlüssel.

Die Zufallszahl r_0 ist ein probabilistischer Faktor zum Schutz vor Choosen Plaintext Angriffen auf den verschlüsselten Text (Ciphertext). Der Angreifer kennt bei einer **Choosen Plaintext Attack** einen oder mehrere Plaintexts und die entsprechenden Ciphertexts dazu und versucht mit Hilfe dieser Information den Schlüssel zu brechen. Durch den probabilistischen Faktor kann ein Angreifer, wenn ihm nur die Ciphertexte vorliegen, zwei identische Plaintexts nicht als solche erkennen.

Den gekapselten Datensatz D_0 erzeugt der Besitzer des Mobilen Agenten durch die Signatur des Ciphertext und des Hash–Wertes h_0. Danach beginnt der erste Protokollschritt, in dem D_0 an das Agentensystem AS_1 geschickt wird. Klar ist, das nicht nur die Liste der gekapselten Datensätze $AS_i \longrightarrow AS_{i+1} : \prod_{k=0}^{i} D_k$ mit $1 \leq i \leq n$, sondern die komplette MA–Instanz, die diese Liste enthält, übertragen wird. Der Protokollschritt fällt mit dem Migrationsschritt zusammen. Zur Verdeutlichung wird aber im Folgenden nur der Datensatz D_i betrachtet.

Wenn der Mobile Agent am Agentensystem AS_i ankommt, enthält er eine Liste von gekapselten Datensätzen inklusive D_{i-1}, aus dem der nächste Hash–Wert h_i berechnet werden kann. Im Allgemeinen besteht ein gekapselter Datensatz D_i $(i > 0)$ aus zwei Teilen,

1. den Daten d_i, probabilistisch verschlüsselt, sodass nur O sie lesen kann und

Aufbau der gekapselten Datensätze

2. einem Hash–Wert des vorhergehenden gekapselten Datensatzes D_{i-1} konkateniert mit der Identität des nächsten Agentensystems AS_{i+1}.

Abbildung 4.19 stellt die Verkettungsrelation nochmals graphisch dar.

KAG–P1: realisiert strenge Vorwärts–Integrität

Die gekapselten Datensätze erfüllen die Eigenschaft der strengen Vorwärts–Integrität. Angenommen ein Angreifer lässt D_m intakt, verändert aber D_k mit $k < m$. Ohne Beschränkung der Allgemeinheit sei $k = m - 1$ und der Angreifer verändert D_{m-1} in D'_{m-1}. Unter der Annahme, dass D_m nicht verändert wurde, hat D_m folgende Form: $D_m = AS_m\{O_p[d_m \bullet r_m] \bullet h_m\}$ mit $h_m = H(D_{m-1} \bullet DN(AS_{m+1}))$. Unter der Annahme, dass die Verkettungsrelation hält, gilt $H(D'_{m-1} \bullet DN(AS_{m+1})) = H(D_{m-1} \bullet DN(AS_{m+1}))$. Dies verletzt jedoch die Annahme, dass der kryptographische Hash–Wert H kollisionsfrei (vgl. Seite 88) ist. Daraus folgt durch Induktion, dass eine Manipulation von D_{m-1} ohne eine gleichzeitige Modifikation von D_m nicht möglich ist, solange die Hash–Funktion kollisionsfrei ist. D.h. es ist nicht möglich D_k mit $k < m - 1$ zu verändern, ohne gleichzeitig D_{k+1} zu manipulieren.

Die Vorwärts–Integrität der Liste kann öffentlich verifiziert werden.

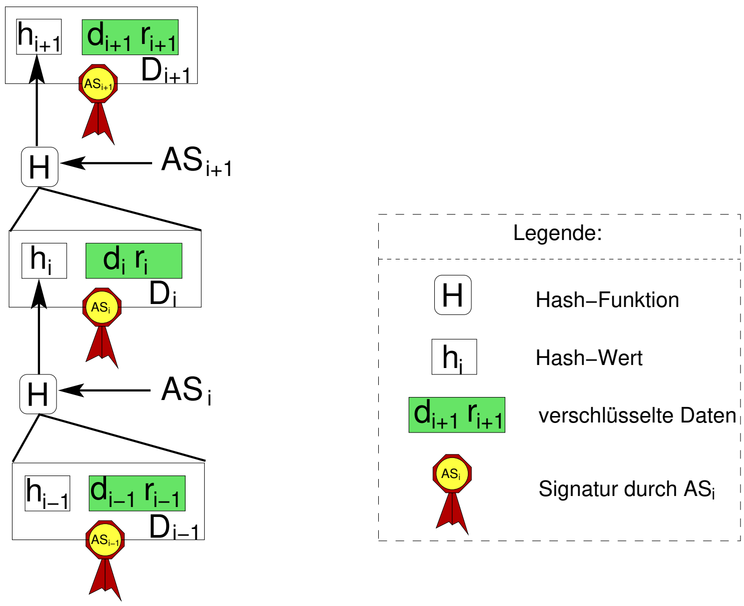


Abbildung 4.19: Verkettungsrelation bei KAG–PI

KAG–P1: öffentliche Verifizierbarkeit der Liste

Betrachtet eine beliebige Entität die Liste $D_0, D_1, \ldots, D_i$, hat jedes D_k folgende Gestalt $D_k = AS_k\{X_k, h_k\}$. Nur die Zeichenkette X_k, der Ciphertext, kann nicht interpretiert werden. Der Betrachter kann aber sowohl die Verkettungsrelation als auch die Gültigkeit der Signatur von AS_k überprüfen.

KAG–P1: relativ sicher gegenüber Verkürzung der Liste

KAG–P1 ist auch relativ sicher gegenüber einer Verkürzung der Liste. Bei einer gegebenen Liste von Datensätzen $D_0, D_1, \ldots, D_m$ führt jede Beschneidung an der Stelle k, wobei $k < m$ gilt, dazu, dass die verketteten Hash–Werte ungültig werden, außer der Angreifer ist in der Lage, gleichzeitig mit der Beschneidung einen neuen Datensatz im Namen von AS_{k+1} einzufügen, d.h. falls mehrere Angreifer zusammenarbeiten, kann die Liste verkürzt werden. Das Protokoll erlaubt einem einzelnen Angreifer an der Stelle m gefälschte Datensätze D_{m+i} einzufügen, bevor der Agent zum Besitzer zurückkehrt. Um sich jedoch erfolgreich als Agentensystem AS_{m+i} maskieren zu können, müsste der Angreifer in der Lage sein, eine gültige Signatur $D_{m+i} = AS_{m+i}\{O_p[d_{m+i} \bullet r_{m+i}] \bullet h_{m+i}\},\ 1 \leq i \leq n - m$ zu erzeugen. Dies ist ihm jedoch nicht möglich, da er nicht im Besitz des privaten Schlüssels von AS_{m+i} ist.

Analog lässt sich zeigen, dass KAG–P1 auch die anderen der auf Seite 126 angegebenen Eigenschaften 1 bis 6 erfüllt (vgl. [KAG 98]).

Vorwärts–Anonymität: KAG–P2

Sollen die Agentensysteme die Identitäten der anderen Agentensysteme, die Daten erzeugt haben, nicht ermitteln können, besteht der Bedarf nach Vorwärts–Anonymität. In diesem Fall muss KAG–P2 verwendet werden. KAG–P2 unterscheidet sich im Wesentlichen von KAG–P1 in der Berechnung der gekapselten Daten:

- gekapselte Daten:

$$D_i = O_p\left[AS_i\{d_i\} \bullet r_i\right] \bullet h_i,\ 0 \leq i \leq n$$

Vertauschung von digitaler Signatur und Verschlüsselung

Durch die Vertauschung von digitaler Signatur und Verschlüsselung wird die Identität von AS_i verschlüsselt und damit für keine andere Entität, außer O, zugänglich. Durch diese Änderung im Protokoll verliert man aber die Möglichkeit der öffentlichen Verifizierbarkeit der Liste (Eigenschaft 4).

Öffentliche Daten: KAG–P1a

In Abbildung 4.19 sieht man, dass alle Nutzdaten d_i bei KAG–P1 verschlüsselt übertragen werden. Damit sind die Daten für andere Agentensysteme AS_i aber auch für den Mobilen Agenten selbst, zu einem späteren Zeitpunkt der Reise nicht mehr nutzbar. Sollen die veränderlichen Daten öffentlich zugänglich bleiben, muss KAG–P1 modifiziert werden. Diese modifizierte Form wird im Folgenden als KAG–P1a bezeichnet.

Bei der Kapselung der Daten wird bei KAG–P1a auf den Verschlüsselungsschritt verzichtet. Da die Zufallszahl r_i lediglich als probabilistischer Faktor zur Stärkung der Verschlüsselungsoperation verwendet wird, kann auch darauf verzichtet werden. Damit erhält man das neue KAG–P1a Protokoll:

Protokollablauf KAG–P1a

- gekapselte Daten:

$$D_i = AS_i\{d_i \bullet h_i\},\ 0 \leq i \leq n$$

- Verkettungsrelation:

  wie bei KAG–P1

- Protokoll:

  wie bei KAG–P1

KAG–P1a erfüllt damit die Eigenschaften 2 bis 6. In Tabelle 4.9 werden die Eigenschaften der vorgestellten Integritätsprotokolle nochmals gegenübergestellt.

4.5.3 Reisehistorie

Die von einem Mobilen Agenten besuchten Agentensysteme werden als **Reisehistorie** bezeichnet. Diese Information ist z.B. für die Zugriffskontrolle wichtig (vgl. Abschnitt 4.4.1), um entscheiden zu können, ob der Mobile Agent bereits ein Agentensystem besucht hat, dessen Vertrauenslevel unter einem bestimmten Wert liegt (nicht vertrauenswürdiges Agentensystem). Daneben muss die Reisehistorie auch Informationen liefern über die Reihenfolge, in der die Agentensysteme besucht wurden.

Der Name des Mobilen Agenten wird vor jeder Migration durch das Quell–Agentensystem digital signiert (vgl. Abschnitt 4.2.4). Dadurch könnte zwar

	KAG–P1	KAG–P2	KAG–P1a
1. Vertraulichkeit	✔	✔	—
2. Non Repudiation	✔	✔	✔
3. Strenge Vorwärts–Integrität	✔	✔	✔
4. Öffentliche Verifizierung der Vorwärts–Integrität	✔	—	✔
5. Stabilität gegenüber nachträglicher Einfügung	✔	✔	✔
6. Stabilität gegenüber Verkürzung	✔	✔	✔
7.Vorwärts–Anonymität	—	✔	—

Tabelle 4.9: Eigenschaften der Integritätsprotokolle nach Karjoth et al.

eine Signaturkette entstehen, die aber nicht als Historie verwendet werden kann. Diese Signaturketten wären z.B. nicht vor Verkürzung geschützt.

KAG Protokolle zur Bildung der Historie

Die in den vorhergehenden Abschnitten vorgestellten Protokolle lassen sich ohne Veränderung verwenden, um eine öffentlich zugängliche ebenso wie eine vertrauliche Historie zu bilden. Wird KAG–P1a eingesetzt, ohne Daten zu verwenden, d.h. d_i bleibt ungenutzt, so entsteht eine Liste der besuchten Agentensysteme, die öffentlich verifizierbar, stabil gegen nachträgliche Einfügung und gegen nachträgliche Verkürzung ist. Auch die strenge Vorwärtsintegrität ist gewährleistet und, da jedes Agentensystem die Verkettungsrelation in Form von h_i signiert, ist die Historie für alle Agentensysteme verbindlich.

KAG–P1a für öffentliche Historie

KAG–P1 für vertrauliche Historie

Sollte es Bedarf nach einer Historie geben, die nur vom Quell–Agentensystem gelesen werden kann, so ist KAG–P1 zu verwenden.

4.5.4 Verbindlichkeit

Die Verbindlichkeit bei der Migrations– bzw. Kommunikationsrelation ist einfacher zu erreichen als bei der Aufrufrelation (vgl. Abschnitt 4.4.3). Die Fragen, die hier von Interesse sind, lauten:

- Wurde der Agent migriert? Wer hat den Agent migriert?
- Fand ein Nachrichtenaustausch zwischen Entitäten statt? Welche Entitäten waren beteiligt?

Verbindlichkeit der Kommunikation durch digitale Signatur

Sowohl die Kommunikation als auch die Migration kann man als Form des Nachrichtenaustausches betrachten (vgl. z.B. Abschnitte 2.5, 4.1). Um die Verbindlichkeit von Nachrichten zu erreichen, wird die digitale Signatur der Nachricht verwendet. Bei der Kommunikationsrelation muss jede Nachricht vor der Übertragung an den Empfänger digital signiert werden.

Verbindlichkeit der Migration durch Historie

Die Verbindlichkeit der Migration wird durch die Historie sichergestellt. Jedes Agentensystem schreibt in h_i den Distinguished Name des nächsten Agentensystems, auf das der Mobile Agent migriert wird. Durch die digitale Signatur und die Eigenschaften der KAG–Protokolle ist eine spätere Leugnung nicht möglich und eine Umlenkung des Mobilen Agenten auf ein anderes Agentensystem würde erkennbar.

4.5.5 Schlussfolgerungen für die Sicherheitsarchitektur

Bei der Kommunikations– und Migrationsrelation steht die Vertraulichkeit und die Integrität der übertragenen Daten im Vordergrund. Vertraulichkeit der Kommunikation lässt sich mit Hilfe **hybrider Verschlüsselungsverfahren** erreichen. Dazu ist es aber erforderlich, dass alle Kommunikationspartner im Besitz von Schlüsselpaaren für asymmetrische Verfahren sind und Schlüssel für ein symmetrisches Verfahren erzeugen können. Ein Mobiler Agent stützt sich für diese Zwecke auf das Agentensystem ab, das stellvertretend für ihn Schlüssel erzeugt und ihm als **LCA** ein Zertifikat ausstellt. Obwohl dieses MA–Zertifikat nur lokale Gültigkeit (auf dem Agentensystem) besitzt, muss das Agentensystem dieses bei einer CA speichern, damit Signaturen des Mobilen Agenten nachträglich verifizierbar bleiben.

Da die Implementierung der Mobilen Agenten möglichst schlank bleiben soll und von einem Entwickler Mobiler Agenten nicht die Implementierung kryptographischer Methoden verlangt werden kann, muss das Agentensystem als Stellvertreter des Mobilen Agenten tätig werden. Der Agent nutzt zur Berechnung von Hash–Werten, MACs und für verschiedene Verschlüsselungsroutinen sowie Signaturen Schnittstellen des Agentensystems.

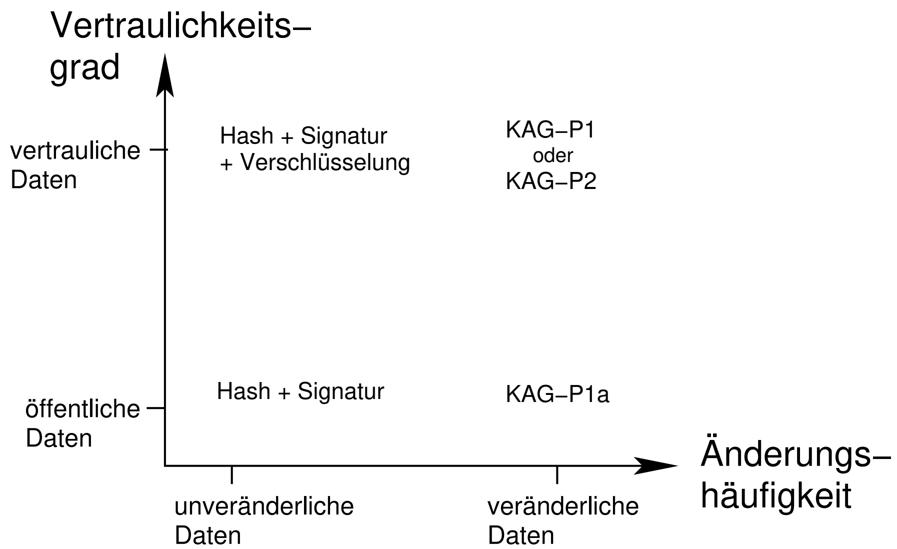


Abbildung 4.20: Integrität der Kommunikations- und Migrationsrelation: Mechanismen

Zur Sicherstellung der Integrität und ggf. der Vertraulichkeit der Instanz–Daten der Mobilen Agenten muss das Agentensystem die KAG–Protokollfamilie implementieren und auch verifizieren können. Abbildung

4.20 fasst die Klassifikation von Instanzdaten und Mechanismen zu deren Sicherung nochmals zusammen.

Das Design Pattern „Mobiler Agent“ muss Attribute zur Aufnahme und Klassifikation einer Reisehistorie enthalten und das Agentensystem muss diese Attribute ausfüllen.

4.6 Zusammenfassung der Sicherheitsdienste

In Abschnitt 2 wurde die Hierarchie der OSI–Sicherheitsdienste vorgestellt (vgl. Abbildung 2.15). In der folgenden Abbildung 4.21 wird diese Hierarchie nochmals aufgegriffen und um spezifische Sicherheitsanforderungen, die in Systemen Mobiler Agenten erforderlich sind, erweitert. Die neu-

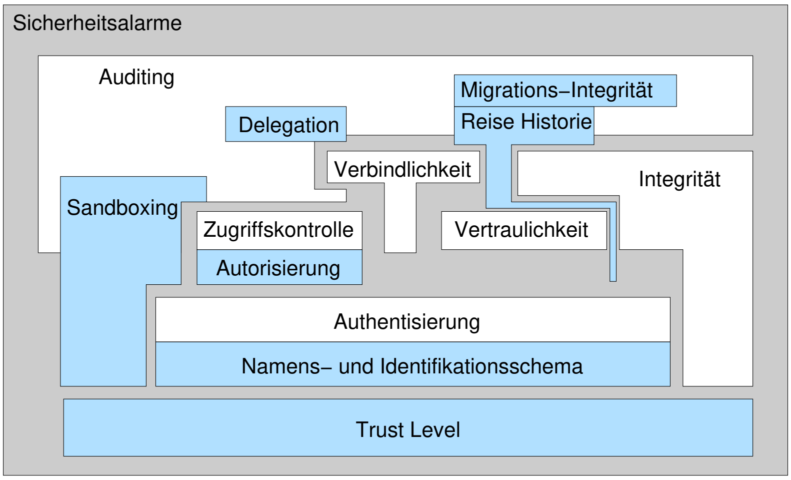


Abbildung 4.21: Hierarchie der Sicherheitsdienste

en Sicherheitsdienste sind grau dargestellt und in die Hierarchie eingeordnet. Überlappende und übereinander liegende Bereiche symbolisieren eine mögliche Abstützung auf andere Sicherheitsdienste. Für die Migrations–Integrität kann beispielsweise ein Dienst, der die Reisehistorie eines Mobilen Agenten bildet, die Dienste die Vertraulichkeit und Integrität realisieren sowie ggf. die Authentisierung Verwendung finden.

Ein wichtiger Basisdienst, der im Szenario des domänenübergreifenden Managements fast allen anderen Diensten zugrunde liegt, ist das Trust–Level Management.

Im nächsten Kapitel wird die Implementierung der in diesem Abschnitt definierten Sicherheitsdienste untersucht.

Kapitel 5

Komponenten der Sicherheitsarchitektur

Inhaltsverzeichnis

In diesem Abschnitt wird eine bausteinbasierte Sicherheitsarchitektur vorgestellt, welche die Sicherheitsmechanismen aus dem vorigen Abschnitt implementiert. Die Sicherheitsmechanismen wurden auf Basis einer prospektiven Risikoanalyse eines allgemeinen Systemmodells entwickelt. Diese Risikoanalyse kann als Basis für einen Betreiber eines Managementsystems dienen, um für seinen konkreten Fall eine unternehmensbezogene und implementierungsabhängige Risikoanalyse durchzuführen. Damit sind die spezifischen Sicherheitsanforderungen des konkreten Anwendungsfalls zu ermitteln. Sobald die konkreten Anforderungen identifiziert sind, können die Bausteine ausgewählt und in das Managementsystem integriert werden, die die erforderlichen Sicherheitsdienste zur Verfügung stellen.

Im ersten Abschnitt dieses Kapitels wird untersucht, welche Entitäten am besten geeignet sind, um Sicherheitsdienste zu implementieren. Dabei wird sich zeigen, dass dies das Agentensystem, als Mittler zwischen Mobilen Agenten und Ressourcen des Endsystems, ist. Der Abschnitt 5.4 wird sich deshalb mit Komponenten beschäftigen, die in ein Agentensystem integriert werden, Sicherheitsdienste realisieren und auch für andere Entitäten zur Verfügung stellen können.

Bei den Sicherheitsdiensten gibt es aber auch solche, die nicht nur einzelne Entitäten betreffen, sondern eine Querschnittsfunktion innerhalb einer Domäne darstellen. Die Komponenten der Architektur werden deshalb unterteilt in Komponenten der Domäne und Komponenten des Agentensystems. In Abschnitt 2.2 wurden Szenarios für den Einsatz Mobiler

Agenten angegeben. Es hat sich gezeigt, dass diese insbesondere für organisationsübergreifende Managementsysteme nutzbringend einsetzbar sind. Zur Realisierung eines domänenübergreifenden Managementsystems sind aber auch domänenübergreifende Querschnittsfunktionen erforderlich. Die Domänen–Komponenten werden deshalb weiter unterteilt in Intra– (vgl. Abschnitt 5.2) und Inter–Domänen–Komponenten (vgl. Abschnitt 5.3).

5.1 Bewertung der Entitäten zur Realisierung von Sicherheitsdiensten

Unabhängig von einzelnen konkreten Sicherheitsdiensten lässt sich ein System Mobiler Agenten daraufhin untersuchen und bewerten, welche Entitäten geeignet und in der Lage sind, Sicherheitsdienste zu implementieren. Im Folgenden wird diese Frage für die Entitäten Mobiler Agent, Agentensystem sowie Endsystem untersucht (vgl. auch [Roel 99a]).

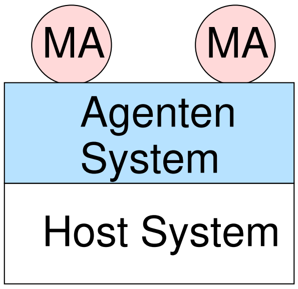
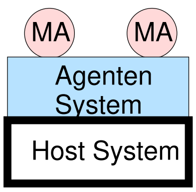


Abbildung 5.1: Architekturmodell für Systeme Mobiler Agenten

Dabei wird das allgemeine Architekturmodell für Systeme Mobiler Agenten verwendet (vgl. Abbildung 5.1). Auf einem Endsystem wird eine Ausführungsplattform für Mobile Agenten, das Agentensystem, ausgeführt. Die Mobilen Agenten wiederum werden vom Agentensystem ausgeführt. Im Folgenden wird untersucht, welchen Beitrag diese Entitäten zur Sicherheitsarchitektur zu leisten im Stande sind und wo deren Grenzen liegen. Jede Entität wird unter der Annahme betrachtet, dass sie die komplette Sicherheitsarchitektur mit allen Sicherheitsdiensten vollständig implementieren muss. Durch diese Betrachtungsweise treten die Stärken, aber auch die Schwächen im Hinblick auf die Implementierung von Sicherheitsdiensten durch eine Entität deutlicher hervor. Insbesondere wird dadurch auch klar, dass es Sicherheitsdienste gibt, die eine bestimmte Entität für andere Entitäten überhaupt nicht implementieren kann.

5.1.1 Realisierung von Sicherheitsdiensten durch das Endsystem



Das Endsystem, bzw. das auf dem Endsystem zum Einsatz kommende Betriebssystem, stellt die Sicherheitsdienste bereit (vgl. nebenstehende Abbildung). Das Agentensystem und die darauf ablaufenden Mobilen Agenten erscheinen aus der Sicht des Endsystems als ein Prozess oder als eine Menge von Threads.

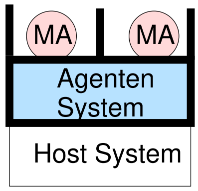
- **Vorteile**
  - Bestehende und ausgereifte Sicherheitsdienste des Betriebssystems können ohne Änderung übernommen und genutzt werden. Die Implementierung der Sicherheitsarchitektur beschränkt sich dabei auf eine geeignete Auswahl von Sicherheitsdiensten, die das Betriebssystem zur Verfügung stellt.
  - Da sich ein Agentensystem nicht von einem „normalen“ Benutzerprozess unterscheidet, lassen sich die Ressourcen des Endsystems völlig unabhängig vom verwendeten Typ des Agentensystems oder den Mobilen Agenten sichern.

- **Nachteile**
  - Das resultierende Sicherheitsmodell ist dadurch allerdings auch sehr starr. Die Dynamik des Systems Mobiler Agenten kommt nicht zum Tragen und lässt sich in der Sicherheitsarchitektur auch nicht abbilden. Bei Betriebssystemen, bei denen Agentensystem und Mobile Agenten als ein Prozess erscheinen (prozess–orientiertes Betriebssystem), hat das Betriebssystem keine Kenntnis von den Entitäten Agentensystem und Mobiler Agent. Deshalb ist es auch nicht möglich, dass das Endsystem spezifische Sicherheitsanforderungen dieser Entitäten erfüllen kann. Das Endsystem kann in diesem Fall verschiedene Mobile Agenten, die auf einem Agentensystem ablaufen, überhaupt nicht als eigene Entitäten erkennen.
    Bei Thread–orientierten Betriebssystemen wäre dies u.U. möglich. Das Agentensystem sowie jeder Mobile Agent könnte als eigener Thread implementiert werden. Sicherheitsdienste, die auf Ebene der Threads arbeiten, könnten dann verwendet werden, um Sicherheitsanforderungen von Mobilen Agenten zu realisieren. In heterogenen Umgebungen kann jedoch nicht davon ausgegangen werden, dass alle Endsysteme Threads anbieten.
    Im Allgemeinen muss davon ausgegangen werden, dass bei einer Implementierung der Sicherheitsarchitektur im Endsystem auch nur die Ressourcen des Endsystems, und nicht Mobile Agenten oder Agentensysteme, verlässlich geschützt werden können.
  - Bei Sicherheitsdiensten auf Betriebssystemebene ist eine dynamische Anpassung nur schwer möglich. Die sicherheitsrelevanten Entscheidungen werden größtenteils beim Start eines Prozesses festgelegt und können dann nicht mehr verändert werden, ohne den Prozess zu stoppen. Beispielsweise wird beim Start eines Prozesses festgelegt, wer für den Prozess verantwortlich ist und welche Rechte der Prozess besitzt. Mobile Agenten — die auf einem Agentensystem „kommen und gehen“ — für die verschiedene Organisationen verantwortlich sind, und denen dynamisch Rechte erteilt werden sollen, lassen sich damit nur sehr schwer oder überhaupt nicht realisieren.

– Eine Endsystem–basierte Sicherheitslösung in heterogener Umgebung ist schwer zu verwalten. Management–Schnittstellen und die Sicherheitsdienste selbst, unterscheiden sich zum Teil erheblich. Eine einheitliche Sicherheitsarchitektur für das gesamte Managementsystem ist nur schwer zu realisieren, da die Ausprägungen der Sicherheitsdienste von heterogenen Endsystemen abhängen.

5.1.2 Realisierung von Sicherheitsdiensten durch das Agentensystem

Die Sicherheitsdienste werden durch das Agentensystem implementiert, überwacht und durchgesetzt. Dabei übernimmt das Agentensystem als Laufzeitumgebung für Mobile Agenten auch die Sicherung der Agenten.

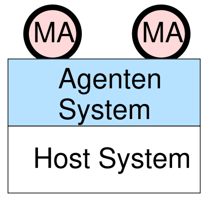


- Vorteile
  - Mobile Agenten können geschützt werden, unabhängig davon, ob sie eigene Sicherungsmaßnahmen implementieren. Die Entwicklung und Implementierung von Mobilen Agenten wird dadurch vereinfacht.
  - Das Agentensystem, als Abstraktionsschicht oberhalb der heterogenen Betriebssysteme, bildet eine homogene Ausführungsplattform für Mobile Agenten. Die Sicherheitsdienste, die das Agentensystem zur Verfügung stellt, realisieren daher eine einheitliche Sicherheitsarchitektur, deren Managementschnittstellen, auch über heterogene Systemgrenzen hinweg, einheitlich sind.
  - Das Agentensystem implementiert auch die Sicherheitsdienste, die zur Sicherung der Ressourcen des Endsystems verwendet werden. Auch diese Aufrufschnittstellen sind auf allen Endsystemen gleich und nicht vom unterliegenden Betriebssystem abhängig.

- Nachteile
  - Agenten müssen sich voll auf die Sicherheitsdienste des Agentensystems verlassen und diesen auch vertrauen können, da sie selbst keinerlei eigene Schutzmaßnahmen implementieren.
  - Auch die Endsysteme müssen sich auf die Sicherheitsdienste des AS verlassen. Endsysteme sind zwar in der Lage, eigene Betriebssystem–spezifische Sicherheitsdienste zu nutzen und z.B. den Zugriff des AS auf Ressourcen des Endsystems zu beschränken. Da das System Mobiler Agenten aber zum Zweck des Managements eingesetzt werden soll, wird das Agentensystem i.d.R. mit Administrator– oder Root–Rechten ausgestattet sein und damit gelten die meisten Einschränkungen für das Agentensystem nicht mehr.

5.1.3 Realisierung von Sicherheitsdiensten durch den Mobilen Agenten

Alle Sicherheitsdienste werden durch den Mobilen Agenten selbst implementiert, überwacht und durchgesetzt. Die Sicherheitsarchitektur wird durch die Gesamtheit der Sicherheitsdienste, die die verschiedenen Mobilen Agenten implementieren, gebildet.



- Vorteile
  - Ein Agent kann sich bezüglich seiner Semantik optimal selbst schützen. Das Wissen, welche Dienste und Ressourcen des Agenten zu schützen sind, ist bei der Implementierung des Agenten am umfangreichsten.
  - Schutzdienste können individuell und feingranular auf jeden einzelnen Agenten abgestimmt werden.
- Nachteile
  - Sicherheitsdienste werden von den verschiedenen Implementierern der Agenten zur Verfügung gestellt. Eine einheitliche Sicht auf die Schnittstellen lässt sich dabei nur schwer durchsetzen. Nicht jedem Implementierer kann vertraut werden. Die Sicherheitsdienste in ihrer Gesamtheit sind immer nur so gut wie die schlechteste Implementierung.
  - Es entsteht eine Sicherheitsarchitektur, die voll auf Agenten basiert, die aber von verschiedensten Organisationen entwickelt wurden. Es ist schwer, dafür ein einheitliches Management zu installieren.
  - Agentensysteme und Endsysteme müssen dem (möglicherweise fremden) Agenten vertrauen.
  - Da ein Agent voll unter der Kontrolle des Agentensystems ausgeführt wird, kann er sich nie vollständig selbst schützen (vgl. dazu auch Abschnitt 4.1)

5.1.4 Schlussfolgerungen

Werden die Nachteile aus den vorhergehenden Abschnitten betrachtet, so wird klar, dass keine der drei Entitäten Endsystem, Agentensystem und Mobiler Agent die vollständige Sicherheitsarchitektur implementieren kann. Dem Grundsatz nach sollte jede Entität die Sicherheitsdienste implementieren, für die sie am besten geeignet ist.

Das Betriebssystem kann seine Ressourcen (z.B. Dateien, Netzwerkverbindungen, u.ä.) selbst schützen. Dabei müssen aber das Agentensystem und alle darauf ablaufenden Agenten als eine Einheit (ein Prozess) betrachtet werden. Die Rechte, die das Betriebssystem vergeben und kontrollieren kann, richten sich nach den Rechten desjenigen, der das Agentensystem startet.

Um feingranulare Sicherheitsmechanismen auf Basis von Mobilen Agenten realisieren zu können, ist das Betriebssystem nicht geeignet. Deshalb wird im Agentensystem, als Mediator zwischen Endsystem und Mobilem Agenten, der größte Teil der Sicherheitsarchitektur realisiert werden. Das Agentensystem stellt eine Art „Agentenbetriebssystem“ dar und ist als solches auch am besten geeignet, um Sicherheitsdienste zu implementieren.

Es gibt aber auch Fälle, in denen nur der Mobile Agent „weiß“ welche seiner Ressourcen zu schützen sind. In diesem Fall muss der Mobile Agent Entscheidungsregeln für den Zugriff auf seine Ressourcen festlegen. Für die technische Durchsetzung kann er sich auf Schnittstellen und Methoden des Agentensystems abstützen. Im Folgenden wird der Fokus deshalb auf Sicherheitskonzepten des Agentensystems und der Mobilen Agenten liegen.

5.2 Intra–Domänen–Komponenten

Grundlegende Basis für eine Sicherheitsarchitektur ist die eindeutige Bezeichnung aller Entitäten (Namensschema, vgl. Abschnitt 4.2.2) sowie die Bindung des Namens an die entsprechende Entität (vgl. Abschnitt 4.2.3). Diese Aufgaben sind nicht auf ein End– oder Agentensystem beschränkt, sondern innerhalb der gesamten Domäne zur Verfügung zu stellen. Eine Identifizierung der Entitäten alleine ist nicht ausreichend, es muss auch Mechanismen zur Lokalisierung geben. Im nächsten Abschnitt wird deshalb ein Namens– und Lokalisierungsdienst vorgestellt. Da für die Bindung des Namens an Entitäten, Zertifikate bzw. digitale Signaturen verwendet werden, wird eine CA–Infrastruktur benötigt. Bei den Intra–Domänen–Komponenten handelt es sich um globale und domänenübergreifende Basisdienste, was jedoch nicht bedeutet, dass diese Dienste auch zentral implementiert und erbracht werden müssen. Eine verteilte Implementierung aller Intra–Domänen–Komponenten erhöht die Ausfallsicherheit und die Skalierbarkeit der Gesamtarchitektur.

5.2.1 Naming– und Location Service

Um Agentensysteme sowie Mobile Agenten lokalisieren und erreichen zu können, ist ein Verzeichnisdienst erforderlich, der Namen auf Objektreferenzen abbildet. Über diese Objektreferenzen sind die Entitäten dann erreichbar. Der MASIF–Standard [MASIF] definiert hierzu den **MAFFinder**, der Methoden zur Registrierung, Löschung und zur Lokalisierung von Agenten, Agentensystemen und Places spezifiziert. Die Methoden zur Lokalisierung erlauben die Suche nach bestimmten Namen, nach Eigenschaften oder über komplexe Suchprofile. Unabhängig von der konkret verwendeten Technologie ist ein domänenübergreifender Naming– und Location Dienst erforderlich, der zumindest die MAFFinder Schnittstelle zur Verfügung stellt.

zweifelsfreie Registrierung muss gewährleistet sein

Wird diese Komponente aus sicherheitstechnischen Gesichtspunkten betrachtet, so ist eine zweifelsfreie und „fälschungssichere“ Registrierung der Namen zu gewährleisten. Dies wird einerseits durch ein global eindeutiges Namensschema (vgl. Abschnitte 4.2.2 und 5.4.8), andererseits durch die systematische Verhinderung von Name–Spoofing und Maskerade–Angriffen gewährleistet.

Agentensysteme dürfen nur „ihre“ Agenten registrieren

Im Naming– und Location Dienst dürfen nur Agentensysteme Namen und Objektreferenzen eintragen und zwar nur für Mobile Agenten, die auf dem Agentensystem ausgeführt werden. Damit wird verhindert, dass ein Agentensystem einen eigenen Mobilen Agenten unter dem Namen eines fremden Agentensystems registriert. Der Naming– und Location Dienst muss also vor jeder Registrierung eines Namens das registrierende Agentensystem authentisieren. Das Agentensystem darf dann nur Namen registrieren, die den Namen des Agentensystems als Teil enthalten. Das heißt, nur der eigene Name des Agentensystems und die Namen „seiner“ Agenten können vom Agentensystem im Verzeichnisdienst eingetragen werden.

Der Naming– und Location Service benötigt, ebenso wie die Agentensysteme, einen Authenticator (vgl. Abschnitt 5.4.2) zur Authentisierung und einen Communication Manager (Abschnitt 5.4.5) für eine verbindliche und vertrauliche Kommunikation.

5.2.2 Certification Authority (CA); Trust Center Infrastruktur

Sehr viele der in Kapitel 4 vorgestellten Sicherheitsmechanismen, die z.B. die Sicherheitsanforderungen Authentisierung, Autorisierung, Verbindlichkeit oder Integrität realisieren, stützen sich auf Public–Key Verfahren, digitale Signaturen und Zertifikate ab. Aus diesem Grund muss eine Zertifizierungsstelle, im Englischen als Certification Authority (CA) oder Trust Center bezeichnet, eingerichtet werden. Dabei handelt es sich ganz klar um eine Querschnittsfunktion, die für alle Mobilen Agenten und alle Agentensysteme einer Domäne erforderlich ist. Die Aufgaben, die von einer CA zu erfüllen sind, wurden auf Seite 85 bereits vorgestellt. In diesem Abschnitt soll der organisatorische Ablauf nochmals näher betrachtet werden. Für eine erste vereinfachende Darstellung wird davon ausgegangen, dass es für eine Domäne genau eine Domänen–CA gibt, d.h. jede Organisation betreibt eine eigene Domänen–CA.

Die wichtigste Aufgabe der CA ist die Ausstellung, der Widerruf und die Speicherung von Zertifikaten für die Entitäten innerhalb der Domäne.

Zertifikatserstellung

In Tabelle 4.3 wurden für alle Entitäten eindeutige Identifikatoren angegeben. Diese Tabelle wird in der Tabelle 5.1 um Mechanismen zur Bindung des Namens an die entsprechende Entität (Identifikation) erweitert. Die Domänen–

Entität		Eindeutiger Name	Bindung des Namens durch
Agenten	Gattung	Eindeutige Bezeichnung des Quellcode–Paketes (z.B. Java Package)	digitale Signatur der Gattung
	Instanz	Eindeutiger Name, MASIF Namensschema mit Erweiterung	digitale Signatur des Namens
Agentensystem		Eindeutiger Name, MASIF Namensschema	Zertifikat
Endsystem		Adresse oder Full Qualified Host Name	Zertifikat, IPSec, DNSSec, ssh
Benutzer	Implementierer, Benutzer, Administrator, . . .	Eindeutiger Name und/oder Rolle (Distinguished Name, z.B. X.500)	Zertifikat

Tabelle 5.1: Bindung von Identifikatoren an Entitäten

CA zertifiziert also alle Benutzer, alle Agentensysteme und alle Endsysteme. Dazu wird der Name (DN) der Entität, deren öffentlicher Schlüssel und ggf. zusätzliche Attribute von der CA digital signiert.

Benutzerzertifikat

Die **Benutzerzertifikate** können als Attribut das Rollen–Set des Benutzers enthalten (vgl. Abschnitt 4.4.1). Der Benutzer kann sein Zertifikat für alle Zwecke verwenden, die den Einsatz seines öffentlichen oder privaten Schlüssels bedürfen wie z.B. zur Authentisierung oder Autorisierung.

Besitzer signiert Namen der MA–Instanz

Ein Besitzer eines Mobilen Agenten, d.h. der Benutzer, der den Mobilen Agenten das erste Mal startet, signiert dessen Namen. Dadurch erklärt er sich einerseits für den Mobilen Agenten verantwortlich. Andererseits wird dadurch auch zum Ausdruck gebracht, dass der Mobile Agent im Auftrag des signierenden Benutzers handelt.

Implementierer signiert MA–Gattung

Auch die Implementierer von Mobilen Agenten erhalten ein Benutzerzertifikat und verwenden den entsprechenden privaten Schlüssel, um die von ihnen implementierten Agenten–Gattungen digital zu signieren. Damit zertifiziert der Implementierer die Agenten–Gattung.

Rollenzertifikat

Für Benutzer können zusätzlich auch eigene **Rollenzertifikate** erstellt werden, die den Namen des Benutzers, eine Menge von Rollen und einen öffentlichen Schlüssel (Rollenschlüssel) enthalten. Soll ein Mobiler Agent im Auftrag eines Benutzers, in einer bestimmten Rolle tätig werden, wird der entsprechende Rollenschlüssel zur Signatur des Namens der MA–Instanz verwendet.

Endsystemzertifikat

Endsystemzertifikate beinhalten neben dem DN und dem öffentlichen Schlüssel auch die IP–Adresse des Endsystems, damit sowohl der Distinguished Name als auch die Adresse des Systems als Identifikator verwendet werden können und um IP–Spoofing Angriffe zu verhindern.

Agentensystem-zertifikat

Damit das Agentensystem als Local Certification Authority (vgl. Abschnitt 5.4.1) handeln kann, um selbst wieder Zertifikate für Mobile Agenten auszustellen, muss das Attribut `CA:TRUE` im **Agentensystem–Zertifikat** gesetzt werden.

Abbildung 5.2 fasst noch einmal zusammen, welche Entitäten direkt von der Domänen–CA zertifiziert werden und wie diese Zertifikate verwendet werden, um Mobile Agenten zu signieren.

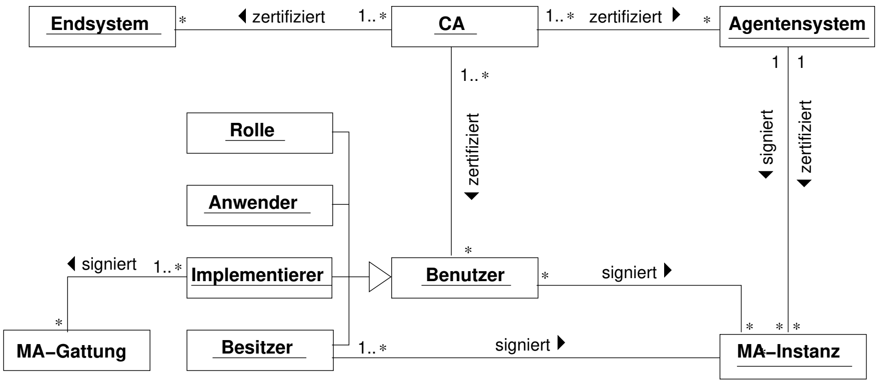


Abbildung 5.2: Zertifikate und Signatur von Entitäten

Widerruf von Zertifikaten

Wird ein privater Schlüssel kompromittiert oder ändern sich die zertifizierten Daten (z.B. der Name), muss die ausstellende CA das gültige Zertifikat, vor dessen Ablauf für ungültig erklären; die CA muss das Zertifikat widerrufen (engl. Revocation). Grundsätzlich geschieht dies durch eine periodische Veröffentlichung von Widerrufslisten, die auch als **Certification Revocation Lists (CRL)** bezeichnet werden.

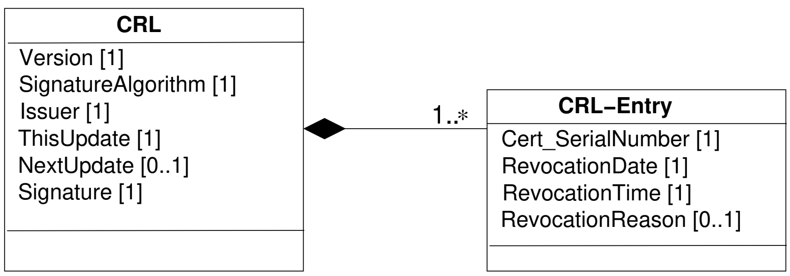


Abbildung 5.3: Widerruf von Zertifikaten: Certification Revocation List

Aufbau einer Widerrufsliste

Eine Widerrufsliste hat die in Abbildung 5.3 angegebene Struktur. Die komplette Datenstruktur wird von der widerrufenden CA (`Issuer`) signiert (`Signature`). Die verschiedenen Listen werden im `Version` Attribut der Reihe nach durchnummeriert. Im Attribut `ThisUpdate` wird der Zeitpunkt, ab dem die CRL gültig ist, eingetragen. Im optionalen Attribut `NextUpdate` wird der Zeitpunkt angegeben, zu dem die nächste aktualisierte CRL erscheint. Der Inhalt der CRL ist eine Liste von CRL–Einträgen. Pro widerrufenem Zertifikat gibt es einen solchen `CRL-Entry`. Das widerrufene Zertifikat kann über die Seriennummer (`Cert_SerialNumber`) in Verbindung mit dem Identifikator der ausstellenden CA (`Issuer` Feld) eindeutig identifiziert werden, da die Seriennummern innerhalb einer CA eindeutig sein müssen. Für jedes widerrufene Zertifikat ist der Zeitpunkt des Widerrufs angegeben, optional kann auch der Grund des Widerrufs (`RevocationReason`) mit angegeben werden.

Mit der Verteilung von CRLs sind zwei Probleme verbunden; die möglichst schnelle und aktuelle Bekanntgabe widerrufener Zertifikate und die effiziente Verteilung der Listen. CRLs werden groß und können auch nicht verkürzt werden. Nur über die CRL kann der Zeitpunkt ermittelt werden, zu dem ein Zertifikat ungültig wurde. Für eine Beweissicherung zu einem späteren Zeitpunkt kann es erforderlich sein, diesen Widerrufszeitpunkt zu kennen. Auch bei einem regulären Ablauf des Zertifikates darf der entsprechende Eintrag in der CRL deshalb nicht gelöscht werden, d.h. CRLs werden nur verlängert und nie verkürzt.

Verteilung der Widerrufslisten

Um die zu übertragenden Datenmengen gering zu halten, wird nur zu bestimmten Zeitpunkten eine komplette CRL übertragen. Zwischen diesen Zeitpunkten werden nur Delta Listen übermittelt, die nur die CRL–Entries enthalten, die seit der letzten Komplettliste widerrufen wurden. Für die Übertragung der CRL gibt es Push und Pull Ansätze. Beim Push–Modell müssen sich alle Interessenten, die Zertifikate verifizieren, bei der CA anmelden und bekommen dann automatisch die CRL zugeschickt. Beim Pull–Modell muss derjenige, der ein Zertifikat verifiziert, die Liste aktiv bei der CA anfordern.

Für Mobile Agenten werden sehr häufig Zertifikate erstellt und auch sehr häufig und in sehr kurzen Perioden widerrufen (vgl. Abschnitt 5.4.1). Das Zertifikat eines bestimmten Mobilen Agenten ist dabei i.d.R. nur für einen sehr kleinen Kreis von Entitäten von Interesse. Ein Push–Modell zur Verteilung der CRLs ist nicht geeignet, da viel zu viele Entitäten mit Informationen überflutet würden, die für sie nicht von Interesse sind. Aus diesem Grund wird ein Pull–Modell mit CRLs und Delta–CRLs verwendet. Dem Grundsatz nach ist jede Entität selbst verantwortlich, bei der Verifikation eines Zertifikates zu überprüfen, ob das Zertifikat nicht bereits widerrufen wurde. Neben den CRLs bietet die CA deshalb zusätzlich eine Schnittstelle an, über die explizit für ein bestimmtes Zertifikat über dessen Seriennummer abgefragt werden kann, ob und zu welchem Zeitpunkt es widerrufen wurde.

Speicherung von Zertifikaten

Speicherung aller Zertifikate

Die CA speichert die Zertifikate aller Entitäten, die von ihr zertifiziert wurden und übermittelt diese an alle anfragenden Entitäten. Die Speicherungspflicht gilt nicht nur für aktuell gültige Zertifikate, sondern auch für abgelaufene oder widerrufene. Nur wenn auch diese gespeichert werden, besteht zu einem späteren Zeitpunkt die Möglichkeit, eine digitale Signatur zu verifizieren, die zu einem Zeitpunkt erstellt wurde, als das Zertifikat noch gültig war.

Zusätzliche Dienste der CA

Neben den Hauptaufgaben — Erstellung, Speicherung und Widerruf von Zertifikaten — stellt die CA ggf. weitere Dienste zur Verfügung.

Schlüsselgenerierung

Wenn nicht davon ausgegangen werden kann, dass alle Entitäten technisch und organisatorisch in der Lage sind, sicheres Schlüsselmaterial zu erzeugen, muss die CA Schlüsselpaare für die Entitäten erzeugen. Dabei muss sichergestellt werden, dass der erzeugte private Schlüssel nur der entsprechenden Entität zugänglich gemacht wird und die Übergabe auf sichere Art erfolgt.

Time Stamping

Beim Zeitstempeldienst werden Daten, die der CA vorgelegt werden, mit einem Zeitstempel versehen und digital signiert.

Notarization

Bei der Notarization werden Vorgänge (z.B. ein Vertragsabschluss), die zwischen Partnern stattfinden, durch eine digitale Signatur der entsprechenden Daten beglaubigt. Der Zeitstempeldienst wird hier verwendet, um den Zeitpunkt, an dem der Vorgang stattgefunden hat, zu dokumentieren. Mit Hilfe von Time Stamping und Notarization lässt sich Verbindlichkeit realisieren.

CA–Infrastruktur, PKI

Bisher wurde davon ausgegangen, dass es innerhalb einer Domäne genau eine CA gibt, die alle Benutzer, Agentensysteme und Endsysteme zertifiziert. In Abschnitt 4.2.4 wurde gezeigt, wie in diesem Fall ein Zertifikat verifiziert wird. Alle Entitäten der Domäne vertrauen der CA und kennen deren öffentlichen Schlüssel mit dem auch alle Zertifikate verifiziert werden können.

skalierbare Infrastruktur ⇒ mehrere CAs

Um skalierbare CA–Infrastrukturen für größere Domänen aufzubauen, ist eine einzige Domänen–CA in der Regel nicht ausreichend. Es sind mehrere CAs erforderlich, um Ausfallsicherheit und Effizienz zu erhöhen. Jeder dieser CAs zertifiziert dann nur noch einen Teil der Domäne (vgl. Abbildung 5.4).

Verifikationsprozess über Zertifizierungspfad

Durch die größere Anzahl an CAs wird der Verifikationsprozess für Zertifikate komplexer. Nun kann es vorkommen, dass ein zu verifizierendes Zertifikat nicht mehr direkt von einer dem Verifizierenden bekannten oder vertrauenswürdigen CA signiert wurde. In diesem Fall muss die Vertrauenswürdigkeit der CA erst durch Zertifikate anderer vertrauenswürdiger CAs bescheinigt werden. Es muss ein so genannter Zertifizierungspfad gefunden werden. Ein **Zertifizierungspfad** (Certification Path) ist eine geordnete Menge von Zertifikaten, die bei einer vertrauenswürdigen CA beginnt und bei der CA endet, die das zu verifizierende Zertifikat ausgestellt hat.

Im Beispiel sei angenommen, Alice möchte das Zertifikat von Bob verifizieren. Sie ist im Besitz des Zertifikates von Bob, das von der CA Y ausgestellt

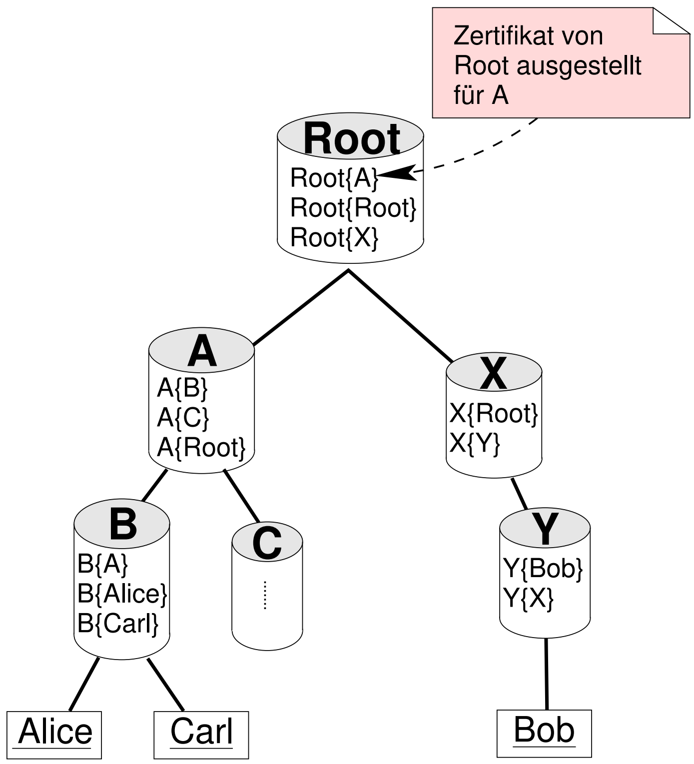


Abbildung 5.4: Hierarchische CA–Infrastruktur

wurde, der Alice weder vertraut, noch deren öffentlichen Schlüssel kennt. Alice kennt den öffentlichen Schlüssel ihrer CA B, der sie auch vertraut. Ausgehend von dieser CA ermittelt Alice den folgenden Zertifizierungspfad zu Y: $B\{A\}, A\{\text{Root}\}, \text{Root}\{X\}, X\{Y\}$. Dabei ist $X\{Y\}$ die Kurzschreibweise für das Zertifikat C_Y^X, das von X für Y ausgestellt wurde. Der Zertifizierungspfad besagt, dass B den Zertifikaten von A vertraut und diese wiederum den Zertifikaten der Root–CA. Die Root–CA wiederum hat X zertifiziert und damit mittelbar auch Y. Über die Zertifikate dieses Zertifizierungspfades bekommt Alice alle öffentlichen Schlüssel der CAs A, $Root$, X und Y und kann damit das Zertifikat von Bob verifizieren.

Um die Berechnung des Zertifizierungspfades zu vereinfachen, und um Organisationsstrukturen nachzubilden, werden, wie in Abbildung 5.4 bereits impliziert, hierarchische Strukturen verwendet. Die Root–CA, der alle Entitäten der Domäne vertrauen, zertifiziert z.B. pro Abteilung, Sub–Level CAs (d.h. Attribut `CA:TRUE`), die ihrerseits weitere Sub–Level CAs oder direkt die Entitäten zertifizieren. Aus dem Beispiel wird auch ersichtlich, dass sich die CAs gegenseitig zertifizieren, d.h. wenn A die CA B zertifiziert, so gibt es auch ein Zertifikat von B für A. Andernfalls müssten alle Berechnungen des Zertifizierungspfades bei der Root–CA beginnen und damit würde ein Flaschenhals bei der Root–CA entstehen.

hierarchische CA–Infrastruktur erleichtert Berechnung des Zertifizierungspfades

Für eine weitere Steigerung der Effizienz werden bestimmte Aufgaben einer CA auf eigene Komponenten ausgelagert. Für die Personalisierung der zu zertifizierenden Benutzer wird in der Praxis oft eine eigene Komponente, die **Registration Authority (RA)** eingeführt. Um den Zugriff auf gespeicherte Zertifikate zu erhöhen, werden eigene **Certification Repositories** aufgesetzt.

Die Kombination aus Repository, RA und von CAs zu einem Verbund wird oft auch als **Public Key Infrastructure (PKI)** [HFPS 99] bezeichnet.

5.3 Inter–Domänen–Komponenten

Im vorhergehenden Abschnitt wurden PKI und CAs betrachtet, die nur für eine Organisationseinheit zuständig sind. Sollen Mobile Agenten organisationsübergreifend eingesetzt werden, bedarf es Mechanismen, um die Zertifikate auch in fremden Domänen verwenden zu können.

Cross Zertifizierung

Cross Zertifizierung verbindet unabhängige Zertifizierungspfade

Damit ein Zertifikat einer Domäne D in einer anderen Domäne E verifizierbar wird, muss es einen Zertifizierungspfad von D zu E und umgekehrt geben. Die Root–CA der Domäne D signiert das Zertifikat von E und umgekehrt. Dieser Vorgang wird als **Cross Zertifizierung** bezeichnet, durch die zwei unabhängige (Teil–) Zertifizierungspfade zu einem neuen domänenübergreifenden Pfad verbunden werden. Die Verbindung von zwei unabhängigen Domänen nur durch eine einzige Cross–Zertifizierung der Root–CAs setzt hierarchische CA–Infrastrukturen in den beteiligten Domänen voraus.

Wird ein Managementsystem über mehr als zwei Domänen hinweg betrieben, kann die Cross Zertifizierung als Netz oder in einer Hub and Spoke Architektur erfolgen.

Im ersten Fall zertifiziert jede Root–CA jede andere, am Managementsystem beteiligte Root–CA (quadratischer Aufwand). Im Fall von **Hub and Spoke** (Nabe und Speiche) wird eine CA als Hub ausgezeichnet, die wechselseitig alle anderen Root–CAs zertifiziert (linearer Aufwand, vgl. Abbildung 5.5). Im Grunde wird dadurch die hierarchische Struktur fortgesetzt. Die als Hub ausgezeichnete CA stellt die domänenübergreifende Root–CA dar.

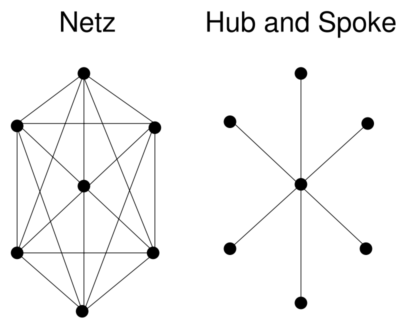


Abbildung 5.5: Cross-Zertifizierung

Ebenso wie die Inter–Domänen CA–Infrastruktur nicht hierarchisch ausgebildet werden kann, kann auch die CA–Infrastruktur innerhalb einer Domäne vermascht sein.

Für den Fall, dass nicht die gesamte Domäne, sondern nur einzelne Teile der Domäne an dem domänenübergreifenden Managementsystem teilnehmen, erfolgt die Cross–Zertifizierung mit der entsprechenden Sub–Level CA der Abteilung und nicht mit der Root–CA. Damit gibt es Querverbindungen zwischen den beiden CA–Infrastrukturen und die hierarchische Struktur wird dadurch zu einem gewissen Grad aufgebrochen.

Rollenzertifikate

Verantwortliche Domäne erstellt Rollenzertifikate

Nachdem die Rollenmodelle für die Zugriffskontrolle autonom von jeder Domäne oder sogar von jedem Betreiber eines Agentensystems spezifiziert werden (vgl. Abschnitt 4.4.2), stellt die entsprechende CA auch für Benutzer aus fremden Domänen die Rollenzertifikate aus.

5.4 Sicherheitskomponenten des Agentensystems

Das Agentensystem stellt den Mittler zwischen Mobilem Agenten und Ressourcen des Endsystems dar. Außerdem ist es Prokurist für seine Mobilen Agenten. Insofern wird, wie bereits in Abschnitt 5.1 festgestellt, der größte Teil der Komponenten der Sicherheitsarchitektur vom Agentensystem implementiert. Die einzelnen Komponenten werden in den folgenden Abschnitten vorgestellt.

5.4.1 Agentensystem als Local Certification Authority (LCA)

Die CA–Infrastruktur zertifiziert Benutzer, Endsysteme und Agentensysteme (vgl. Abbildung 5.2), aber keine Mobilen Agenten. Wie in den Abschnitten 4.2.4 und 4.5.1 bereits erwähnt, benötigt jede Entität, die aktiv im Managementsystem handelt, private Schlüssel und entsprechende Zertifikate. Damit ein Mobiler Agent z.B. vertraulich (d.h. verschlüsselt) kommunizieren kann, braucht er Schlüssel und Zertifikat(e).

Agentensystem als LCA für Mobile Agenten

Wegen des Grundsatzes des Vertrauens durch Einbettungsbeziehung (Abschnitt 4.1) und dem Trust Level Management (vgl. Abschnitt 4.1.2) kann der Mobile Agent das Agentensystem als CA und auch zur Erzeugung seiner Schlüssel verwenden. Das Agentensystem ist Local Certification Authority für alle Mobilen Agenten, die vom Agentensystem ausgeführt werden.

Das Agentensystem, dessen primäre Funktion die Bereitstellung einer Laufzeitumgebung für Mobile Agenten darstellt, soll möglichst „schlank“ bleiben, um auch auf kleineren Geräten ausführbar zu bleiben. Daher soll das Agentensystem keine Daten — außer seiner eigenen Schlüssel — permanent speichern. Es kann nicht davon ausgegangen werden, dass dem Agentensystem Hintergrundspeicher in größeren Mengen zur Verfügung steht. Im Extremfall verliert das Agentensystem bei einem Systemausfall alle Zustandsinformationen und alle flüchtigen Daten. Dies bedeutet, dass nicht alle Aufgaben einer CA vom Agentensystem voll erbracht werden, dies gilt insbesondere für die Speicherung (Repository) und die Verwaltung von widerrufenen Zertifikaten (Revocation Management). Für diese Funktionalität stützt sich das Agentensystem auf seine eigene CA, d.h. auf die CA, die das Zertifikat des Agentensystems erstellt hat. Die CA des Agentensystems übernimmt also die Repository Funktion auch für MA–Zertifikate, die das Agentensystem erstellt hat. Damit lassen sich langfristig die Zertifikatspfade berechnen.

CA der LCA übernimmt Repository Funktion

In Abbildung 5.6 ist ein Sequenzdiagramm angegeben, dass den Ablauf von Schlüssel– und Zertifikatgenerierung sowie deren Speicherung und Widerruf darstellt. Der Mobile Agent kann über die Schnittstelle `get_cert`, die Ausstellung eines Zertifikates anfordern. Das Agentensystem erzeugt die Schlüssel stellvertretend für den Mobilen Agenten und generiert das Zertifikat `cert`, das an die CA zur Speicherung übermittelt wird. Sobald die Speicherung bestätigt wurde, wird das Schlüsselpaar und das Zertifikat dem Mobilen Agenten zur Verfügung gestellt. Dieser kann die Schlüssel und das Zertifikat so lange nutzen, wie die Gültigkeitsdauer des Zertifikates nicht abläuft oder der Mobile Agent das Agentensystem nicht verlässt. Der zweite Fall ist im Sequenzdiagramm angegeben. Nach der Migration ändert sich der Name des Mobilen Agenten und das Ziel–Agentensystem muss ihm als LCA ein neues Zertifikat ausstellen. Deshalb muss vor der Migration des Mobilen Agenten das Quell–Agentensystem das Zertifikat im Repository als widerrufen markieren und kann danach das lokal gespeicherte Zertifikat löschen. Erst danach darf der Mobile Agent migriert werden.

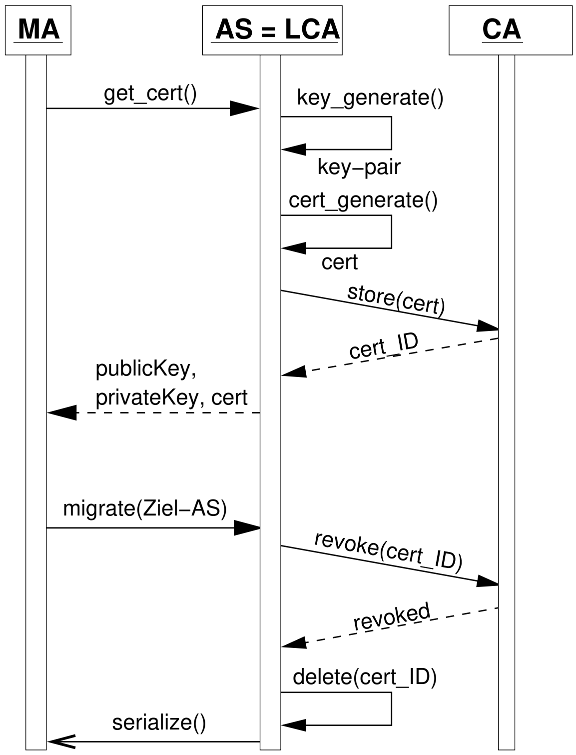


Abbildung 5.6: Agentensystem als Local Certification Authority

Für alle Mobilen Agenten, die aktuell auf dem Agentensystem ablaufen, hält neben der CA auch das Agentensystem die Zertifikate vor und übermittelt diese auf Wunsch auch an anfragende Entitäten.

5.4.2 Authenticator (Verifier)

Die als **Authenticator** bezeichnete Komponente des Agentensystem implementiert die Funktionen, die das Agentensystem als Verifier im Authentisierungsprozess erbringen muss (vgl. Abschnitt 4.2.3 und Tabelle 4.6). Das Agentensystem authentisiert Anwender, Mobile Agenten und andere Agentensysteme sowohl lokal als auch entfernt. Das Endsystem, auf dem das Agentensystem selbst gestartet wurde, kann und muss vom Agentensystem wegen der Einbettungsbeziehung nicht authentisiert werden.

Im Authenticator werden für alle Entitäten, auch für Anwender, starke Authentisierungsverfahren mit starken zertifikatsbasierten kryptographischen Verfahren verwendet.

Authentisierung von Anwender und Agentensystem

Jede Entität besitzt ein oder mehrere Schlüsselpaare und entsprechende Zertifikate. Der Ablauf bei der Authentisierung von Anwendern und Agentensystemen ist der gleiche. In Abbildung 5.7 wird dieser Ablauf exemplarisch für einen Anwender dargestellt. Falls das Agentensystem ein anderes Agentensystem authentisiert, kann, ohne sonstige Veränderungen, das Anwender–Objekt im Sequenzdiagramm durch das AS–Objekt ersetzt werden.

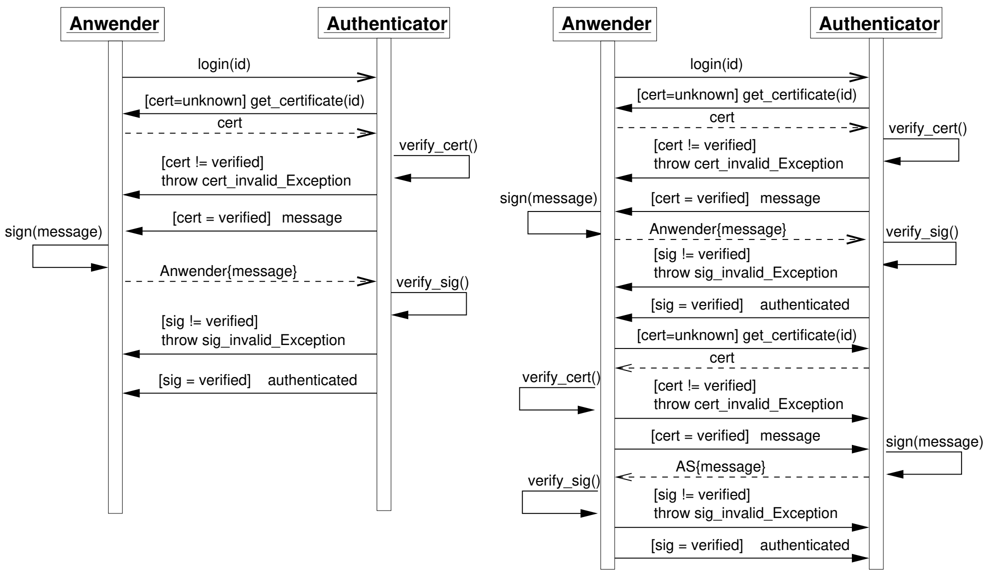


Abbildung 5.7: Authentisierung von Anwendern und Agentensystemen durch den Authenticator (einseitig und beidseitig)

Ablauf der Authentisierung

Der Anwender meldet sich, über seine Client–Software, beim Agentensystem an (`login`). Der Authenticator fordert, falls er das Zertifikat des Anwenders noch nicht kennt, dieses an. Mit dem öffentlichen Schlüssel der signierenden CA wird das Zertifikat (wie in Abbildung 4.7 auf Seite 91 dargestellt) verifiziert. Falls das Anwender–Zertifikat von einer CA stammt, deren öffentlicher

Schlüssel dem Authenticator nicht bekannt ist, muss er versuchen einen Zertifizierungspfad zu berechnen (vgl. Abschnitt 5.2.2), um an ein Zertifikat der zertifizierenden CA zu gelangen und damit das vorgelegte Zertifikat zu verifizieren. Falls die Verifikation erfolgreich war, muss der Anwender noch belegen, dass er auch im Besitz des entsprechenden privaten Schlüssels ist. Die Signatur einer Nachricht des Authenticators ist dafür hinreichend. Denn nur wenn der korrekte private Schlüssel zur Signatur verwendet wurde, kann der Authenticator die Nachricht mit dem öffentlichen Schlüssel aus dem Zertifikat entschlüsseln. Danach ist der Anwender erfolgreich angemeldet.

zweiseitige Authentisierung

Der beschriebene Ablauf stellt die einseitige Authentisierung des Anwenders durch das Agentensystem dar, d.h. das Agentensystem identifiziert den Anwender. Der Anwender kann sich jedoch nicht sicher sein, mit dem „richtigen“ Agentensystem zu kommunizieren. Von einer beidseitigen Authentisierung wird gesprochen, wenn auch der Anwender das Agentensystem authentisiert. In diesem Fall ergibt sich das, in Abbildung 5.7 rechts dargestellte Sequenzdiagramm.

Die Sequenzdiagramme sind für die lokale und auch die entfernte Authentisierung gleich. Unabhängig davon, ob sich der Anwender bzw. dessen Client–Programm auf demselben Endsystem wie das Agentensystem befinden oder sich von einem entfernten Rechner aus anmelden, bleibt der Vorgang der Authentisierung derselbe.

Lokale Authentisierung von MA–Instanz und MA–Gattung

Nachdem der Anwender angemeldet ist, kann er z.B. einen Mobilen Agenten starten und migrieren. An diesem Szenario lässt sich die Authentisierung von MA–Instanz und –Gattung erläutern (vgl. Abbildung 5.8).

Verifikation der Repository– und Implementierer–Signatur

Der Anwender gibt in der `startMA` Methode an, welche MA–Gattung aus welchem Code–Repository er instantiiert haben möchte. Das Agentensystem lädt daraufhin die Gattung. Das Repository signiert die PDE, welche die Gattung enthält. Der Authenticator verifiziert diese Signatur und stellt dadurch sicher, dass die Gattung aus dem gewünschten Repository geladen wurde und verhindert damit Maskerade. Danach wird die Signatur der MA–Gattung selbst verifiziert, um den Implementierer des Agenten eindeutig zu identifizieren. Außerdem ist dadurch gewährleistet, dass der Programm–Code nicht verändert wurde (Integrität durch signierten Hash).

Damit die beiden Signaturen geprüft werden können, muss der Authenticator entweder die öffentlichen Schlüssel des Repository und des Implementierers kennen oder deren Zertifikate anfordern. Bei der MA–Gattung wird das Zertifikat des Implementierers gleich mit übertragen und der Authenticator muss nur noch sicherstellen, dass dieses Zertifikat verifiziert werden kann. Das Zertifikat des Repository kann der Authenticator entweder direkt vom Repository oder von der zuständigen CA anfordern und dann verifizieren.

Am Ende dieses Verifikationsprozesses ist zweifelsfrei gewährleistet, dass die

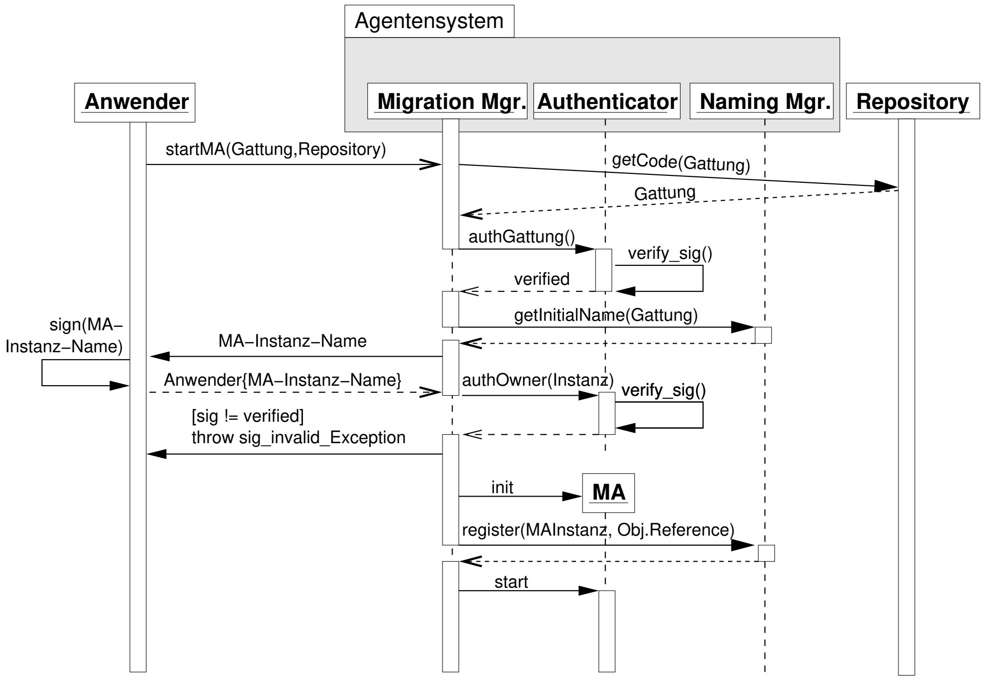


Abbildung 5.8: Instantiierung eines Mobilen Agenten; Authentisierung der Gattung

MA–Gattung aus dem gewünschten Repository stammt, nicht verändert wurde und die Identität des oder der Implementierer ist zweifelsfrei belegt.

Das Agentensystem — genauer der Naming Manager — legt nun den Namen für die MA–Instanz fest (`getInitialName`) und lässt diesen Namen vom Anwender signieren. Durch die digitale Signatur des Namens wird der Anwender zum Besitzer (Authority) des Mobilen Agenten (vgl. Abschnitt 4.2.4). Erst nachdem sich der Besitzer durch die Signatur als für den Mobilen Agenten verantwortlich erklärt hat, wird dieser auch tatsächlich instantiiert (`init`). Nachdem die MA–Instanz über die `register` Schnittstelle des Naming Managers auch im Naming Service registriert wurde, kann der Mobile Agent gestartet werden (`start`).

Feststellung des Verantwortlichen: Verifikation der Signatur im `Authority` Feld

Nach einer Migration kann die Signatur des initialen Namens des Mobilen Agenten, die im `Authority`–Attribut der MA–Instanz steht, jederzeit verifiziert und damit der für den Mobilen Agenten verantwortliche Besitzer bestimmt werden. Der Authenticator bietet hierfür die Schnittstelle `authOwner`. In dieser Methode wird das Zertifikat des Signierenden überprüft. Gegebenfalls muss dies vom Authenticator aus einem Certification Repository geladen und verifiziert werden. Anschließend wird die digitale Signatur des initialen MA–Namens im `Authority` Attribut überprüft. Im Erfolgsfall, d.h. wenn sowohl Zertifikat als auch Signatur gültig sind, wird der Distinguished Name des Signierenden und dessen öffentlicher Schlüssel zurückgegeben. Im Fehlerfall wird eine `NotAuthenticated` Exception erzeugt.

Die Instanz selbst wird über die Verifikation der Signatur des Quell–Agentensystems authentisiert. Hierfür stellt der Authenticator die Methode

Authentisierung der Instanz: Verifikation der Signatur des Quell–AS

`authInstance` zur Verfügung, die analog zur `authOwner` Funktion arbeitet. Da die Authentisierung sowohl von MA–Gattung als auch –Instanz über eine Verifikation einer digitalen Signatur erfolgen, ist diese nur im lokalen Fall sinnvoll und überhaupt möglich.

Entfernte Authentisierung von Mobilen Agenten

Authentisierung über Zertifikat

Falls ein Mobiler Agent, der auf einem entfernten Agentensystem ausgeführt wird, Ressourcen des lokalen Agentensystems nutzen möchte, muss der entfernte Mobile Agent authentisierbar sein. Eine direkte Authentisierung über die Signatur, wie im vorigen Abschnitt beschrieben, ist nicht möglich. Daher wird im entfernten Fall über ein Zertifikat authentisiert. Das entfernte Agentensystem erstellt dem Mobilen Agenten als LCA ein Zertifikat. Dieses wird vom lokalen Agentensystem angefordert und der Mobile Agent kann dann analog zu dem in Abbildung 5.7 dargestellten Ablauf authentisiert werden.

5.4.3 Laufzeitumgebung, Places

Mobile Agenten werden vom Agentensystem innerhalb einer Laufzeitumgebung ausgeführt. Diese Laufzeitumgebung implementiert die Sicherheitsmechanismen, welche die Ausführungsrelation betreffen (vgl. Abschnitt 4.3). Dies betrifft im Wesentlichen die Vertraulichkeit und den Schutz der verschiedenen Mobilen Agenten voreinander. Das heißt, die Laufzeitumgebung muss Speicher– und Laufzeitschutz sowie eine Trennung der Namensräume gewährleisten.

Verwendung von Java realisiert Speicherschutz

Wie in Abschnitt 4.3.2 auf Seite 101 bereits erläutert, kann der Speicherschutz durch die Verwendung geeigneter Programmiersprachen, die keine Methoden zur direkten Speichermanipulation oder Pointer bieten, erreicht werden. Bei den meisten Mobilen Agenten und den entsprechenden Agentensystemen ist der Speicherschutz systemimmanent, durch die Verwendung von Java als Implementierungssprache, gegeben. Für die wenigen Systeme, die nicht in Java implementiert sind, muss ein Speicherschutz nachträglich implementiert werden. Im Folgenden wird von der Verwendung von Java ausgegangen, da dies die mit Abstand am häufigsten verwendete Sprache für die Implementierung von Agentensystemen ist (vgl. auch Abbildung 2.5 auf Seite 25).

Thread Groups und Permission Manager realisieren Laufzeitschutz

Der Laufzeitschutz für Mobile Agenten wird von der Laufzeitumgebung und dem Permission Manager (vgl. Abschnitt 5.4.7) erbracht. Der Permission Manager sorgt dafür, dass nur Berechtigte die Methoden des Mobilen Agenten nutzen dürfen. Bei der Implementierung der Laufzeitumgebung ist darauf zu achten, dass Mobile Agenten sich nicht gegenseitig, an den von ihnen zur Verfügung gestellten Schnittstellen vorbei, beeinflussen können (Circumvention Attack). Das heißt insbesondere, Low–Level Aufrufe von Methoden des Laufzeitsystems sind gesondert zu betrachten. Deshalb sollte jeder Mobile Agent mit allen seinen Sub–Threads eine eigene Thread Group bilden, denn dann kann dieser Agent sehr einfach vor dem Zugriff von Threads aus anderen

Thread Groups geschützt werden [OaWo 99, Oaks 98]. Ein Low–Level Aufruf über Methoden zur Thread Manipulation kann damit verhindert werden.

Um Namenskonflikte zwischen verschiedenen Agenteninstanzen sowie zwischen Agenteninstanz und Agentensystem — die beim Instantiieren von Objekten und beim Aufruf von Methoden auftreten können — zu verhindern, ist eine Trennung der Namensräume erforderlich (vgl. Abschnitt 4.3.2). Dies lässt sich in Java (ab Java 2 SDK [GJSB 00]) durch den Class Loader realisieren. Der Class Loader ist für das Laden und Instantiieren von Java Byte Code zuständig, von der Class Loader Klasse können eigene Class Loader Objekte abgeleitet und instantiiert werden. Alle Klassen, die von einem bestimmten Class Loader Objekt geladen wurden, bilden einen Namensraum [Gong 98, Java2]. Um Namenskonflikte, wie in Abbildung 4.11 dargestellt, zu verhindern, wird für jeden Mobilen Agent ein eigener Class Loader verwendet. Zur Laufzeit werden Objekte nicht nur durch ihren Namen, sondern durch ihren vollständigen Package Name und den Class Loader, der die entsprechende Klasse geladen hat, identifiziert. Der Class Loader wird daher auch als Defining Class Loader bezeichnet [Gong 98a, LiYe 99].

Class Loader realisiert Trennung der Namensräume

Die Thread Group, innerhalb derer ein Mobiler Agent ausgeführt wird, und sein Namespace, der durch den Defining Class Loader bestimmt wird, stellen den Ausführungskontext eines Mobilen Agenten dar. Dieser Ausführungskontext wird im MASIF Standard [MASIF] als Place bezeichnet (vgl. Abb. 2.4).

5.4.4 Migration Manager

Die Komponente des Agentensystems, die für die Migration von Mobilen Agenten verantwortlich ist, wird als **Migration Manager** bezeichnet. Er implementiert eine oder mehrere der in Abschnitt 2.5 beschriebenen Migrationssemantiken. Der Migration Manager ist sowohl für abgehende als auch für ankommende Mobile Agenten verantwortlich. Der Migration Manager ist die erste und die letzte Komponente, die den Lebenszyklus eines Mobilen Agenten auf einem Agentensystem beeinflusst. Aus diesem Grund implementiert er auch einige grundlegende Sicherheitsmechanismen. In den nächsten Abschnitten werden diese beschrieben. Dazu wird der Ablauf der Migration eines Mobilen Agenten auf ein anderes Agentensystem betrachtet. Es werden abgehende und ankommende Mobile Agenten unterschieden.

Abgehende Mobile Agenten

Der Migration Manager stellt für die Migration eines Mobilen Agenten die Aufrufschnittstelle `MigrateAgent` zur Verfügung. Beim Aufruf muss der Name des zu migrierenden Mobilen Agenten und der Name des Ziel–Agentensystems angegeben werden. Die Methode kann sowohl vom Mobilen Agenten selbst als auch von anderen, entsprechend autorisierten Entitäten aufgerufen werden.

Ermittlung des Vertrauenslevel des Ziel–AS

Falls der Mobile Agent selbst die Funktion nutzt, muss er vorher den Vertrauenslevel des Ziel–Agentensystems bestimmen können. Der Mobile Agent ruft dazu, mit dem Namen des Ziel–Agentensystems, die Schnittstelle `TrustLevel` des Migration Managers auf (vgl. auch Sequenzdiagramm in Abbildung 5.9). Dieser implementiert das in Abschnitt 4.1.2 beschriebene Verfahren. Als Ergebnis erhält der Mobile Agent einen Vertrauenslevel zwischen -1 und 4 und kann in Abhängigkeit davon entscheiden, ob er auf das Ziel–Agentensystem migriert. Nach einem Aufruf von `MigrateAgent` muss

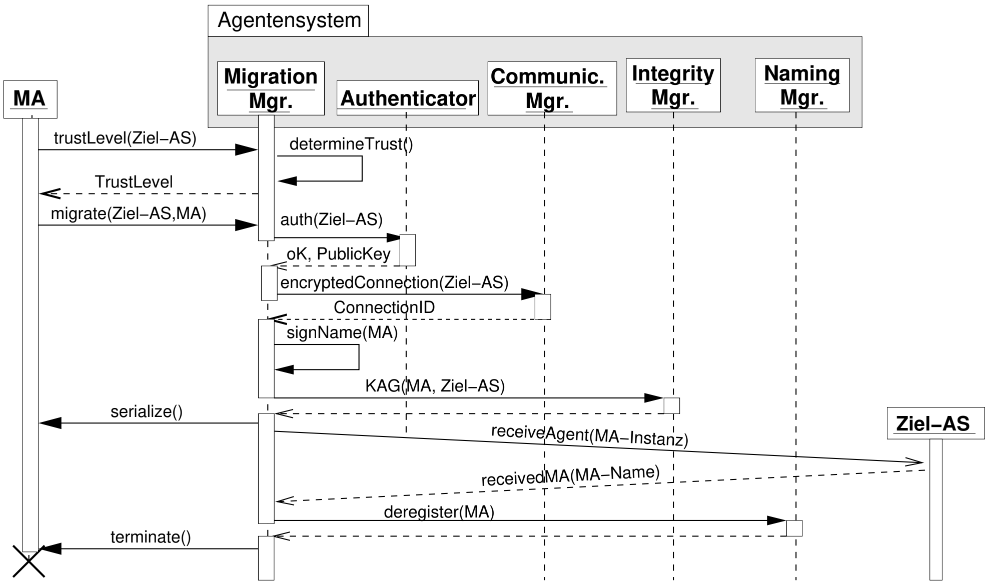


Abbildung 5.9: Migration Manager; Mobiler Agent verlässt Agentensystem (Sequenzdiagramm)

der Migration Manager sicherstellen, dass der Mobile Agent nur auf das entsprechende Ziel–Agentensystem übertragen wird. Er muss die Identität des Ziel–Agentensystems zweifelsfrei bestimmen und eine Umleitung oder Duplizierung des Mobilen Agenten bei dessen Übertragung verhindern. Für die Bestimmung der Identität des Ziel–Agentensystems kann sich der Migration Manager auf den Authenticator abstützen. Dieser bestimmt, wie in Abschnitt 5.4.2 beschrieben, die Identität und den öffentlichen Schlüssel des Ziels. Mit dem Schlüssel kann der in Abschnitt 5.4.5 beschriebene Communication Manager eine verschlüsselte Verbindung (sicherer und vertraulicher Kanal) zum Ziel–Agentensystem aufbauen. Das Quell–Agentensystem kommuniziert ab diesem Zeitpunkt ausschließlich über diesen Kanal — der über die `Connection_ID` identifiziert wird — mit dem Ziel–Agentensystem. Der Mobile Agent wird über diese verschlüsselte Verbindung zum Ziel–Agentensystem übertragen.

Identifikation des Ziel–AS

Aufbau eines vertraulichen Kanals zum Ziel

Umleitung und Replay–Angriff werden verhindert

Eine Umleitung des Mobilen Agenten bzw. der Protokoll–Dateneinheiten, die den Mobilen Agenten enthalten, ist zwar weiterhin möglich, aber nur das Ziel–Agentensystem kann den Mobilen Agenten entschlüsseln und ausführen. Andere Agentensysteme, die umgeleitete Mobile Agenten empfangen, werden diese verwerfen.

Ein böswilliges System, das sich zwischen Quell– und Ziel–Agentensystem befindet, kann die (verschlüsselten) Protokoll–Dateneinheiten zwar kopieren, aber weder entschlüsseln noch für einen Replay–Angriff verwenden. Bei diesem Angriff versucht der Angreifer den kopierten Mobilen Agenten zu einem späteren Zeitpunkt erneut zum Ziel–Agentensystem zu übertragen und dort ausführen zu lassen. Der Replay–Angriff wird durch Sequenznummern für Nachrichten, die innerhalb der verschlüsselten Sitzung übertragen werden, erkannt (vgl. auch Abschnitt 5.4.5). Der aktuelle Name des Mobilen Agenten, in den auch der Name des ausführenden Agentensystems eingeht, wird vom Quell–Agentensystem vor der Übertragung digital signiert (vgl. Abschnitt 4.2.4, Seite 95). Über diese Signatur kann das Ziel–Agentensystem die Identität der MA–Instanz verifizieren und anschließend neu festsetzen.

Quell–AS signiert Namen des MA

Danach wird der Integrity Manager (vgl. Abschnitt 5.4.6) aufgerufen, um bei Bedarf ein Protokoll der KAG–Klasse auf den Daten des Mobilen Agenten auszuführen und die Verkettungsrelation zu bilden. Im Anschluss daran wird der Mobile Agent serialisiert und über den Aufruf der `receiveAgent` Methode beim Ziel–Agentensystem auf dieses migriert. Nach der bestätigten Ankunft auf dem Ziel–AS wird der Naming Manager dazu veranlasst (`deregister`) den Mobilen Agenten unter seinem alten Namen aus dem Naming Service zu entfernen.

Integrity Manager bildet KAG–Verkettungsrelation und History

Ankommende Mobile Agenten

Über die `receiveAgent` Schnittstelle wird die serialisierte Agenteninstanz vom Quell– zum Ziel–Agentensystem übertragen. Diese Methode wird vom Migration Manager implementiert. Bevor der ankommende Mobile Agent instantiiert und gestartet werden kann, sind sicherheitsrelevante Tests durchzuführen (vgl. Sequenzdiagramm in Abbildung 5.10). Zuerst wird der Mobile Agent und der verantwortliche Besitzer ermittelt und authentisiert. Der Migration Manager ruft dazu die Funktion `authOwner` und `authInstance` beim Authenticator auf. Im ersten Fall wird die Signatur im Authority Attribut verifiziert. Bei einer erfolgreichen Authentisierung wird der Distinguished Name des Besitzers und dessen öffentlicher Schlüssel zurückgegeben. Durch die Verifikation der digitalen Signatur der MA–Instanz wird auch gewährleistet, dass jede Veränderung erkannt wird. Nachdem Besitzer, Quelle und Instanz identifiziert sind, kann die Gattung aus einem Repository geladen werden. Auch die Gattung wird, wie in Abschnitt 5.4.2 beschrieben, authentisiert.

Besitzer ermitteln

Instanz authentisieren

Der Naming Manager (vgl. Abschnitt 5.4.8) vergibt dann einen neuen eineindeutigen Namen für den Mobilen Agenten. Unter diesem Namen wird der Mobile Agent instantiiert und danach vom Naming Manager beim Namens– und Lokalisierungsdienst (z.B. MAFFinder in [MASIF]) registriert. Die integritätsgesicherten Daten, die der Mobile Agent transportiert, werden vom Integrity Manager überprüft. Er testet bspw., ob die KAG–Verkettungsrelation noch hält oder verletzt wurde. Als letzter Schritt wird der Mobile Agent gestartet.

Namensvergabe

Integritätstest

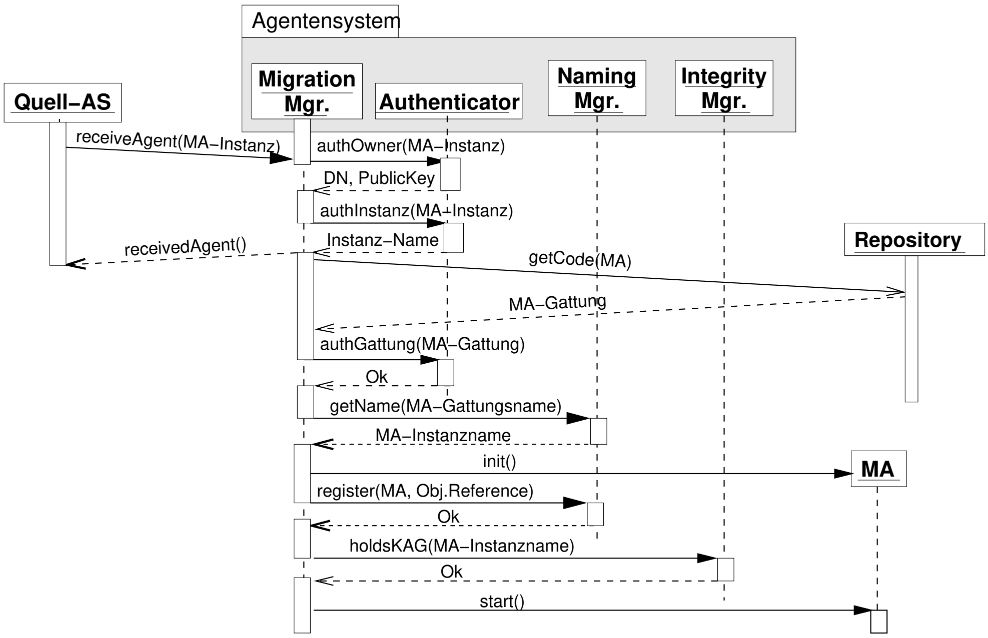


Abbildung 5.10: Migration Manager: Ankunft eines Mobilen Agenten

Reisehistorie des Mobilen Agenten

Signaturen der besuchten Agentensysteme bilden Historie

Die Reisehistorie eines Mobilen Agenten wird aus der Summe der Migration Manager der besuchten Agentensysteme als „Nebenprodukt" der Authentisierung gebildet. Jeder Migration Manager bzw. jedes Agentensystem signiert den Namen des Agenten bevor dieser das Agentensystem verlässt. Dadurch bildet sich eine Kette von signierten Agentennamen. Diese Kette ist allerdings per se nicht geschützt. Von einem Angreifer, z.B. einem böswilligen Agentensystem auf dem Reiseweg des Agenten, könnte die Reihenfolge verändert oder es könnten Teile der Kette gelöscht werden.

Schutz der Historie durch KAG Protokoll

Um eine gesicherte, d.h. unveränderbare Historie zu erhalten, muss auf die Elemente der Kette lediglich ein Protokoll aus der KAG–Klasse, entsprechend den jeweils konkreten Anforderungen (vgl. Tabelle 4.9, Seite 131), angewendet werden. Der Migration Manager trägt den signierten Namen in eine vom Integrity Manager gebildete History–Liste ein. Die Bildung dieser Listen durch den Integrity Manager wird in Abschnitt 5.4.6 beschrieben.

5.4.5 Communication Manager

verbindungsorientiertes, verschlüsselndes Protokoll

Die Aufgabe des **Communication Managers** ist die Realisierung der Kommunikationsrelation zwischen Entitäten. Technisch gesehen ist der Communication Manager eine Protokollinstanz für ein verbindungsorientiertes, verschlüsselndes Protokoll, das sich aus einem Handshake und einem Session Protokoll zusammensetzt. Entitäten nutzen den Communication Manager über die Schnittstelle `connection(Ziel)`, um eine Verbindung zur Ziel–Entität zu bekommen. Der Communication Manager kommuniziert mit dem

Communication Manager des Ziel–AS, über den die eigentliche Ziel–Entität erreichbar ist. Um einen verschlüsselten Kanal zu etablieren, wird zuerst eine beidseitige Authentisierung der beiden beteiligten Agentensysteme durchgeführt (vgl. Abbildung 5.7) für die der Authenticator verantwortlich ist. Danach ist jeder Communication Manager im Besitz des öffentlichen Schlüssels der Partner–Instanz, mit dessen Hilfe ein **Handshake Protokoll** zur Aushandlung des Verfahrens und der Parameter der Verschlüsselung durchgeführt wird.

In Abbildung 5.11 ist dieser Handshake dargestellt, der vom Communication Manager des Quell–AS, im Folgenden als Quelle bezeichnet, initiiert wurde.

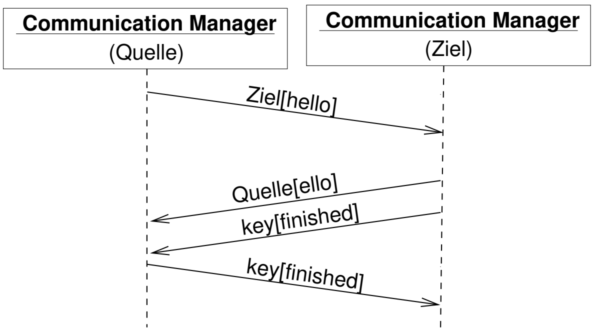


Abbildung 5.11: Handshake zwischen zwei Communication Managern

Die Quelle schickt eine `hello` Nachricht an das Ziel, die mit dem öffentlichen Schlüssel des Ziels verschlüsselt ist. Das `hello` (vgl. Abbildung 5.12) besteht aus einer Liste von Security Parameter Objekten und einem Key_Material Objekt. Ein Security Parameter Objekt definiert die Eigenschaften der Verbindung. In der `hello` Nachricht teilt die Quelle ihre Präferenzen bezüglich der anzuwendenden Verschlüsselungs–, MAC– und Kompressions–Algorithmen — und zwar in absteigender Priorität — mit. Aus den Daten des Key_Material wird der Sitzungsschlüssel für das ausgewählte Verschlüsselungsverfahren berechnet.

Handshake Protokoll

Aushandlung der zu verwendenden Algorithmen

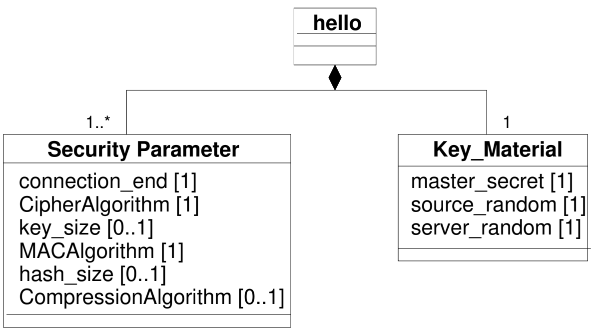


Abbildung 5.12: Struktur der `hello` Nachricht

Das Ziel antwortet auf die `hello` Nachricht mit einer `ello` Nachricht, die nur noch das Security Parameter Objekt enthält, das die Eigenschaften der

Verbindung festlegt. Außerdem wird das Key_Material ergänzt und ebenfalls mit zurückgeschickt. Im unmittelbaren Anschluss daran sendet das Ziel eine `finished` Nachricht, die bereits mit dem vereinbarten Verfahren und dem entsprechenden Schlüssel verschlüsselt ist. Die Quelle antwortet ebenfalls mit einer verschlüsselten `finished` Nachricht und ab diesem Zeitpunkt ist der vertrauliche Kanal zwischen Quelle und Ziel etabliert.

Session Protokoll

Ab diesem Zeitpunkt beginnt das **Session Protokoll** zur Übertragung der Nutzdaten. Jeder Communication Manager verwendet Sequenznummern für seine abgehenden Nachrichtenpakete. Die Nutzdaten werden mit der Sequenznummer konkateniert und darüber wird der vereinbarte MAC berechnet. Nutzdaten, Sequenznummer und MAC werden in einer Session Protocol Entity zusammengefasst, die vor der Übertragung mit dem vereinbarten Schlüssel und dem gewählten Verfahren verschlüsselt wird.

Vertraulichkeit durch Verschlüsselung

Die Vertraulichkeit der Kommunikationsrelation zwischen Quell– und Ziel–Entität wird durch die Verschlüsselung gewährleistet. Auch die Vertraulichkeit der Migrationsrelation ist sichergestellt, da auch die MA–Instanzen nur über verschlüsselte Kanäle zum Ziel–AS übertragen werden.

Replay Protection durch Sequenznummern

Über die Sequenznummern im Session Protokoll können Replay–Angriffe erkannt werden. Der Communication Manager, der eine Session Protocol Entity mit einer ungültigen Sequenznummer empfängt, verwirft diese.

Integrität durch MAC

Durch den MAC werden Veränderungen während der Übertragung erkannt. Falls der vom Communication Manager berechnete MAC nicht mit dem übertragenen übereinstimmt, wird die Session Protocol Entity verworfen und neu angefordert.

Der vom Communication Manager zur Verfügung gestellte Kanal zwischen den beiden kommunizierenden Entitäten realisiert damit folgende Sicherheitsanforderungen:

- Vertraulichkeit
- Integrität
- Schutz vor Wiedereinspielung (Replay Protection)

Der Communication Manager wird so ins Agentensystem integriert, dass nur über ihn eine Kommunikations– oder Migrationsrelation etabliert werden kann. Dadurch wird z.B. eine direkte und unverschlüsselte Kommunikation zwischen zwei Entitäten — und damit eine Schwächung der Sicherheitsarchitektur — verhindert.

5.4.6 Integrity Manager

KAG–Protokollklassen zum Schutz transportierter Daten

Der **Integrity Manager** ist für einen Teilaspekt der Integrität der Migrationsrelation verantwortlich. Er schützt die von der MA–Instanz transportierten veränderlichen und unveränderlichen Daten durch die Implementierung der KAG–Protokollklassen (vgl. Abschnitt 4.5.2). Die Integrität der Kommunikationsrelation sowie der Protokolldateneinheiten, in denen der Mobile Agent vom Quell– zum Ziel–Agentensystem übertragen wird, wird vom Communication Manager (vgl. Abschnitt 5.4.5) gesichert.

Der Besitzer des Agenten bzw. der Mobile Agent selbst muss am Beginn der Reise festlegen können, welche Daten durch welche KAG–Protokollklasse gesichert werden sollen. Der Integrity Manager bietet über die in Abbildung 5.13 dargestellte Datenstruktur die Möglichkeit, dies zu spezifizieren.

Datenstruktur für KAG–Integritätslisten

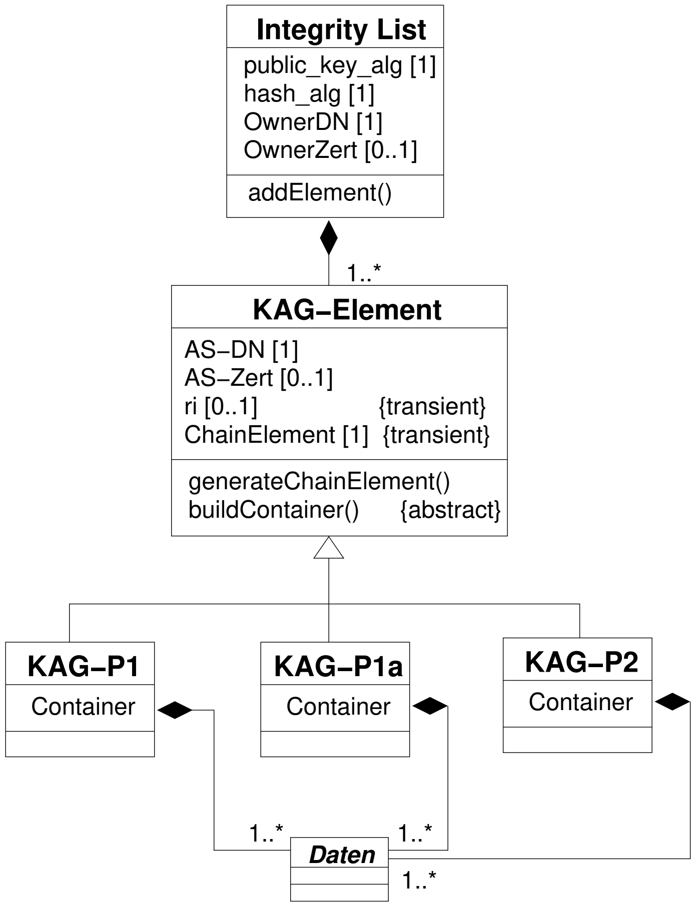


Abbildung 5.13: Integrity Manager: Datenstruktur für KAG–Integritätslisten

Die Auswahl erfolgt über die Instantiierung von KAG–x Objekten. Für alle Daten, die mit demselben KAG–Protokoll gesichert werden sollen, wird ein entsprechendes KAG–x Objekt instantiiert. Für unterschiedlich zu schützende Daten können auch verschiedene KAG–x Objekte und damit verschiedene Integritätslisten gebildet werden. Die Datenstruktur für die Integritätsliste wird vom Integrity Manager zur Verfügung gestellt und der Mobile Agent nutzt sie durch Instantiierung von KAG–x Objekten. Der Agentenprogrammierer muss sich nicht um die Implementierung der Methoden des KAG–Protokolls kümmern und der Agent bleibt „schlank“.

Auswahl der Protokollklasse über Instantiierung von KAG–x Objekten

Eine Integritätsliste besteht aus `KAG-Element` Objekten. Pro besuchtem Agentensystem und KAG–Protokollklasse gibt es ein KAG–Element. Diese Klasse beinhaltet Attribute, die alle KAG–Protokollklassen gemeinsam

KAG-Element: Attribute aller Protokollklassen

haben. Hier wird der Name des Agentensystems und optional auch dessen Zertifikat, die Zufallszahl r_i und das eigentliche Verkettungselement (ChainElement) zusammengefasst. Mit der Methode generateChainElement wird dieses Verkettungselement berechnet. Der zugrunde liegende Algorithmus ist für alle Protokollklassen gleich. Es wird dabei der Hash–Wert über den Container des Agentensystems, das der Mobile Agent als letztes besucht hat und dem DN des (Ziel–) Agentensystems, das er als nächstes besuchen wird, gebildet:

$$\texttt{ChainElement}_i := H\left(\texttt{Container}_{i-1} \bullet DN(AS_{i+1})\right)$$

Protokollklassen unterscheiden sich durch buildContainer Methode

Das Verkettungselement, in das Daten des vorausgehenden und des nachfolgenden Agentensystems eingehen, wird mit den eigenen zu schützenden Daten verknüpft. An dieser Stelle unterscheiden sich die verschiedenen KAG–Protokollklassen; sie implementieren hierzu verschiedene Algorithmen. Aus diesem Grund wird die Methode buildContainer() im KAG–Element abstrakt definiert. Jedes Objekt der Klasse KAG–x muss diese Methode selbst implementieren. Im Falle von KAG–P1 werden die zu schützenden Daten mit der Zufallszahl r_i konkateniert und dann mit dem öffentlichen Schlüssel (O_p) des Besitzers der Liste verschlüsselt. Dieser Schlüssel lässt sich entweder direkt aus dem OwnerZert aus der Liste oder mittelbar über den Distinguished Name aus OwnerDN ermitteln. Die so verschlüsselten Daten werden mit dem ChainElement verknüpft und dann vom Agentensystem digital signiert. Die verschiedenen Algorithmen die in der Methode buildContainer bei den KAG–x Objekten implementiert werden, sind in Tabelle 5.2 zusammengefasst.

Protokollklasse	buildContainer Algorithmus; $\texttt{Container}_i :=$
KAG–P1	$AS_i\,\{O_p\,[\prod \text{Daten} \bullet r_i] \bullet \texttt{ChainElement}_i\}$
KAG–P1a	$AS_i\,\{\prod \text{Daten} \bullet \texttt{ChainElement}_i\}$
KAG–P2	$O_p\,[AS_i\,\{\prod \text{Daten}\} \bullet r_i] \bullet \texttt{ChainElement}_i$

Tabelle 5.2: Algorithmen zur Bildung von KAG–Integritätslisten

An dieser Stelle wird auch ersichtlich, wieso $\texttt{ChainElement}_i$ und r_i als transient spezifiziert sind. **Transiente Attribute** oder Objekte werden bei der Serialisierung nicht mit serialisiert und sie werden damit auch nicht Teil der migrierenden MA–Instanz. Die Zufallszahl r_i darf gar nicht im Klartext auf anderen Agentensystemen lesbar sein, sonst würde die Zahl nicht mehr als probabilistischer Faktor wirken. Sie wird deshalb nur verschlüsselt, innerhalb des Containers, übertragen. Das ChainElement geht als Teil der Signatur, bzw. bei KAG–P2 im Klartext, mit in den Container ein.

Der Integrity Manager selbst bietet zwei Schnittstellen KAG(MAInstanz : Name, ZielAS : Name) und KAG_hold(MAInstanz : MAInstanz). Beim Aufruf der ersten Methode wird die Verkettungsrelation gebildet und die Integritätsliste fortgeschrieben. Der Integrity Manager ruft dazu auf dem letzten KAG–Element der Integritätsliste `generateChainElement` und `buildContainer` auf. Dies wird für alle vom Agenten transportierten Integritätslisten wiederholt. `KAG()` wird vom Migration Manager aufgerufen, bevor der Mobile Agent das Agentensystem verlässt (vgl. auch Abbildung 5.9).

Der Integrity Manager überprüft in der `KAG_hold` Methode, ob die Verkettungsrelation noch hält, d.h. ob die Integrität der Daten gewahrt ist. Dazu muss er für das letzte KAG–Element der Liste folgendes testen:

Verifikation der Verkettungsrelation

$$H(\text{Container}_{i-1} \bullet DN(AS_{i+1})) \stackrel{?}{=} \text{ChainElement}_i$$

Er entnimmt aus dem jeweiligen KAG–Element das `ChainElement`$_i$. Dann berechnet er das `ChainElement` selbst aus den mitübertragenen Daten, d.h. er bildet den Hash über den `Container`$_{i-1}$ und seinen eigenen Distinguished Name ($= DN(AS_{i+1})$). Wenn beide Werte gleich sind, hält die Verkettungsrelation. Falls die Werte verschieden sind, ist die Integrität nicht mehr gewahrt. Der Integrity Manager muss dann das `ChainElement`$_k$ finden, für das die Verkettungsrelation noch hält. Allen Daten, die von Agentensystemen vor dem Agentensystem k erzeugt wurden, kann vertraut werden. Ab dem Agentensystem k ist die Integrität nicht mehr sichergestellt. Ursache dafür kann eine Manipulation der Daten des Agentensystems k sowie eine Einfügung oder die Löschung von Containern aus der Liste sein (vgl. auch Tabelle 4.9).

In Abbildung 5.14 ist beispielhaft die Reise eines Mobilen Agenten von Agentensystem AS_0 bis zum Agentensystem AS_3 dargestellt. Die vom jeweiligen Agentensystem erzeugten Container, die vom Mobilen Agenten als Teil der Integritätsliste transportiert werden, sind rechts im Bild dargestellt. Wie der Container$_i$ in die Berechnung des `ChainElement`$_{i+1}$ eingeht, ist ebenfalls symbolisiert. Das Agentensystem AS_3 überprüft die Integrität der Liste, indem es die Integrität des letzten `ChainElement` der Liste testet.

5.4.7 Permission Manager

Der **Permission Manager** implementiert das in Abschnitt 4.4.1 beschriebene rollenbasierte Zugriffskontrollmodell RBAC. Jeder Permission Manager verwaltet Agentensystem–lokal seine Zugriffsmatrix $M' = |R| \times |O|$. In den Zellen der Matrix, die durch Rollen und Objekte indiziert werden, stehen die Rechte, welche die Rolle an dem Objekt besitzt. Jede zu schützende Ressource, z.B. ein Attribut, eine Methode oder eine Entität, kann ein Objekt innerhalb der Matrix sein.

Wie bereits in Abschnitt 4.4.2 erläutert, wird eine benutzerbestimmbare Autorisierung (DAC) favorisiert. Der Permission Manager muss also Möglichkeiten zur Vergabe sowie zur Durchsetzung von Rechten bieten.

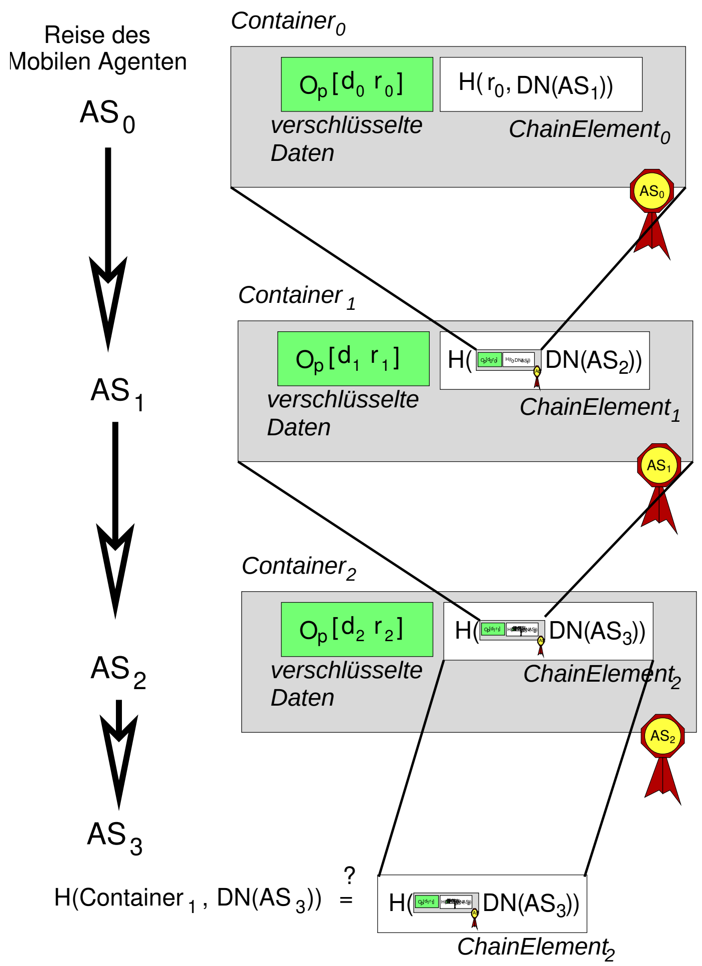


Abbildung 5.14: Beispiel für eine KAG–P1 Integritätsliste

Autorisierung, Rechtespezifikation

Permission Manager als zentrale Komponente zur Rechtedurchsetzung

Der Permission Manager ist für die Durchsetzung von Rechten an Ressourcen des Agentensystems, des Endsystems und — in seiner Stellvertreterrolle — auch für die Rechte an Ressourcen der Mobilen Agenten verantwortlich. Die zu schützenden Objekte des Agentensystems und des Endsystems sind relativ statisch und im Vorhinein bekannt. Der jeweilige Besitzer von Agentensystem, Endsystem und Mobilem Agenten spezifiziert die Rechte an seinen Entitäten, d.h. der Besitzer einer Entität darf in alle Objektspalten der Matrix, die seine Entität betreffen, schreibend zugreifen.

Um die Matrix aufbauen zu können, müssen sowohl die Rollen als auch die Objekte bekannt sein. Das der Matrix zugrunde liegende Rollenmodell wird lokal beim Agentensystem bzw. für alle Agentensysteme in einer abgeschlossenen Domäne, vom Sicherheitsadministrator festgelegt. Die statischen Objekte des Agentensystems und des Endsystems können sofort in die Matrix eingetragen werden. Anders stellt sich der Sachverhalt für Mobile Agenten dar. Die zu schützenden Objekte des Mobilen Agenten müssen dem Agentensystem bekannt gemacht werden, da nicht angenommen werden kann, dass das Agentensystem alle Objekte aller Mobilen Agenten „kennt“. Die Rechte werden vom Besitzer der Mobilen Agenten spezifiziert. Der Agent trägt eine Rechtebeschreibung für seine Objekte mit sich. Wenn er aus einer fremden Domäne kommt, in der ein anderes Rollenmodell verwendet wird, müssen auch diese neue Rollen dem Agentensystem bekannt gemacht werden. Der Mobile Agent trägt dazu gewissermaßen seine eigene „kleine“ Zugriffsmatrix mit sich.

Besitzer spezifiziert Rechte an seinen Entitäten

Nachdem ein Mobiler Agent auf dem Agentensystem ankommt, wird diese Zugriffsmatrix mit in die Matrix des Agentensystems übernommen. Dabei stellt der Permission Manager sicher, dass nur Rechte übernommen werden, die Objekte des Mobilen Agenten betreffen. Ein Mobiler Agent darf keine Rechte an Objekten anderer Entitäten verändern. Auch die neuen Rollen aus der Rechtespezifikation des Mobilen Agenten werden mit in die Matrix übernommen. Den Rollen, die bisher noch nicht in der Matrix eingetragen waren, dürfen entsprechend auch nur Rechte an den Objekten des Mobilen Agenten eingeräumt werden.

Rechtespezifikation des Mobilen Agenten wird in die Matrix des AS übernommen

Die bisher dargestellte Autorisierung ist statisch, da die Rechte immer am Beginn des Lebenszyklus einer Entität auf einem Agentensystem gesetzt werden. Um den dynamischen Aspekten und kooperativen Problemlösungsstrategien gerecht werden zu können, müssen Rechte auch dynamisch, d.h. zur Laufzeit einer Entität verändert, hinzugefügt und entzogen werden können.

Der Permission Manager stellt hierzu die Schnittstellen `addPerm (Object, Role, Permission)` und `removePerm (Object, Role, Permission)` zur Verfügung, mit denen die Zellen der Matrix editiert werden können. Der Aufruf gilt der Zelle der Matrix, die durch `Object` und `Role` indiziert wird. In dieser Zelle wird `Permission` eingetragen bzw. entfernt. Auch die Verwendung dieser Methoden unterliegt der Zugriffskontrolle durch den Permission Manager.

dynamische Autorisierung

Falls der Mobile Agent das Agentensystem wieder verlässt, wird die MA–lokale Zugriffsmatrix nochmals betrachtet. Alle Rechte und alle Objekte, die darin spezifiziert sind, werden aus der Matrix entfernt. Für die Rollen gilt grundsätzlich dasselbe, außer der Rolle wurden von anderen Entitäten zusätzliche Rechte erteilt. Dann darf der Rolleneintrag nicht entfernt werden. Der Permission Manager überprüft die Rollenzeile falls dort nur Rechte am zu migrierenden Mobilen Agenten eingetragen sind, kann die Rolle aus der Matrix gelöscht werden.

Bereinigung der Matrix beim Verlassen des MA

Rechtedurchsetzung

Eine wichtige Anforderung an das vom Permission Manager durchzusetzende Rechtekonzept ist eine feingranulare objektspezifische Kontrolle. Es muss möglich sein, den Zugriff auf Methodenebene der Entität zu kontrollieren. Das bedeutet, dass während der Ausführung der Entität der **Focus of Control (Fokus der Ausführung)** an den Permission Manager übergehen muss.

Permission Manager unterbricht zur Rechtedurchsetzung die Aufrufrelation

Der Permission Manager entscheidet dann in Abhängigkeit der Einträge in der Matrix, ob die aufrufende Entität berechtigt ist, die Methode zu verwenden. Falls ja, geht der Focus of Control zurück zur Entität und die Methode wird ausgeführt. Falls der Aufrufer nicht berechtigt ist, wird eine Security Exception, als Antwort auf den Aufruf, generiert.

Der Permission Manager wird damit zur zentralen Kontrollinstanz zwischen aufrufender und aufgerufener Entität. Er unterbricht die Aufrufrelation, um Zugriffskontrollentscheidungen zu treffen.

Eine Implementierung der Unterbrechung der Aufrufrelation muss zwei Anforderungen genügen:

1. Der zusätzliche Aufwand für den Programmierer einer Entität, den er zum Schutz einer Methode aufwenden muss, soll so klein wie möglich sein.
2. Die Rechteüberprüfung durch den Permission Manager muss für einen berechtigten Benutzer so transparent wie möglich erfolgen.

Für die Umsetzung einer Unterbrechung der Aufrufrelation gibt es folgende Möglichkeiten:

Kapselung des zu schützenden Objektes durch ein Guard–Objekt

- **Guard–Objekt**: Das zu schützende Objekt, bzw. die zu schützende Methode, wird innerhalb des Guard–Objektes gekapselt [GoSc 98]. Dies erfordert im Normalfall einen zusätzlichen Methodenaufruf im Programmtext, um das Guard–Objekt zu instantiieren. Um bspw. den Zugriff auf das `Accounting`–Objekt zu sichern, muss die in Abbildung 5.15 dargestellte Programmzeile in den Programm–Code eines Mobilen Agenten, der das `Accounting`–Objekt implementiert, eingebaut werden.

```
AccountGuard = new GuardObjekt(Accounting,
                               PermissionManager)
```

Abbildung 5.15: Verwendung eines Guard–Objektes

Problem: nicht transparent für berechtigte Benutzer

Der Aufwand für den Programmierer ist relativ gering, Anforderung eins wird damit erfüllt. Das Objekt, das im Guard–Objekt gekapselt wird, ist dann allerdings nicht mehr direkt erreichbar. Das Guard–Objekt (im Bsp. `AccountGuard`) besitzt als einzige Schnittstelle die Methode `getObject()`. In dieser Funktion wird die Kontrolle an den Permission Manager übergeben, der die Rechte des Aufrufers überprüft. Falls dieser das entsprechende Recht besitzt, wird eine Referenz auf das `Accounting`–Objekt zurückgeliefert. Dieser Indirektionsschritt kann für den Aufrufer nicht mehr transparent realisiert werden und er muss wissen, dass das `Accounting`–Objekt, das er eigentlich nutzen möchte, durch das Guard–Objekt `AccountGuard` gekapselt ist. Der Benutzer kann das `Accounting`–Objekt nicht mehr direkt aufrufen. Dies widerspricht klar Anforderung zwei. Aus diesem Grund sollten Guard–Objekte nicht für die Implementierung des Permission Managers verwendet werden.

```
public double getThroughput() {
        PermissionManager.check();
        ⋮
}
```

Abbildung 5.16: Direkter Aufruf des Permission Managers durch eine Check–Methode

- Direkter Aufruf des Permission Managers durch eine **Check–Methode** als erste Anweisung innerhalb der zu schützenden Methode: Dadurch wird der Focus of Control explizit an den Permission Manager übergeben. In Abbildung 5.16 ist als Beispiel angegeben, wie die Methode `getThroughput` des `Accounting`–Objektes gesichert werden kann. Der Aufwand mit einem zusätzlichen Methodenaufruf für jede zu sichernde Methode ist gering. Für den berechtigten Benutzer ist der Wechsel des Focus of Control transparent. Der Check–Aufruf liefert einem berechtigten Aufrufer als Return–Wert `True`. Bei einem unberechtigten Benutzer wird eine Security–Exception generiert.

Check–Methode innerhalb der zu schützenden Methode

- **Interceptor**: Das Prinzip des Interceptors wurde mit CORBA [OMG 01-02-33] als eine Erweiterung des ORB (Object Request Broker) eingeführt. Logisch gesehen unterbricht der Interceptor den Aufruf– und den Antwort–Pfad zwischen Client und Server in einer verteilten Anwendung (vgl. Abbildung 5.17 und Abschnitt 3.1). Ein **Request–Level Interceptor** unterbricht diesen Pfad auf Ebene der Aufrufe, ein **Message–Level Interceptor** auf Nachrichtenebene. Durch das Interceptor Konzept lassen sich sehr einfach zusätzliche ORB–Dienste, wie z.B. ein Security Service [OMG 2001-03-08], in einen bestehenden ORB „einhängen". Bei jedem Aufruf zwischen Client und Server wird dieser vom Security Interceptor unterbrochen, um sicherheitsrelevante Entscheidungen zu treffen. Das Interceptor–Prinzip, das für eine verteilte

Interceptor unterbricht Aufrufe und Antworten zwischen Client und Server

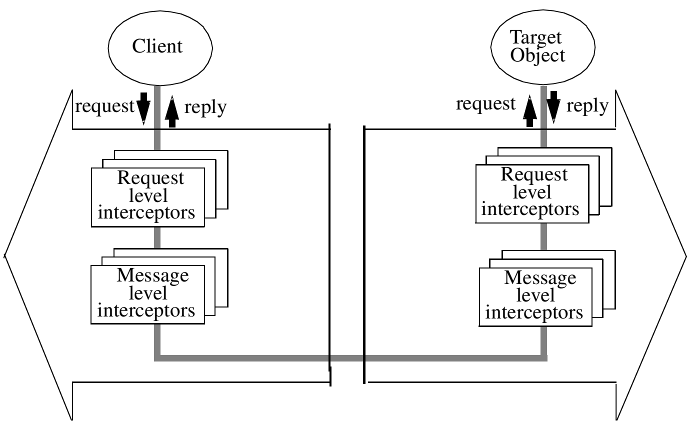


Abbildung 5.17: CORBA Interceptor Konzept aus [OMG 01-02-33]

Anwendung spezifiziert wurde, ließe sich auch lokal auf einem Agentensystem implementieren, um die Aufrufrelation zu unterbrechen. Der

große Vorteil des Interceptors ist seine vollkommene Transparenz sowohl für den Programmierer als auch für den berechtigten Benutzer. Der Programmierer muss keinen zusätzlichen Code einfügen, um den Interceptor anzusprechen.
Allerdings sprechen zwei gravierende Nachteile gegen die Verwendung eines Interceptors als Mittel der Zugriffskontrolle auf einem Agentensystem.

Problem: hoher Aufwand der Implementierung sowie bei der Ausführung

1. Für den Interceptor auf Ebene der Aufrufrelation ist eine Unterstützung durch das Laufzeitsystem, auf dem das Agentensystem aufsetzt (z.B. die Java Virtual Machine), erforderlich, um auch jeden Aufruf durch den Interceptor unterbrechen zu können.
2. Weil der Interceptor auch für den Programmierer transparent sein soll, müssen alle Aufrufe abgefangen werden. Auch die Aufrufe der Entität auf eigene Methoden würden vom Interceptor unterbrochen.

Der Interceptor ist ein mächtiges Konzept für verteilte Anwendungen, aber nicht unbedingt geeignet, um lokal auf einem Agentensystem eine Zugriffskontrolle zu implementieren.

Aus diesen Gründen wird ein direkter Aufruf des Permission Managers mittels Check–Methode bevorzugt.

Rollenzuweisung und Rollenaktivierung

Jede Entität des Managementsystems kann aktiv im System handeln, d.h. als Subjekt auf Objekte zugreifen. Durch das RBAC Modell wird zwischen Subjekt und Objekt die Rolle eingefügt, an die die eigentlichen Rechte geknüpft sind (vgl. Abschnitt 4.4.1). Damit eine Entität aktiv im System handeln kann, muss ihr mindestens eine Rolle zugewiesen werden. Diese Rollenzuweisung wird durch die *sa*–Abbildung (Subject Assignment) festgelegt.

Rollenzuweisung durch Erteilung eines Rollenzertifikates

Technisch gesehen wird eine Entität durch ein Rollenzertifikat Mitglied dieser Rolle. Da das Rollenmodell domänenspezifisch festgelegt wird, kann auch nur die entsprechende Domäne ein bestimmtes Rollenzertifikat erteilen. Als Beispiel soll ein Mess Agent des Providers dienen, der auf dem Kundensystem QoS–Messungen durchführt. Vom Sicherheitsadministrator des Kunden wird das entsprechende Rollenzertifikat (`QoSMeter`), an das die benötigten Rechte geknüpft sind, erstellt. Der Kunde erteilt dem Agenten des Providers damit ein Rollenzertifikat aus der eigenen Domäne.

Explizite Aktivierung der Rolle

Eine Entität kann im Besitz mehrerer Rollenzertifikate sein. Das aufgabenorientierte Zugriffskontrollmodell und das Prinzip der minimalen Rechte erfordert eine Aktivierung der Rolle durch die Entität. Nur die handelnde Entität „weiß“, welche Aufgabe zu erledigen ist und in welchen Rollen sie aktiv tätig werden möchte. Der Permission Manager bietet eine Schnittstelle `activate(rolename)`, mit der eine Entität Rollen aktivieren kann. Der Permission Manager überprüft, ob die Entität im Besitz eines entsprechenden Rollenzertifikates ist, das dann vom Authenticator verifiziert wird. Falls dies erfolgreich war, wird die Rolle vom Agentensystem für die Entität aktiviert. Analog dazu gibt es die Schnittstelle `deactivate(rolename)` zur Deaktivierung der Rolle `rolename`.

Ob eine Entität nur eine einzige Rolle oder eine Menge von Rollen zum gleichen Zeitpunkt aktivieren kann, hängt von der lokal gültigen Sicherheitspolicy ab. Im ersten Fall wird auf die `deactivate` Methode verzichtet und mit erneutem Aufruf von `activate` wird automatisch die vorher aktivierte Rolle deaktiviert.

Der Fall, dass eine Entität die `activate` Methode nicht nutzt oder versucht, eine Rolle zu aktivieren, die es im lokalen Rollenmodell nicht gibt, muss gesondert betrachtet werden. Der letzte Fall kann insbesondere bei domänenübergreifender Migration eines Mobilen Agenten auftreten.

Implizite Aktivierung der Rolle

Für diese Fälle wird in der Matrix eine Default–Rolle mit minimalen Rechten spezifiziert und jede Entität erhält bei der Initiierung als „Basis"–Rolle diese Default–Rolle. Sobald die Entität dann die `activate` Methode nutzt, wird — je nach Semantik — die Default–Rolle durch die neu aktivierte Rolle ersetzt bzw. das Rollen–Set um die neue Rolle erweitert.

5.4.8 Naming Manager

Der **Naming Manager** ist für die Bildung und Vergabe von Namen an Mobilen Agenten und für das Agentensystem selbst verantwortlich. Dazu implementiert er das in Abschnitt 4.2.2 beschriebene MASIF–Namensschema. Ein Name setzt sich dabei aus der Identity, der Authority und dem Agent System Type zusammen. Die Authority eines Mobilen Agenten bzw. eines Agentensystems ist die verantwortliche Entität. In beiden Fällen ist dies der Besitzer, d.h. derjenige, der das Agentensystem bzw. den Mobilen Agenten erstmals instantiiert und gestartet hat.

Im Folgenden wird die Bildung und Registrierung des Namens von Agentensystemen und Mobilen Agenten beschrieben.

Bildung eines Agentensystem–Namens

Beim Start eines Agentensystems muss der Benutzer im Besitz eines digitalen Zertifikates sein, das vom Authenticator verifiziert wird und den DN des Benutzers liefert. Nur ein authentisierter und autorisierter Benutzer ist in der Lage ein Agentensystem zu starten. Durch das erfolgreiche Starten des Agentensystems wird der Benutzer zum Besitzer des Agentensystems. Der Name des Endsystems, auf dem das Agentensystem abläuft, wird als Identity ebenfalls Teil des Namens.

Der Naming Manager bildet aus dem Namen des Endsystems, dem DN des Besitzers und dem AS–Type den Compound Name des Agentensystems. Die Bestandteile des Compound Name werden durch "!" getrennt. In Abbildung 5.18 ist ein Beispiel für einen Agentensystem–Namen angegeben.

$$[\underbrace{\text{pcheger0.nm.informatik.uni-muenchen.de}}_{\textbf{\textit{Identity}}}!\underbrace{\text{MASA Admin}}_{\textbf{\textit{Authority}}}!\underbrace{\text{4}}_{\textbf{\textit{AS-Type}}}]$$

Abbildung 5.18: Beispiel eines Agentensystem–Namens

Der Name wird in Verbindung mit einer Objektreferenz, über die das Agentensystem erreicht werden kann, beim Naming– und Location Service (vgl. Abschnitt 5.2.1) registriert.

Bildung des Namens einer MA–Instanz

Bei der Bildung eines Namens für einen Mobilen Agenten ist zu unterscheiden, ob der initiale Name beim erstmaligen Start des Mobilen Agenten oder ein neuer Instanz–Name nach einer Migration berechnet wird.

Beim erstmaligen Start (vgl. Ablauf in Abbildung 5.8) wird die Identity des Mobilen Agent aus dem Package Identificator der Agenten(haupt)klasse gebildet. Dieser wird aus der MA–Gattung ermittelt. Der DN des Besitzers ist zu diesem Zeitpunkt bereits bekannt, da der Besitzer vom Authenticator bei der Authentisierung ermittelt wurde. Aber nicht nur der Besitzer ist Authority für den Mobilen Agenten, sondern auch das Agentensystem als Executor des Agenten. Die Authority als Teil des Namens des Mobilen Agenten setzt sich deshalb aus dem Namen des ausführenden Agentensystems und dem Namen des Besitzers zusammen. Der Agent System Type ist ein Attribut, das ebenfalls aus der Gattung ermittelt werden kann.

Wenn dieses Namensschema in der angegebenen Form verwendet würde, bestünde das Problem, dass auf einem Agentensystem genau eine MA–Instanz pro Gattung ablaufen könnte, da die Instanznamen verschiedener Instanzen der gleichen Gattung nicht mehr unterscheidbar wären. Der Naming Manager implementiert deshalb einen globalen Zähler, der die bisher benannten Agenten durchzählt. Bei der Namensbildung einer Instanz wird die Identity um diesen Zähler erweitert. Damit wird der Name jeder Agenteninstanz global eindeutig.

In Abbildung 5.19 ist ein Beispiel für einen Agenten–Namen angegeben, der auf dem Agentensystem aus dem vorherigen Beispiel ausgeführt wird.

Auch hier wird der Name und die Objektreferenz vor dem Start des Mobilen Agenten im Naming– und Location Service registriert.

Nach einer Migration kann der neue Name der MA–Instanz sehr einfach aus seinem letzten Namen ermittelt werden. In der Authority des alten Namens wird der Name des Quell–Agentensystems durch den Namen des Ziel–Agentensystems ersetzt.

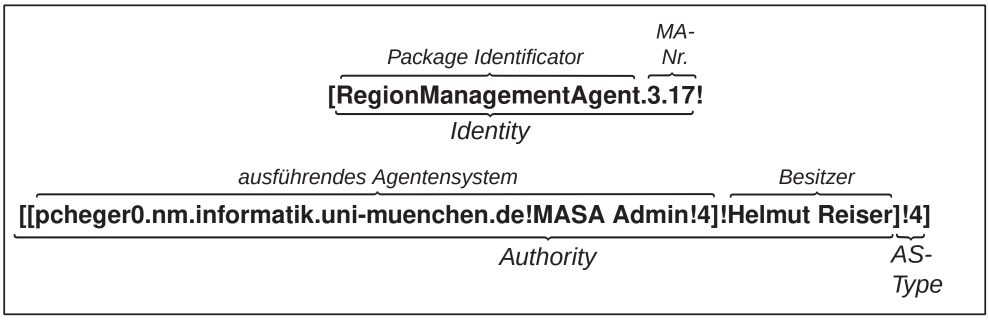


Abbildung 5.19: Beispiel eines Agenten–Namen

5.4.9 Auditing und Logging

Event–basiertes Logging

Zur Überwachung der Entitäten und zur Protokollierung der Aktivitäten innerhalb eines Agentensystems ist eine Auditing– und Logging–Komponente erforderlich. Durch das Logging, ggf. mit zusätzlicher digitalen Signatur, lässt sich Verbindlichkeit, auch der Aufrufrelation, realisieren. Der **Logger** ist die Komponente, die das Auditing implementiert. Er arbeitet Event–basiert. Sobald eine Aktion ausgeführt wird, die als zu protokollierend definiert wurde, wird ein entsprechender Event erzeugt und die Aktion protokolliert. Dabei wird der Verursacher der Aktion, der Zeitpunkt, das Objekt, auf dem die Aktion ausgeführt wurde, und ggf. weitere Informationen in eine Log–Datei geschrieben.

flexibel konfigurierbar

Der Logger muss in hohem Maße konfigurierbar sein. Um beispielsweise die Menge der protokollierten Daten und die Ausführlichkeit der Information einstellen zu können, werden verschiedene Log–Level unterstützt. Diese sind hierarchisch aufgebaut, d.h. ein Log–Level einer niedrigeren Stufe umfasst immer alle Log–Level höherer Stufen. Standardmäßig ist die Abstufung `FATAL`, `ERROR`, `WARNING` und `INFO`. Im Log–Level `FATAL` werden nur sehr schwerwiegende Fehler oder äußerst sicherheitskritische Aktionen protokolliert. Im Gegensatz dazu wird im Log–Level `INFO` sehr viel und sehr ausführliche Information auch über das „normale“ Systemverhalten in die Log–Datei geschrieben. Wird der Log–Level `INFO` für ein Objekt aktiviert, werden automatisch alle Events der anderen (höherstehenden) Log–Level mitprotokolliert, ohne dass dies gesondert konfiguriert werden müsste.

Log kann umgeleitet werden

Auch die Objekte, auf denen die Aktionen ausgeführt werden, die den Event auslösen, sind frei konfigurierbar. Es ist jederzeit möglich, dem Logger neue Objekte bekanntzumachen und für diese Logging–Events zu definieren. Die Log–Level können für jedes Objekt einzeln eingestellt werden. Damit ist es möglich, zu bestimmten Objekten ausführlichere Informationen zu speichern als für andere Objekte. Auch das Ausgabeziel des Loggers kann konfiguriert werden. Der Logger kann in eine lokale Datei und/oder auf einen oder mehrere Logging Server protokollieren. Durch die Umlenkung auf einen anderen Rechner werden die Auditing Daten, auch bei einer Kompromittierung der lokalen Maschine, vor nachträglicher Veränderung geschützt.

5.4.10 Zusammenfassung

Zusammenfassung und Gegenüberstellung: Systemmodell, Sicherheitsmechanismen, Komponenten der Architektur und Angriffe

In Kapitel 2 wurden die Sicherheitsanforderungen von Systemen mobiler Agenten anhand des dort vorgestellten abstrakten Systemmodells (vgl. auch Abschnitt 2.4) untersucht. Kapitel 4 hat sich mit den Mechanismen beschäftigt, welche die Sicherheitsanforderungen durchsetzen können. Abbildung 5.20 bildet den zusammenfassenden Überblick dieser beiden Aspekte, indem die verschiedenen Teile des Systemmodells den dafür erforderlichen Sicherheitsmechanismen gegenüber gestellt werden.

Modell		erforderliche Sicherheitsmechanismen
Entitätenmodell		Identifikation Authentisierung Autorisierung Anonymität Reisehistorie
Relationenmodell		Zugriffskontrolle Verbindlichkeit
	Ausführungs–relation	Vertraulichkeit Integrität Sandboxing
	Kommuni–kations–relation	Vertraulichkeit Integrität Reisehistorie
	Aufruf–relation	Sandboxing

Abbildung 5.20: Entitäten– und Relationenmodell; erforderliche Sicherheitsmechanismen

Die Sicherheitsmechanismen selbst werden, wie in diesem Kapitel beschrieben, durch Komponenten der Sicherheitsarchitektur implementiert bzw. realisiert. Dieser Zusammenhang wird in Abbildung 5.21dargestellt.

Das eigentliche Ziel der Sicherheitsmechanismen und der Komponenten, die diese Mechanismen implementieren, ist es, Angriffe zu verhindern. Wie aber bereits in Abschnitt 2 dargestellt, ist eine strenge Eins zu Eins Abbildung zwischen Angriff und verhinderndem Mechanismus weder wünschenswert noch besonders sinnvoll. Dieses Faktum zeigt sich auch in Abbildung 5.22. Hier werden die Angriffe auf Entitäten– und Relationenmodell den Sicherheitsmechanismen gegenübergestellt, die in der Lage sind, einen Beitrag zur Verhinderung des speziellen Angriffs zu leisten.

In der bisherigen Arbeit wurden alle Überlegungen auf ein abstrakteres Modell von Systemen Mobiler Agenten bezogen. Damit wird erreicht, dass ei-

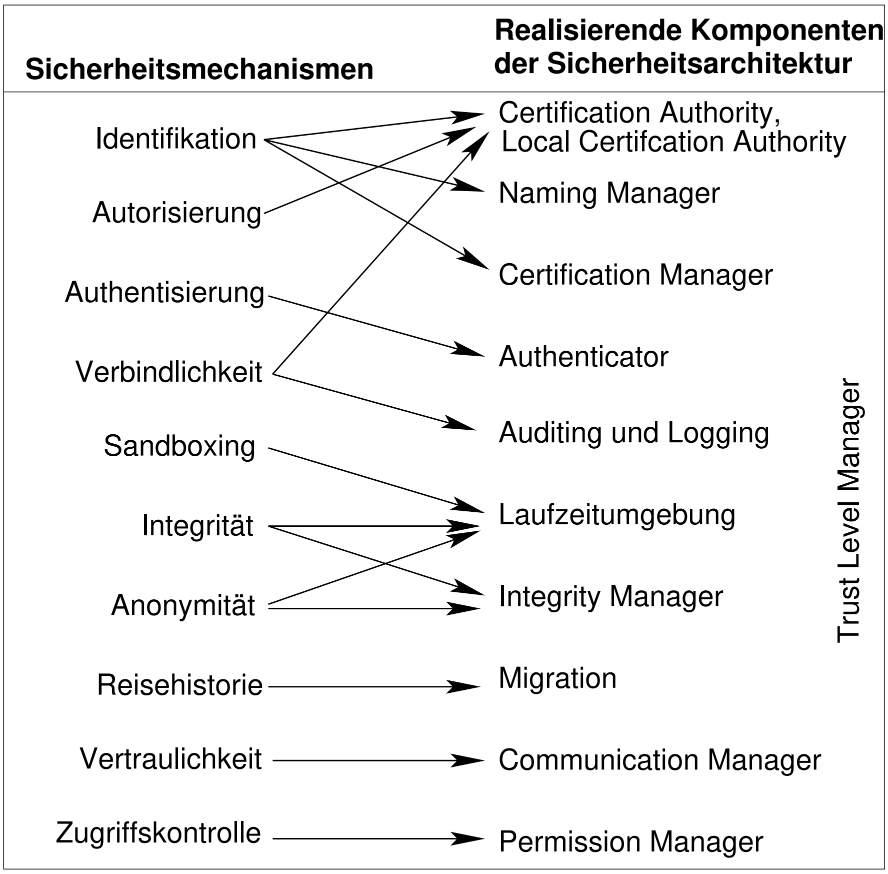


Abbildung 5.21: Sicherheitsmechanismen und realisierende Komponente

ne prospektive und allgemeingültigere Architektur entsteht und die Konzepte nicht nur auf eine Implementierungsklasse anwendbar sind, sondern allgemein anwendbar bleiben. Trotzdem muss, um die Tragfähigkeit der Architektur zu zeigen, eine Implementierung innerhalb eines konkreten Systems Mobiler Agenten erfolgen. Mit dieser Fragestellung wird sich das folgende Kapitel befassen.

Sicherheits-mechanismus	**bietet Schutz vor Angriffen**		
	Entitätenmodell		**Relationenmodell**
Identifikation und Authentisierung	Maskerade	Trust Level Management	Rechtemissbrauch Rechtediebstahl Leugnung Vervielfältigung Widereinspielung Umleitung
Autorisierung und Zugriffskontrolle			Rechtemissbrauch Rechtediebstahl Umgehung der Schnittstellen Denial of Service Denial of Execution
Vertraulichkeit	Maskerade		Abhören Rechtediebstahl Vervielfältigung Man–in–the–Middle Widereinspielung Umleitung
Verbindlichkeit			Leugnung
Integrität			Veränderung Vervielfältigung Wiedereinspielung Umleitung Manipulation des Ausführungspfades
Sandboxing			Umgehung der Schnittstellen
Ressourcen–Beschränkung			Denial of Service Ressourcenmissbrauch
Auditing und Logging			Leugnung

Abbildung 5.22: Zusammenfassung von Sicherheitsmechanismen und Angriffen

Kapitel 6

Prototypische Implementierung

Inhaltsverzeichnis

Für die prototypische Implementierung der in den letzten beiden Kapiteln vorgestellten Konzepte wurde die Mobile Agent System Architecture (MASA) verwendet, die im nächsten Abschnitt kurz vorgestellt wird. Abschnitt 6.2 beschreibt dann die prototypische Realisierung der Sicherheitsarchitektur in MASA.

6.1 Mobile Agent System Architecture

Die Mobile Agent System Architecture (MASA) [KRRV01, GHR 99, Roel 99a, Kemp 98] wurde an der Lehr– und Forschungseinheit „Kommunikationssysteme und Systemprogrammierung" entwickelt. Grundlegende Prämisse bei Design und Implementierung von MASA war es, ein hinsichtlich des Agentensystems völlig plattform– und betriebssystemunabhängiges System zu schaffen. Das Agentensystem muss alle systemspezifischen Dinge für den Agenten transparent machen. Es bildet eine für die Agenten homogene Ausführungsumgebung. Java bietet die Möglichkeit, viele Plattformen ohne spezifische Anpassungen zu erreichen und wurde vor allem deshalb als Implementierungssprache gewählt. Das Einsatzgebiet IT-Management bedingt aber auch, dass der Mobile Agent bei Bedarf auch bis zu den Endsystem–Ressourcen vordringen kann.

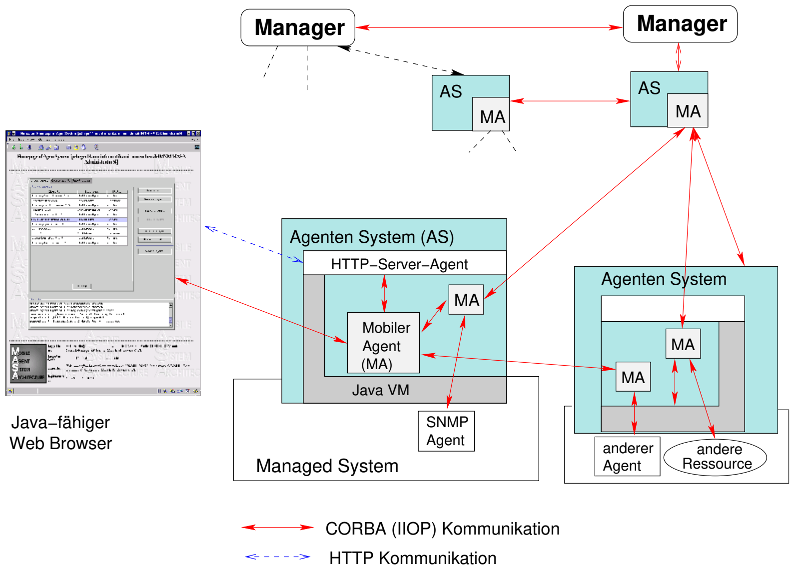


Abbildung 6.1: Architektur und Kommunikationsrelationen in MASA

Java und CORBA als Basis

Abbildung 6.1 zeigt exemplarisch eine MASA–Region, bestehend aus mehreren Agentensystemen. Ein Agentensystem ist als Java Anwendung realisiert, die innerhalb einer Java Virtual Machine abläuft. Die Mobilen Agenten sind ebenfalls in Java implementiert und laufen im Kontext und unter der Kontrolle des Agentensystems ab. In Abbildung 6.1 sind auch die Kommunikationsrelationen zwischen den einzelnen MASA–Entitäten dargestellt, die alle über CORBA [OMG 01-02-33] (konkret über das Internet Inter ORB Protocol (IIOP)) kommunizieren.

Applets als GUI für AS und MA

In MASA besitzt jeder Agent und auch das Agentensystem selbst ein Applet, das die Benutzerschnittstelle realisiert. Ein Benutzer, der ein Applet laden möchte, baut eine HTTP–Verbindung zum HTTP–Server–Agenten des Agentensystems auf, der das entsprechende Applet zur Verfügung stellt. Zu diesem Zweck ist jedem Agentensystem ein (stationärer) HTTP–Server–Agent zugeordnet. Sobald das Applet im Browser gestartet wurde, kommuniziert dieses wieder über IIOP mit „seinem“ Agenten. Die Schnittstellen aller Agenten und auch des Agentensystems werden über CORBA IDL spezifiziert.

Schnittstellen spezifiziert durch IDL

Die Implementierungsarchitektur von MASA, dargestellt in Abbildung 6.2, gliedert sich in vier Hauptschichten. Die unterste Schicht ist das Betriebs– und Transportsystem. Diese werden durch das jeweilig zu administrierende System determiniert und deshalb nicht weiter betrachtet.

MASA Schichtenarchitektur

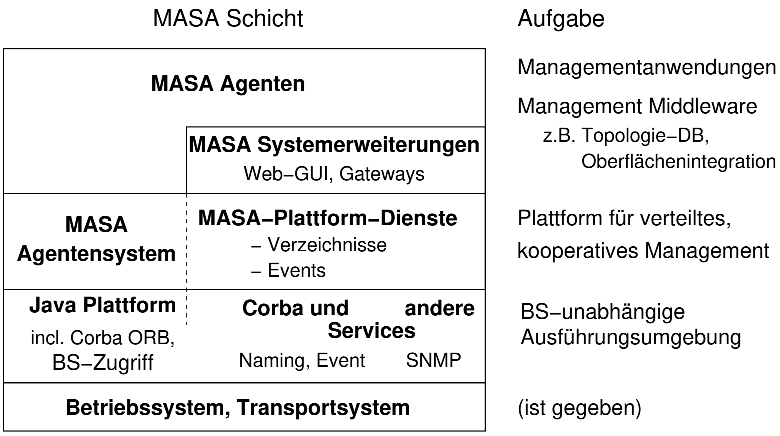


Abbildung 6.2: Schichtenarchitektur von MASA

Darüber wird eine einheitliche Umgebung für die verteilte Anwendung MASA geschaffen. Java wird als Ausführungs– und Laufzeitsystem verwendet, CORBA als Transportsystem und als Lieferant essentieller Basisdienste wie z.B. dem Naming– oder Event–Service. Darauf aufbauend liegt das MASA–Agentensystem, welches die eigentliche Plattform für das kooperative und verteilte Management darstellt, und die MASA Plattform–Dienste wie z.B. Verzeichnisdienste und ein angepasster Event–Service. In der obersten Schicht befinden sich die MASA–Agenten sowie MASA–Systemerweiterungen, die schließlich die konkreten Managementanwendungen bzw. Management–spezifische Middleware–Komponenten realisieren.

Das Agentensystem selbst, dessen Hauptaufgabe die Verwaltung Mobiler Agenten darstellt, ist modular aufgebaut. Im Folgenden werden einige der zentralen Klassen des Agentensystems kurz vorgestellt:

zentrale Bestandteile des Agentensystems

- Klasse `AgentSystem` `AgentSystem` ist der zentrale Teil des Agentensystems und implementiert alle in MASIF definierten Funktionen der Interfaces `MAFAgentSystem` und stellt entsprechende CORBA Schnittstellen bereit. Die darin enthaltene Funktionalität umfasst:
  - Erzeugen, Migrieren und Terminieren von Agenten
  - Anhalten und Fortsetzen der Ausführung von Agenten
  - Registrieren und Entfernen von Agenten und Agentensystemen aus dem MASIF–Namensverzeichnis
  - Abfragen und Durchsuchen des MASIF–Namensverzeichnisses

`AgentSystem` übernimmt somit hauptsächlich die Aufgabe eines Dispatchers. Die eigentliche Funktionalität wird in den Agent Managern erbracht.

- Klasse `AgentManager` `AgentManager` realisiert den Manager für MASA–Agenten und implementiert die von `AgentSystem` angebotenen MASIF–Funktionen für Agenten des Typs MASA. Neben den MASIF–Funktionen werden zusätzliche, MASA–spezifische Funktionen implementiert. Diese dienen zur Vereinfachung der Kommunikation zwischen Agenten, der Speicherung von serialisierten Agenten in Dateien sowie der Lieferung umfangreicher Statusinformationen.
  Neben den Agenten vom Typ MASA sollen auch andere Agententypen (z.B. Voyager– [Voyager] oder Aglets–Agenten [Aglets]) unterstützt werden können. Die benötigte Gateway–Funktionalität wird in spezifischen Manager–Klassen (z.B. `VoyagerManager` [Bran 99]) zusammengefasst, die den `AgentManager` um die spezifischen Funktionen erweitern.

Alle Agenten werden von der Klasse `.masa.Agent` abgeleitet, die in die Klassen `MobileAgent` und `StationaryAgent` verfeinert wird. Nur die Klasse `MobileAgent` implementiert Funktionen zur Migration, da stationäre Agenten das Agentensystem nicht verlassen können.

Auf dieser Infrastruktur wurden in Forschungsprojekten, Diplomarbeiten und praktischen Studienarbeiten viele Management–Agenten und –Anwendungen entwickelt [Ense 01a, Ense 01b, Heil 00, Grus 99, Saca 01, Lore 01, Gigl 00, Thae 00, Maul 99, Demm 99, Bran 99, Gerb 99, Goli 99, Maul 99, Roel 99b, Demm 98, Allg 98, Coeh 98, Radi 98, Kemp 98].

6.2 Implementierung der Sicherheitsarchitektur

Die MASA Plattform wurde verwendet, um die in den beiden vorhergehenden Kapiteln beschriebenen Sicherheitskonzepte zu implementieren. Im folgenden Teilabschnitt werden Naming Service und CA als Komponenten der Domäne bzw. einer Region vorgestellt. Abschnitt 6.2.2 beschreibt dann die innerhalb eines Agentensystems implementierten Bausteine.

6.2.1 Intra–Domänen–Komponenten

Naming Service

Der MASIF–Standard definiert einen eigenen Verzeichnisdienst für Mobile Agenten und Agentensysteme, den MAFFinder. In MASA wird als Namens–

und Lokalisierungsdienst jedoch der Standard CORBA Naming Service verwendet, der von fast allen Produkten, die CORBA implementieren, mitgeliefert wird. In diesem Verzeichnisdienst werden Namen auf Objektreferenzen — genauer auf CORBA IOR's — abgebildet. Über eine IOR (Interoperable Object Reference) kann ein CORBA Objekt ortstransparent über den ORB adressiert und angesprochen werden.

Allerdings wird der Naming Service durch die im Rahmen von [Gigl 00] entwickelte Klasse `MASAFinder` gekapselt. Ein direkter Zugriff auf den Naming Service ist nicht mehr möglich. Außerdem kann jedes Agentensystem nur die Namen der eigenen Agenten registrieren, d.h. derjenigen, die auch auf dem entsprechenden Agentensystem ausgeführt werden. `MASAFinder` implementiert die Schnittstellen des MAFFinder. Damit sind die Vorteile des Naming–Service als einfacher und doch effizienter Verzeichnisdienst mit der mächtigeren Schnittstelle des MAFFinder vereint. Dadurch, dass ein direkter Zugriff auf den Naming Service nicht mehr möglich ist und alle Agentensysteme beim Zugriff authentisiert werden, kann Name–Spoofing verhindert werden.

`MASAFinder` implementiert MAFFinder und kapselt Naming Service

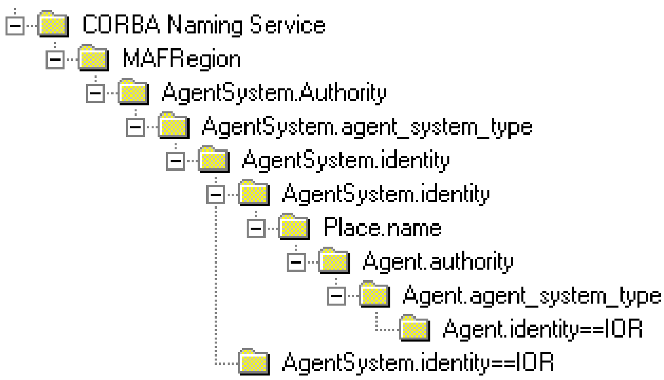


Abbildung 6.3: Struktur innerhalb des Naming Service

In Abbildung 6.3 ist die allgemeine Verzeichnisstruktur, die vom MASAFinder aufgebaut wird, dargestellt. Der Name einer Entität wird dabei durch die Teilnamen von der Wurzel bis zu einem Blatt gebildet. Die IOR's werden immer nur an Blätter geknüpft. Für die Vergabe von eineindeutigen Namen ist der in Abschnitt 6.2.2 beschriebene Naming Manager verantwortlich.

Certification Authority

Die in MASA implementierte CA ist zweigeteilt in die `MASA-CA` und die `JDK-developer-CA`. Die `MASA-CA` erzeugt klassische X.509v3 Zertifikate, um Benutzer, Agentensysteme und MA–Instanzen zu zertifizieren. Die CA–Implementierung stützt sich hinsichtlich der kryptographischen Algorithmen auf die Programmpakete IAIK–JCE 2.5 und OpenSSL [OpenSSL]. Diese werden im Abschnitt 6.2.2 auf den Seiten 182 und 184 noch näher vorgestellt.

X509v3 Zertifikate für alle Entitäten

Daneben gibt es eine eigene (Teil–) CA für Implementierer–Zertifikate, die `JDK-developer-CA`. Für die Signatur der Gattung ist es erforderlich, dass der Implementierer seinen Agenten–Code in einer so genannte Java Archive Datei (`jar`) zusammenfasst. Diese muss er dann digital signieren. Dazu verwendet er das Werkzeug `jarsigner`, das Teil der Java Entwicklungsumgebung ist und das Zertifikate in einem bestimmten Format erwartet. Der Entwickler muss einen `keystore` besitzen, der sein Zertifikat enthält. Die `JDK-developer-CA` erzeugt ein entsprechendes Implementierer–Zertifikat und integriert dieses in den `keystore` des Implementierers.

`jarsigner` Zertifikate für Implementierer

6.2.2 Komponenten des Agentensystems

Naming Manager

MASIF Namensschema

Der Naming Manager ist für die Vergabe und Registrierung der Namen Mobiler Agenten und des Agentensystems selbst verantwortlich. In MASA wurde das in Abschnitt 4.2.2 beschriebene MASIF Namensschema realisiert.

erweitert um globale Zähler

Um mehrere Instanzen derselben Agentengattung auf einem Agentensystem starten zu können, wurde das MASIF Namensschema, wie in Abschnitt 5.4.8 erläutert, um Region–globale Zähler erweitert. Jedes Agentensystem erhält nach dem Start eine innerhalb der Region eindeutige Nummer. Diese Nummer wird durch den Naming Service für die gesamte Region verwaltet und an das jeweilige Agentensystem vergeben. Jeder Agent erhält nach dem Start eine für das entsprechende Agentensystem lokal eindeutige Nummer. Der Instanzname des Mobilen Agenten wird um die Nummer des Agentensystems und seine eigene Agentennummer erweitert. Dabei bleibt die Nummer des Agentensystems, auf dem der Agent erstmalig gestartet wurde, über seine gesamte Lebenszeit Teil seines Namens. Die Agentennummer wird nach jeder Migration vom Naming Manager des entsprechenden Ziel–Agentensystems neu vergeben. Durch dieses Konzept wird jeder Agent innerhalb der Region und über seinen gesamten Lebenszyklus hinweg eineindeutig identifizierbar.

dadurch eineindeutige Identifikation aller Entitäten

Beispiel: Region Management Agent

In Abbildung 6.4 ist die Benutzeroberfläche des Region Management Agents abgebildet. Der Name des Agenten `[RegionManagement-Agent.1.3![[pcheger14.in.nm.informatik.uni-muen-chen.de!MASA Admin!4]!Helmut Reiser!4]` (vgl. auch Abbildung 5.19) steht in Kurzform hinter „Homepage of Agent“. Im angegebenen Beispiel besteht der Name aus der Identity „RegionManagement-Agent.1.3“ (in Kurzform). Die Nummer des Agentensystems (im Bsp. `pcheger14.in.nm...`), auf dem der Agent gestartet wurde, ist „1“; die Nummer des Agenten „3“. Die verantwortliche Authority ist „Helmut Reiser“ und der Agent ist vom Typ „4“, d.h. ein MASA–Agent. Der Region Management Agent dient zur Verwaltung und zur Visualisierung aller Agentensysteme mit ihren Agenten innerhalb einer Region. In der linken Spalte sind die Agentensysteme angegeben. Sobald man ein Agentensys-

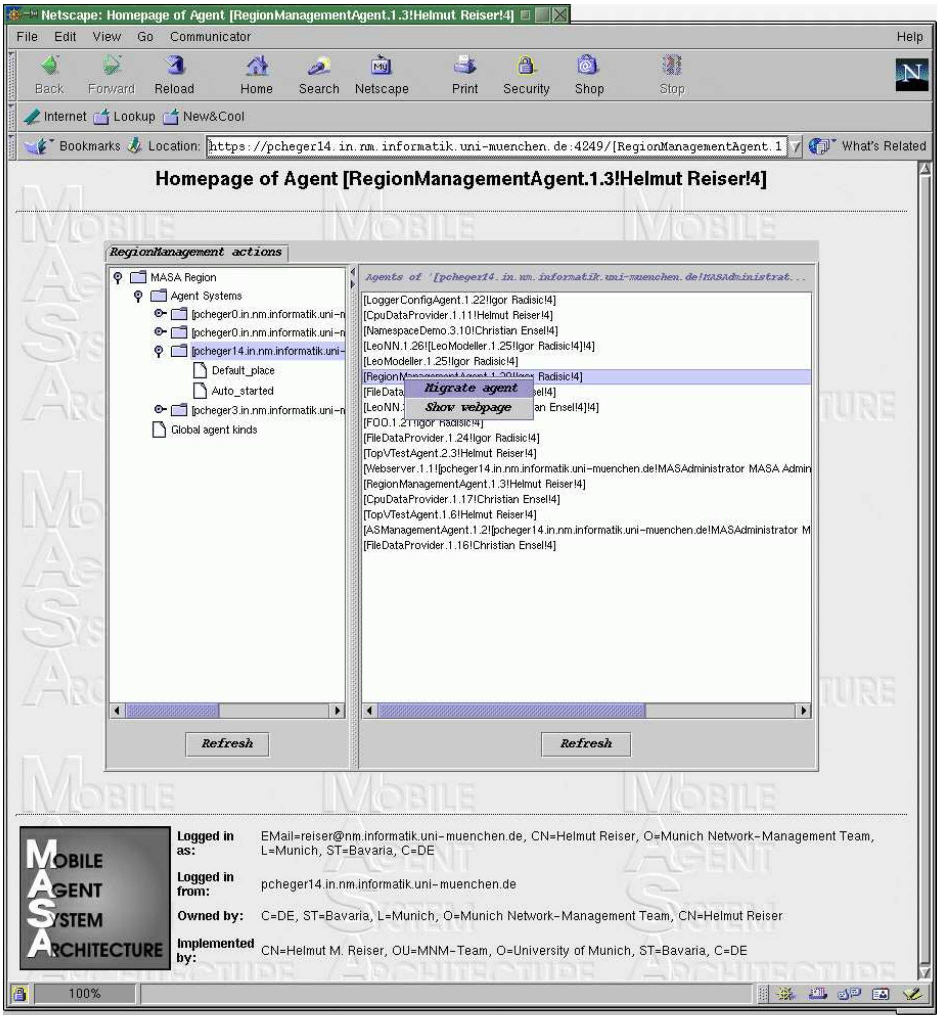


Abbildung 6.4: GUI des Region Management Agent

tem auswählt, werden in der rechten Spalte alle Agenten, die auf diesem Agentensystem ausgeführt werden, angezeigt. Im vorgestellten Fall laufen auch mehrere Agenteninstanzen derselben Gattung auf dem Agentensystem `pcheger14.in.nm....`. Von `CPUDataProvider` werden Nummer 1.11 und 1.17 ausgeführt und auch von den Agenten der Gattung `LeoNN` und `TopVTestAgent` werden jeweils zwei Instanzen auf demselben Agentensystem ausgeführt. Aus der Abbildung geht auch hervor, dass die Nummer des initialen Agentensystems über die Lebenszeit des Agenten erhalten bleibt. Das Agentensystem `pcheger14.in.nm....` hat die Nummer 1, `pcheger3` die Nummer 3. Der Agent `NamespaceDemo3.10` wurde auf `pcheger3` erstmalig gestartet und erst später auf das Agentensystem `pcheger14` migriert.

Certification Manager

Der Certification Manager des Agentensystems ist die Local Certification Authority für Mobile Agenten. Er wird implementiert von der Klasse `Agent-`

Certification Manager als LCA

`SystemCertManager`. Der Certification Manager muss für Agenteninstanzen, die auf dem Agentensystem ablaufen, Schlüssel erzeugen, den Instanzen Zertifikate ausstellen, diese lokal speichern und an die globale CA zur langfristigen Speicherung übergeben (vgl. auch Abschnitt 5.4.1).

Caching von Schlüsseln zur Performance Optimierung

Der Prozess der Schlüsselerzeugung ist relativ rechenaufwendig und kann daher zu Performance–Einbußen führen, falls die Schlüssel immer „on Demand“ erzeugt werden (vgl. auch Abschnitt 6.3). Um die Performance und den in Abbildung 5.6 dargestellten Ablauf zu optimieren, wird vom Certification Manager ein `KeyCache` implementiert. Die Größe dieses Caches kann vom Agentensystem–Betreiber frei gewählt werden. Der Certification Manager startet bei seiner Initialisierung einen Thread mit sehr niedrigerer Priorität, der den Cache sukzessive mit Schlüsselpaaren füllt. Durch die niedrige Priorität des Threads erfolgt eine Schlüsselberechnung „auf Vorrat“ nur dann, wenn die Last auf dem Agentensystem nicht zu groß ist. Wenn für einen Agent ein Zertifikat angefordert wird, können die Schlüsselpaare aus dem Cache entnommen werden. Es muss nur noch die Datenstruktur für das Zertifikat erzeugt und dann digital signiert werden.

technische Umsetzung über Cryptography Provider

Für die technische Umsetzung des Certification Managers sind Algorithmen zur Schlüsselgenerierung, zur Berechnung von MACs, Verschlüsselungsverfahren sowie Algorithmen zur digitalen Signatur erforderlich. Diese Verfahren werden innerhalb der Java Cryptography Architecture [JCA] zusammengefasst. Da in den Vereinigten Staaten Exportrestriktionen für Verschlüsselungsalgorithmen gelten, darf nach Europa nur eine sehr eingeschränkte Anzahl von Implementierungen exportiert werden. Um eigene Algorithmen in die JCA integrieren zu können, wurde die Java Cryptography Extension [JCE, JCE-imp] spezifiziert. Hier wird eine so genannte Provider Infrastruktur definiert. Die Cryptography Provider definieren einheitliche Schnittstellen, die von allen Programmen genutzt werden können. Die Klassen, welche die Funktionalität dann tatsächlich implementieren, können zur Laufzeit ermittelt und dynamisch gebunden werden. Damit besteht die Möglichkeit, Implementierungen von Drittanbietern sehr einfach in die JCA zu integrieren. Die Drittanbieter implementieren die durch die Cryptography Provider definierten Schnittstellen.

In MASA wurde als JCE Provider das von der Technischen Universität Graz entwickelte IAIK–JCE 2.5 [IAIK-JCE] verwendet. Damit lassen sich Message Authentication Codes, RSA–Schlüssel sowie X.509v3 Zertifikate erzeugen. Alle dafür benötigten Verschlüsselungsverfahren werden von diesem Provider implementiert.

Authenticator

Authenticator für lokale Authentisierung

Der Authenticator führt nur lokale Authentisierung durch. Die entfernte Authentisierung wird vom Communication Manager (vgl. Seite 184) übernommen. Der Authenticator prüft die digitalen Signaturen von MA–Instanzen und Gattungen (vgl. Abschnitt 5.4.2). Er ermittelt also die Autho-

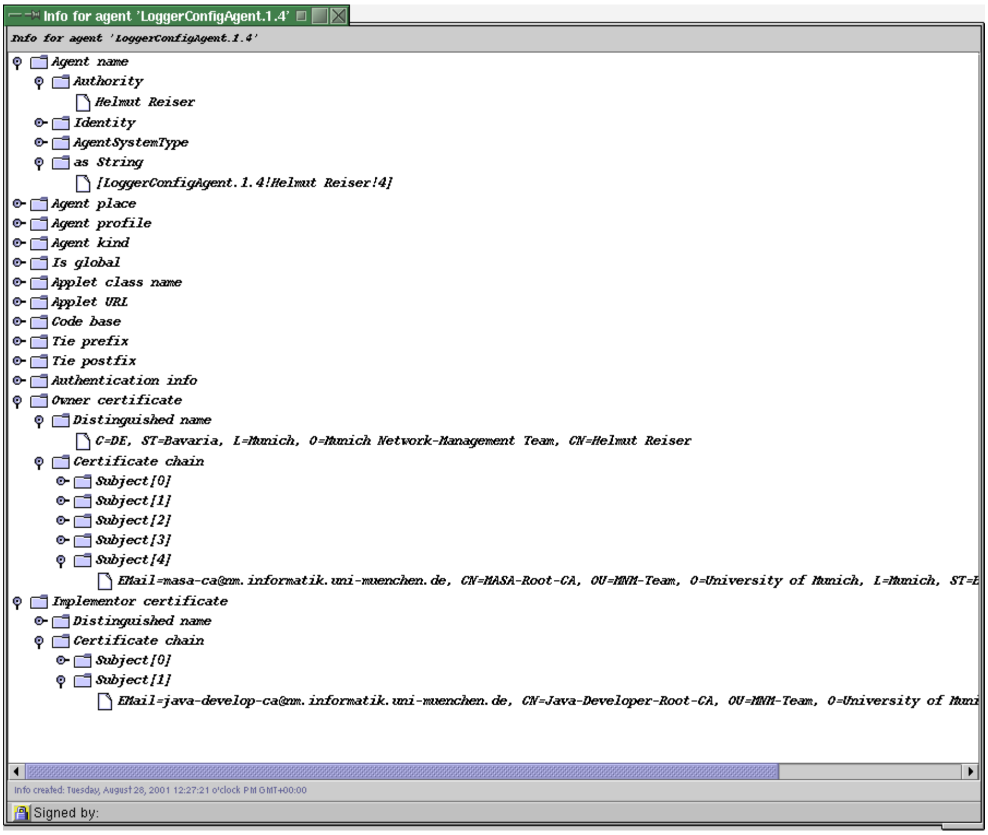


Abbildung 6.5: Informationen über Mobilen Agenten

rity eines Mobilen Agenten und dessen Implementierer. Abbildung 6.5 zeigt das Informationsfenster eines Agenten, im Beispiel das des `LoggerConfigAgent`. Der Besitzer (`Authority`) des Agenten wird vom Authenticator ermittelt. Das entsprechende Zertifikat, bzw. der Distinguished Name wird ebenfalls im Info Feld `Owner Certificate` angegeben. Der Zertifizierungspfad, den der Authenticator berechnet hat, kann unter `Certification chain` eingesehen werden. Im Feld `Subject[0]` findet sich das aktuelle Zertifikat des Besitzers, in den anderen Feldern entsprechend die Zertifikate aus dem Zertifizierungspfad. Im angegebenen Beispiel endet der Pfad (`Subject[4]`) bei der MASA–Root–CA. Analog kann der Implementierer und dessen Zertifizierungspfad aus dem Feld `Implementor certificate` gelesen werden. Im angegeben Beispiel endet dieser Pfad bei der Java–Developer–Root–CA.

Der verantwortliche Besitzer eines Mobilen Agenten geht als Authority Attribut mit in dessen Namen ein und wird auf der Homepage des entsprechenden Agenten oder des Agentensystems angezeigt. Daneben wird der Besitzer auch im Feld „Owned by“, der Implementierer unter „Implemented by“ auf der GUI angezeigt.

In der Abbildung des Region Management Agents (vgl. Abb. 6.4) werden auf der rechten Seite alle Agenten angezeigt, die sich auf dem Agentensystem `pcheger14....` befinden. Die Authority steht hinter dem ersten Ausrufezeichen. In der Abbildung ist auch erkennbar, dass der Agent `LeoNN.1.26` nicht von einem Benutzer direkt, sondern von einem anderen Agenten `Leo–Modeller.1.25` gestartet wurde, für den wiederum Igor Radisc verantwortlich ist.

Communication Manager

Der Communication Manager soll, wie in Abschnitt 5.4.5 beschrieben, ein verbindungsorientiertes verschlüsselndes Protokoll zwischen beliebigen Entitäten realisieren, d.h. mit Hilfe des Communication Manager können vertrauliche Kommunikationsrelationen etabliert werden.

konsequente Integration von SSL/TLS

In der Sicherheitsarchitektur von MASA wurde der Communication Manager durch die konsequente Integration von **SSL** (Secure Socket Layer) bzw. **TLS** (Transport Layer Security) realisiert. SSL wurde von der Firma Netscape entwickelt und 1996 als Internet Draft bei der IETF (Internet Engineering Task Force) eingereicht [SSL 3.0], aber nicht weiter verfolgt. Im Standardisierungsprozess wurde das Protokoll dann in TLS umbenannt und schließlich als RFC standardisiert [DiAl 99]. Im Allgemeinen Sprachgebrauch wird jedoch meist weiterhin die Abkürzung SSL verwendet.

Handshake Layer zur Authentisierung

Record Layer zur Verschlüsselung

SSL/TLS gliedert sich in zwei Protokoll–Teile, den Record Layer und den Handshake Layer. Der Handshake Layer realisiert eine ein– oder zweiseitige Authentisierung auf Basis von X.509 Zertifikaten und einem Challange Response Verfahren sowie die Aushandlung von Verschlüsselungs– und MAC–Algorithmen und den Austausch von Schlüsselmaterial. Mit Hilfe von SSL/TLS ist es also möglich, dass sich zwei Kommunikationspartner gegenseitig authentisieren. Am Ende des Handshake Layer haben sich die Partner auf Algorithmen geeinigt und der Record Layer kann mit Hilfe des ausgehandelten Schlüsselmaterials Sitzungsschlüssel berechnen, um die gesamte folgende Kommunikation zu verschlüsseln.

Die Kommunikation innerhalb von MASA wird grundsätzlich über IIOP abgewickelt. Lediglich für das Laden eines Applets (GUI für Agent oder Agentensystem) wird eine HTTP–Verbindung zum HTTP–Server–Agent des entsprechenden Agentensystems aufgebaut. In MASA wurden alle Kommunikationsrelationen (vgl. Abbildung 6.1) durch SSL/TLS gesichert. Das bedeutet, dass nur noch IIOP/SSL–Verkehr zwischen Entitäten möglich ist. Der HTTP–Server–Agent akzeptiert nur HTTPS–Anfragen. Für die Realisierung von IIOP/SSL (IIOP over SSL) wurde der CORBA–ORB *Orbacus 3.1.2* der Firma Object Oriented Computing [OOC] um SSL/TLS erweitert. Hierzu wurden die SSL–Implementierungen OpenSSL sowie das von der Universität Graz entwickelte IAIK–iSaSiLk 2.5.1 [iSaSiLk] verwendet. Das letztgenannte Programmpaket wurde auch eingesetzt, um die HTTPS–Kommunikation (HTTP over SSL) zu realisieren.

Durch die Verwendung von SSL ist es möglich, alle Kommunikationskanäle transparent zu verschlüsseln. Zusätzlich dazu wird auch die lokale Kommunikation — mittels Aufrufrelation — zwischen Mobilen Agenten unterbunden. Auch lokal können Agenten nur über IIOP/SSL und nicht direkt über Java Aufrufe miteinander kommunizieren. Damit werden alle Methodenaufrufe zwischen Agenten durch das Agentensystem „vermittelt“. Damit kann das Agentensystem diese Aktionen auch mitprotokollieren, was bei direkten Java Aufrufen nicht so einfach zu realisieren wäre. Außerdem kann bei einem systematischen Einsatz von SSL der Handshake Layer zur beidseitigen Authentisierung von Entitäten verwendet werden. Wie in Abschnitt 6.2.2 beschrieben, authentisiert der Authenticator nur die Authority und den Implementierer eines Mobilen Agenten. Alle anderen Entitäten, d.h. insbesondere Benutzer und MA–Instanzen, werden in MASA über den Handshake Layer authentisiert.

lokale Kommunikation wird unterbunden

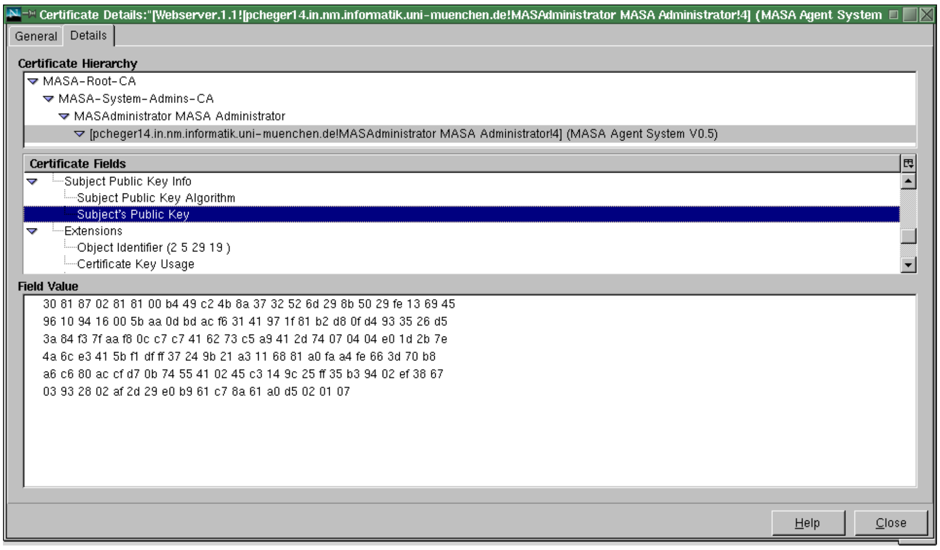


Abbildung 6.6: Authentisierung des Agentensystems über SSL

So besitzt beispielsweise jeder Benutzer ein SSL Zertifikat. Sobald er mit einem Browser beim HTTP–Server–Agent eines Agentensystems eine Benutzeroberfläche anfordert, beginnt der Handshake Layer. Zuerst authentisiert sich das Agentensystem beim Benutzer. Der Benutzer bekommt das in Ab-

Beipiel: Authentisierung eines Benutzers

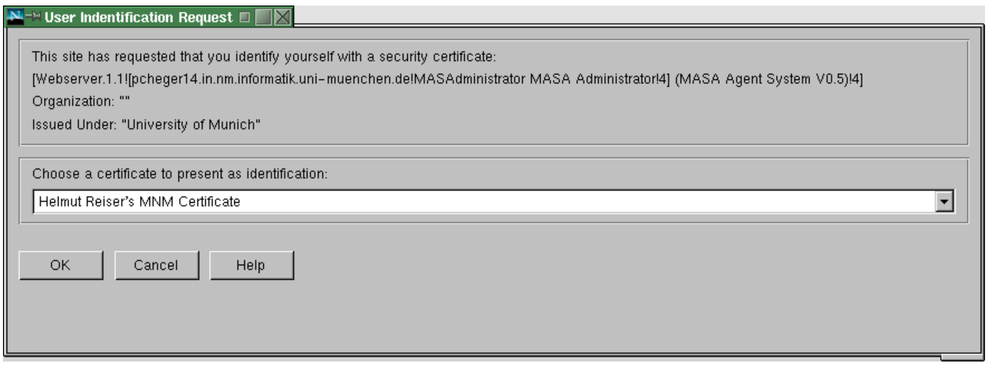


Abbildung 6.7: Authentisierung eines Benutzers über SSL

bildung 6.6 dargestellte Fenster und kann sich detaillierte Informationen zum präsentierten Zertifikat anzeigen lassen, um dann zu entscheiden, ob er das

Zertifikat akzeptiert. Falls er der signierenden CA vertraut und das Zertifikat akzeptiert, muss er sich gegenüber dem Agentensystem authentisieren. Dazu muss er eines seiner Zertifikate auswählen (vgl. Abbildung 6.7), dass dann im weiteren Verlauf des Handshake Layer zur Authentisierung verwendet wird.

Die Authentisierungsinformationen, die vom Authenticator und vom Communication Manager ermittelt wurden, werden in den Fußzeilen auf der Benutzeroberfläche angezeigt (vgl. Abbildung 6.8 und 6.4). Es wird angezeigt,

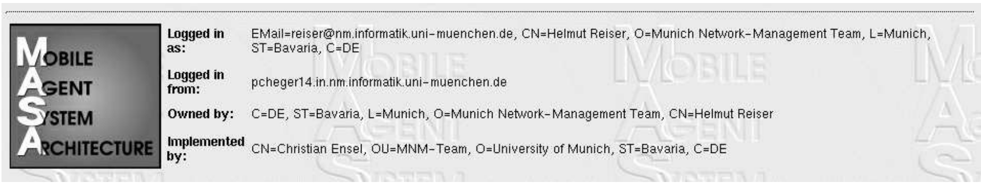


Abbildung 6.8: Authentisierungsinformationen dargestellt auf der Benutzeroberfläche

wer den Agenten oder das Agentensystem im Moment nutzt (Logged in as:), wer für die Entität verantwortlich ist (Owned by:) und wer sie implementiert hat.

Migration Manager

In MASA wird die Migration innerhalb des `AgentManager` implementiert (vgl. Abschnitt 6.1), der auch alle anderen Methoden zur Manipulation von Agenten implementiert. Der `AgentManager` ist damit die zentrale Komponente für Agent Management und Agent Transfer (vgl. Abschnitt 2.3.1). Aus diesen Gründen wurde der Migration Manager als Teil des `AgentManager` implementiert.

Migration Manager implementiert Trust Level Management

Der Migration Manager koordiniert die sicherheitsrelevanten Tests, die vor bzw. nach einer Migration durchzuführen sind. Er stellt im Grunde nur eine Art „Trader“ dar, der die zu erledigenden Aufgaben an die entsprechenden Komponenten der Sicherheitsarchitektur verteilt (vgl. Abschnitt 5.4.4). Die einzige Aufgabe, die er selbst erledigen muss, ist das Trust–Level–Management (vgl. Abschnitt 4.1.2).

Integrity Manager

Damit die Integrität und Vertraulichkeit zwischen Agenten auf einem Agentensystem sichergestellt werden kann, ist eine konfigurierbare Abschottung zwischen einzelnen Agenten erforderlich. Die hierzu erforderlichen Maßnahmen sind Speicherschutz, eigene Namensräume und Laufzeitschutz (vgl. Abschnitt 4.3.2).

Speicherschutz

Der Speicherschutz wird durch die Verwendung von Java und CORBA erreicht. Für die Agenten sind keine direkten Speicherzugriffe möglich. Aufrufe von Methoden anderer Agenten sind nur über IIOP/SSL zulässig und damit kontrollierbar.

Sichtbarkeit

Neben dem Speicherschutz ist auch die Sichtbarkeit und Manipulierbarkeit von Attributen zu betrachten. Grundsätzlich ist ein direkter Zugriff auf Attribute nicht möglich, sondern ein lesender Zugriff muss über `get`– ein schreibender über `set`–Methoden erfolgen. Der Visibilitätsmodifikator von Attributen sollte grundsätzlich auf `private` gesetzt werden. Die Zugriffsmethoden können entsprechend den Anforderungen auf `public`, `protected`, `package` oder auch auf `private` gesetzt werden.

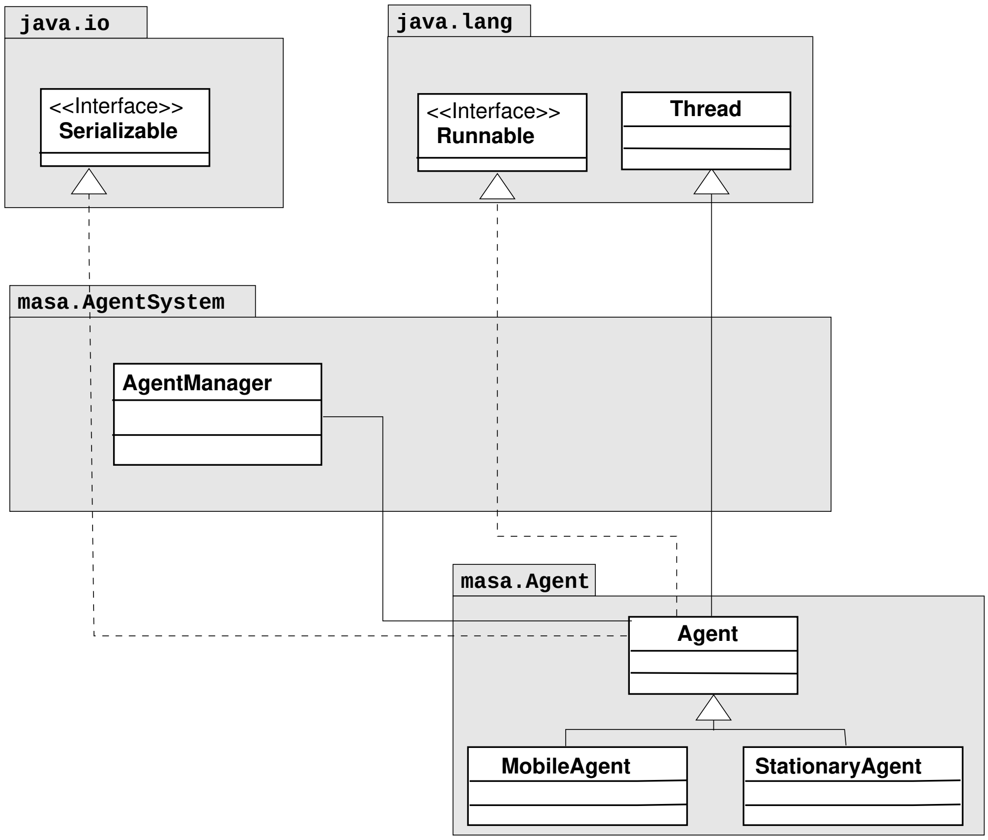


Abbildung 6.9: Ursprüngliche Vererbungs– und Package Struktur in MASA

Es gibt aber auch Attribute des Agenten, die von diesem nicht verändert werden dürfen. Als Beispiel sei hier nur der Name des Agenten angeführt. Dieser wird vom Naming Manager vergeben und durch das Agentensystem gesetzt. Wenn ein Agent sein Namens–Attribut ändern kann, so kann er seine Identität verändern und sich als anderer Agent ausgeben (Maskerade). Um Agentenattribute und Methoden, die vom Agenten selbst nicht genutzt bzw. verändert werden können, zu realisieren, musste die Klassenstruktur von MASA geändert werden. In Abbildung 6.9 ist die ursprüngliche Klassenstruktur vereinfacht dargestellt. Alle Attribute und Methoden, die in der Klasse Agent definiert werden, können von der Klasse Agent und ggf. auch von deren Subklassen verändert werden. In Tabelle 6.1 ist die Sichtbarkeit nochmals zusammengefasst (aus [Flan 97]).

Sichtbar für:	Visibilitätsmodifikator public	protected	package	private
Definierende Klasse	ja	ja	ja	ja
Klasse im selben Package	ja	ja	ja	nein
Subklasse aus anderem Package	ja	ja	nein	nein
andere Klasse aus anderem Package	ja	nein	nein	nein

Tabelle 6.1: Visibilitätsmodifikatoren und Sichtbarkeit in Java

Um das angesprochene Problem zu lösen, wurde eine neue Vaterklasse von `Agent` eingeführt, die aber zum Package von `AgentSystem` gehört. Die neue Klasse `AgentEnvironment` fasst alle Attribute und Methoden zusammen, die zwar zum Agenten gehören, von diesem aber nicht manipuliert werden dürfen. Werden Attribute weiterhin auf `private` und `set`–Methoden auf `package` gesetzt, kann nur das Agentensystem das Attribut setzen. Falls die `get`–Methode auf `protected` gesetzt wird, können neben dem Agentensystem nur die abgeleiteten Agentenklassen die Attribute auslesen. Abbildung 6.10 zeigt die neue Package– und Vererbungsstruktur.

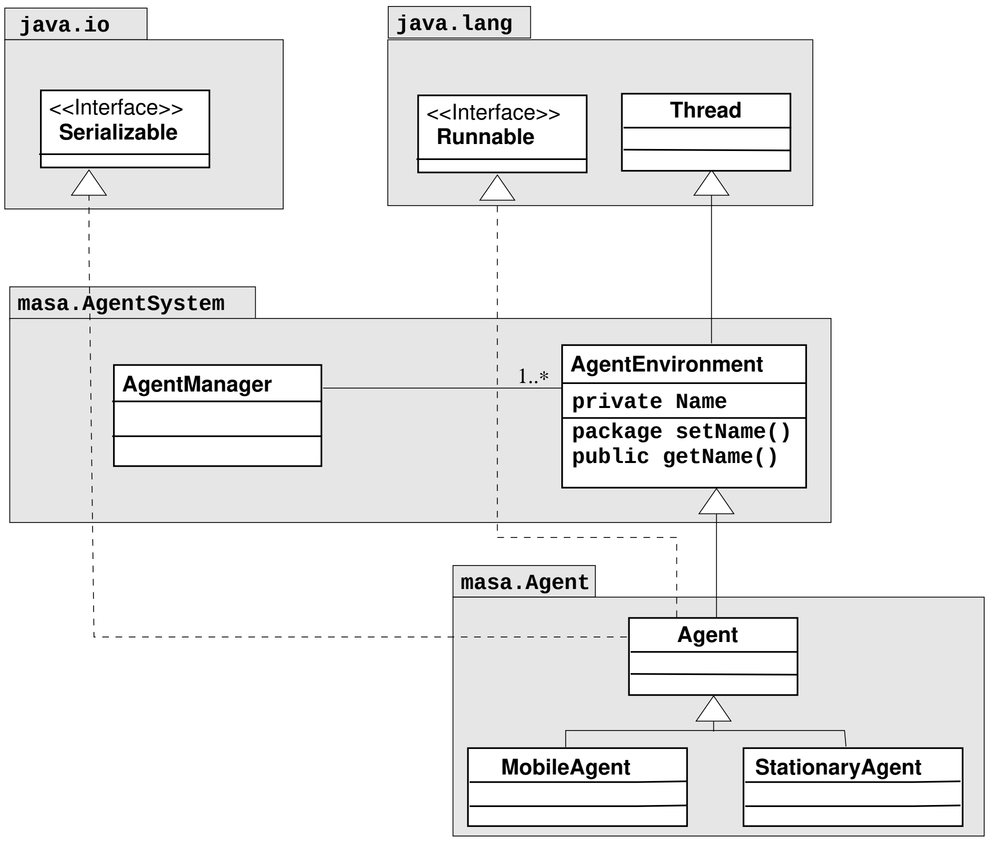


Abbildung 6.10: Geänderte Vererbungs- und Package–Struktur in MASA

Für die Realisierung eigener Namensräume wurde der `AgentClassLoader` implementiert, der vom System Class Loader und dem Security Class Loader abgeleitet ist. Grundsätzlich wird jeder Agent von einer eigenen `AgentClassLoader` Instanz geladen, die damit zum Defining Class Loader wird. Alle vom Defining Class Loader geladenen Klassen werden in der Java Virtual Machine mit der ID des Class Loaders markiert und bilden dadurch einen abgeschlossenen Namensraum (vgl. auch Abschnitt 5.4.3). Für Agenten, die gemeinsame Basisklassen verwenden und auch gemeinsam für die Erledigung einer Aufgabe verantwortlich sind, d.h. eine Gruppe bilden, muss ein gemeinsamer Defining Class Loader verwendet werden. Andernfalls würden die Basisklassen von jedem Agent der Gruppe neu in seinen eigenen Namensraum geladen.

eigene Namensräume

Der Laufzeitschutz, d.h. die Sicherung der Methoden des Agentensystems zur Manipulation von Agenten, wird größtenteils durch den auf Seite 189 beschriebenen Permission Manager realisiert, der die Zugriffskontrolle auf Methoden und Objekte des Agentensystems implementiert. Lediglich die Manipulation eines Agenten durch direkte Aufrufe der Java API — und hier insbesondere die Methoden zur Manipulation von Threads — sind gesondert zu betrachten. Der Zugriff auf Thread–Methoden und Thread Gruppen lässt sich durch entsprechende Policy–Einträge in der `masa.boot.policy` (wie auf Seite 191 erläutert) einschränken [OaWo 99].

Laufzeitschutz

Permission Manager

Die Implementierung des Permission Managers (`AgentSystemPermissionManager`) stützt sich stark auf die Java 2 Sicherheitsarchitektur [Gong 98, GMPS 97, Oaks 98] ab und hier insbesondere auf die Java Klassen `SecurityManager` und `AccessController`. Diese Klassen können für eigene Zwecke erweitert und ergänzt werden. In Abbildung 6.11 wird die Einordnung dieser beiden Konzepte in die Java–Architektur dargestellt.

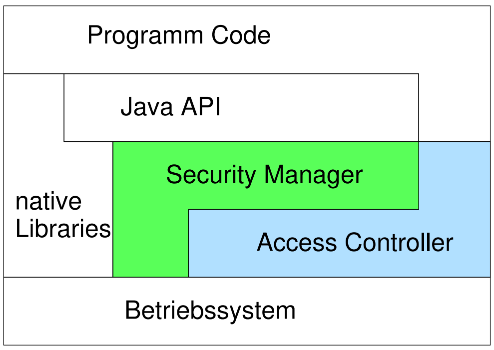


Abbildung 6.11: Einordnung von `AccessController` und `SecurityManager` in die Java Architektur

Der Security Manager realisiert die seit Java 1.0 bekannte Sandbox. Der Zugriff von Applets auf Ressourcen des Betriebssystems wird dabei vom Secu-

Security Manager realisiert Sandbox

rity Manager verhindert. Mit diesem "Alles oder Nichts" Sandbox–Konzept von Java 1.0 sind zwei Probleme verbunden. Auf der einen Seite gibt es keine Möglichkeit für ein Applet die Sandbox zu durchdringen, auf der anderen Seite kann für eine normale Anwendung — wie sie das Agentensystem darstellt – keine Sandbox instantiiert werden. Das erste Problem wurde mit Java 1.1 durch Signed Applets behoben. Für digital signierte Applets kann konfiguriert werden, über welche Schnittstellen sie die Sandbox verlassen dürfen. Das zweite Problem ließe sich durch die Implementierung und Instantiierung eines eigenen Security Managers lösen. Da dies relativ aufwendig und fehleranfällig ist, wurde mit Java 1.2 der Access Controller eingeführt.

Access Controller für die einfache Parametrisierung und Konfiguration

Der Security Manager bleibt zwar weiterhin die zentrale Kontrollinstanz für die Zugriffskontrolle, nutzt aber für seine Entscheidungen den Access Controller, der einfacher zu konfigurieren und zu parametrisieren ist. Für die Entscheidung, ob ein Zugriff gewährt oder abgelehnt wird, benötigt der Access Controller folgende Java–Objekte:

- Die `CodeSource` wird durch eine URL und eine Menge von öffentlichen Schlüsseln determiniert. Über die Code Source lässt sich die Identität der zugreifenden Entität bzw. einer Gruppe von Entitäten abbilden.
- Das Java–Objekt `Permission` kapselt eine bestimmte Operation. Die Permission besitzt zwei verschiedene Anwendungskontexte. Wenn die Permission (über die Code Source) einer Entität zugeordnet wird, repräsentiert sie ein erteiltes Recht. Im anderen Fall kann eine Entität ein bestimmtes Permission Objekt instantiieren, um den Security Manager zur fragen, ob die Entität das spezifizierte Recht besitzt.
  Eine Permission besteht aus drei Teilen:
  1. Der **Type** bestimmt die Art der Permission (z.B. `FilePermission`)
  2. **Name** bezeichnet das konkrete Objekt für das die Permission gelten soll.
  3. **Action** kennzeichnet die durchzuführende Aktion.

  Mit Hilfe einer `PermissionCollection` kann eine Menge von Einzel–Permissions zu einem Objekt zusammengefasst werden.
- Das `Policy` Objekt kapselt alle Permissions aller Entitäten, d.h. über das Policy Objekt können Entitäten Rechte zugeordnet werden. Pro Anwendung kann nur ein Policy Objekt instantiiert werden, d.h. pro Agentensystem gibt es nur eine Policy, in der die Rechte aller Entitäten, die das Agentensystem nutzen, zusammengefasst werden müssen.
- Eine `ProtectionDomain` ist ein Gruppierungsobjekt, um alle Rechte die einer bestimmten Code Source zugeordnet werden, zusammenzufassen.

Der Access Controller kann nicht direkt instantiiert werden, da sein Konstruktor als `private` deklariert ist. Nur wenn ein Security Manager aktiv ist, wird auch der Access Controller instantiiert. Die Besonderheit des Security Managers ist, dass er nur ein einziges Mal instantiiert und dann nicht mehr gelöscht werden kann. Damit wird auf Ebene der Virtual Machine verhindert, das ein einmal aktivierter Security Manager wieder deaktiviert wird.

Der MASA Permission Manager nutzt für die Zugriffskontrolle den Security Manager und den Access Controller. Er instantiiert den vom System Security Manager (`java.lang.SecurityManager`) abgeleiteten `AgentSecurityManager`, der wiederum den Access Controller aktiviert.

MASA Permission Manager instantiiert Security Manager

Der Access Controller stellt eine einzige Methode zur Verfügung — `checkPermission(Permission p)` — die vom Security Manager genutzt wird. Diese Methode testet, ob das Policy Objekt die Permission `p` für das aufrufende Subjekt enthält, d.h. ob das Subjekt die Permission besitzt. Falls ja, kehrt die Methode normal zurück, falls das erforderliche Recht nicht im Policy Objekt ist, wird eine `AccessControlException` generiert.

Dem Security Manager wird über `check`–Methoden signalisiert, wann eine Zugriffskontrollentscheidung zu fällen ist.

Alle Rechte, die auf einem Agentensystem gelten, werden in einem Policy Objekt spezifiziert. Beim Start eines Agentensystems wird dieses Policy Objekt erzeugt. Die statischen Rechte an den Schnittstellen des Agentensystems und des Endsystems werden in der `masa.boot.policy` spezifiziert und mit in das neu erzeugte Policy Objekt übernommen.

statische Rechte werden in der `masa.boot.policy` spezifiziert

Die allgemeine Syntax für Policy Einträge ist in Abbildung 6.12 angegeben [Gong 98].

```
grant [SignedBy ”signer_names”] [, CodeBase ”URL”] {
    permission permission_class_name [ ”target_name” ]
               [, ”action”] [, SignedBy ”signer_names”];
    permission ...
};
```

Abbildung 6.12: Syntax für Einträge im Policy Objekt

Das Schlüsselwort `grant` spezifiziert die in geschweiften Klammern angegebenen Permissions. Die Permission kann an eine Code Base gebunden werden. Wird auch das `SignedBy` Schlüsselwort angegeben, dann wird die Permission nur an Subjekte erteilt, die von der Entität mit dem oder den Namen `signer_names` signiert sind. Der Verknüpfungsoperator für die `grant` Bedingungen ist UND. Das heißt, falls mehrere Signer und eine Code Base angegeben sind, muss das zugreifende Java–Objekt von allen signiert sein und von der entsprechenden Code Base stammen. Über das `SignedBy` Schlüsselwort wird das Rollenkonzept abgebildet.

Das Schlüsselwort `permission` bezeichnet das entsprechende Recht mit Type, Name des Objektes und Action. Auch hier kann wieder ein Signer angegeben werden. Dann wird das Recht nur gewährt, falls die Implementierung des Permission–Objektes entsprechend signiert ist. Damit können Maskerade Angriffe auf Permissions verhindert werden. Falls die Permission–Objekte nicht signiert werden, könnte ein Angreifer seinem Mobilen Agenten eine entsprechende Permission Implementierung mitgeben, die das Recht immer erteilt.

Beispiel aus masa.boot.policy

In Abbildung 6.13 ist ein Ausschnitt der `masa.boot.policy` angegeben. Die spezifizierten Rechte gelten für alle Agenten, deren Gattungs-

```
grant codeBase
"systemresource://de/unimuenchen/informatik/mnm/masa/agent/-"
{
   permission java.util.PropertyPermission
   "de.unimuenchen.informatik.mnm.masa.runSecure", "read";

   permission de.unimuenchen.informatik.mnm.masa.
   agentSystem.ClassAllowUsePermission
   "de.unimuenchen.informatik.mnm.masa.event.-";
   ⋮
}
```

Abbildung 6.13: Ausschnitt aus der `masa.boot.policy`

name mit `de.unimuenchen.informatik.mnm.masa.agent` beginnt. Das „-“ Zeichen ist ein Wildcard–Operator, der auf alle Erweiterungen dieses Präfixes zutrifft. Die Rechte gelten also beispielsweise sowohl für ...`agent.MessAgent` als auch für ...`agent.Abteilung21.Unterabteilung2.QoSAgent`.

Die erste Permission erlaubt es allen über die Code Base spezifizierten Agenten die Property ...`masa.runSecure` auszulesen. Damit können die Agenten bestimmen, ob das Agentensystem, das sie gerade besuchen, im Secure Mode betrieben wird. Die zweite Permission erteilt allen Agenten das Recht, alle Klassen des Event Service zu benutzen.

Versucht ein Agent ein Event Objekt zu instantiieren, wird der Security Manager über eine check–Methode darüber informiert. Der Security Manager befragt über die `checkPermission` Methode den Access Controller, ob der Agent das entsprechende Recht besitzt.

dynamische Rechtevergabe für neue Objekte

Das bisher beschriebene Verfahren ist statisch, da die Permissions alle bereits vor dem Start des Agentensystems bekannt und spezifiziert sein müssen. Zur Verdeutlichung der dynamischen Rechteverteilung wird im Folgenden angegeben, wie eine Methode eines Mobilen Agenten geschützt werden kann. Als Beispiel dient wiederum ein Mess Agent, der eine Methode `getThroughput` besitzt, die zu schützen ist. Außerdem wird angenommen, dass diese Methode als Objekt der Zugriffskontrolle auf dem Ziel–Agentensystem unbekannt ist.

1. Schritt: neue Permission Klasse implementieren

Zuerst muss der Agentenprogrammierer eine neue Permission Klasse `getThroughputPermission` erzeugen. Dazu muss er die Klasse `java.security.Permission` verfeinern:

```
public class getThroughputPermission extends Permission {
    protected int mask;
    static private int EXEC = 0x01;

    public getThroughputPermission(String name) { this(name, "exec"); }

    public getThroughputPermission(String name, String action) {
        super(name);
        if (action == null)
             action = "exec";
        parse(action);
    }

    private void parse(String action) {
        StringTokenizer st = new StringTokenizer(action, ",\t ");
        ⋮
    }

    public boolean implies(Permission permission) {
        if (!(permission instanceof getThroughputPermission))
            return false;

        getThroughputPermission p = (getThroughputPermission) permission;
        String name = getName();
        if (!name.equals("*") && !name.equals(p.getName()))
            return false;
        if ((mask & p.mask) != p.mask)
            return false;
        return true;
    }

    public boolean equals(Object o) {
        if (!(o instanceof getThroughputPermission))
            return false;

        getThroughputPermission p = (getThroughputPermission) o;
        return ((p.getName().equals(getName())) && (p.mask == mask));
    }

    public int hashCode() { return getName().hashCode() ^ mask; }

    public String getActions() {
        if (mask == 0)
            return "";
        else if (mask == EXEC)
            return "exec";
        else throw new IllegalArgumentException("Unknown action");
    }

    public PermissionCollection newPermissionsCollection() {
        return new getThroughputPermissionCollection();
    }
}
```

Abbildung 6.14: Implementierung der `getThroughputPermission`

Innerhalb der `getThroughputPermission` werden alle möglichen Aktionen spezifiziert. Im Beispiel ist dies als einzige Methode die Ausführungsoperation `exec`.

Die wichtigste Methode der Permission Klasse ist `implies`. In dieser Methode wird entschieden, ob eine gegebene Permission die implementierte Permission impliziert, d.h. ob die Entität berechtigt ist, die Aktion auf dem Objekt auszuführen. Diese Methode wird vom Access Controller verwendet, um die Policy Objekte mit den Permissions zu vergleichen. Im angegebenen Beispiel wird FALSE zurückgegeben, wenn die vorgelegte Permission nicht vom Typ `getThroughputPermission` ist. Auch wenn der Name des Objektes, auf das zugegriffen werden soll, nicht übereinstimmt, wird FALSE zurückgeliefert. Hier sieht man, dass innerhalb der `implies` Methode auch das Wildcard Matching implementiert wird. Zuletzt wird noch überprüft, ob die richtige Aktion ausgewählt wurde.

2. Schritt: Implementierung einer check–Methode

Neben der eigentlichen Permission muss auch eine check–Methode implementiert werden (vgl. Abbildung 6.15), die den Access Controller befragt, ob die aufrufende Entität das Ausführungsrecht (`exec`) an `getThroughput` besitzt.

```
public void checkThroughput() {
  AcessController.checkPermission(
    new getThroughputPermission("exec"));
}
```

Abbildung 6.15: Implementierung der `checkTroughput` Methode

3. Schritt: zu schützende Methode durch check–Methode sichern

Die check–Methode und die neue Permission Klasse werden mit in die Agenten Gattung des Mess Agenten übernommen. Die zu schützende Methode muss dann noch erweitert werden. Als erster Methodenaufruf innerhalb von `getThroughput` ist ein Aufruf der check–Methode einzutragen. Damit ist die Durchsetzung der Zugriffskontrolle auf die `getThroughput` Methode implementiert.

4. Schritt: Rechtevergabe über Ergänzung des Policy Objektes

Die eigentliche Rechtevergabe erfolgt über einen Policy–Eintrag, der exemplarisch in Abbildung 6.16 angegeben ist.

```
grant signedBy "Harald Roelle"
{
   permission
   de.unimuenchen.informatik.mnm.masa.MessAgent.getThroughput
   "de.unimuenchen.informatik.mnm.masa.MessAgent.*" "exec";
   signedBy "QoSManager"
}
```

Abbildung 6.16: Beispiel eines Policy Eintrags zum Schutz von `getThroughput`

Diese `grant`–Strophe wird von dem Mess Agenten transportiert und, sobald er auf dem Ziel–Agentensystem ankommt, kann er über die Schnittstelle `addPolicy` des Permission Managers seine Rechtespezifikation in die Agentensystem–Policy mit übernehmen. Im angegebenen Beispiel haben alle Entitäten, die von Harald Rölle signiert sind, das Recht die Methode `getThrouphput` des `MessAgent` auszuführen. Zum Schutz vor Maskerade muss der Bytecode, der die `getThroughputPermission` implementiert vom QoS Manager signiert sein.

Ausblick: Mit dieser Technik lässt sich die Dynamik in einem System Mobiler Agenten relativ gut behandeln. Es können dynamisch neue Rechte vergeben werden. Die volle Dynamik, wie sie wünschenswert wäre, lässt sich mit der gegenwärtigen Implementierung des Policy Objektes nicht erreichen. In Java 1.2 ist das Policy Objekt nur semi–dynamisch. Das Policy Objekt kann zwar mit der `refresh()` Methode beliebig erneuert werden, aber eine Entität, der ein Recht erteilt wurde, behält dieses bis zum Ende ihres Lebenszyklus. In der gegenwärtigen Implementierung fehlt eine `revoke()` Methode, mit der Rechte explizit entzogen werden können. Um dies zu realisieren, müsste die Policy Klasse entsprechend geändert werden, (vgl. z.B. [NiPa 99]). Dies wurde im Prototyp nicht realisiert.

Problem: Java Policy nur semi–dynamisch

Das Beta–Release in der Version 1.4 des Java 2 SDK beinhaltet erstmals den Java Authentication and Authorization Service (JAAS) [LGKN 99, JAAS]. JAAS unterstützt das Konzept des Principal sowie rollenbasierte Zugriffskontrollkonzepte. Die Einträge in den Policies können entsprechend erweitert werden, vgl. Abbildung 6.17.

JAAS unterstützt Rollenkonzept

```
grant Codebase "http://uni–muenchen.de"
signedBy "Harald Roelle"
Principal MNM.Team "Administrator"
{
   permission
   java.io.FilePermission
   "/etc/passwd" "read, write";
}
```

Abbildung 6.17: JAAS: Beispiel eines Policy Eintrags

Neu ist hier die `Principal` Zeile. Es wird nicht mehr nur betrachtet, woher der Code kommt, der die Passwort Datei lesen oder schreiben möchte, und wer diesen signiert hat, sondern welcher Principal das entsprechende Objekt „beauftragt“ hat. Im angegebenen Beispiel würde das Recht erteilt, wenn das Objekt, das auf die Datei zugreifen möchte, von der Uni München kommt, von Harald Rölle signiert wurde und von einem Benutzter aus dem MNM Team gestartet wurde, der die Rolle „Administrator“ hat.

Auditing und Verbindlichkeit

Die Logger–Komponente (implementiert in `masa.tools.Debug` und `masa.tools.DebugControl`) des Agentensystems schreibt eine Log–Datei, in der alle Systemaktivitäten protokolliert werden können. Der Logger

Logger protokolliert Systemaktivität

lehnt sich bezüglich seiner Funktionalität an den aus Unix bekannten syslog–Mechanismus an. Es können verschiedene Log–Level (z.B. `Fatal`, `Error`, `Warning`, `Info`, usw.) konfiguriert werden. Die Quelle des Logging Eintrags, d.h. die einzelne Komponente des Agentensystems, wird als *Facility* bezeichnet. Die Agenten bilden ebenfalls eine Facility Klasse. Mit Hilfe des in Abbildung 6.18 dargestellten `LoggerConfigAgent` können für jede Facility Klasse die aufzuzeichnenden Log–Level ausgewählt werden. In Abbil-

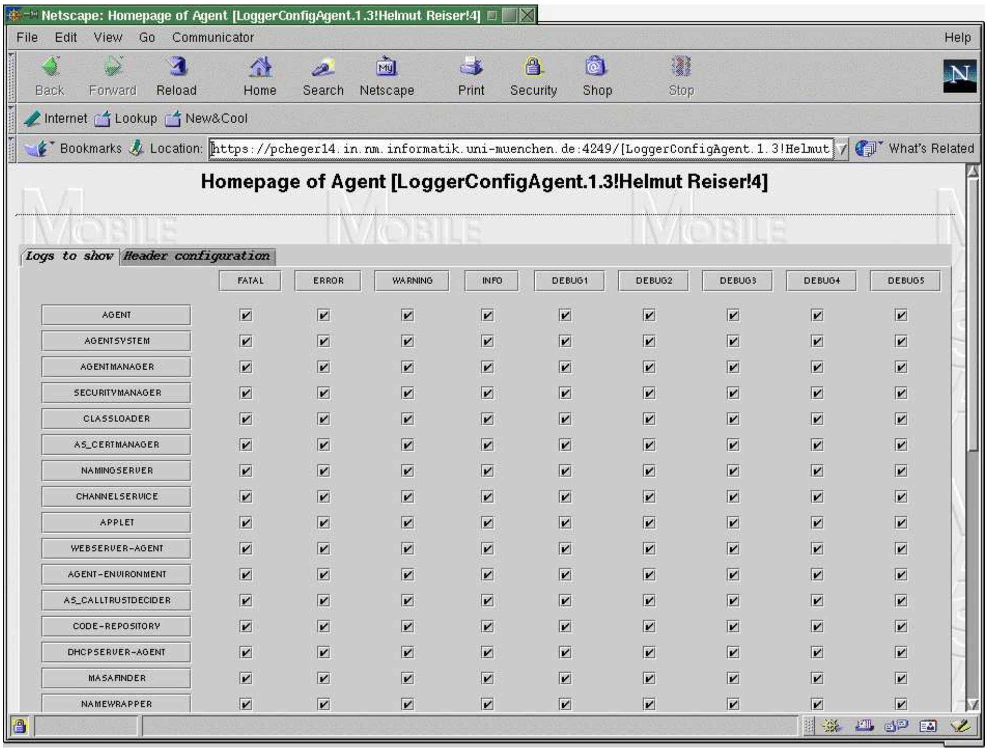


Abbildung 6.18: Logger Config Agent zur Konfiguration des Auditings

```
<Aug 30><12:07:24><2001>
<WEBSERVER–AGENT><ERROR>
<[pcheger14.in.nm.informatik.uni–muenchen.de!MASAdministrator MASA
   Administrator!4]>
<Thread[Thread−1,2,[Webserver.1.1![pcheger14.in.nm.informatik.
   uni–muenchen.de!MASAdministrator MASA Administrator!4]
   (MASA Agent System V0.5)!4]−group]>
<at ...agent.Webserver.HttpsServer.run(HttpsServer.java:260)>
    SSL Exception
    Client has no certificate!
    iaik.security.ssl.SSLException: Client has no certificate!
        at iaik.security.ssl.r.e(Unknown Source)
```

Abbildung 6.19: Beispiel eines Log–Eintrags

Beispiel eines Log–Eintrags

dung 6.19 ist ein Logging Eintrag angegeben. Die einzelnen Bestandteile des Eintrags werden in „< >“ eingeschlossen. Im Beispiel beginnt der Eintrag mit dem Datum, der Uhrzeit und dem Jahr des Events. Daran anschließend erscheint der Name der Facility, der Log–Level, danach der Name des Agentensystems, die Thread–ID und der Name des Agenten, der diesen Event ausgelöst hat sowie die Stelle im Code, an der der Event produziert wurde. Am Ende wird noch die entsprechende Meldung des Agenten selbst ausgegeben. Im Beispiel hat der HTTP–Server–Agent (`Webserver.1.1`) eine HTTPS–Anfrage ohne gültiges Zertifikat erhalten und die Verbindung abgelehnt.

Über den `LoggerConfigAgent` kann auch der Umfang der Logging–Einträge festgelegt werden. Jeder einzelne Bestandteil eines Logging–Events (geklammert durch „< >“) kann aktiviert oder deaktiviert werden.

Werden die Log–Einträge zusätzlich zur Speicherung in einer lokalen Datei auch noch auf einem oder mehreren Logging–Servern geschrieben, kann eine nachträgliche Veränderung der lokalen Log–Dateien erkannt werden.

Um die Logs justitiabel zu machen, müsste eine digitale Signatur der einzelnen Log–Einträge durch das Agentensystem implementiert werden. Dies ließe sich innerhalb der MASA–Sicherheitsarchitektur sehr einfach realisieren. Es wäre auch denkbar, zusätzlich eine Signatur des Log–Eintrags durch die auslösende Entität zu implementieren. Damit würde die Entität ihre Aktionen bestätigen.

6.3 Evaluation der Performance

In einem Managementsystem werden Sicherheitsmechanismen i.d.R. zusätzlich zur eigentlichen Managementfunktionalität implementiert. Naturgemäß führt diese zusätzliche Funktionalität auch zu einem erhöhten Ressourcenverbrauch. Um diesen zusätzlichen Aufwand abschätzen zu können, wurde die vorgestellte Implementierung mit Hilfe eines Profilers untersucht.

Profiler findet Hot–Spots

Ein **Profiler** dient u.a. dazu, so genannte **Hot–Spots** in einer Anwendung zu finden; d.h. die Teile eines Programms, in denen die meiste Rechenzeit verbraucht oder die Objekte, von denen der meiste Speicher belegt wird. In der Java Virtual Machine wird das Profiling durch das **Java Virtual Machine Profiler Interface (JVMPI)** unterstützt [JVMPI, ViLi 00]. Mit Hilfe dieses Interfaces und eines entsprechenden Profiling–Agenten lässt sich der Ressourcenverbrauch von Java Anwendungen messen. Das MASA Agentensystem, das einen Großteil der Sicherheitsarchitektur implementiert und die Ausführungsplattform für Mobile Agenten darstellt, wurde mit Hilfe des HPROF Profiling–Agenten [JVMPI] untersucht.

MASA Betriebsmodi: secure oder insecure

Die Sicherheitsarchitektur wurde so implementiert, dass MASA in einem „secure“ und einem „insecure“ Modus betrieben werden kann. Im insecure Modus werden alle Bestandteile der Architektur, die auf JCE basieren, deaktiviert. In diesem Fall sind alle Komponenten der Sicherheitsarchitektur, die Zertifikate oder starke Verschlüsselung verwenden, nicht aktiv. Die Gründe für zwei verschiedene Betriebsmodi liegen zum einen in der Exportproblematik für Kryptographie–Software, zum anderen in Linzenzbedingungen von verwendeten Software–Bibliotheken. In einigen Ländern (z.B. Frankreich) dürfen Produkte, die starke Verschlüsselungstechniken einsetzen, nicht verwendet werden. Außerdem dürfen die verwendeten JCE–Provider (vgl. Seite 182) kostenlos nur in Forschung und Lehre eingesetzt werden. Nachdem die oberste Prämisse bei der Entwicklung von MASA eine möglichst breite Anwendbarkeit war, werden alle Teile, die kritisch im angegebenen Sinne sind, im insecure Modus ausgeblendet. Damit kann gewährleistet werden, dass MASA von allen Kooperationspartnern für ihre Zwecke eingesetzt werden kann.

Testfall: Start und Stopp des Agentensystems

Für das Profiling wurde ein Agentensystem im secure und insecure Modus gestartet, dann wieder beendet und dabei Profiling Daten gewonnen, um den Laufzeit–Overhead, der durch die Sicherheitsarchitektur entsteht, zu ermitteln. Die Messungen wurden auf einem Pentium III Prozessor (1 GHz) mit 512 Mb Hauptspeicher unter SuSE Linux 7.0 (Kernel 2.2.18) durchgeführt. In den Abbildungen 6.20 und 6.21 ist jeweils ein kleiner Ausschnitt der Informationen, die mit Hilfe des Profilers HPROF ermittelt wurden, dargestellt. Grundlage der Daten in Abbildung 6.20 war der Start des Agentensystems im insecure Modus, in Abbildung 6.21 entsprechend im secure Modus. Dabei wurde der Verbrauch von CPU–Zeit ermittelt.

```
CPU SAMPLES BEGIN (total = 652) Tue Nov  6 14:07:07 2001
rank   self  accum   count trace method
   1  9.05%  9.05%      59   214 java/io/BufferedReader.<init>
   2  7.36% 16.41%      48   167 java/io/BufferedReader.<init>
   3  5.67% 22.09%      37   169 java/io/BufferedReader.<init>
   4  3.68% 25.77%      24   249 java/io/BufferedReader.<init>
   5  2.61% 28.37%      17    15 java/lang/ClassLoader.defineClass0
   6  2.15% 30.52%      14   243 java/lang/Throwable.printStackTrace0
   7  1.69% 32.21%      11   409 java/io/BufferedReader.<init>
   8  1.69% 33.90%      11   285 java/lang/StringBuffer.<init>
  :
 292  0.15% 100.00%      1   356 java/util/zip/ZipFile.open
CPU SAMPLES END


TRACE 214: (thread=11)
        java/io/BufferedReader.<init>(BufferedReader.java:81)
        java/io/BufferedReader.<init>(BufferedReader.java:92)
        java/io/LineNumberReader.<init>(LineNumberReader.java:47)
        de/unimuenchen/informatik/mnm/masa/tools/ClassContextTool.
                            getCallStack(ClassContextTool.java:21)

TRACE 15: (thread=1)
        java/lang/ClassLoader.defineClass0(ClassLoader.java:Native method)
        java/lang/ClassLoader.defineClass(ClassLoader.java:477)
        java/security/SecureClassLoader.defineClass(SecureClassLoader.java:109)
        java/net/URLClassLoader.defineClass(URLClassLoader.java:248)

THREAD START (obj=861fb20, id = 11, name="MASA_AgentManager", group="main")

THREAD START (obj=81c6fa0, id = 1, name="main", group="main")
```

Abbildung 6.20: Ausschnitt aus Profiling Information des Agentensystems im insecure Modus

Der Profiler untersucht dazu periodisch die Stacks aller aktiven Threads in der Java Virtual Machine. Eine solche „Stichprobe“ wird als **Sample** und die Betrachtung der Stacks als **Stack Trace** bezeichnet. Aus einem Stack Trace lässt sich ermitteln, welche Objekte bzw. Methoden zum Stichprobenzeitpunkt aktiv sind; diese werden in den Abbildungen in der Spalte `method` angegeben. Die `count` Spalte gibt an, wie oft eine Methode innerhalb des gesamten betrachteten Zeitraumes aktiv war. Die Methoden, die am häufigsten aktiv waren, sind die Hot–Spots innerhalb des Agentensystems. Die Gesamtlaufzeit der Anwendung ist jeweils in der ersten Zeile der Abbildungen unter `total` angegeben. Die Maßeinheit hierbei sind keine realen Zeitinformationen, sondern die Anzahl der Samples. Der Profiling–Agent gibt auch die Stack–Traces selbst mit aus. Daraus lässt sich ermitteln, von welchen Methoden die gerade aktive Methode aufgerufen wurde. Da diese Traces sehr groß werden können, wurde deren maximale Tiefe in der Messung auf vier beschränkt.

```
1  CPU SAMPLES BEGIN (total = 1371) Tue Nov  6 13:59:55 2001
2  rank   self  accum   count trace method
3     1 16.27% 16.27%     223   260 java/lang/System.currentTimeMillis
4     2 12.04% 28.30%     165   262 sun/security/provider/SeedGenerator.run
5     3  2.33% 30.63%      32    68 java/lang/ClassLoader.defineClass0
6     4  1.82% 32.46%      25   541 java/io/BufferedReader.<init>
7     5  1.53% 33.99%      21   435 java/lang/Throwable.printStackTrace0
8     ⋮
9   574  0.07% 100.00%      1   160 java/io/UnixFileSystem.canonicalize
10 CPU SAMPLES END
11
12
13 TRACE 260: (thread=6)
14         java/lang/System.currentTimeMillis(System.java:Native method)
15         sun/security/provider/SeedGenerator.run(SeedGenerator.java:135)
16         java/lang/Thread.run(Thread.java:479)
17
18 TRACE 262: (thread=6)
19         sun/security/provider/SeedGenerator.run(SeedGenerator.java:135)
20         java/lang/Thread.run(Thread.java:479)
21
22 TRACE 68: (thread=1)
23         java/lang/ClassLoader.defineClass0(ClassLoader.java:Native method)
24         java/lang/ClassLoader.defineClass(ClassLoader.java:477)
25         java/security/SecureClassLoader.defineClass(SecureClassLoader.java:109)
26         java/net/URLClassLoader.defineClass(URLClassLoader.java:248)
27
28 THREAD START (obj=86d5c30, id = 6, name="SeedGenerator Thread",
29               group="SeedGenerator ThreadGroup")
```

Abbildung 6.21: Ausschnitt aus Profiling Information des Agentensystems im secure Modus

Wird die Gesamtlaufzeit der beiden Modi verglichen, zeigt sich, dass diese für den angegebenen Testfall im secure Modus um den Faktor zwei höher ist als im insecure Modus (1371 samples im Vergleich zu 652). Dabei verbrauchen die ersten fünf Traces ein Drittel der gesamten Rechenzeit (Zeile 7, Spalte `accum` in Abbildung 6.21). Bei der Betrachtung der einzelnen Traces zeigt sich, dass die ersten beiden, mit der Nummer 260 und 262 (Zeile 3 und 4), fast 30% der gesamten Rechenzeit verbrauchen. Von Zeile 13 bis Zeile 20 sind die Aufruf Stacks der beiden Traces dargestellt. Der `SeedGenerator` (Trace 262) dient dazu, Zufallszahlen zur Schlüsselberechnung zu erzeugen. Dieser ruft wiederum die Methode `currentTimeMillis` auf, die den Trace 260 ausmacht.

Laufzeit–Overhead durch Schlüsselberechnung

Das heißt, das Starten eines Agentensystems im secure Modus dauert fast doppelt so lange wie im insecure Modus und dabei werden fast 30 % der Rechenzeit zur Berechnung von Zufallszahlen für die Schlüsselberechnung verbraucht.

In der folgenden Aufzählung werden die wesentlichen Aktionen, die beim Start und Stopp eines Agentensystems ausgeführt werden, aufgeführt. Die Aktionen, die im secure Modus *zusätzlich* nötig sind, werden kursiv dargestellt:

1. Initialisierung des Agentensystems und seiner Komponenten (Anbindung an das Kommunikationssystem, Aufbau des Event Channels, Verbindung zum Naming Service, *Instantiierung der Komponenten der Sicherheitsarchitektur, Authentisierung des Besitzers des AS, Berechnung der Sitzungsschlüssel für das AS*)
2. Registrierung des Agentensystems und aller Places über MASAFinder beim Naming Service
3. Initialisierung und Registrierung des HTTP–Server Agent, *Erzeugung der Zertifikate und Sitzungsschlüssel für den Agenten, Aufbau verschlüsselter Verbindungen durch den Communication Manager*
4. Initialisierung und Registrierung des AS Management Agent (GUI des Agentensystems), *Authentisierung des Agenten, Erzeugung der Zertifikate und Sitzungsschlüssel für den Agenten, Aufbau verschlüsselter Verbindungen*
5. Befehl zum Stopp des Agentensystems
6. Stopp und Deregistrierung des AS Management Agent und des HTTP–Server Agenten
7. Stopp des Agent Manager
8. Deregistrierung des Agentensystems
9. Terminierung des Agentensystems

Der Start eines Agentensystems im secure Modus ist unter Laufzeitaspekten der worst–case für die Sicherheitsarchitektur. Alle Komponenten der Sicherheitsarchitektur müssen initialisiert werden und der Certification Manager muss Schlüssel für das Agentensystem selbst, sowie Zertifikate und Schlüssel für alle Agenten erzeugen.

Beim Start eines Agentensystems gehen damit alle Mechanismen bzw. Komponenten der Sicherheitsarchitektur, die beim oben angegebenen Ablauf zusätzlich ausgeführt werden, mit in den Runtime–Overhead ein. Das heißt, die folgenden Mechanismen der Sicherheitsarchitektur wurden im Testfall (mit–) gemessen:

- Authentisierung der Entitäten (vgl. Abschnitt 4.2.3, 4.2.4 und 5.4.2)
- LCA– und Naming–Management (vgl. Abschnitt 5.4.1 sowie 4.2.2 und 5.4.8)

- Aufbau der Laufzeitumgebung für Mobile Agenten; Speicherschutz, Laufzeitschutz, getrennte Namensräume (vgl. Abschnitt 4.3.2 und Abschnitt 5.4.3)
- Integritätstest der Agenten–Gattung (vgl. Abschnitt 4.5.2 und 5.4.4)
- Zugriffskontrolle (vgl. Abschnitt 4.4.1 und 5.4.7)
- Vertrauliche Kommunikation zwischen den Entitäten Agentensystem, Agenten und dem Naming Service (vgl. Abschnitt 4.5.1 und 5.4.5)
- Auditing und Logging (vgl. Abschnitt 5.4.9)

Da ein verteilter Profiler nicht zur Verfügung stand, wurden die Aspekte der Sicherheitsarchitektur, an denen mehr als ein Agentensystem beteiligt sein muss, nicht evaluiert. Entsprechend wurden folgende Mechanismen und Abläufe nicht gemessen:

- Trust–Level Management (vgl. Abschnitt 4.1.2)
- Migration von Agenten (vgl. Abschnitt 5.4.4)
- Aufbau und Verifikation der KAG–Integritätslisten (vgl. Abschnitt 4.5.2, 4.5.3 und 5.4.6)

Optimierung durch Key–Cache

Im laufenden Betrieb des Agentensystems lässt sich der Engpass, der durch die Schlüsselerzeugung entsteht, entzerren. Dazu wurde, wie in Abschnitt 6.2.2 auf Seite 182 beschrieben, der Key–Cache implementiert. Dieser Cache wird von einem Thread mit niedriger Priorität sukzessive gefüllt. Die teuren Operationen werden also nur ausgeführt, wenn die Last auf dem Agentensystem nicht hoch ist. Eine Anpassung an das jeweilige Agentensystem erfolgt über die konfigurierbare Größe des Cache. Bei einem Agentensystem, das sehr viele oder sehr häufig neue Agenten besuchen, muss der Key–Cache entsprechend größer konfiguriert werden als bei einem Agentensystem, das wenige oder selten neue Agenten besuchen, um dasselbe Laufzeitverhalten zu bekommen.

Zufallszahlen: Optimierung durch Hardware–Unterstützung

Auch die Generierung der Zufallszahlen lässt sich optimieren. Werden diese nicht mehr durch die doch sehr aufwendigen algorithmischen Verfahren softwaremäßig, sondern durch Hardware unterstützt berechnet, so kann der Overhead, der dadurch entsteht, fast auf Null gesenkt werden. In vielen Chip–Sätzen heutiger Rechner sind bereits Komponenten zur Hardware–gestützten Generierung von Zufallszahlen integriert. Als Beispiel seien hier nur die Intel Chip–Sätze der 800er Serie genannt, auf denen ein Zufallszahlengenerator integriert ist, der auf thermischem Rauschen basiert [IRNG 99, i8xx]. Um diesen oder auch andere Hardware–basierte Generatoren in die Sicherheitsarchitektur zu integrieren, müsste lediglich der `SeedGenerator` (vgl. Abb. 6.21) so angepasst werden, dass er die entsprechenden Register des (Hardware–) Zufallszahlengenerators ausliest. Damit ließe sich die Laufzeit im Testfall für den secure Modus um fast 30% reduzieren. Eine Anpassung der Java Virtual Machine ist hierzu nicht erforderlich.

Kapitel 7

Zusammenfassung und Ausblick

Inhaltsverzeichnis

Die Flexibilität, die Möglichkeit Heterogenität bis zu einem gewissen Grade zu überwinden, das intuitive Programmierparadigma des Mobilen Agenten und schließlich die Möglichkeit, den Agenten mit seinen Daten **und** Zustandsinformationen zu migrieren, machen diese Technik so interessant für die Anwendung im Management. Allerdings stehen einer breiten Anwendung dieser Technologie zwei Sachverhalte entgegen. Zum einen die mangelnde Interoperabilität und Standardisierung der verschiedenen Systeme Mobiler Agenten. Was aber noch wichtiger ist für den Einsatz, in für die Praxis relevanten Szenarien, ist die mangelnde Sicherheit bestehender Systeme. Diese Arbeit befasste sich mit der zweiten Fragestellung.

7.1 Ergebnisse dieser Arbeit

Das Problem bei einem Sicherheitskonzept für ein organisationsübergreifendes Managementsystem auf der Basis Mobiler Agenten ist eng verknüpft mit der mangelnden Interoperabilität bestehender Systeme. Häufig ist es so, dass Sicherheitsmechanismen nur für eine konkrete Implementierungsklasse eines Systems Mobiler Agenten entwickelt und dann auch nur auf dem konkreten System implementiert wird.

allgemeines Systemmodell als Basis der Anforderungsanalyse

Diese Arbeit hat bewusst mit diesem Ansatz gebrochen. Die Anforderungsanalyse wurde nicht für ein konkretes System durchgeführt, sondern es wurde ein allgemeineres Modell entwickelt. Dieses Systemmodell, dass sich grob in ein Entitäten– und Rollenmodell aufteilen lässt, ist so allgemeingültig, dass es auf alle Systeme Mobiler Agenten angewendet werden kann. Die Sicherheitsanforderungen, die an ein System Mobiler Agenten gestellt werden, wurden dann auch anhand der verschiedenen Klassen des Modells abgeleitet. Das Ergebnis dieser Analyse war eine Liste von Sicherheitsanforderungen.

Ableitung der Sicherheitsmechanismen

Für diese Anforderungen wurden systematisch Sicherheitsmechanismen abgeleitet, die in der Lage sind, diese Anforderungen zu erfüllen. Dabei wurde ganz bewusst auf offene Szenarien verzichtet und das Anwendungsgebiet auf organisationsübergreifendes Management eingeschränkt. Die Tatsache, dass zwischen Unternehmen vertragliche Beziehungen bestehen und sich die Beteiligten bereits kennen, erleichtert den Aufbau einer Sicherheitsarchitektur.

Trust Level Management als Basis

Es existieren bereits Vertrauensbeziehungen zwischen den Beteiligten, die sich durch das in dieser Arbeit entwickelte Konzept zum Trust Level Management auf eine formalere Art und Weise beschreiben und auch berechnen lassen.

Ein weiteres in dieser Arbeit entwickeltes Prinzip ist der Grundsatz des Vertrauens durch Einbettungsbeziehung. Dies besagt zum einen, dass ein Mobiler Agent dem Agentensystem, voll ausgeliefert ist , sobald er auf dieses migriert. Andererseits lassen sich aus diesem Prinzip aber auch wieder Vertrauensbeziehungen zwischen dem Agenten und seinem Agentensystem ableiten. Dies führt dazu, dass das Agentensystem gewissermaßen als Prokurist des Mobilen Agenten handelt und stellvertretend für diesen sicherheitsrelevante Aktionen ausführt oder ihm Schnittstellen und Implementierungen für sicherheitskritische Dienste zur Verfügung stellt. Dies ist der fundamentale Unterschied zu offenen Szenarien. Hier gilt, dass der Mobile Agent seinem Agentensystem grundsätzlich nicht vertrauen kann.

AS als Prokurist des MA

Trennung des MA in Gattung und Instanz

Im Rahmen der Untersuchungen zur Identifikation und Authentisierung wurde erkannt, dass der Mobile Agent nicht als eine Einheit betrachtet werden kann, sondern aus zwei Teilen besteht, der Agentengattung und der Agenteninstanz. In der Literatur wurde diese Trennung bisher so nicht getroffen. Der Mobile Agent wird immer als Gesamtheit seiner statischen und dynamischen Anteile gesehen. Wir haben festgestellt, dass Agenteninstanz und –gattung verschiedene Teile des Lebenszyklus betreffen und dass für beide im Regelfall unterschiedliche Authorities verantwortlich sind. Für die Gattung — die auch aus einer anderen Quelle stammen kann als die Instanz — ist der Implementierer, für die Instanz ist der Benutzer bzw. der Besitzer verantwortlich. An diesem kleinen Beispiel wird bereits klar, dass die beiden Teile des Agenten nicht zusammengefasst und dann einheitlich behandelt werden dürfen.

Klassifikation und Schutz der Nutzdaten

Auch für die von der Agenteninstanz transportierten Nutzdaten wurden in der Arbeit Mechanismen angegeben, wie diese geschützt werden können. Die schützenswerten Daten wurden klassifiziert und für jede Klasse wurden Verfahren oder Protokolle angegeben, um die Daten entsprechend der konkreten Anforderungen, schützen zu können.

Das dieser Arbeit zugrunde liegende Szenario des domänenübergreifenden Managements erfordert eine skalierbare Infrastruktur. Die Sicherheitsarchitektur unterscheidet Komponenten des Agentensystems, Intra–Domänen und Inter–Domänen–Komponenten. Jedes beteiligte Unternehmen kann seine eigene Domäne mit Agentensystemen und Intra–Domänen–Komponenten eigenverantwortlich und ggf. auch autark absichern. Die Sicherheitsarchitektur basiert sehr stark auf einer CA–Infrastruktur. Diese wird neben den originären Aufgaben einer CA auch zur Identifizierung zur Rollenzuteilung und zur Autorisierung verwendet. Die CA–Infrastruktur ist durchgängig vom „Kleinen“ zum „Großen“. Jedes Agentensystem ist für seine Agenten LCA und damit Teil der CA–Infrastruktur. Innerhalb einer Domäne kann es mehrere Abteilungs–CAs und Domänen–CAs geben. Die Struktur kann sowohl hierarchisch als auch vernetzt aufgebaut werden. Die CA verwaltet neben den Identitäts–Zertifikaten auch Rollenzertifikate, da Autorisierung und Zugriffskontrolle auf einem rollenbasierten Zugriffskontrollmodell gründen. Sobald Domänengrenzen überschritten werden, können domänenübergreifende CAs aufgesetzt werden. Lokale Infrastrukturen innerhalb der Domäne müssen dazu kaum angepasst oder geändert werden. Es ist damit technisch sehr einfach, eine bisher eigenständige Management–Domäne in einen größeren Kontext einzugliedern. Wir sind uns allerdings auch bewusst, dass der organisatorische Aufwand, der z.B. durch die Abbildung oder Integration verschiedener Rollenmodelle entstehen kann, nicht unerheblich ist (vgl. auch Abschnitt 7.3).

skalierbare Infrastruktur

CA als integrierende Komponente

Um die Tragfähigkeit und Anwendbarkeit der entwickelten Konzepte zu zeigen, haben wir einen Großteil der spezifizierten Komponenten der Architektur innerhalb von MASA implementiert und integriert. Dabei zeigte sich auch, dass die Konstruktionsprinzipien für alle Systeme Mobiler Agenten verwendet werden können. Die meisten der entwickelten Komponenten der Architektur ließen sich auch auf andere Java–basierte Systeme Mobiler Agenten übertragen.

Allerdings konnten, auch für das eingeschränkte Szenario, nicht alle Problemstellungen vollständig erörtert werden; die Fragen, die offen bleiben mussten, werden in Abschnitt 7.3 zusammengefasst.

7.2 Einsatzgebiete der Sicherheitsarchitektur im Management

Diese Arbeit muss sich der Frage nach der Anwendbarkeit und der Einsatzfähigkeit der Sicherheitsarchitektur stellen. Gibt es Bereiche des Managements, für welche die Sicherheitsarchitektur besonders geeignet oder solche, für die sie überhaupt nicht geeignet ist?

domänenübergreifendes Management

Die vorgestellte Lösung wurde vor dem Hintergrund und für Szenarien entwickelt, in denen ein domänenübergreifendes Managementsystem erforderlich ist. Solche Managementsysteme werden auch als interorganisational bezeichnet [Nerb 01]. Sie werden von verschiedenen, rechtlich selbständigen und unabhängigen Unternehmen oder Organisationen aufgebaut und betrieben. Ein weiteres Beispiel hierfür sind Customer Service Management (CSM) Systeme [Lang 01, Nerb 01], in denen der Kunde passiven oder auch aktiven Zugriff auf die Managementsysteme seiner Provider erhält. Wie in Abschnitt 2.2 bereits angedeutet, kommen die Betreiber und Nutzer von domänenübergreifenden Managementsystemen nicht umhin, starke Sicherheitsverfahren und –mechanismen einzusetzen, auch wenn dadurch ein erhöhter Aufwand entsteht.

erhöhter Ressourcenbedarf als mögliche Beschränkung

Dieser erhöhte Aufwand resultiert aus den doch relativ hohen Hardware–Anforderungen der Java Virtual Machine, durch die Laufzeitdefizite, verursacht durch die Interpretation von Java Byte Code sowie durch die algorithmisch aufwendigen und rechenzeitintensiven kryptographischen Verfahren (vgl. auch Abschnitt 6.3). Auch die Kommunikationsinfrastruktur, die im Falle des Prototyps durch CORBA bereit gestellt und über IIOP/SSL abgesichert wird, verursacht zusätzlichen Ressourcenbedarf. Dies kann zu Einschränkungen der Anwendbarkeit bspw. im Netzmanagement führen. Netzkomponenten, die keine Möglichkeit bieten ein Agentensystem zu installieren, können nicht direkt durch Mobile Agenten verwaltet werden. Hier bietet das Proxy Konzept jedoch die Möglichkeit einen Mobilen Agenten auf ein Agentensystem zu migrieren, das sich möglichst nahe bei der Komponente befindet, um diese dann über klassische Managementschnittstellen zu administrieren.

Der erhöhte Hardware–Aufwand lässt sich aber auch durch Konzepte wie die Java Micro Edition [J2ME] und minimumCORBA [OMG 01-09-34a], die für Systeme mit beschränkter Hardware entwickelt wurden, relativieren. Auch die durch Java verursachten Laufzeitdefizite lassen sich durch Just in Time Compiler lösen. Lediglich der Runtime–Overhead, der durch die kryptographischen Algorithmen verursacht wird, ist systemimmanent.

Für die relativ teure Berechnung der Schlüssel und hier insbesondere für die Berechnung der Zufallszahlen kann aber, wie in Abschnitt 6.3 beschrieben, auf Hardware–basierte Lösungen zurückgegriffen werden. Auch wenn Hardware–Zufallszahlengeneratoren nicht auf jedem System zur Verfügung stehen, kann selbst der Mobile Agent auf seiner Reise die entsprechende Spezial–Hardware nutzen. In [BHR 01, BrRe 01] wurde gezeigt, wie Mobile Agenten entwickelt werden können, die sich selbst an spezielle Systemumgebungen anpassen können.

Einsatz in den verschiedenen Funktionsbereichen

Die Frage nach den Einsatzbereichen der Sicherheitsarchitektur lässt sich auch anhand der Management–Funktionsbereiche und –Disziplinen (nach [HAN 99a]) untersuchen. Für welche Funktionsbereiche bzw. Management–Disziplinen sind starke Sicherheitsmechanismen erforderlich? Gibt es Funktionsbereiche, bei denen auf die Sicherheitsarchitektur verzichtet werden kann?

- Im Konfigurationsmanagement geht es primär um die aktive Gestaltung eines verteilten IT–Systems, d.h. um das Setzen oder Ändern von Parametern, die das Verhalten des Systems bestimmen.

- Das Fehlermanagement beschäftigt sich mit dem Entdecken, Eingrenzen und Beheben von anormalem Systemverhalten.
- Das Abrechnungsmanagement sorgt für eine Umlegung der Kosten, die durch den Betrieb einer IT–Infrastruktur entstehen, auf die Nutzer der Infrastruktur.
- Das zentrale Schlagwort des Leistungsmanagements ist die Dienstgüte und Ziel ist es, die Güte der Dienst zu optimieren.
- Alle Funktionen und Aufgaben, die das Management der Sicherheit in einem verteilten System zum Ziel haben, werden im Funktionsbereich Sicherheitsmanagement zusammengefasst.

Jeder dieser Funktionsbereiche ist in unterschiedlicher Ausprägung in den verschiedenen Managementdisziplinen vom Netz–, System–, Anwendungs–, Dienst– bis hin zum Enterprise Management, zu erbringen. Die Tätigkeiten des Konfigurations–, Fehler– und Leistungsmanagements sind äußerst kritisch für die Funktion und Verfügbarkeit des gesamten IT–Systems und als solche unbedingt vor Angriffen oder unberechtigter Nutzung zu schützen. Alle Funktionen, die direkte oder indirekte monetäre Auswirkungen auf Beteiligte haben, werden von diesen nur akzeptiert werden, wenn sichergestellt ist, dass die Nutzung und der Betrieb gesichert ist, sowie die zugrunde liegenden Daten korrekt ermittelt, unverändert und verbindlich sind. Ein Sicherheitsmanagement, das auf der Grundlage eines ungesicherten Managementsystem basiert, ist ein Widerspruch in sich. Das heißt, dass in keinem Funktionsbereich, in dem Mobile Agenten zum Einsatz kommen sollen, auf die Sicherheitsarchitektur verzichtet werden kann. Jedes System, insbesondere ein solch komplexes System wie ein Managementsystem es darstellt, ist immer nur so sicher wie sein schwächstes Glied.

7.3 Offene Fragestellungen

Die Fragen, die in dieser Arbeit offen bleiben mussten, kommen im Wesentlichen aus dem Management–Bereich sowie aus verwandten Gebieten.

Management–Fragestellungen

- Der Aufbau eines domänenübergreifenden Managementsystems, in dem alle beteiligten Organisationen auch *aktiv* am Managementprozess beteiligt werden können und nicht nur passiven Zugriff auf eingeschränkte Datenbestände eines mehr oder weniger zentralisierten Managementsystems erhalten, ist ein bisher kaum untersuchtes und infolgedessen ungelöstes Problem. Diese Arbeit liefert einen ersten Ansatz für solche Systeme, indem es die Sicherheit für domänenübergreifende Systeme auf Basis Mobiler Agenten erhöht. Im vorhergehenden Abschnitt wurde bereits erwähnt, dass der Aufwand bei der Bildung einer übergreifenden

Domäne jedoch nicht unerheblich ist. Dies betrifft im Wesentlichen die semantische Integration bzw. die Abbildungsproblematik der von den verschiedenen Organisationen unabhängig voneinander entwickelten und verwalteten Informationen. Ein völlig neues Problem, das hier entsteht, ist die Integration verschiedener Rollenmodelle zu einem gemeinsamen übergreifenden Modell. Wie können bereits bestehende Rollenmodelle zu einem (Domänen–) übergreifenden Rollenmodell zusammengefasst werden? Wie können unterschiedliche Rechte syntaktisch und semantisch auf ein übergreifendes Autorisierungsmodell abgebildet werden? Ähnliche Fragen stellen sich beim Trust–Level Management. Wie lässt sich ein effizientes domänenübergreifendes Trust–Level–Management aufsetzen? Wie lassen sich Rollen, Rechte und Vertrauensbeziehungen aus einer Domäne auf semantisch gleiche Rollen, Rechte bzw. Vertrauenslevel der anderen Domäne abbilden?

- Das vorgeschlagene Trust–Level Management basiert ausschließlich auf der Identität der beteiligten Entitäten. Auch die Klassifikation von Vertrauen ist sehr einfach gehalten. Alle anderen Informationen, die mit in die Berechnung des Vertrauens eingehen, werden implizit an die Identität geknüpft und sind damit für die Beteiligten weder erkennbar noch nachvollziehbar. Wie kann ein domänenübergreifendes Trust–Level Management etabliert werden, das Informationen, wie z.B. über den Kontext, über den Anwendungsfall oder über die bisherige Historie der Entität formalisiert und dadurch explizit macht, und trotzdem effizient bleibt?
- Der Problemkreis der Delegation von Rechten, der in dieser Arbeit nur am Rande betrachtet wurde, umfasst einige sehr interessante, sowohl technische als auch organisatorische Fragestellungen. Hier wäre zu untersuchen, wie Rechte oder Teile von Rechten an andere Entitäten weitergegeben werden können. Wie kann diese Rechteweitergabe kontrolliert und ggf. wieder zurückgenommen werden? Wie lassen sich Gruppen von kooperierenden Mobilen Agenten realisieren, bei denen, für jedes Mitglied der Gruppe, das Prinzip der minimalen Rechte gilt? Lassen sich Verfahren finden, mit denen der Delegationsprozess selbst überwacht und gesichert werden kann?
- In dieser Arbeit wurde die Anforderung nach Ressourcenbeschränkung nur als „ja/nein"–Entscheidung im Rahmen der Zugriffskontrolle betrachtet. Insbesondere vor dem Hintergrund der zunehmenden Zahl von DoS und DDoS (Distributed denial of service) Angriffen sollten hier feingranulare Konzepte entwickelt werden. Wie kann bspw. für einen Mobilen Agenten die Rechenzeit, der Speicherverbrauch, die Anzahl der Kommunikationsverbindungen u.ä. beschränkt werden?

Verwandte Gebiete

Durch die Einführung von UMTS wurde ein Anstoß für neue Entwicklungen im Hinblick auf Netztechnologie– und Dienste–Integration gegeben. In den nächsten Jahren werden in diesem Umfeld zwei Probleme von besonderem

Interesse sein:

1. Die Entwicklung neuer, z.B. kontextsensitiver Dienste, deren Nutzung nicht an bisherigen Provider– oder technologiebedingten Grenzen endet. Das Ziel, welches erreicht werden soll, lässt sich sehr gut durch das Schlagwort von der Dienstnutzung „anywhere, anyhow and anytime“ beschreiben. Der Dienstnutzer kann seinen Dienst an jedem Ort, zu jeder Zeit und mit beliebigen Endgeräten nutzen.

2. Eng verbunden mit der Entwicklung neuer Dienste ist auch der Fragenkomplex der Dienstkomposition. Es wird nicht mehr nur die klassischen Individual– und Massendienste geben, sondern es wird sich eine Vielzahl von Diensten entwickeln, die zwischen diesen beiden Extremen liegen. Der Benutzer wird sich künftig spezielle Dienste aus Basisdiensten zusammensetzen.

   Resultierend aus dem UMTS Geschäftsmodell wird es auf der Anbieterseite eine Vielzahl von unterschiedlichen Unternehmen geben, die als Network Operator, Service–, Value–added Service– oder auch als Content–Provider auftreten und auf dem Markt konkurrieren. Trotz der Konkurrenzsituation müssen diese Unternehmen kooperieren, um für den individuellen Kunden spezifische Dienste, wie z.B. spezielle personalisierte Dienste, anbieten zu können.

Die Diensterbringung und das Management der Dienste und Infrastrukturen muss nicht nur organisatorische, sondern auch technische Domänengrenzen überschreiten. Dabei sollen die system–inhärenten Grenzen, z.B. zwischen Organisationen oder verschiedenen Technologiebereichen, für den Benutzer so transparent wie möglich überwunden werden.

Auf der anderen Seite werden die verschiedenen Endgeräte des Benutzers (wie z.B. Mobiltelefone, PDAs, mobile und stationäre Rechner) als Ausführungsplattform für die Dienste und Dienstbausteine der unterschiedlichsten Diensterbringer genutzt.

Viele Fragen, die in dieser Arbeit untersucht wurden, stellen sich auch im Umfeld der Mobilkommunikation. Allerdings ist das Umfeld sehr viel inhomogener und die Beziehungen zwischen den beteiligten Entitäten sind noch dynamischer. Eine Untersuchung, inwieweit domänenübergreifende Sicherheitskonzepte für Mobile Agenten hier angewendet werden können, um eine verteilte Dienste–Architektur oder ein domänenübergreifendes Service–Management für UMTS zu etablieren, ist sicherlich von großem Interesse.

ABKÜRZUNGEN

$A\{m\}$ digitale Signatur von m durch A, s. auch $A_s[m]$

$A \longrightarrow B : m$ A schickt Nachricht m an B

A_p öffentlicher Schlüssel von A (public key of A)

$A_p[m]$ Verschlüsselung der Nachricht m mit dem öffentlichen Schlüssel von A

A_s privater Schlüssel von A (secret key of A)

$A_s[m]$ digitale Signatur von m durch A

act Aktivierung einer Rolle

AS_0 Ausgangs–Agentensystem des Mobilen Agenten; Startpunkt der Reise

AS_i Agentensystem Nr. i auf der Reise des MA

C_A^{CA} von CA ausgestelltes Zertifikat für A

d, m, n Nachricht, Daten

D_i von AS_i gekapselte (geschützte) Daten

d_i Daten auf Agentensystem AS_i erzeugt

$DN(x)$ Funktion zur Ermittlung des Identifikators (Distinguished Name) der Entität x

h_i von AS_i erzeugter Hash–Wert

H_m Hash–Wert H der Nachricht m

M Zugriffsmatrix

M' Zugriffsmatrix des Agentensystems

$MAC_s(m)$ MAC der Nachricht m, gesichert mit dem Schlüssel S

O Menge der Objekte

O_x Eigentümer (Owner) der Entität x

P Menge der Rechte

p Recht $p \in P$

pa Rechtezuweisung (Permission Assignment)

r_i Rolle $r_i \in R$

r_i Zufallszahl, generiert von AS_i

$rtv(Z)$ mittelbarer (recommended) Trust Level für Z

S Menge der Subjekte

S Schlüssel für ein symmetrisches Verschlüsselungsverfahren

s Subjekt $s \in S$

$S[m]$ Verschlüsselung von m mit dem Schlüssel S für ein symmetrisches Verfahren

sa Rollenmitgliedschaften eines Subjektes (Subject Assignment)

sig_m^A von A erstellte (digitale) Signatur der Nachricht m

$tv(B_i)$ Trust Level von B_i

$tv_p(Z)$ Trust Level für Z, abgeleitet über den Pfad p

Z_A Datenstruktur für das Zertifikat von A

ACID *A*tomicity, *C*onsistency, *I*solation, *D*urability

AH Authentication Header

AIAG Automotive Industry Action Group

ANX Automotive Network eXchange

API Application Programming Interface

AS Agentensystem

ATM Assynchronous Transfer Mode

B2B Business to Business Commerce

CA Certification Authority

CBC Cipher Block Chaining Mode

CEF Computing with encrypted Functions

CERT/CC Computer Emergency Response Team Coordination Center

CN Corporate Network

CORBA Common Object Request Broker Architecture

CRL Certification Revocation List

CSI Common Secure Interoperability Feature

CSM Customer Service Management

DAC Discretionary Access Control

DDoS Distributed denial of service

DES Data Encryption Standard

DNSSec Domain Name System Security Extensions

DNS Domain Name System

DN Distinguished Name

DoE Denial of execution

DoS Denial of service

ECB Electronic Codebook Mode

ESP Encapsulation Security Payload

FQHN Full Qualified Host Name

GUI Graphical User Interface

HMAC Hashed Message Authentication Code

HTTPS HTTP over SSL

HTTP Hypertext Transfer Protocol

IDL Interface Definition Language

IETF Internet Engineering Task Force

IIOP Internet Inter ORB Protocol

IOR Interoperable Object Reference

IPSec Sicherheitserweiterung für IP

ISDN Integrated Service Digital Network

JAAS Java Authentication and Authorization Service

jar Java Archive

JCA Java Cryptography Architecture

JCE Java Cryptography Extension

JVMPI Java Virtual Machine Profiler Interface

KAG–P1 Integritätsprotokoll Nr. 1 von Karjoth et al. [KAG 98]

LCA Local Certification Authority

MAC Mandatory Access Control

MAC Message Authentication Code

MASA Mobile Agent System Architecture

MASIF Mobile Agent System Interoperability Facility

MA Mobiler Agent

MbD Management by Delegation

MD5 Message Digest No. 5

MIB Management Information Base

MO Managementobjekt

OMG Object Management Group

ORB Object Request Broker

OSI Open Systems Interconnection

PDA Personal Digital Assistant

PDE Protokoll–Dateneinheit

PDU Protocol Data Unit

PKI Public Key Infrastructure

POP Point–of–Presence

PRAC Partial Result Authentication Code

QoS Quality of Service

RA Registration Authority

RBAK Role based Access Control

RM Referenzmodell

RRQ RecommendationRequest

RR RecommendationReply

RR Resource Record

SAP Service Access Point

SECIOP Secure IIOP

SLA Service Level Agreement

SNMP Simple Network Management Protocol

ssh Secure Shell

SSL Secure Socket Layer

TCB Trusted Computer Base

TLS Transport Layer Security

TTP Trusted Third Party

UMTS Universal Mobile Telecommunications System

URL Uniform Ressource Locator

VHE Virtual Home Environment

VPN virtuelles privates Netz

ABBILDUNGEN

TABELLEN

LITERATUR

[AABB 98] ABELSON, H., R. ANDERSON, S. M. BELLOVIN, J. BENALOH, M. BLAZE, W. DIFFIE, J. GILMORE, P. G. NEUMANN, R. L. RIVEST, J. I. SCHILLER und B. SCHNEIER: *The Risks of Key Recovery, Key Escrow & Trusted Third Party Encryption*. 1998, `http://www.crypto.com/key_study/`.

[AbHa 97] ABDUL-RAHMAN, A. und S. HAILES: *A Distributed Trust Model*. In: [NSPW 97].

[Aglets] *Aglets*, `http://www.trl.ibm.co.jp/aglets`.

[AIAG] *Automotive Industrie Action Group (AIAG)*, `http://www.aiag.org`.

[Albe 98] ALBERT, E.: *Bewertung von Java–basierten Client/Server–Architekturen im Global Information Services (GIS) der BMW AG*. Diplomarbeit, Ludwig-Maximilians-Universität München, Dezember 1998.

[Allg 98] ALLGEYER, P.: *Entwicklung und Implementierung einer Managementschnittstelle für einen PC-basierten Switch/Router*. Diplomarbeit, Technische Universität München, August 1998.

[AnKu 96] ANDERSON, R. J. und M. G. KUHN: *Tamper Resistance - a Cautionary Note*. In: *The Second USENIX Workshop on Electronic Commerce Proceedings*, Seiten 1–11, Oakland, California, November 1996. USENIX, `http://www.cl.cam.ac.uk/~mgk25/tamper.pdf`.

[AnKu 97] ANDERSON, R. J. und M. G. KUHN: *Low Cost Attacks on Tamper Resistant Devices*. In: CHRISTIANSON, B., B. CRISPO, M. LOMAS und M. ROE (Herausgeber): *Security Protocols, 5th International Workshop*, LNCS 1361, Seiten 7–9. Springer, 1997, `http://www.cl.cam.ac.uk/~mgk25/tamper2.pdf`.

[Ante 01] ANTERSBERGER, S.: *Evaluation und Anwendung des Rahmenrollenmodells der BMW AG mit Hilfe des Tivoli SecureWay Policy Director*. Diplomarbeit, Technische Universität München, Oktober 2001.

[ASA/MA 99] *First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents (ASA/MA 99)*, Palm Springs, California, October, 3–6 1999. IEEE.

[Avit 98] AVITABILE, M.: *An Examination of Requirements for Meta–Policies in Policy–Based Management*. Diplomarbeit, Technische Universität München, November 1998, `http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Diplomarbeiten/avit98/avit98.shtml`.

[BBCM 98] BREUGST, M., I. BUSSE, S. COVACI und T. MAGEDANZ: *Grasshopper - A Mobile Agent Platform for IN Based Service Environments*. In: *Proceedings of IEEE IN Workshop*, Seiten 279–290, Bordeaux, France, Mai 1998. .

[BeLa 73] BELL, D.E. und L. J. LAPADULA: *Secure Computer Systems: Mathematical Foundations and Model*. Technischer Bericht M74–244, MITRE Corp., Bedford, MA, USA, May 1973.

[BGS 98] BERKOVITS, S., J. D. GUTTMAN und V. SWARUP: *Authentication for Mobile Agents*. In: VIGNA, G. [Vign 98], Seiten 114–136.

[BHR 01] BRANDT, R., C. HÖRTNAGL und H. REISER: *Dynamically Adaptable Mobile Agents for Scaleable Software and Service Management*. Journal of Communications and Networks, 3(4):307–316, Dezember 2001.

[BHR 97] BAUMANN, J., F. HOHL und K. ROTHERMEL: *Mole - Concepts of a Mobile Agent System*. Fakultätsbericht 1997/15, Universität Stuttgart, 1997, `http://www.informatik.uni-stuttgart.de/cgi-bin/ncstrl_rep_view.pl?/inf/ftp/pub/library/ncstrl.ustuttgart_fi/TR-1997-15/TR-1997-15.bib`.

[BKR 98] BREDIN, J., D. KOTZ und D. RUS: *Market-based Resource Control for Mobile Agents*. In: *Autonomous Agents*, Seiten 197–204. ACM Press, Mai 1998, `ftp://ftp.cs.dartmouth.edu/pub/kotz/papers/bredin:market.ps.Z`.

[Blak 96] BLAKLEY, B.: *The Emperor's Old Armor*. In: NATIONAL SECURITY AGENCY (NSA) [NISSC 96], `http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper204/SECOND.PDF`.

[BoKa 01] BOHN, J. und G. KARJOTH: *Sicherheitsdienste für mobile Agentenanwendungen*. In: KILLAT, U. und W. LAMMERSDORF (Herausgeber): *Kommunikation in verteilten Sytemen (KiVS)*, Hamburg, Februar 2001. Gesellschaft für Informatik (GI), Springer.

[BPW 98] BIESZCZAD, A., B. PAGUREK und T. WHITE: *Mobile Agents for Network Management*. IEEE Communication Surveys, 1(1), 1998, `http://dsp.jpl.nasa.gov/members/payman/swarm/bieszczad98-comsoc.pdf`.

[Bran 01] BRANDT, R.: *Dynamic Adaptation of Mobile Code in Heterogeneous Environments*. Diplomarbeit, Technische Universität München, Februar 2001, `http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Diplomarbeiten/bran01/bran01.shtml`.

[Bran 99] BRANDT, R.: *Interoperabilität von CORBA ORBs in Mobilen Agentensystemen*. Systementwicklungsprojekt, Technische Universität München, Mai 1999, `http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Fopras/bran99/bran99.shtml`.

[BrNa 89] BREWER, D. F. C. und M. J. NASH: *The Chinese Wall Security Policy*. In: *Proceedings 1989 IEEE Symposium on Security and Privacy*, Seiten 206–214. IEEE, 1989.

[BrRe 01] BRANDT, R. und H. REISER: *Dynamic Adaptation of Mobile Agents in Heterogenous Environments*. In: PICCO, G. P. (Herausgeber): *Mobile Agents: Fifth International Conference, MA 2001, Proceedings*, Nummer 2230 in *Lecture Notes in Computer Science (LNCS)*, Seiten 70–87, Atlanta, Georgia, USA, Dezember 2001. Springer, `http://www.nm.informatik.uni-muenchen.de/Literatur/MNMPub/Publikationen/brre01/brre01.shtml`.

[BSJ 97] BERTINO, E., P. SAMARATI und S. JAJODIA: *An Extended Authorization Model for Relational Databases*. IEEE Transactions on Knowledge and Data Engineering, 9(1):85–101, 1997.

[cameleon] *The European ACTS Research Project cameleon*, `http://www.dfv.rwth-aachen.de/project/cameleon/cameleon.html`.

[CERT] *Computer Emergency Response Team Coordination Center (CERT/CC)*, `http://www.cert.org`.

[CFSD 90] CASE, J. D., M. FEDOR, M. L. SCHOFFSTALL und C. DAVIN: *RFC 1157: Simple Network Management Protocol (SNMP)*. RFC, IETF, Mai 1990, `ftp://ftp.isi.edu/in-notes/rfc1157.txt`.

[CGHL 95] CHESS, D., B. GROSOF, C. HARRISON, D. LEVINE, C. PARRIS und G. TSUDIK: *Itinerant Agents for Mobile Computing*. IBM Research Report RC 20010, IBM Research Division, 1995.

[Chen 00] CHEN, Z.: *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*. Addison Wesley, Juni 2000.

[CMRW 93] CASE, J., K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *RFC 1442: Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)*. RFC, IETF, April 1993, `ftp://ftp.isi.edu/in-notes/rfc1442.txt`.

[Coeh 98] COEHN, C.: *Integriertes Management verteilter Email-Systeme auf der Basis flexibler Managementagenten*. Diplomarbeit, Ludwig-Maximilians-Universität München, August 1998, `http://www.nm.informatik.uni-muenchen.de/common/Literatur/MNMPub/Diplomarbeiten/coeh98/coeh98.shtml`.

[Concordia] *Concordia*, `http://www.merl.com/projects/concordia/`.

[DAgents] *D'Agents*, `http://agent.cs.dartmouth.edu`.

[DeFr 89] DESMEDT, Y. und Y. FRANKEL: *Threshold cryptosystems*. In: BRASSARD, G. (Herausgeber): *Advances in Cryptology — CRYPTO '89*, Nummer 435 in *Lecture Notes in Computer Science (LNCS)*, Seiten 307 — 315. Springer, 1989.

[DeFr 91] DESMEDT, F. und Y. FRANKEL: *Shared generation of authenticators and signatures*. In: FEIGENBAUM, J. (Herausgeber): *Advances in Cryptology — CRYPTO '91*, Nummer 576 in *Lecture Notes in Computer Science (LNCS)*, Seiten 457 — 469. Springer, 1991.

[Demm 98] DEMMEL, S.: *Implementierung eines portierbaren Java-DHCP-Servers mit einer CORBA-Managementschnittstelle*. Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, September 1998.

[Demm 99] DEMMEL, S.: *Design und prototypische Implementierung eines policybasierten Dienstmanagements für das Nomadic Computing*. Diplomarbeit, Ludwig-Maximilians-Universität München, November 1999.

[Denn 76] DENNING, D. E.: *A Lattice Model of Secure Information Flow*. Communications of the ACM, 19(5):236–243, Mai 1976, `http://www.cs.duke.edu/~geha/cps296/pdf/lattice.pdf`.

[DiAl 99] DIERKS, T. und C. ALLEN: *RFC 2246: The TLS Protocol Version 1.0*. RFC, IETF, Januar 1999, `ftp://ftp.isi.edu/in-notes/rfc2246.txt`.

[EAC 98] EDJLALI, G., A. ACHARYA und V. CHAUDHARY: *History-based Access Control for Mobile Code*. In: *Proc. of the 5th ACM Conf. on Computer and Communications Security*. ACM, 1998, `http://www.cs.ucsb.edu/~acha/publications/ccs98-submitted.html`.

[EaKa 97] EASTLAKE, D. und C. KAUFMAN: *RFC 2065: Domain Name System Security Extensions*. RFC, IETF, Januar 1997, `ftp://ftp.isi.edu/in-notes/rfc2065.txt`.

[EC] *Eurocontrol*, `http://www.eurocontrol.be`.

[Ecke 01] ECKERT, C.: *IT–Sicherheit: Konzepte — Verfahren — Protokolle*. Oldenbourg, München, 2001.

[Ecke 98] ECKERT, C.: *Sichere, verteilte Systeme — Konzepte, Modelle und Systemarchitekturen*. Habilitationsschrift, Technische Universität München, November 1998.

[ECOOP 98] DEMEYER, S. und J. BOSCH (Herausgeber): *Object-Oriented Technology. ECOOP'98 Workshop Reader*, Nummer 1543 in *Lecture Notes in Computer Science (LNCS)*, Berlin, Heidelberg, Juli 1998. Springer.

[EEC 00] *Annual Report 2000*. Report, Eurocontrol Experimental Centre (EEC), 2000, `http://www.eurocontrol.fr/public/annual_report/2000/eecrap2000.pdf`.

[EEC 01] *Innovative Slot Allocation: an Overview*. EEC Note 10, Eurocontrol Experimental Centre (EEC), Bretigny–sur–Orge, France, Mai 2001, http://www.eurocontrol.fr/public/reports/eecnotes/2001/10.htm.

[EEC 01a] *A Simplified Approach to Conflict Probability Estimation*. EEC Note 11, Eurocontrol Experimental Centre (EEC), Bretigny–sur–Orge, France, Mai 2001, http://www.eurocontrol.fr/public/reports/eecnotes/2001/11.htm.

[Ense 01a] ENSEL, C.: *New Approach for Automated Generation of Service Dependency Models*. In: *Network Management as a Strategy for Evolution and Development; Second Latin American Network Operation and Management Symposium (LANOMS 2001)*, Belo Horizonte, Brazil, August 2001. IEEE, http://www.dcc.ufmg.br/lanoms/.

[Ense 01b] ENSEL, C.: *A Scalable Approach to Automated Service Dependency Modeling in Heterogeneous Environments*. In: *5th International Enterprise Distributed Object Computing Conference (EDOC '01)*, Seattle, USA, September 2001. IEEE, IEEE Publishing, http://edoc.doc.ic.ac.uk/.

[Essi 97] ESSIN, DANIEL J.: *Patterns of Trust and Policy*. In: [NSPW 97].

[Fack 00] FACKELMANN, D.: *IPSec Integration in einer Linux Umgebung*. Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, August 2000, http://www.nm.informatik.uni-muenchen.de/common/Literatur/MNMPub/Fopras/fack00/fack00.shtml.

[Fack 01] FACKELMANN, D.: *Rechtekonzept für die Mobile Agent System Architecture*. Diplomarbeit, Ludwig-Maximilians-Universität München, April 2001.

[FeKu 92] FERRAIOLO, D. und R. KUHN: *Role–Based Access Controls*. In: *15th NIST-NCSC National Computer Security Conference*, Seiten 554–563, Baltimore, MD, October 13-16 1992. , http://csrc.nist.gov/rbac/.

[FGS 96] FARMER, W. M., J. D. GUTTMAN und V. SWARUP: *Security for Mobile Agents: Issues and Requirements*. In: NATIONAL SECURITY AGENCY (NSA) [NISSC 96], http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper033/SWARUP96.PDF.

[Flan 97] FLANAGAN, D.: *Java in a Nutshell*. O'Reilly, Zweite Auflage, Mai 1997, http://www.oreilly.com/catalog/javanut2/.

[Funf 98] FÜNFROCKEN, S.: *Transparent Migration of Java–Based Mobile Agents*. In: ROTHERMEL, K. und F. HOHL [RoHo 98].

[Gerb 99] GERBER, S.: *Entwicklung einer Benutzerschnittstelle mit Java/CORBA für die Mobile Agent System Architecture (MASA)*. Systementwicklungsprojekt, Technische Universität München, April 1999, http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Fopras/gerb99/gerb99.shtml.

[GHR 99] GRUSCHKE, B., S. HEILBRONNER und H. REISER: *Mobile Agent System Architecture — Eine Plattform für flexibles IT–Management*. Technischer Bericht 9902, Ludwig-Maximilians-Universität München, Institut für Informatik, München, August 1999, http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Publikationen/ghr99/ghr99.shtml.

[Gigl 00] GIGL, J. G.: *Implementierung der MAFFinder–Schnittstelle für die Mobile Agent System Architecture*. Studienarbeit für das Aufbaustudium Informatik, Technische Universität München, Juli 2000, http://www.nm.informatik.uni-muenchen.de/common/Literatur/MNMPub/Fopras/gigl00/gigl00.shtml.

[GJSB 00] GOSLING, J., B. JOY, G. STEELE und G. BRACHA: *The Java Language Specification*. Addison–Wesley, Reading, Mass., Zweite Auflage, Juni 2000, http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html.

[GMPS 97] GONG, L., M. MUELLER, H. PRAFULLCHANDRA und R. SCHEMERS: *Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java(TM) Development Kit 1.2*. In: *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, California, USA, Dezember 1997. USENIX, http://java.sun.com/security/usenix-jdk12security.ps.

[Gold 96] GOLDSZMIDT, G. S.: *Distributed Management by Delegation*. Doktorarbeit, Graduate School of Arts and Sciences, Columbia University, 1996.

[Goli 99] GOLIAS, D.: *Entwicklung und Implementierung einer Benutzerschnittstelle für Policy-Managementagenten*. Fortgeschrittenenpraktikum, Technische Universität München, Oktober 1999.

[Gong 97] GONG, L.: *Surviveable Mobile Code is Hard to Build*. In: *Foundations for Secure Mobile Code Workshop*, 1997, http://www.cs.nps.navy.mil/research/languages/statements/.

[Gong 98] GONG, L.: *Java 2 Plattform Security Architecture*. Technischer Bericht Version 1.0, Sun Microsystems, Inc., Palo Alto, CA, October 2 1998, ftp://ftp.java.sun.com/docs/j2se1.3/security-spec.pdf.

[Gong 98a] GONG, L.: *Secure Java Class Loading*. IEEE Internet Computing, 2(6):56–61, November 1998.

[GoSc 98] GONG, L. und R. SCHEMERS: *Signing, Sealing, and Guarding Java Objects*. In: VIGNA, G. [Vign 98], Seiten 206–216, http://java.sun.com/people/gong/papers/lncs98.ps.gz.

[GoYe 95] GOLDSZMIT, G. und Y. YEMINI: *Distributed Managment by Delegation*. In: *Proceedings of the 15th International Conference on Distributed Computing Systems*, Juni 1995.

[GoYe 98] GOLDSZMIDT, G. und Y. YEMINI: *Delegated Agents for Network Management*. IEEE Communications Magazine, 36(3):66–70, März 1998.

[GPSN 96] GHODOSI, H., J. PIEPRZYK und R. SAFAVI-NAINI: *Dynamic Threshold Cryptosystems: A New Scheme in Group Oriented Cryptography*. In: PRIBYL, J. (Herausgeber): *International Conference on the Theory and Applications of Cryptology (PRAGOCRYPT '96)*, Seiten 370 — 379, Prague, Czech Republic, 1996. CTU Publishing.

[Grasshopper] *Grasshopper*, `http://www.grasshopper.de`.

[Gray 95] GRAY, R. S.: *Agent Tcl: Alpha Release 1.1*, 1995, `ftp://agent.cs.dartmouth.edu/pub/agents/doc.1.1.ps.gz`.

[Gres 99] GRESSER, C.: *Konzept für den Betrieb eines Trust Centers der BMW Bank*. Diplomarbeit, Ludwig-Maximilians-Universität München, Dezember 1999.

[Grus 99] GRUSCHKE, B.: *Entwurf eines Eventkorrelators mit Abhängigkeitsgraphen*. Dissertation, Ludwig-Maximilians-Universität München, November 1999.

[GSWA 01] GROLIK, S., T. STOCKHEIM, O. WENDT, S. ALBAYRAK und S. FRICKE: *Dispositive Supply–Web–Koordination durch Multiagentensysteme*. Wirtschaftsinformatik: Agententechnologie — Kooperierende Softwareagenten im betrieblichen Einsatz, 43(2):143 — 156, April 2002.

[Gypsy] *Gypsy*, `http://www.infosys.tuwien.ac.at/Gypsy`.

[HAN 99] HEGERING, H.-G., S. ABECK und B. NEUMAIR: *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999. 651 p.

[HAN 99a] HEGERING, H.-G., S. ABECK und B. NEUMAIR: *Integriertes Management vernetzter Systeme — Konzepte, Architekturen und deren betrieblicher Einsatz*. dpunkt-Verlag, ISBN 3-932588-16-9, 1999, `http://www.dpunkt.de/produkte/management.html`. 607 S.

[HaRe 00] HAUCK, R. und H. REISER: *Monitoring Quality of Service across Organizational Boundaries*. In: LINNHOFF-POPIEN, C. und H.-G. HEGERING (Herausgeber): *Trends in Distributed Systems: Towards a Universal Service Market. Proceedings of the third International IFIP/GI Working Conference, USM 2000*, Nummer 1890 in *Lecture Notes in Computer Science (LNCS)*, Munich, Germany, September 2000. Springer, `http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Publikationen/hare00/hare00.shtml`.

[HaRe 99] HAUCK, R. und H. REISER: *Monitoring of Service Level Agreements with flexible and extensible Agents*. In: *Workshop of the OpenView University Association (OVUA'99)*, Bologna, Italy, Juni 1999. HPOVUA, http://www.hpovua.org/PUBLICATIONS/PROCEEDINGS/6_HPOVUAWS/Papers/hauck_reiser.pdf.

[Hauc 01] HAUCK, R.: *Architektur für die Automation der Managementinstrumentierung bausteinbasierter Anwendungen*. Dissertation, Ludwig-Maximilians-Universität München, Juli 2001, http://www.nm.informatik.uni-muenchen.de/Literatur/MNMPub/Publikationen/hauc01/hauc01.shtml.

[HCK 95] HARRISON, C. G., D. M. CHESS und A. KERSHENBAUM: *Mobile Agents: Are they a good idea?* Research Report, IBM Research Division, 1995.

[Heil 00] HEILBRONNER, S.: *Konzeption einer Architektur für das integrierte Management der Ressourcennutzung nomadischer Systeme in Datennetzen*. Dissertation, Ludwig-Maximilians-Universität München, Januar 2000.

[HFPS 99] HOUSLEY, R., W. FORD, W. POLK und D. SOLO: *RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile*. RFC, IETF, Januar 1999, ftp://ftp.isi.edu/in-notes/rfc2459.txt.

[HGF 98] HARTMANN, J., C. GOERG und P. FARJAMI: *Agent Technology for the UMTS VHE Concept*. In: *Proceeding of the First ACM International Workshop on Wireless Mobile Multimedia in Conjunction with ACM/IEEE MobiCom'98*, Seiten 203–208, University of North Texas, Dallas, USA, Oktober 1998. ACM.

[Hoar 69] HOARE, C.A.R.: *An axiomatic basis for computer programming*. Communications of the ACM, 12:576–583, 1969.

[Hohl 98] HOHL, F.: *Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts*. In: VIGNA, G. [Vign 98], Seiten 92–113.

[Hojn 99] HOJNACKI, M.: *Einsatz des Java Dynamic Management Kit (JDMK) zur Antwortzeitüberwachung bei der DeTeSystem*. Diplomarbeit, Technische Universität München, Mai 1999, http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Diplomarbeiten/hojn99/hojn99.shtml.

[HoLo 98] HOWARD, J. D. und T. A. LONGSTAFF: *A Common Language for Computer Security Incidents*. Technischer Bericht SAND98–8667, Sandia National Laboratories, Albuquerque, New Mexico, Oktober 1998, http://www.cert.org/research/taxonomy_988667.pdf.

[i8xx] INTEL CORPORATION: *Intel 82802 Firmware Hub: Random Number Generator*. Programmer's Reference Manual 298029–001, Intel

Corporation, Dezember 1991, `http://developer.intel.com/design/chipsets/manuals/298029.pdf`.

[IAIK-JCE] TECHNISCHE UNIVERSITÄT GRAZ: *IAIK Java Cryptography Extension (IAIK-JCE)*, `http://jcewww.iaik.tu-graz.ac.at/products/jce/index.php`.

[IRNG 99] INTEL PLATFORM SECURITY DEVISION: *The Intel Random Number Generator*. Technical Paper, Intel Corporation, 1999, `http://developer.intel.com/design/security/rng/techbrief.htm`.

[iSaSiLk] TECHNISCHE UNIVERSITÄT GRAZ: *IAIK-iSaSiLk; A Java Implementation of the SSLv3 protocol*, `http://jcewww.iaik.at/products/isasilk/index.php`.

[ISO 10181-1] *Information Technology – Open Systems Interconnection – Security Frameworks in Open Systems – Overview*. IS 10181-1, International Organization for Standardization and International Electrotechnical Committee, November 1995.

[ISO 7498] *Information Processing Systems – Open Systems Interconnection – Basic Reference Model*. IS 7498, International Organization for Standardization and International Electrotechnical Committee, 1984.

[ITU- 93] ITU–T: *X.500 – Information technology – Open systems Interconnection – The Directory: Overview of concepts, models and services*. ITU–T Recommendation, International Telecommunication Union, November 1993.

[J2ME] SUN MICROSYSTEMS, INC.: *Java 2 Platform, Micro Edition (J2ME Platform)*. Technischer Bericht, Sun Microsystems, Inc., `http://java.sun.com/j2me/`.

[JAAS] SUN MICROSYSTEMS, INC.: *Java Authentication and Authorization Service (JAAS) — Reference Guide for Java 2 SDK, Standard Edition, v 1.4*, `http://java.sun.com/j2se/1.4/docs/guide/security/jaas/JAASRefGuide.html`.

[Java2] SUN MICROSYSTEMS, INC.: *Java 2 SDK, Standard Edition Documentation*, `http://java.sun.com/products/jdk/1.2/docs/index.html`.

[JCA] SUN MICROSYSTEMS, INC.: *Java Cryptography Architecture — API Specifications & Reference*. Technischer Bericht, Sun Microsystems, Inc., 1999, `http://java.sun.com/products/jdk/1.2/docs/guide/security/CryptoSpec.html`.

[JCE-imp] SUN MICROSYSTEMS, INC.: *How to Implement a Provider for the Java Cryptography Architecture*. Technischer Bericht, Sun Microsystems, Inc., 1998, `http://java.sun.com/products/jdk/1.2/docs/guide/security/HowToImplAProvider.html`.

[JCE] SUN MICROSYSTEMS, INC.: *JavaTM Cryptography Extension 1.2.1 — API Specification & Reference*. Technischer Bericht, Sun Microsystems, Inc., 2000, `http://java.sun.com/products/jce/doc/guide/API_users_guide.html`.

[JeWo 98] JENNINGS, N. und M. WOOLRIDGE (Herausgeber): *Agent Technology*. Springer, 1998.

[Joha 98] JOHANSEN, D.: *Mobile Agent Applicability*. In: ROTHERMEL, K. und F. HOHL [RoHo 98].

[JRS 95] JOHANSEN, D., R. VAN RENESSE und F. B. SCHNEIDER: *Operating System Support for Mobile Agents*. In: *Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems*, Orcas Island, Wa, USA, Mai 1995. .

[JVMPI] SUN MICROSYSTEMS, INC.: *Java Virtual Machine Profiler Interface (JVMPI)*, `http://java.sun.com/products/jdk/1.2/docs/guide/jvmpi/`.

[KAG 98] KARJOTH, G., N. ASOKAN und C. GÜLCÜ: *Protecting the Computation Results of Free-Roaming Agents*. In: ROTHERMEL, K. und F. HOHL [RoHo 98], Seiten 195–207.

[KaSt 00] KAVASSERI, R. und B. STEWART: *RFC 2982: Distributed Management Expression MIB*. RFC, IETF, Oktober 2000, `ftp://ftp.isi.edu/in-notes/rfc2982.txt`.

[KaVo 98] KASSAB, L. L. und J. VOAS: *Agent Trustworthiness*. In: DEMEYER, S. und J. BOSCH [ECOOP 98], Seiten 121–133, `http://cui.unige.ch/~ecoopws/ws98/paperList.html`.

[KeAt 98a] KENT, S. und R. ATKINSON: *RFC 2401: Security Architecture for the Internet Protocol*. RFC, IETF, November 1998, `ftp://ftp.isi.edu/in-notes/rfc2401.txt`.

[KeAt 98b] KENT, S. und R. ATKINSON: *RFC 2402: IP Authentication Header*. RFC, IETF, November 1998, `ftp://ftp.isi.edu/in-notes/rfc2402.txt`.

[KeAt 98c] KENT, S. und R. ATKINSON: *RFC 2406: IP Encapsulating Security Payload (ESP)*. RFC, IETF, November 1998, `ftp://ftp.isi.edu/in-notes/rfc2406.txt`.

[Kell 98] KELLER, A.: *CORBA-basiertes Enterprise Management: Interoperabilität und Managementinstrumentierung verteilter kooperativer Managementsysteme in heterogener Umgebung*. Dissertation, Technische Universität München, Dezember 1998.

[Kemp 98] KEMPTER, B.: *Entwurf eines Java/CORBA-basierten Mobilen Agenten*. Diplomarbeit, Technische Universität München, August 1998.

[KeRe 99] KELLER, A. und H. REISER: *Dynamic Management of Internet Telephony Servers: A Case Study based on JavaBeans and JDMK*. In: *Proceedings Third International Enterprise Distributed Object Computing Conference (EDOC '99)*, Seiten 135–146, Mannheim, Germany, Sept., 27-30 1999. IEEE Publishing, `http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Publikationen/kere99/kere99.shtml`.

[Klus 99] KLUSCH, M. (Herausgeber): *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*. Springer, Berlin, Heidelberg, 1999.

[Knoe 99] KNÖCHLEIN, H.: *Management eines Internet Telefonie Servers mittels JDMK*. Diplomarbeit, Technische Universität München, Februar 1999, `http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Diplomarbeiten/knoe99/knoe99.shtml`.

[KRRV01] KEMPTER, B., H. REISER, H. ROELLE und G. VOGT: *Implementierung eines MASIF konformen Agentensystems — Die Mobile Agent System Architecture (MASA)*. PIK – Praxis der Informationsverarbeitung und Kommunikation, 24(3):141–148, September 2001.

[KRS 99] KOVACS, E., K. RÖHRLE und B. SCHIEMANN: *Adaptive Mobile Access to Context–aware Services*. In: [ASA/MA 99], Seiten 190–201.

[Küp 01] KÜPPER, A.: *Nomadic Communication in Converging Networks*. Dissertation, RWTH Aachen, VDE Verlag, Berlin, Offenbach, 2001.

[KuPa 98] KÜPPER, A. und A. S. PARK: *Stationary vs. Mobile User Agents in Future Mobile Telecommunication Networks*. In: ROTHERMEL, K. und F. HOHL [RoHo 98], Seiten 112–123.

[Lang 01] LANGER, M.: *Konzeption und Anwendung einer Customer Service Management Architektur*. Dissertation, Technische Universität München, März 2001, `http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2001/langer.html`.

[LaOs 98] LANGE, D. und M. OSHIMA: *Programming and Deploying Java[tm] Mobile Agents with Aglets[tm]*. Addison Wesley Longman, 1998.

[LaPa 96a] LAPADULA, L.: *Secure Computer Systems: Mathematical Foundations. An electronic reconstruction by Len LaPadula of the original MITRE Technical Report 2547, Volume I titled "Secure Computer Systems: Mathematical Foundations" by Leonhard J. LaPadula and D. Elliott Bell dated 1 March 1973*. November 1996, `http://www.mitre.org/resources/centers/infosec/secure_computers/secure_comp_math.pdf`.

[LaPa 96b] LAPADULA, L.: *Secure Computer Systems: A Mathematical Model. An electronic reconstruction by Len LaPadula of the original MITRE Technical Report 2547, Volume II titled "Secure Computer Systems: Mathematical Foundations" by Leonhard J.*

LaPadula and D. Elliott Bell dated 31 May 1973. November 1996, http://www.mitre.org/resources/centers/infosec/secure_computers/secure_comp.pdf.

[LeSc 99a] LEVI, D. und J. SCHOENWAELDER: *RFC 2591: Definitions of Managed Objects for Scheduling Management Operations*. RFC, IETF, Mai 1999, ftp://ftp.isi.edu/in-notes/rfc2591.txt.

[LeSc 99b] LEVI, D. und J. SCHOENWAELDER: *RFC 2592: Definitions of Managed Objects for the Delegation of Management Scripts*. RFC, IETF, Mai 1999, ftp://ftp.isi.edu/in-notes/rfc2592.txt.

[LGKN 99] LAI, C., L. GONG, L. KOVED, A. NADALIN und R. SCHEMERS: *User Authentication and Authorization in the Java Platform*. In: *15th Annual Computer Security Applications Conference*, Phoenix, AZ, USA, Dezember 1999. IEEE Computer Society.

[LiHe 00] LINNHOFF-POPIEN, C. und H.-G. HEGERING (Herausgeber): *Trends in Distributed Systems: Towards a Universal Service Market. Proceedings of the third International IFIP/GI Working Conference, USM 2000*, Nummer 1890 in *Lecture Notes in Computer Science*, Munich, Germany, September 2000. Ludwig-Maximilians-University, Springer.

[LiYe 99] LINDHOLM, T. und F. YELLIN: *The JavaTM Virtual Machine Specification*. Addison–Wesley, Reading, Mass., Zweite Auflage, April 1999, http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html.

[Lore 01] LORENZ, B.: *Erweiterung der MASA-Umgebung um Accounting-Funktionalität*. Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, August 2001, http://www.nm.informatik.uni-muenchen.de/common/Literatur/MNMPub/Fopras/lore01/lore01.shtml.

[MASIF] *Mobile Agent System Interoperability Facilities Specification*. OMG TC Document orbos/98-03-09, Object Management Group, März 1998.

[Maul 99] MAUL, O.: *Implementierung des CORBA Notification Service in der MASA-Java-Agentenumgebung*. Fortgeschrittenenpraktikum, Technische Universität München, Dezember 1999.

[Mead 97] MEADOWS, C.: *Detecting Attacks on Mobile Agents*. In: *Foundations for Secure Mobile Code Workshop*, 1997, http://www.cs.nps.navy.mil/research/languages/statements/.

[Merz 99] MERZ, M.: *Electronic commerce: Marktmodelle, Anwendungen und Technologien*. dpunkt–Verlag, Heidelberg, 1999.

[MITRE] *The MITRE Corporation*, http://www.mitre.org.

[Mits 98] MITSUBISHI ELECTRIC ITA: *Mobile Agent Computing – A White Paper*. Technischer Bericht, Mitsubishi Electric ITA, Januar 1998.

[Mole] *Mole*, `http://mole.informatik.uni-stuttgart.de`.

[Moun 97] MOUNTZIA, M.-A.: *Flexible Agents in Integrated Network and Systems Management*. Dissertation, Technische Universität München, Dezember 1997.

[Moun 97a] MOUNTZIA, M.-A.: *A Distributed Management Approach based on Flexible Agents*. Journal of Interoperable Communication Networks, Special Issue on Telecommunication Service Engineering, 1997.

[NeLe 98b] NECULA, G. C. und P. LEE: *Safe, Untrusted Agents Using Proof–Carrying Code*. In: VIGNA, G. [Vign 98], Seiten 61–91.

[Nerb 01] NERB, M.: *Customer Service Management als Basis für interorganisationales Dienstmanagement*. Dissertation, Technische Universität München, März 2001, `http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2001/nerb.html`.

[NiPa 99] NIKANDER, P. und J. PARTANEN: *Distributed Policy Management for JDK 1.2*. In: *Proceedings of the 1999 Network and Distributed Systems Security Symposium (NDSS 99)*, San Diego, California, Februar 1999. Internet Society, `http://www.isoc.org/ndss99/proceedings/`.

[NISSC 96] NATIONAL SECURITY AGENCY (NSA) (Herausgeber): *19th National Information Systems Security Conference Proceedings*, Oktober 1996, `http://csrc.nist.gov/nissc/1996/papers/NISSC96/`.

[NSPW 97] ACM SIGSAC: *Proceedings of the Workshop on New Security Paradigms (NSPW'97)*. Association for Computing Machinery, 1997.

[Oaks 98] OAKS, S.: *Java Security*. The Java Series. O'Reilly, 1998.

[OaWo 99] OAKS, S. und H. WONG: *Java Threads*. O'Reilly, Sebastopol, CA, USA, Zweite Auflage, 1999.

[OKO 98] OSHIMA, M., G. KARJOTH und K. ONO: *Aglets Specification 1.1 Draft*. Technischer Bericht Draft 0.65, IBM Research Labs Tokio, September 1998, `http://www.trl.ibm.co.jp/aglets/spec11.html`.

[OMG] *Object Management Group*, `http://www.omg.org`.

[OMG 00-06-41] *A Discussion of the Object Management Architecture*. OMG Specification formal/00-06-41, Object Management Group, Januar 1997, `http://www.omg.org/cgi-bin/doc?formal/00-06-41`.

[OMG 01-02-33] *The Common Object Request Broker: Architecture and Specification, v2.4.2*. OMG Specification formal/01-02-33, Object Management Group, Februar 2001, `http://cgi.omg.org/cgi-bin/doc?formal/01-02-33`.

[OMG 01-09-34] *The Common Object Request Broker: Architecture and Specification, v2.5*. OMG Specification formal/01-09-34, Object Management Group, September 2001, http://www.omg.org/cgi-bin/doc?formal/01-09-34.

[OMG 01-09-34a] *The Common Object Request Broker: Architecture and Specification*, Kapitel 23: MinimumCORBA, Seiten 23.1–23.30. Object Management Group, 2001, http://www.omg.org/cgi-bin/doc?formal/01-09-34.

[OMG 2001-03-08] *Security Service Specification — Version 1.7*. OMG Specification formal/2001-03-08, Object Management Group, März 2001, http://www.omg.org/technology/documents/formal/security_service.htm.

[OOC] *Object Oriented Computing*, http://www.ooc.com.

[OpenSSL] *The OpenSSL Project*, http://www.openssl.org/.

[Pain 99] PAINTMAYER, T.: *Einbettung von Sicherheitsverfahren in ein offenes heterogenes Sicherheitsmanagement auf der Basis der OSI Management Architektur*. Dissertation, Ludwig-Maximilians-Universität München, Mai 1999.

[Perr 00] PERRY, T. S.: *Tracking weather's flight path*. IEEE Spectrum, 37(9):38 — 45, September 2000.

[Pets 01] PETSCH, M.: *Aktuelle Entwicklungsumgebungen für mobile Agenten und Multiagentensysteme*. Wirtschaftsinformatik: Agententechnologie — Kooperierende Softwareagenten im betrieblichen Einsatz, 43(2):175 — 182, April 2001.

[PhKa98] PHAM, A. und A. KARMOUCH: *Mobile Software Agents: An Overview*. IEEE Communications Magazine, 36(7):26–37, Juli 1998.

[PlWe 01] PLÖSCH, R. und R. WEINREICH: *Ein agentenbasierter Ansatz für die Ferndiagnose und –überwachung von Automatisierungssystemen*. Wirtschaftsinformatik: Agententechnologie — Kooperierende Softwareagenten im betrieblichen Einsatz, 43(2):167 —173, April 2001.

[Python] *Python Documentation*. WWW, http://www.python.org/doc.

[Radi 98] RADISIC, I.: *Konzeption eines policy-basierten Konfigurationsmanagements für nomadische Systeme in Intranets*. Diplomarbeit, Ludwig-Maximilians-Universität München, August 1998, http://www.nm.informatik.uni-muenchen.de/common/Literatur/MNMPub/Diplomarbeiten/radi98/radi98.shtml.

[Raep 98] RAEPPLE, MARTIN: *Sicherheitskonzepte für das Internet*. dpunkt–Verlag, Heidelberg, 1998.

[Reis 99] REISER, H.: *Einsatz des Java Dynamic Management Kit für flexible, dynamische Managementsysteme*. PIK – Praxis der Informationsverarbeitung und Kommunikation, 22(4):204–212, Dezember 1999, `http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Publikationen/reis99/reis99.shtml`.

[ReVo 00] REISER, H. und G. VOGT: *Threat Analysis and Security Architecture of Mobile Agent based Management Systems*. In: HONG, J. W. und R. WEIHMAYER (Herausgeber): *NOMS 2000 IEEE/IFIP Network Operations and Managment Symposium — The Networked Planet: Management Beyond 2000*, Seite 979, Honolulu, Hawaii, USA, April, 10–14 2000. IEEE, `http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Publikationen/revo00/revo00.shtml`.

[ReVo 00a] REISER, H. und G. VOGT: *Security Requirements for Management Systems using Mobile Agents*. In: TOHME, S. und M. ULEMA (Herausgeber): *Proceedings of the Fifth IEEE Symposium on Computers & Communications*, Seiten 160–165, Antibes - Juan Les Pins, France, Juli 2000. IEEE, `http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Publikationen/revo00a/revo00a.shtml`.

[Rieg 99] RIEGER, T.: *Sicherheitstechnische Einordnung und Bewertung der Extranets und Intranets der BMW AG*. Diplomarbeit, Technische Universität München, November 1999.

[RiSc 98] RIORDAN, J. und B. SCHNEIER: *Environmental Key Generation Towards Clueless Agents*. In: VIGNA, G. [Vign 98], Seiten 15–24.

[Rive 92] RIVEST, R.: *RFC 1321: The MD5 Message-Digest Algorithm*. RFC, IETF, April 1992, `ftp://ftp.isi.edu/in-notes/rfc1321.txt`.

[RJB 98] RUMBAUGH, J., I. JACOBSON und G. BOOCH: *Unified Modeling Language — Reference Manual*. Addison–Wesley, 1998.

[Roel 99a] RÖLLE, H.: *Authentisierung und Autorisierung für das Java/CORBA–Agentensystem MASA*. Diplomarbeit, Technische Universität München, August 1999, `http://www.nm.informatik.uni-muenchen.de/common/Literatur/MNMPub/Diplomarbeiten/roel99/roel99.shtml`.

[Roel 99b] RÖLLE, H.: *Prototypische Implementierung des CORBA Topology Service*. Fortgeschrittenenpraktikum, Technische Universität München, Februar 1999.

[RoHo 98] ROTHERMEL, K. und F. HOHL (Herausgeber): *Mobile Agents (MA '98)*, Band 1477 der Reihe *Lecture Notes in Computer Science (LNCS)*, Berlin; Heidelberg, 1998. Springer.

[RoPo 97] ROTHERMEL, K. und R. POPESCU-ZELETIN (Herausgeber): *Mobile Agents (MA '97)*, Band 1219 der Reihe *Lecture Notes in Computer Science (LNCS)*, Berlin, April 1997. Springer.

[Rose 96] ROSE, M. T.: *The Simple Book — An Introduction to Internet Management*. Prentice-Hall, überarbeitete zweite Auflage, 1996.

[Roth 98] ROTH, V.: *Secure Recording of Itineraries through Cooperating Agents*. In: DEMEYER, S. und J. BOSCH [ECOOP 98], `http://cui.unige.ch/~ecoopws/ws98/papers/vroth98c.ps`.

[Saca 01] SACA, V.: *Aufbau eines Testbeds für das IP Accounting und dessen Einbindung in die Mobile Agent System Architecture (MASA)*. Fortgeschrittenenpraktikum, Technische Universität München, 2001.

[SaSa 94] SANDHU, R. S. und P. SAMARATI: *Access Control: Principles and Practice*. IEEE Computer, 32(9):40–48, September 1994.

[SaSc 75] SALTZER, J. H. und M. D. SCHROEDER: *The Protection of Information in Computer Systems*. Proceedings of the IEEE, 63(9):1278–1308, September 1975, `http://web.mit.edu/Saltzer/www/publications/protection/`.

[SaTs 98] SANDER, T. und C. F. TSCHUDIN: *Protecting Mobile Agents Against Malicious Hosts*. In: VIGNA, G. [Vign 98], Seiten 44–60, `http://www.star-lab.com/sander/publications/MA-protect.ps`.

[SaTs 98a] SANDER, T. und C. F. TSCHUDIN: *On Software Protection Via Function Hiding*. In: *Proceedings of the 2nd International Workshop on Information Hiding*, 1998, `http://www.star-lab.com/sander/publications/hiding.ps`.

[SaTs 97] SANDER, T. und C. F. TSCHUDIN: *Towards Mobile Cryptography*. Technischer Bericht TR-97-049, International Computer Science Institute, Berkeley, California, November 1997, `http://www.star-lab.com/sander/publications/satschu.ps`.

[SCFY 96] SANDHU, R. S., E. J. COYNE, H. L. FEINSTEIN und C. E. YOUMAN: *Role–Based Access Control Models*. IEEE Computer, 29(2):38–47, Februar 1996.

[Scheme] STEELE, G.-L. und G.-J. SUSSMAN: *Scheme - A LISP dialect*, `http://www.swiss.ai.mit.edu/projects/scheme/index.html`.

[Schm 01] SCHMIDT, H.: *Entwurf von Service Level Agreements auf der Basis von Dienstprozessen*. Dissertation, Ludwig-Maximilians-Universität München, 2001. To be published.

[Schn 97] SCHNEIDER, F. B.: *Towards Fault–tolerant and Secure Agentry*. In: *11th International Workshop on Distributed Algorithms*, Nummer 1320 in *Lecture Notes in Computer Science (LNCS)*, Saarbrücken, September 1997. Springer, `http://ncstrl.cs.cornell.edu:80/Dienst/UI/1.0/Display/ncstrl.cornell/TR97-1636`.

[Schn 96] SCHNEIER, BRUCE: *Applied Cryptography*. Wiley & Sons, Zweite Auflage, 1996.

[Schr 99] SCHREIER, J.: *Evaluierung des Einsatzes von IPv6 und IPSEC bei der Händlervernetzung der BMW AG* . Diplomarbeit, Ludwig-Maximilians-Universität München, Februar 1999.

[Schw 00] SCHWINN, A.: *Keymanagement mit DNSSec*. Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, Juli 2000, `http://www.nm.informatik.uni-muenchen.de/common/Literatur/MNMPub/Fopras/schw00/schw00.shtml`.

[Schw 01] SCHWINN, A.: *Einsatz von IPSec bei der BMW AG*. Diplomarbeit, Ludwig-Maximilians-Universität München, Januar 2001.

[ScQu 99] SCHÖNWÄLDER, J. und J. QUITTEK: *Secure Management by Delegation within the Internet Management Framework*. In: SLOMAN, M., S. MAZUMDAR und E. LUPO (Herausgeber): *Integrated Network Management VI (IM'99)*, Seiten 687–700, Boston, MA, Mai 1999. IEEE Publishing.

[Sham 79] SHAMIR, A.: *How to Share a Secret*. Communications of the ACM (CACM), 22(11):612–613, November 1979.

[ShGe 98] SHOUP, V. und R. GENNARO: *Securing Threshold Cryptosystems against Chosen Ciphertext Attack*. In: NYBERG, K. (Herausgeber): *Advances in Cryptology — EUROCRYPT '98*, Nummer 1403 in *Lecture Notes in Computer Science (LNCS)*, Seiten 1–16, Helsinki, Finland, 1998. Springer, `http://www.research.ibm.com/security/tcc.ps`.

[Smit 97] SMITH, N.: *Stack Smashing Vulnerabilities in the UNIX Operating System*. Technischer Bericht, Southern Connecticut State University, 1997, `http://www.bronzesoft.org/docs/security/bufov/nate-buffer.pdf`.

[Sole 95] SOLEY, R. M. (Herausgeber): *Object Management Architecture Guide*. John Wiley & Sons, Inc., Dritte Auflage, Juni 1995.

[SSL 3.0] FREIER, A. O., P. KARLTON und P. C. KOCHER: *The SSL Protocol Version 3.0*. Internet Draft, IETF, November 1996, `http://home.netscape.com/eng/ssl3/draft302.txt`.

[Stal 97] STALLINGS, W.: *Operating Systems: Internals and Design Principals*. Prentice Hall, Dritte Auflage, 1997.

[Stal 98] STALLINGS, W.: *Cryptography and Network Security — Principles and Practice*. Prentice Hall, 1998.

[SuBu 01] SUNDERMEYER, K. und S. BUSSMANN: *Einführung der Agententechnologie in einem produzierenen Unternehmen — ein Erfahrungsbericht*. Wirtschaftsinformatik: Agententechnologie — Kooperierende Softwareagenten im betrieblichen Einsatz, 43(2):135 — 141, April 2002.

[Tacoma] *Tacoma*, `http://www.tacoma.cs.uit.no`.

[Tane 92] TANENBAUM, A. S.: *Modern Operating Systems*. Prentice Hall, Inc., 1992.

[Thae 00] THAENS, K.: *Entwicklung eines Discovery-Managementagenten*. Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, Mai 2000.

[TS 123 127] *Universal Mobile Telecommunications System (UMTS); Virtual Home Environment*. Technical Specification ETSI TS 123 127 V4.1.0, European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France, April 2001, `http://www.etsi.org`.

[TS 22.121] *Virtual Home Environment (Release 5)*. Technical Specification 3GPP TS 22.121 V5.1.0, 3rd Generation Partnership Project (3GPP), Valbonne, France, Juni 2001, `http://www.3gpp.org`.

[Tsch 99] TSCHUDIN, C. F.: *Mobile Agent Security*. In: KLUSCH, M. [Klus 99], Seiten 431–445.

[Vign 98a] VIGNA, G.: *Cryptographic Traces for Mobile Agents*. In: VIGNA, G. [Vign 98], Seiten 137–153.

[Vign 98] VIGNA, G. (Herausgeber): *Mobile Agents and Security*, Nummer 1419 in *Lecture Notes in Computer Science (LNCS)*, Berlin, Heidelberg, 1998. Springer.

[ViLi 00] VISWANATHAN, D. und S. LIANG: *Java Virtual Machine Profiler Interface*. IBM Systems Journal, 39(1):82–95, 2000, `http://www.research.ibm.com/journal/sj/391/viswanathan.html`.

[Voyager] *Voyager*, `http://www.objectspace.com/products/voyager/`.

[Weid 98] WEIDMANN, M.: *Vergleich von Werkzeugen für das Sicherheitsmanagement bei der BMW Bank*. Diplomarbeit, Ludwig-Maximilians-Universität München, November 1998.

[Whit 00] WHITE, K.: *RFC 2925: Definitions of Managed Objects for Remote Ping, Traceroute, and Lookup Operations*. RFC, IETF, September 2000, `ftp://ftp.isi.edu/in-notes/rfc2925.txt`.

[WSB 00] WILHELM, U.G., S.M. STAAMANN und L. BUTTYÁN: *A Pessimistic Approach to Trust in Mobile Agent Platforms*. Internet Computing, 4(5):40–48, September 2000.

[X.509] ITU–T: *X.509 — Information technology — Open Systems Interconnection — The Directory: Authentication framework*. ITU–T Recommendation, International Telecommunication Union, August 1997.

[X.800] ITU: *X.800 — Data Communication Networks; Open Systems Interconnection (OSI); Security Structure and Application — Security Architecture for Open Systems Interconnection for CCITT Applications*. ITU–T Recommendation, International Telecommunication Union, Geneva, 1991.

[X.810] ITU: *X.810 — Data Networks and Open System Communications Security — Information Technology — Open Systems Interconnection — Security Frameworks for Open Systems: Overview*. ITU–T Recommendation, International Telecommunication Union, Geneva, 1996. also published as ISO/IEC International Standard 10181-1.

[X.811] ITU: *X.811 — Data Networks and Open System Communications Security — Information Technology — Open Systems Interconnection — Security Frameworks for Open Systems: Authentication Framework*. ITU–T Recommendation, International Telecommunication Union, Geneva, 1995. also published as ISO/IEC International Standard 10181-2.

[X.812] ITU: *X.812 — Data Networks and Open System Communications Security — Information Technology — Open Systems Interconnection — Security Frameworks for Open Systems: Access Control Framework*. ITU–T Recommendation, International Telecommunication Union, Geneva, 1995. also published as ISO/IEC International Standard 10181-3.

[X.814] ITU: *X.814 — Data Networks and Open System Communications Security — Information Technology — Open Systems Interconnection — Security Frameworks for Open Systems: Confidentiality Framework*. ITU–T Recommendation, International Telecommunication Union, Geneva, 1995. also published as ISO/IEC International Standard 10181-5.

[X.815] ITU: *X.815 — Data Networks and Open System Communications Security — Information Technology — Open Systems Interconnection — Security Frameworks for Open Systems: Integrity Frameworks*. ITU–T Recommendation, International Telecommunication Union, Geneva, 1995. also published as ISO/IEC International Standard 10181-6.

[X.816] ITU: *X.816 — Data Networks and Open System Communications Security — Information Technology — Open Systems Interconnection — Security Frameworks for Open Systems: Security Audit and Alarms Framework*. ITU–T Recommendation, International Telecommunication Union, Geneva, 1995. also published as ISO/IEC International Standard 10181-7.

[Yee 97] YEE, B. S.: *A Sanctuary for Mobile Agents*. In: *DARPA Foundations for Secure Mobile Code Workshop*, Seiten 21–27, Monterey, CA, USA, 26 - 28 March 1997. DARPA, `http://www.cs.nps.navy.mil/research/languages/statements/bsy.ps`.

[YGY 91] YEMINI, Y., G. GOLDSZMIDT und S. YEMINI: *Network Management by Delegation*. In: KRISHNAN, I. und W. ZIMMER (Herausgeber): *Proceedings of the 2nd International Symposium on Integrated Network Management, Washington*. IFIP, North-Holland, April 1991.

[YKSR 01a] YLONEN, T., T. KIVINEN, M. SAARINEN, T. RINNE und S. LEHTINEN: *SSH Protocol Architecture*. Internet Draft, IETF, Juli 2001, `http://www.ietf.org/html.charters/secsh-charter.html`.

[YKSR 01b] YLONEN, T., T. KIVINEN, M. SAARINEN, T. RINNE und S. LEHTINEN: *SSH Connection Protocol*. Internet Draft, IETF, Juli 2001, `http://www.ietf.org/html.charters/secsh-charter.html`.

[YKSR 01c] YLONEN, T., T. KIVINEN, M. SAARINEN, T. RINNE und S. LEHTINEN: *SSH Authentication Protocol*. Internet Draft, IETF, Juli 2001, `http://www.ietf.org/html.charters/secsh-charter.html`.

[YKSR 01d] YLONEN, T., T. KIVINEN, M. SAARINEN, T. RINNE und S. LEHTINEN: *SSH Connection Protocol*. Internet Draft, IETF, Januar 2001, `http://www.ietf.org/html.charters/secsh-charter.html`.

INDEX

B

C

D

E

Q

R

S