INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Rechtekonzept für die Mobile Agent System Architecture (MASA)

Dietmar Fackelmann

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Rechtekonzept für die Mobile Agent System Architecture (MASA)

Bearbeiter:	Dietmar Fackelmann
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Helmut Reiser Harald Rölle

Abgabetermin: 13. März 2001

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 13. März 2001

. . . .

(Unterschrift des Kandidaten)

Zusammenfassung

Mobile Agenten führen Tätigkeiten auf verschiedenen Systemen flexibel und ortsungebunden aus. Dazu werden die Agenten an die lokale Umgebung eines Systems gebunden. Neben der Ausführung des Agenten auf dem System sind oft auch Zugriffe auf Ressourcen des Systems notwendig. Eine Agenteninstanz kann ebenso Ressourcen bereitstellen, auf die ein Agentensystem oder weitere Agenteninstanzen zugreifen können. Dabei müssen unautorisierte Zugriffe und der Missbrauch von Informationen des Systems ausgeschlossen werden. Für die Akzeptanz einer Agentensystemarchitektur spielt die Sicherheit eine große Rolle.

Diese Arbeit erstellt für das CORBA/Java basierte mobile Agentensystem MASA ein Rechtekonzept.

Beginnend mit einer Einführung in die Grundlagen zur Sicherung der Informationssicherheit werden die Anforderungen an eine Agentensystemarchitektur analysiert. Eine konkrete Untersuchung von MASA zeigt vorhandene Schwachstellen der momentanen Implementierung auf. Auf Basis der Sicherheitsanforderungen und den MASA zugrundeliegenden Standards wird ein Rechtekonzept entwickelt, welches eine geeignete Policy-Sprache bereitstellt, um anwendungsspezifische Zugriffskontrollpolitiken zu formulieren.

Anhand eines Implementierungskonzepts wird die Integration des Rechtekonzepts in das mobile Agentensystem MASA beschrieben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Durch den Einsatz Mobiler Agentensysteme sind generell alle IT-Ressourcen gemeinsam benutzbar. Mobile Agenten können auf Agentensysteme migrieren, auf Ressourcen von Agentensystemen zugreifen, aber auch selbst Ressourcen bereitstellen. Durch eine Agentensystemarchitektur besitzt eine Anwendung eine höhere Flexibilität und Dynamik. Agenten können dynamisch um neue Funktionalitäten erweitert werden und diese durch Migration der Agenten auf den unterschiedlichen Agentensystemen zur Verfügung stellen. Eine Beispielanwendung ist die Verwendung einer Agentensystemarchitektur als ein verteiltes Managementsystem. Die Vorteile liegen hier in der durch das Agentensystem ermöglichten, dynamischen Änderung der Managementfunktionalität. Beispielsweise kann so eine neu hinzugefügte Ressource dynamisch in ein Endsystem eingebunden werden. Die ansonsten nötige manuelle Konfiguration vor Ort entfällt.

Die Vorteile der Agentensystemarchitektur resultieren aus der Mobilität der Agenten. Bezüglich der Sicherheit birgt die Mobilität jedoch neue Risiken. Aus der Perspektive eines Systems kann ein Agent potenziell die Informationssicherheit des Systems gefährden. Ein bösartiger Agent kann beispielsweise versuchen, unberechtigt auf sensible Daten des Systems zuzugreifen. Eine weitere Gefahr besteht in der Störung des Funktionsablaufs durch einen Agenten. Betrachtet man die Agenten, so können auch sie sensible Daten mit sich führen. Auch diese Daten müssen vor unautorisierten Zugriffen und Missbrauch geschützt werden.

Betrachtet man die möglichen Einsatzgebiete von Agentensystemarchitekturen, wie z.B. den Einsatz als Managementsystem, werden an das Agentensystem hohe Sicherheitsanforderungen gestellt. Eine Hauptanforderung ist die Sicherstellung der Informationssicherheit, so dass Informationen vor unautorisierten Zugriffen und Missbrauch geschützt sind. Der Schutz der Informationssicherheit ist schon bei Einzelplatzsystemen und verteilten Systemen keine leichte Aufgabe. Durch die Eigenschaften einer Agentensystemarchitektur wird diese Aufgabe noch zusätzlich erschwert. Hier gilt es, Ressourcen des Agentensystems vor Agenten und Ressourcen der Agenten vor Agentensystemen zu schützen.

Der Hauptunterschied einer Agentensystemarchitektur gegenüber herkömmlichen Architekturen ist die Ausführung von fremden Programmen auf dem System. Dieser Fall tritt ein, sobald ein Agent auf einem Agentensystem ausgeführt wird. Der Agent muss dynamisch in die Ausführungsumgebung des Agentensystems eingebunden und ausgeführt werden.

Java-Applets, die von einem Rechner über das World Wide Web heruntergeladen werden und auf dem System innerhalb eines Browsers zur Ausführung gebracht werden, können als Agenten, wenn auch mit eingeschränkter Funktionalität, betrachtet werden. Sicherheitshinweise, die vor der Ausführung von Java-Applets warnen, lassen die Sicherheitsproblematiken, die durch die Ausführung von fremden Programmen auf dem System entstehen, erahnen.

Ein Rechtekonzept eines Agentensystems hat die Aufgabe, die Anforderungen bezüglich der Informationssicherheit zu erfüllen. Dabei werden durch das Rechtekonzept die Rechtevergabe, Rechtedurchsetzung und deren Verwaltung betrachtet. Die Auswahl der eingesetzten Mechanismen und Techniken hat dabei direkte Einflüsse auf die Eigenschaften und Funktionalität der Agentensystemarchitektur. Unterschiedliche Konzepte können sich unterschiedlich sowohl auf die Migration, Autonomie und die Kommunikationsmöglichkeiten der Agenten als auch auf die Effizienz und Performance auswirken.

Letztendlich wird ein Rechtekonzept benötigt, welches die positiven Eigenschaften einer Agentensystemarchitektur unterstützt, eine sichere Umgebung bereitstellt und die Anforderungen bezüglich der Performance und der Funktionalität erfüllt.

1.1 Überblick über die Mobile Agent System Architecture (MASA)

MASA ist eine Agentensystemarchitektur, welche von Bernhard Kempter [KEM 98] entwickelt und implementiert wurde. Die plattformunabhängige Agentensystemarchitektur ist eine Laufzeitarchitektur für mobile Agenten. Der Kerngedanke bestand darin, das zentrale Management verteilter Systeme durch ein verteiltes, flexibles Management abzulösen. Hierzu werden Agenten eingesetzt, die unterschiedliche Funktionalitäten besitzen. Die Agenten sind in der Lage unterschiedliche Managementaufgaben auszuführen, wobei ihnen ihre Mobilität und die Kommunikationsfähigkeit mit anderen Agenten zugute kommt. Durch den Einsatz der Agentensystemarchitektur werden Nachteile des zentralen Managements gelöst. So kann eine Agentensystemarchitektur besser mit der Heterogenität und der großen Anzahl der zu managenden Komponenten und Protokollen umgehen. Ebenso ist die Architektur besser geeignet, sich ändernden Gegebenheiten im Netz anzupassen. Die Agentensystemarchitektur hat auch nicht mehr den Nachteil des "Single Point of Failure".

Um die Agentensystemarchitektur zu nutzen, wird auf einem Endsystem jeweils ein Agentensystem zur Ausführung gebracht. Ein Agentensystem stellt dabei die Ausführungsumgebung für Agenten zur Verfügung. Für Agenten ist das Agentensystem auch die Schnittstelle zum Endsystem. Agenten sind in der Lage, über das Agentensystem auf Ressourcen des Endsystems zuzugreifen.

MASA ist vollständig in der Programmiersprache Java implementiert [JAW 01]. Die Plattformunabhängigkeit des Agentensystems ist durch die Java Virtual Machine, welche die Basis für das Agentensystem darstellt, gegeben. Als Kommunikationsinfrastruktur für die Agenten wird die Common Object Request Broker Architecture (CORBA) eingesetzt. Agenten können über die CORBA-Schnittstelle miteinander kommunizieren. Die Mobilität der Agenten ist durch die Möglichkeit der selbständigen Migration von Agenten gegeben. Die graphische Bedienoberfläche wird durch einen Webbrowser dargestellt. Durch den Browser kann der Webserver, Bestandteil eines jeden Agentensystems, angesprochen werden. Der Webserver wird zur Steuerung der Agenten und des Agentensystems herangezogen. Der Benutzer greift über einen Browser auf das Agentensystem zu. Der Zugriff auf einen Agenten erfolgt durch ein Java-Applet des Agenten, welches vom Webserver angefordert werden kann. Die weitere Kommunikation mit dem Agenten über das Applet erfolgt über CORBA. Die MASA Architektur und ihre Komponenten werden ausführlich in [HGR 99] behandelt.

Die Authentisierung und Autorisationsanforderungen für MASA wurden in der Arbeit von Harald Rölle betrachtet [ROE 99]. Dabei wurde ein Sicherheitskonzept für MASA entwickelt, welches der Entwicklung des Rechtekonzepts zugute kommt.
Das Sicherheitskonzept realisiert grundlegende Sicherheitseigenschaften, auf die das Rechtekonzept aufbauen kann. Einzelne Entitäten können durch die Verwendung von Public-Key Verfahren und Zertifikate authentifiziert und identifiziert werden. Die Kommunikationskanäle werden gesichert und es werden getrennte Ausführungsumgebungen der Entitäten bereitgestellt. Statische und dynamische Informationen der Agenten können unterschieden werden, wobei die statischen Informationen eines Agenten geschützt sind.

1.2 Die Sicherheitsproblematik anhand eines Beispiels

Die Sicherheitsproblematik, welche die Entwicklung eines Rechtekonzepts erfordert, wird auf Grundlage eines Beispielszenarios der Managementanwendung in Anlehnung an [ROE 99] betrachtet (vgl. Abb. 1).

In der Managementanwendung wird ein Händler über einen Provider mit dem Automobilhersteller vernetzt. Der Händler hat so z.B. die Möglichkeit, die Dienste des Bestell-Centers des Automobilherstellers zu nutzen. Die Händler sind dabei über Wählleitungen an den Provider

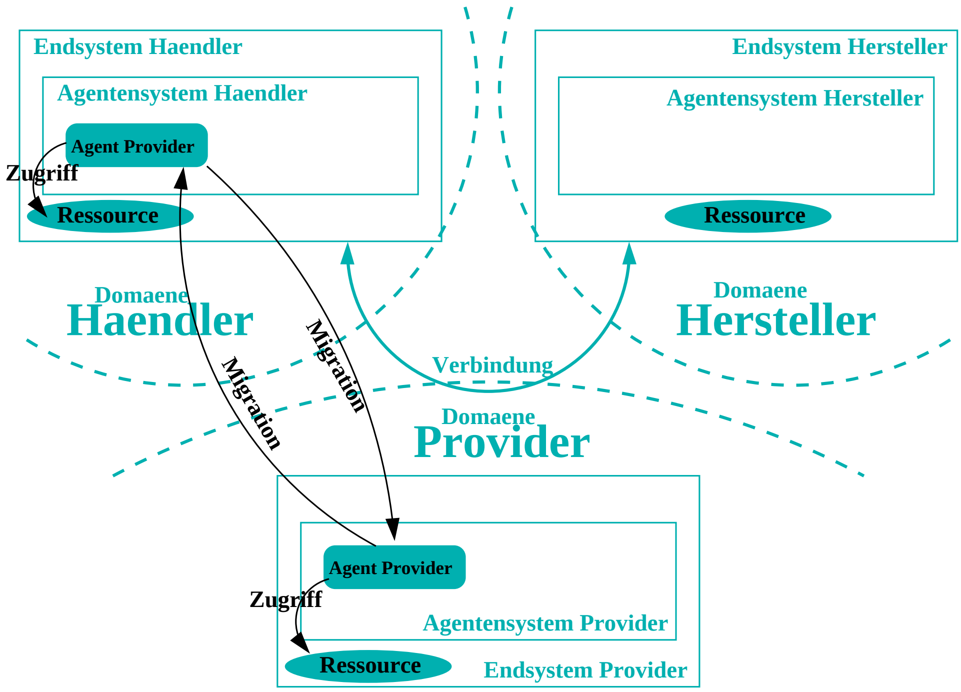


Abbildung 1: Beispielszenario

angeschlossen, wobei die Verbindung als ein Virtuelles Privates Netzwerk (VPN) realisiert ist. Mit dem Provider werden in einem sogenannten Service Level Agreement (SLA) Quality of Service (QoS) Parameter festgelegt. Ein QoS-Parameter stellt z.B. der Netzdurchsatz vom Händler zum Bestell-Center des Automobilherstellers dar.

Damit die Einhaltung der QoS-Parameter überprüft werden kann, sind Messungen der relevanten QoS-Parameter nötig. Die Messung des Netzdurchsatzes vom Händler zum Automobilhersteller erfolgt am Besten vom Händler aus. Aufgrund der Möglichkeit von sich ändernden Gegebenheiten ist zudem die Flexibilität des Management-Centers des Providers gefragt. So können beispielsweise neue QoS-Parameter hinzu kommen oder geändert werden.

Der Einsatz von MASA eignet sich hier besonders gut. Für die Durchsatzermittlung kann das Management-Center des Providers einen mobilen Agenten zum Agentensystem des Händlers senden, wobei dort die nötigen Messungen durch diesen mobilen Agenten vorgenommen werden können. Ändern sich die QoS-Parameter oder kommen neue hinzu, kann ein modifizierter mobiler Agent erstellt werden. Für eine neue Aufgabe kann jeweils ein entsprechender Agent eingesetzt werden, welcher die geforderte Funktionalität bereitstellt und auf das entsprechende Agentensystem migriert, um dort die neue Funktionalität zu integrieren. Die Möglichkeit der Aufgabenteilung ist ein weiterer Vorteil von Agentensystemen. Ein Agent ist beispielsweise in der Lage, weitere Agenten zu beauftragen, um die Aufgabe möglichst effektiv zu bearbeiten.

Die Problematik bezüglich der Sicherheit liegt in den unterschiedlichen Administrationsbereichen

- des Händlers
  (Der Administrator des Händlers ist verantwortlich für sein internes LAN)
- des Providers (Management-Center)
  (Der Management-Provider ist verantwortlich für die Netzverbindung vom Händler zum Hersteller und der Einhaltung der festgelegten QoS-Parameter)

- des Automobilherstellers
  (Der Anwendungs-Administrator ist verantwortlich für das Firmennetz und die Anwendungen des Bestell-Centers)

Ein Agent durchwandert u.U. unterschiedliche Administrationsbereiche. Migriert beispielsweise der Agent des Providers auf das Agentensystem des Händlers, um dort Messungen vorzunehmen, müssen die Sicherheitspolitiken beider beteiligter Partner erfüllt werden. Der Missbrauch von Informationen und unautorisierte Zugriffe auf Informationen der Agentensysteme und der Agenten sind auszuschließen.

Anforderungen des Händlers
Die Sicherheitsanforderungen des Händlers bestehen darin, dem Agenten des Providers nur die Zugriffe zu gewähren, die er für die Messung des Netzdurchsatzes unbedingt benötigt. Der Händler muss daher in der Lage sein, eine möglichst feingranulare Sicherheitspolitik formulieren zu können. Zugriffsrechte müssen an einzelne spezifische Entitäten, aber auch an bestimmte Gruppen von Entitäten vergeben werden können. Die Sicherheitspolitik kann unter Umständen sehr groß werden, wobei sich die Eintragungen dynamisch ändern können.
Außerdem muss der Händler sicherstellen können, dass der Agent im Auftrag des Providers handelt. Bei der Entscheidungsfindung kann dabei auch die Vorgeschichte des Agenten eine Rolle spielen. Bezüglich des Schutzes der Informationssicherheit ist jeder Zugriff einer Zugriffskontrolle zu unterziehen. Die Zugriffskontrolle hat dabei die Aufgabe, die Sicherheitspolitik des Agentensystems durchzusetzen.

Anforderungen des Providers
Die Sicherheitsanforderungen des Provider liegen im Schutz seines Agenten und dessen Ressourcen. Der Provider muss in der Lage sein, eine Sicherheitspolitik für spezifische Agenten erstellen zu können, welche ebenfalls die zugreifenden Entitäten spezifisch adressieren kann. Auch hier müssen feingranulare Zugriffsrechte vergeben werden können. Dabei ist wichtig, dass jeder Zugriff auf den Agenten einer Zugriffskontrolle unterzogen wird. Die Zugriffskontrolle muss dabei die Sicherheitspolitik des Providers durchsetzen. Gegebenenfalls sollte der Agent auch in der Lage sein, seine Aufgaben an weitere Agenten zu delegieren, wobei der Agent die Möglichkeit haben sollte, seine Zugriffsrechte an einen Agenten delegieren zu können, damit dieser die für die Bearbeitung der Aufgabe notwendige Berechtigung besitzt.

1.3 Aufgabenstellung

Für die Mobile Agent System Architecture (MASA) ist ein Rechtekonzept zu entwickeln, welches die Sicherung der Informationen vor Missbrauch und unautorisierten Zugriffen realisiert. Das Konzept soll die Unversehrtheit und Vertraulichkeit der Informationen sicherstellen. Neben dem Aussehen der Zugriffsrechte, muss die Art der Vergabe und deren Durchsetzung festgelegt werden. Die ausgewählten Mechanismen und Techniken sollten Eigenschaften wie die benutzerfreundliche Verwaltung der Sicherheitspolitik und der einfachen und transparenten Nutzung der Ressourcen berücksichtigen. Da Sicherheitsüberprüfungen immer mit einem gewissen Overhead verbunden sind, spielen auch die Performanceansprüche bei der Entwicklung des Rechtekonzepts eine wichtige Rolle.

Die Zugriffskontrolle impliziert eine vorherige Autorisation der Subjekte. Das Rechtekonzept muss daher ein geeignetes Konzept der Authentisierung und Identifizierung vorweisen. Erfolgt die Autorisation durch eine Zugriffskontrollpolitik, sollte die eingesetzte Policy-Sprache einem formalen Modell zugrundeliegen. Die Kriterien, die zur Entscheidungsfindung herangezogen werden, müssen spezifiziert und klassifiziert werden. Dabei ist darauf zu achten, dass die Kriterien den Anforderungen der Mobile Agent System Architecture (MASA) entsprechen.

Damit Zugriffe anwendungsspezifisch autorisiert werden können, sollte eine möglichst feingranulare Zugriffskontrolle realisiert werden. Die Zugriffskontrollpolitik sollte handhabbar und benutzerfreundlich konfigurierbar sein. Die Sicherheitspolitiken müssen überschaubar und eindeutig interpretierbar sein.
Ein Agent soll weiter in der Lage sein, seine Rechte einem anderen Agenten weiterzugeben. Die Rechtedelegation ist als eine erweiterte Funktion des Rechtekonzepts anzusehen.

Sicherheitsrelevante Fragen sollten beantwortbar sein. Dazu gehört z.B. die Frage, welches Subjekt ein spezifisches Zugriffsrecht besitzt, oder wer in den letzten zwei Stunden auf das Objekt "XY" zugegriffen hat. Die Beantwortbarkeit dieser oder ähnlicher Fragen stellt die Basis für ein sicheres System, auch in Bezug auf die Rückverfolgung eventuell unautorisierter Zugriffe.

Es gilt, die Autorisation von der Rechtedurchsetzung und die Rechtedurchsetzung von der Systemfunktionalität möglichst sauber zu trennen, wodurch der Austausch einzelner Module ermöglicht wird.

Die Entwicklung eines Rechtekonzepts für MASA umfasst Thematiken der unterschiedlichsten Gebiete der Informatik. So spielen neben einer formalen Beschreibung einer Policy-Sprache, Mechanismen und Techniken von softwarebasierten Schutzmechanismen und unterschiedliche Architekturen eine Rolle.

Zusammenfassung der Zielsetzung:

- Jedes Objekt sollte vor unautorisierten Zugriffen geschützt sein und lediglich durch autorisierte Subjekte zugreifbar sein und somit einer Zugriffskontrolle unterstehen.
- Jeder Zugriff auf jedes Objekt sollte überprüft werden.
- Ein Subjekt sollte erst nach einer expliziten Autorisation Zugriff auf ein spezifisches Objekt haben. Erfolgte keine explizite Autorisation, so sollte der Zugriff nicht erlaubt werden.
- Fehler, die die Sicherheit betreffen, sollten durch die Überprüfung der Programme auffindbar sein.
- Mobile Agenten sollen die Möglichkeit der Rechtedelegation, also der Weitergabe der Rechte haben.
- Das Rechtekonzept sollte möglichst flexibel und erweiterungsfähig sein.
- Durch ein geeignetes Code Auditing- (Logging-) Verfahren, sollten Attacken zurückverfolgt werden können.
- Durch einen Accountability Mechanismus sollten evtl. schädliche Programme identifiziert werden können.
- Privilegien sollten lokalisierbar sein.
- Ressourcen sollten limitierbar sein.
- Die Performanceansprüche sollten erfüllt sein.

1.4 Vorgehensweise

Im Folgenden wird der Aufbau der Arbeit beschrieben.

Kapitel 2 beschäftigt sich mit Grundlagen, die für die Entwicklung des Rechtekonzepts relevant sind. Es werden grundlegende Begriffe eingeführt, die in der gesamten Arbeit verwendet werden. Nach Betrachtung der Sicherheitsbedrohungen und Sicherheitsanforderungen, werden bekannte

Sicherheitsmodelle vorgestellt. Hier werden unterschiedliche Konzepte und Mechanismen hinsichtlich ihrer Vor- und Nachteile untersucht. Ein weiteres Unterkapitel beschreibt Mechanismen, die zur Formulierung einer Sicherheitspolitik eingesetzt werden. Das darauffolgende Unterkapitel konzentriert sich auf die Komponenten, die zur Realisation der in der Sicherheitspolitik formulierten Anforderungen benötigt werden. Auch hier existieren eine Reihe von Konzepten und Mechanismen, die genauer betrachtet werden. Nachdem unterschiedliche Anforderungen der unterschiedlichen Architekturen behandelt worden sind, werden Modelle, die zur Bewertung und Prüfung der Sicherheit herangezogen werden, analysiert. Am Ende des Kapitels werden die in Java eingesetzten Mechanismen und Techniken bezüglich der Sicherheit eingeführt. Dazu gehört auch die Vorstellung von existierenden und bereits eingesetzten Mechanismen.

Spezielle Anforderungen eines Rechtekonzepts für eine Agentensystemarchitektur werden in Kapitel 3 betrachtet. Die Schwerpunkte liegen hier auf der Rechtevergabe und Rechtedurchsetzung.

In Kapitel 4 wird die Mobile Agent Management Architecture (MASA) bezüglich der in Kapitel 3 spezifizierten Anforderungen analysiert. Dabei werden sowohl Stärken als auch Schwächen der momentanen Implementierung aufgezeigt. Die Risikoanalyse aus Kapitel 4 verdeutlicht, inwieweit MASA die Sicherheitsanforderungen erfüllt bzw. wo noch Schwachstellen liegen.

Das Kapitel 5 betrachtet Standards bezüglich eines Rechtekonzepts. Desweiteren liefert das Kapitel einen Überblick über Konzepte, die zum Schutz der Agentensystemarchitektur eingesetzt werden können. Außerdem wird der realisierte Schutz verschiedener Agentensystemarchitekturen betrachtet.

Aus den in Kapitel 3 gewonnenen allgemeinen Anforderungen und den in Kapitel 4 analysierten Anforderungen für MASA, wird in Kapitel 6 ein Rechtekonzept für MASA entwickelt. Die einzelnen Komponenten des Rechtekonzepts werden vorgestellt und deren Funktionsweise beschrieben. Dazu gehört, neben der Einführung einer Politik-Sprache, das Aussehen der Zugriffsrechte, die Realisation der Rechtevergabe und deren Durchsetzung. Das Zusammenwirken der einzelnen Komponenten des Rechtekonzepts wird am Ende anhand von Beispielszenarien verdeutlicht.

Das Implementierungskonzept von Kapitel 7 integriert das in Kapitel 6 entwickelte Rechtekonzept in die MASA Umgebung. Mechanismen und Techniken, die für die Realisation des Rechtekonzepts verwendet werden können, werden vorgestellt. Das Kapitel beschreibt die notwendige Vorgehensweise, um das in Kapitel 6 vorgestellte Rechtekonzept in MASA zu integrieren und zu nutzen.

Das Kapitel 8 enthält eine Zusammenfassung der Arbeit und einen Ausblicke auf Weiterentwicklungsmöglichkeiten.

2 Grundlagen

Das folgende Kapitel behandelt Grundlagen, die für die Entwicklung eines Rechtekonzepts relevant sind. Nachdem im Kapitel 2.1 grundlegende Begriffe eingeführt werden, wird die Notwendigkeit eines Rechtekonzepts anhand von Sicherheitsbedrohungen und den daraus basierenden Sicherheitsanforderungen motiviert (Kapitel 2.2 und Kapitel 2.3). In Kapitel 2.4 werden anhand von existierenden Sicherheitsmodellen Sicherheitsanforderungen und deren Modellierung und Realisation erläutert. Dabei gibt das Unterkapitel über die Sicherheitspolitiken (Kapitel 2.5) einen Einblick in unterschiedliche Zugriffskontrollmodelle. Die Vorstellung der einzelnen Komponenten einer Zugriffssteuerung und deren Realisationsmodelle in Kapitel 2.6 geben einen Einblick in den Ablauf der Zugriffssteuerung. In Kapitel 2.7, wird anhand der unterschiedlichen Architekturen die jeweiligen Problematik der Realisation einer Zugriffskontrolle verdeutlicht. In Kapitel 2.8 werden Modelle, die zur Bewertung und Prüfung der Sicherheit eines Systems herangezogen werden können, vorgestellt. Abschließend wird in Kapitel 2.9 die Sicherheitsarchitektur von Java betrachtet. Dabei werden einzelne Konzepte und Mechanismen der Java 2 Security Architecture und ihre Grenzen vorgestellt.

2.1 Grundlegende Begriffe

Der folgende Abschnitt dient der Einführung der für das Thema wesentlichen Begriffe.

Information
"Information ist ein Abstraktum, das in einem Rechensystem in Form von Datenobjekten repräsentiert, d.h. konkretisiert wird. Die Information ist das zu schützende Gut eines rechtssicheren Systems. Der Zugriff auf Information ist zu beschränken und zu kontrollieren." [ECK 98]

Informationssicherheit
"Unter der Rechtssicherheit bzw. Informationssicherheit eines Systems verstehen wir die Eigenschaft eines funktionssicheren Systems, nur solche Systemzustände anzunehmen, die zu keiner unautorisierten Informationsveränderung und -gewinnung führen." [ECK 98]

Objekt
Objekte sind konkrete Informationen in Form eines passiven Objekts, wie z.B. eine Datei oder aber auch in Form eines aktiven Objekts, wie z.B. einem Programm. Ein Objekt stellt den passiven Part eines Zugriffs dar.

Subjekt
Subjekte greifen auf Objekte zu. Ein Benutzer, aber auch ein Programm, welches unter der Anordnung eines Benutzers abläuft, kann ein Subjekt sein. Ein Subjekt ist daher immer der aktive Part.

Aktion
Eine Aktion beschreibt die Art und Weise, wie auf ein Objekt zugegriffen wird.

Greift beispielsweise ein Benutzer auf eine Datei zu, um sie zu lesen, so ist der Benutzer das Subjekt, die Datei das Objekt und der lesende Zugriff die Aktion.

Zugriff
Man spricht von einem Zugriff von einem Subjekt auf ein Objekt, sobald das Subjekt mit dem Objekt Informationen jeglicher Art austauscht.

Zugriffssteuerung
Die Zugriffssteuerung (engl. Access Control) beinhaltet die Kontrolle, die Identifikation und Authentifikation der Subjekte, die Rechtedurchsetzung und -verwaltung und die Protokollierung und die Auswertung von Zugriffen.

Authentifikation
Die Authentifikation dient der Verifikation einer Identität und wird zumeist von der Zugangskontrolle realisiert.

Rechtedurchsetzung
Die Rechtedurchsetzung (Zugriffskontrolle) setzt die Rechte durch, die an Subjekte vergeben wurden. Dabei wird anhand der Zugriffskontrollpolitik entschieden, ob das zugreifende Subjekt für diesen Zugriff autorisiert ist.

Rechtevergabe
Die Rechtevergabe ist zuständig für die Vergabe und Rücknahme von Zugriffsrechten (Autorisation).

Zugriffskontrollpolitik
Die Zugriffskontrollpolitik enthält Regeln, welche Zugriffe auf Objekte autorisieren oder verbieten.

Zugriffsrecht
Ein Zugriffsrecht beschreibt das Recht, auf ein Objekt zuzugreifen. Zugriffsrechte beinhalten in der Regel die spezifische Aktion, also auf welche Art und Weise auf das Objekt zugegriffen werden darf. Je nach eingesetzten Mechanismen kann ein Zugriffsrecht zusätzlich das Objekt spezifizieren, für welches das Zugriffsrecht bestimmt ist.

unautorisierter Zugriff
Besitzt ein Subjekt nicht die Berechtigung, also wurde ihm kein Zugriffsrecht vergeben, darf es nicht auf das Objekt zugreifen. Kann das Subjekt dennoch auf das Objekt zugreifen, handelt es sich um einen unautorisierten Zugriff.

Entscheidungskriterium
Ein Entscheidungskriterium ist ein Kriterium, anhand dessen entschieden werden kann, ob der Zugriff gewährt oder abgelehnt wird.

Darf nur ein von Person "X" signiertes Programm auf den Drucker zugreifen, so ist die Signatur ein Entscheidungskriterium, das bei der Entscheidungsfindung herangezogen werden muss.

2.2 Sicherheitsbedrohungen

Ein Rechensystem ist vielen Bedrohungen (engl. threats) ausgesetzt, die die Informationssicherheit des Systems gefährden. Die häufigste Bedrohung ist dabei der Versuch, unautorisiert auf Objekte des Systems zuzugreifen. Es können aktive Angriffe, d.h. Objekte werden manipuliert und passive Angriffe, hier bleiben Objekte unverändert, unterschieden werden [REI 97].

Allgemein kann man drei Klassen von Bedrohungen unterscheiden [REI 97]:

- **Enthüllung der Information (Disclosure of Information)**
- **Angriff auf die Verfügbarkeit eines Dienstes (Denial-of-Service Attack)**
- **Korruption der Information (Corruption of Information)**

Die Informationssicherheit wird in der Regel in allen drei Fällen verletzt. Die Zugriffskontrolle, welche eine geeignete Autorisation der zuvor authentifizierten Entitäten voraussetzt, hat die Aufgabe, unautorisierte Zugriffe zu unterbinden. Durch die Kenntnisnahme existierender Bedrohung können unter Umständen schon bei der Entwicklung eines Rechtekonzepts geeignete Maßnahmen getroffen werden, um den Bedrohungen entgegen zu wirken. Eine nachträgliche Integration von bestimmten Sicherheitsdiensten kann dann in der Regel entfallen.
Anhand der analysierten Bedrohungen können anwendungsspezifische Anforderungen analysiert und aufgestellt werden.

2.3 Sicherheitsanforderung

Die Sicherheitsanforderungen betreffen in erster Linie die Forderung nach der Gewährleistung der Informationsvertraulichkeit, der Verfügbarkeit und der Datenintegrität [ECK 98].

Die Sicherheitsanforderungen basieren hauptsächlich darauf, dass nur autorisierte Subjekte auf ein Objekt zugreifen können. Um Subjekte autorisieren zu können, müssen sie authentifizierbar sein, d.h. Subjekte müssen identifiziert und ihre Identität muss verifiziert werden können. Nur authentifizierbare Subjekte können so einer Zugriffskontrolle unterzogen werden.
Innerhalb der Zugriffssteuerung muss sichergestellt sein, dass alle Subjekte und Objekte, die innerhalb der Sicherheitspolitik auftauchen, von der Rechtevergabe erfasst werden können. Änderungen bzgl. der Rechte dürfen nur von autorisierten Entitäten vorgenommen werden. Eine Verletzung der Sicherheitspolitik muss entsprechend verarbeitet werden. Desweiteren ist jeder Zugriffsversuch einer Zugriffskontrolle zu unterwerfen. Es darf keine Möglichkeit geben, die Zugriffskontrolle zu umgehen um unautorisiert auf Objekte zuzugreifen.

Wurde von einem Subjekt ein Zugriff auf eine Ressource getätigt, so sollte der Zugriff dem Subjekt eindeutig zugeordnet werden können. Die eindeutige Zuordnung ist eine Voraussetzung, um Zugriffe abrechnen zu können. Sind beispielsweise Zugriffe mit Kosten verbunden, müssen sie geeignet abgerechnet werden können. Die Abrechnungsmöglichkeit ist vor allem aber auch dann relevant, wenn Ressourcen nur beschränkt vorhanden sind und eine gerechte Verteilung der Ressourcen erfolgen muss.

Bei der Erstellung eines Rechtekonzepts müssen die Anforderungen des Systems an das Rechtekonzept detailliert beschrieben werden. In der Literatur [SAS 75, ECK 98, SUM 97] finden sich unterschiedliche Anforderungen die ein Sicherheitskonzept betreffen. Dabei handelt es sich oft um allgemeingültige Anforderungen, die erfüllt werden sollten, um hohen Sicherheitsanforderungen zu genügen. Folgend werden die relevanten Anforderungen stichpunktartig aufgezählt und erläutert. Die einzelnen Anforderungen nach [SUM 97] können als Basis für einen Kriterienkatalog herangezogen werden.

- **Least-Privilege / Need-to-Know**
  Subjekte sollten nur die Berechtigungen zugeteilt bekommen, welche sie unbedingt benötigen, um ihre Aufgabe zu erfüllen.

- **Economy of Mechanism**
  Eingesetzte Mechanismen sollten benutzerfreundlich sein. Dabei ist gegebenfalls z.B. ein automatischer Mechanismus gefragt.

- **Resource Limits**
  Die Beschränkung der Zugriffe sollte sich auch auf die Systemressourcen wie z.B. den Prozessor und den Arbeitsspeicher beziehen. Die Systemressourcen sind i. Allg. beschränkt und müssen aufgeteilt werden. Durch die Ressourcenlimitierung kann ein reibungsloser Ablauf garantiert werden, wobei hier z.B. Denial-of-Service Attacken erschwert werden.

- **Psychological Acceptability**
  Die Psychologische Akzeptanz ist ein nicht zu unterschätzender Punkt. Ein Sicherheitssystem sollte benutzerfreundlich und einfach zu konfigurieren sein.

- **Accountability**
  Zum Accountability Prinzip gehört das Auditing, die Identifikation und die Authentifikation. Zugriffsrechte können missbraucht werden. Das System sollte daher sicherheitskritische Abläufe aufzeichnen können. Durch die Aufzeichnung sollte z.B. das Subjekt, das Objekt, die ausgeführte Aktion und der Zugriffszeitpunkt erkennbar sein.

- **Least Common Mechanism**
  Durch die Teilung von Mechanismen untereinander ist die Möglichkeit verdeckter Kommunikationskanäle gegeben. Je weniger Mechanismen gemeinsam genutzt werden, desto geringer ist die Gefahr, die aus den verdeckten Kanälen resultiert.

- **Fail-Safe Defaults**
  Es sollte grundsätzlich jeder Zugriff, der nicht explizit autorisiert worden ist, unterbunden werden.

- **Performance**
  Das System muss bestimmten Performanceansprüchen genügen. Sicherheitsüberprüfungen verursachen immer einen Overhead, der jedoch nicht zu groß sein darf.

- **Compatibility**
  Die Kompatibilität sollte möglichst erhalten bleiben. Werden Änderungen vorgenommen, die inkompatibel zu bestehenden Standards sind, kann dies zu Problemen beim Einsatz des Systems führen.

- **Open-Design**
  Die Sicherheit darf nicht von der Geheimhaltung spezieller Verfahren abhängen.

- **Complete Mediation**
  Jeder Zugriff auf jedes Objekt sollte überprüft werden.

- **Code Auditability**
  Programmcode, der möglicherweise Fehler enthalten kann, sollte überprüfbar sein.

- **Separation of Policy and Mechanism**
  Generell sollte die Sicherheitspolitik von den Mechanismen, die die Sicherheitspolitik implementieren, getrennt sein. Für denselben Mechanismus sollte es möglich sein, multiple Politiken zu unterstützen.

- **Proper Weight to other Goals**
  Die Sicherheit ist nur ein Ziel, wobei oft Konflikte mit anderen Anforderungen entstehen (Funktionalität, Benutzerfreundlichkeit, Performance und Kosten). Hier ist ein gesundes Verhältnis gefragt.

Unterschiedliche Sicherheitsmodelle versuchen dabei, einzelne Aspekte der Anforderungen zu erfüllen.

2.4 Sicherheitsmodelle

Folgend wird das Sicherheitsmodell der Zugriffsmatrix vorgestellt. Es ist für bestimmte Einsatzbereiche flexibel nutzbar und kann in Bezug auf die Datenintegrität anwendungsangepasst eingesetzt werden. Das Zugriffsmatrix-Modell eignet sich zudem besonders gut, um die subjekt- und objektbasierten Mechanismen der Rechtevergabe zu beschreiben.

2.4.1 Sicherheits- / Zugriffsmatrix

Das Zugriffsmatrix-Modell ist ein einfaches Sicherheitsmodell. Durch die Zugriffsmatrix werden die Zugriffsrechte von Subjekten auf Objekte modelliert. In dem Zugriffsmatrix-Modell werden keine Vorgaben gemacht, die die Granularität der Subjekte und der Objekte betreffen. Dadurch können sie anwendungsspezifisch, d.h. auf Basis der Anforderungen einer spezifischen Anwendung hin, festgelegt werden. Der Schutzzustand eines Systems wird durch eine
| Anzahl der Subjekte | x | Anzahl der Objekt | -Matrix modelliert [ECK 98, SUM 97].

Das Modell der Zugriffsmatrix kommt aufgrund seiner Einfachheit und seiner Vielseitigkeit auch in vernetzten Systemen zum Einsatz. Dennoch besitzt das Modell auch einige Schwachstellen. Das Zugriffsmatrix-Modell im herkömmlichen Sinn hat Einschränkungen in Bezug auf die Darstellung besonderer Sicherheitspolitiken, die z.B. von zeitlichen Abläufen, sich dynamisch ändernden Eigenschaften oder mehreren Nutzerautorisationen abhängen.
In der Regel können jedoch alle Sicherheitspolitiken anhand des Zugriffsmatrix-Modells modelliert werden. In der Praxis werden unterschiedliche Mechanismen und Techniken eingesetzt, um das Zugriffsmatrix-Modell umzusetzen.

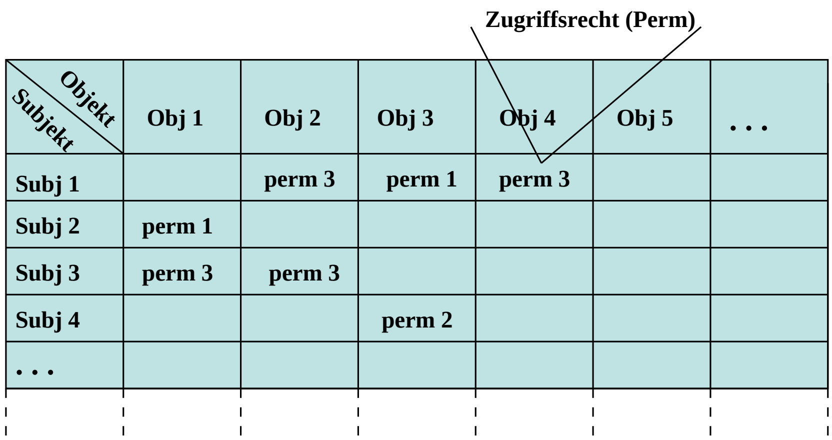


Abbildung 2: Zugriffskontrollmatrix

Das Konzept der Sicherheits- oder Zugriffsmatrix stützt sich dabei unter anderem auf die Voruntersuchungen von Dennis, van Horn, Graham, Denning, Jones und Lampson. Es wurde 1976 von Harrison, Ruzzo und Ullmann umfassend begründet [SUM 97].

In der Abb. 2 werden den Subjekten (s. 1. Spalte) Objekte (s. 1.Zeile) und die dazugehörigen Zugriffsrechte zugeordnet. Objekte können dabei Daten oder auch Funktionen sein [WAE 93].

Das Zugriffsmatrix-Modell hat jedoch auch Schwächen. Folgende Problematiken sind zu berücksichtigen [SUM 97]:

- **Berücksichtigung wertabhängiger Datenobjekte**
  Eine weitere Differenzierung muss zwischen wertabhängigen und wertunabhängigen Objekten getroffen werden. Der Zugriff auf ein wertabhängiges Objekt hängt von der Erfüllung einer auf den Inhalt bezogenen Bedingung ab. Werte können tatsächliche Wertefelder sein oder aber auch Schlüssel. Um den zusätzlichen Aufwand zu umgehen, können notfalls wertabhängige Objekte ausgeblendet werden. Der Zugriffsmatrix fehlt hier die Einbindung von dynamischen Komponenten.

- **Speicherung der Zugriffskontrollmatrix**
  Die Matrix könnte zeilen- oder spaltenweise gespeichert werden. Dabei hat die zeilenweise Speicherung den Vorteil, dass subjektbezogene Änderungen schnell zu realisieren wären. (Streichen oder Erweitern von Subjekten der Matrix). Jede Zeile stellt dabei den Befugnisumfang (Capability List) eines Subjekts dar.
  Der Vorteil einer spaltenweisen Speicherung ist eine einfachere Ausführung von Veränderungen in Art und Umfang der zu schützenden Objekte und der auf ihnen erlaubten Funktionen. Die Spalten drücken dabei den Berechtigungsumfang (Authority List) an auf dieses Objekt bezogenen Operationen aus. Beide Vorteile können jedoch vereint werden, indem man eine

simultane Abspeicherung vornimmt (s. [HAE 78]). Die physische Realisation erfolgt zumeist mithilfe von Bit-Ketten, deren Bits nur im Falle einer Zugriffsberechtigung gesetzt werden.

- **Die Zugriffskontrollmatrix implementiert eine benutzerbestimmbare Zugriffskontrollpolitik**
  Objekte, auf die Subjekte Zugriffsrechte besitzen, können kopiert werden, und somit anderen unberechtigten Subjekten zugänglich gemacht werden. Durch den Einsatz einer systembestimmten Zugriffskontrollpolitik kann auch hier eine höhere Vertraulichkeitsstufe erreicht werden.

Es existieren neben dem Zugriffsmatrix-Modell noch weitere Sicherheitsmodelle, die sich in der Erfüllung der unterschiedlichen Sicherheitsanforderungen unterscheiden.
Gegenüber dem Zugriffsmatrix-Modell wird z.B. durch das Bell-LaPadula-Modell die Vertraulichkeit der Information besser geschützt, da hier der Informationsfluss betrachtet wird [SUM 97]. Dabei werden oft Sicherheitsmodelle erweitert oder es werden neue Mechanismen und Techniken eingesetzt. So besteht beispielsweise ein weiterer Realisierungsansatz aus gerichteten Graphen. Ein Modell mit Einfluss der Graphentheorie ist z.B. das Nehmen-Gewähren-Modell des Bell-LaPadula-Modells oder das Gittermodell von Denning. Beide versuchen die Zugriffsmatrix mit den Schutzringmechanismen bzw. den Trust-Level-Implementierungen zu verbinden, um die bestehenden Mängel der Zugriffsmatrix zu reduzieren [BIS 90]. Ein Modell das die Vertraulichkeit stärker behandelt als die Integrität ist das Biba Integrity Model. Das Denning Information Flow Model betrachtet die Fliesrichtung der Information und löst dadurch das Problem der Sicherheitsklassifikationen, dadurch sind spezifischere Sicherheitsaussagen möglich [CLS 94, SUM 97]. Das Rushby Separability Modell geht näher auf ein Referenz Monitor Konzept ein, wobei eine Policy-Spezifikation des Monitors nötig wird. Das Take-Grant Modell ist ein Capability basiertes Modell indem ein Protection Graph eingesetzt wird.

Eine grundsätzliche Schwachstelle aller Modelle liegt jedoch in der einseitigen Berücksichtigung der Anforderungen [POW 93, VOS 95, SUM 97].

Sicherheitsmodell	Besonderer Aspekt
Bell LaPadula	Vertraulichkeit (hierarchisch)
Denning	Vertraulichkeit (Informationsfluss)
Biba	Integritaet (hierarchisch)
Goguen Meseguer	Nicht Beeinflussung
Clark Wilson	Integritaet: Aufgabenteilung
Chines. Mauer	Kombination (wahlfreie/festgelegte Zugriffskontrolle)
Terry Wismann	Vertraulichkeit und Integritaet

Abbildung 3: Einordnung der Sicherheitsmodelle in Anlehnung an [VOS 95]

Letztendlich ermöglicht kein existierendes Sicherheitsmodell die Modellierung anwendungsspezifischer Sicherheitspolitiken. Eine anwendungsspezifische Politikspezifikation müsste differenzierte benutzerbestimmbare Festlegungen und anwendungsspezifische Informationsflussbeschränkungen enthalten. Desweiteren müssten noch weitere Eigenschaften, wie Authentifikation und Auditing festgelegt werden [ECK 98].

2.5 Sicherheitspolitik

Die Sicherheitspolitik stellt die Sicherheitsrichtlinien eines Systems auf. Hierzu gehören die einzusetzenden Maßnahmen, aber auch die formelle Beschreibung der Autorisation von Subjekten auf Objekte in einer bestimmten Art und Weise zuzugreifen (Zugriffskontrollpolitik).

Formell stellt eine Zugriffskontrollpolitik (engl. Security Policy) eine Menge an Ausführungen dar, welche definiert, was erlaubt ist oder was nicht. Einfache Zugriffskontrollverfahren stellen die Zugriffskontrollpolitik durch z.B. Zugriffskontrolllisten oder Zugriffsrechte dar. Die Definition und Überprüfung der Policies wird durch die Größe und Anzahl der eingesetzten Zugriffskontrollmechanismen komplexer und schwieriger [FAL 00].

Die Zugriffskontrollpolitiken sollte von dem Durchsetzungsmechanismus unabhängig und getrennt sein. Dadurch sind sie nicht unnötig eingeschränkt. Zudem kann ein jeweils passender Mechanismus ausgewählt und so auch leicht ausgetauscht werden.

Betrachtet man Zugriffskontrollpolitiken, unterscheiden sie sich oft dadurch, wer die Zugriffskontrollpolitik bestimmt. Zugriffskontrollpolitiken können vom System (MAC), von einem Benutzer (DAC) oder von von beiden festgelegt sein.

Betrachtet man bestehende Sicherheitskonzepte, so kommt z.T. nur eine systemglobale Politik zum Einsatz. In diesem Fall können in der Regel jedoch die Sicherheitsanforderungen der spezifischen Anwendung und der vorhandenen Ausführungsumgebungen oft nicht erfüllt werden. Kommt nur eine systemglobale Sicherheitspolitik zum Einsatz, so wie beispielsweise in Java, stehen nur die Anforderungen der jeweiligen Ausführungsumgebung im Vordergrund [ECK 98].

2.5.1 Sicherheitsklassen (DAC - MAC)

Es existieren zwei unterschiedliche Klassen von Zugriffskontrollpolitiken, welche im Folgenden vorgestellt werden sollen [ECK 98, BRO 92, CLS 94, SUM 97, GAL 87]. Einerseits handelt es sich dabei um eine benutzerbestimmbare Zugriffskontrollpolitik (Discretionary Access Control (DAC)) und andererseits um eine systembestimmte Zugriffskontrollpolitik (Mandatory Access Control (MAC)).

Bei der benutzerbestimmbaren Zugriffskontrollpolitik (DAC) können Zugriffe beliebig autorisiert werden. Dabei ist durch Administrationsrechte festgelegt, wer welche Zugriffe autorisieren darf. In der Regel kann der Eigentümer des Objekts die Zugriffskontrollpolitiken seines Objekts bestimmen. Benutzerbestimmbare Zugriffskontrollpolitiken ermöglichen dem Eigentümer eines Objekts, Zugriffsrechte an dritte weiterzugeben. Wird ein Zugriffsrecht an eine dritte Entität weitergegeben, kann nicht kontrolliert werden, an wen die Informationen weitergeleitet werden.
Die benutzerbestimmbare Zugriffskontrollpolitik ist die gebräuchlichste Form der Zugriffskontrolle. Der Vorteil ist die feinere Granularität, die dadurch gegeben ist, dass jeder Eigentümer die gewünschten Zugriffsrechte auf seine Objekte selbst setzt.
Betrachtet man die systembestimmte Zugriffskontrolle, so stellt die benutzerbestimmbare Zugriffskontrolle keinen Ersatz, sondern eine Ergänzung dar. Eine systembestimmte Zugriffskontrolle wird so um feinere Einstellungen bezüglich der Zugriffskontrolle erweitert. Der Einsatz einer systembestimmten Zugriffskontrolle ist anzuraten, da eine alleinige benutzerbestimmbare Zugriffskontrolle verletzbarer ist.

Definition: DAC (nach [GAL 87, 5. DEFINITIONS]):
"A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject."

Ein potentielles Problem der benutzerbestimmbaren Zugriffskontrolle stellt das löschen von Subjekten und Objekten dar. Nach dem Löschen muss sichergestellt werden, dass die Referenzen eines gelöschten Objekts keinen ungewollten Zugriff auf ein neu erzeugtes Objekt ermöglichen. Dabei kommt es auf den eingesetzten Mechanismus an. Wird ein subjektbasierter Mechanismus eingesetzt, so gestaltet sich das Löschen der Objekte schwierig, im Gegensatz dazu ist das Löschen von Subjekten bei einem objektbasierten Mechanismus problematisch.

Wird zum Beispiel die Zugriffskontrollliste (ACL) als subjektbasierter Mechanismus verwendet, ist das Löschen von Objekten problemlos möglich, da die ACL mit dem Objekt verbunden ist und damit ebenfalls gelöscht wird. Beim Löschen eines Subjekts hingegen besitzen Zugriffskontrolllisten Einträge, die nicht mehr gültig sind, bzw. bei der Erzeugung eines neuen Subjekts mit demselben Namen z.B. dazu führen können, dass das Subjekt ungewollterweise autorisiert ist.

Die systembestimmte Zugriffskontrolle (MAC) regelt die systemglobalen Einstellungen. Die Regeln der Zugriffskontrolle, die festlegen, welche Zugriffe zugelassen sind, sind fest vorgegeben. Ein Implementierungsbeispiel ist z.B. das Multilevel Security Konzept. Es stellt klassifizierte Umgebungen (Trust-Levels) bereit, die unterschiedliche Sicherheitslevels darstellen. Eine Entität gehört jeweils einem Sicherheitslevel an. Die Sicherheitslevels bilden dabei eine Hierarchie, die auch die Zugriffsrechte regelt [GAL 87]. Es besteht eine zwingende, verbindliche Zugriffskontrolle (Mandatory Access Control). Ein weiteres Beispiel der Implementierung einer systembestimmten Zugriffskontrolle ist das Konzept des Ringschutzes (Ring Structures) [SAL 74] von MULTICS [SUM 97].

Die Sicherheitslevels können je nach Anwendungsart unterteilt sein, wobei sie Arbeitsbereiche bzw. schutzbedürftige Umgebungen widerspiegeln. Ein Beispiel einer systembestimmten Zugriffskontrollpolitik ist z.B. die Festlegung von zwei Sicherheitslevels: [secure] und [default]. Jeder Entität wird nun ein Sicherheitslevel zugeteilt. Beispielsweise könnte ein Benutzer A und die Datei "ausgaben.txt" dem Sicherheitslevel [secure] angehören. Auf die Datei können nun nur Benutzer des Sicherheitslevels [secure] zugreifen. Jeder Zugriff von einem Benutzer des Sicherheitslevels [default] scheitert. Betrachtet nun der Benutzer A die Datei, so kann er sie nicht an einen Benutzer des Sicherheitslevels [default] weiterreichen. Auch nicht, wenn er der Eigentümer der Datei ist.

Die systembestimmte Zugriffskontrolle bietet aus diesem Grund einen höheren Schutz. So schützen die systembestimmten Zugriffskontrollen z.B. auch vor Trojanischen Pferden. Hat z.B. ein Angreifer (Benutzer "B"), der dem Sicherheitslevel [default] angehört, dem Benutzer "A" ein Programm zur Ausführung gegeben, das zusätzlich zu seiner eigentlichen Aufgabe noch die Datei ausgaben.txt an Benutzer "B" sendet, so scheitert dies aufgrund des unterschiedlichen Sicherheitslevels. Wird ein MAC Mechanismus implementiert, so sind die Subjekte und Objekte mit Sicherheitsmarken zu kennzeichnen, welche die Zugehörigkeit der Entität ausdrückt [OPP 97].

2.5.2 Modularisierung in Teil-Politiken

Eine Sicherheitspolitik kann in einem großen Umfeld sehr schnell sehr umfangreich werden. Da die Verwaltung einer umfangreichen Sicherheitspolitik schwer zu realisieren ist, liegt es nahe, die Sicherheitspolitik in Teilpolitiken aufzuteilen. Die einzelnen Teilpolitiken sind weniger umfangreich und somit leichter zu verwalten. Zudem kann die Verwaltung der einzelnen Teilpolitiken delegiert werden. Es ist z.B. möglich, unabhängige Administrationsbereiche zu bilden, um so die Administration der Sicherheitspolitiken strukturieren zu können. Ein weiterer Vorteil besteht in der Wahl der Durchsetzungsmechanismen, da hier nun leicht für jede Teilpolitik ein spezifischer Mechanismus ausgewählt und eingesetzt werden kann.

Andererseits kann es auch vorkommen, dass aufgrund unterschiedlich eingesetzter Konzepte oder Mechanismen einzelne Teilpolitiken existieren, die jedoch von einer Stelle aus verwaltet werden sollen. Die Schwierigkeit bei der Verwaltung einzelner Teilpolitiken besteht in den oft unterschiedlich eingesetzten Policysprachen, den unterschiedlichen Einträgen und dem fehlenden Gesamtüberblick. So ist es beispielsweise nur sehr schwer möglich, Zusammenhänge der einzelnen Politiken zu erkennen bzw. nachvollziehen zu können. Wünschenswert ist, eine Gesamtpolitik über alle Teilpolitiken zu besitzen. Änderungen an dieser Gesamtpolitik wirken sich dann jeweils auf die einzelnen Teilpolitiken aus. Betrachtet man eine Gesamtpolitik, so ist es erforderlich, dass sich aus ihr einzelne Zugriffskontrollpolitiken für die jeweils eingesetzten Zugriffskontrollmechanismen extrahieren lassen, die dann an der richtigen Stelle durchgesetzt werden können.

Eine Möglichkeit, Teilpolitiken in eine Gesamtpolitik zu überführen, besteht in der Verknüpfung von Zugriffsrechtemengen [FAL 00]. Die Verknüpfung basiert dabei auf definierten Operatoren. Zugriffskontrollpolitiken können so konjunktiv und disjunktiv verknüpft werden. Die einzelnen Zugriffskontrollpolitiken können auf Teilmengen ihrer Attribute projiziert werden, wodurch nur die für die Entscheidungsfindung wichtigen Kriterien aus den beteiligten Zugriffskontrollpolitiken herausgefiltert werden.

2.5.3 Domänen

Oft will man Teilmengen von Ressourcen aus organisatorischen Gründen in Domänen gruppieren. Dabei muss festgelegt werden, wie eine solche Gruppe gebildet wird, wie man Zuständigkeiten zuordnet und wie man die Zuordnung dynamisch modifizieren kann. Die Domänenbildung ist eine Möglichkeit, die Ressourcen nach Policy-Gesichtspunkten zu strukturieren. Es existieren so z.B. Organisations- und Verwaltungsdomänen [HAN 99].

Organisationsdomänen (Functional Domains) sind Gruppierungen von einzelnen Komponenten, die nach ihren funktionellen Aufgaben zusammengefasst werden. Betrachtet man einzelne Funktionsbereiche, so können auch hier wiederum Komponenten zusammengefasst werden, damit z.B. für diese Komponenten eine gemeinsame Sicherheitspolitik durchgesetzt werden kann.
Die Einteilung in Verwaltungsdomänen (Administrative Domains) wird vorgenommen, um Domänen mit genau einer Verwaltungsautorität zu beschreiben [HAN 99].

Die einzelnen Domänen können dabei als Schutzdomänen aufgefasst werden. In verteilten Systemen können Administratortätigkeiten auch in zentrale und in lokale Schutzdomänen aufgeteilt werden. Zentrale Sicherheitspolitiken (systemspezifische Sicherheitspolitiken) können auf einer Unternehmens-Ebene getroffen werden, lokale (anwendungsspezifische Sicherheitspolitiken) jeweils auf der betreffenden lokalen Ebene.
Zentrale Sicherheitspolitiken und die jeweiligen dazugehörenden Operationen müssen demnach zentral spezifiziert werden. So können hier z.B. spezifische Rollen und ihre enthaltenen Zugriffsrechte definiert werden. Oder es können Zugriffsrechte an bestimmte Subjektgruppen vergeben werden. Die Gewährung bzw. Entziehung der Zugehörigkeit der spezifischen Rollen bzw. die Zuteilung der Subjekte in Gruppen könnten dann von den Administratoren auf der lokalen Seite vorgenommen werden.

Eine Sicherheitspolitik ist immer für einen bestimmten Bereich zuständig. Um Bereiche abstecken zu können, eignet sich die Einteilung in unterschiedliche Domänen. Finden domänenübergreifende Zugriffe statt, nimmt die Komplexität der Zugriffskontrolle zu. Hier kann es vorkommen, dass Zugriffskontrollpolitiken von zwei bzw. mehreren Parteien durchgesetzt werden sollen [FAL 00].

Müssen mehrere Sicherheitspolitiken durchgesetzt werden, kann es zu Unstimmigkeiten und nicht auflösbaren Konflikten kommen. Diese Konflikte können z.B. durch eine domänenübergreifende Sicherheitspolitik (Metapolicy), die die beteiligten Politiken widerspiegeln, beseitigt werden. Dabei stellt die domänenübergreifende Politik eine Politik der Teilpolitiken dar. Die Metapolicy muss dabei die jeweils in den Teilpolitiken eingesetzten Policysprachen verstehen und umsetzen können.

Um mehrere Sicherheitspolitiken (Teilpolitiken) durchsetzen zu können, existieren unterschiedliche Verknüpfungsmöglichkeiten:

- Interoperation
  Bei einer Interoperation werden alle Teilpolitiken erfüllt.
- Kombination
  Bei der Kombination der Teilpolitiken ist die letztendlich durchgesetzte Sicherheitspolitik

nicht unbedingt konsistent gegenüber den Teilpolitiken.

Die Verknüpfung der Sicherheitspolitiken kann entweder direkt definiert sein, wobei die enthaltenen Zugriffsrechte definiert sind, oder es handelt sich um eine Sicherheitspolitik, die aus verknüpften, bereits definierten Sicherheitspolitiken besteht.

Betrachtet man die Einträge innerhalb der Sicherheitspolitiken, so besteht ein Eintrag zumeist aus [FAL 00]:

- einer Menge von Subjekten,
- einer Menge von Objekten
- einem Prädikat über die Subjekte und Objekte
- einer Aussage, ob der Zugriff erlaubt oder verboten ist

Betrachtet man nun die Verknüpfungsmöglichkeiten der einzelnen Sicherheitspolitiken, so existieren folgende Möglichkeiten:

- die Verknüpfung der Sicherheitspolitiken ist verboten
- das Ergebnis der Verknüpfung ist die Schnittmenge
- das Ergebnis der Verknüpfung ist die Vereinigungsmenge
- die Prädikate können disjunktiv oder konjunktiv verknüpft werden

Zwei Einträge der Sicherheitspolitiken, die sich nicht widersprechen, können so verknüpft werden, dass die resultierende Sicherheitspolitik die Zugriffe

- erlaubt, falls
  - mindestens ein Eintrag der beiden Sicherheitspolitiken den Zugriff erlaubt
  - der Eintrag der einen und der anderen Sicherheitspolitik den Zugriff erlaubt
- verbietet, falls
  - mindestens ein Eintrag der beiden Sicherheitspolitiken den Zugriff verbietet
  - der Zugriff bei beiden Sicherheitspolitiken verboten ist

Letztendlich gilt es einen Weg zu finden, die einzelnen Teilpolitiken zu verbinden ohne die Informationssicherheit zu gefährden.

2.5.4 Formale Sicherheitspolitik

Die in dem Kapitel 2.5 geforderte formelle Beschreibung der Autorisation muss einem formalen Modell zugrundeliegen. Das formale Modell beschreibt die für die Entscheidungsfindung relevanten Kriterien und wie Einträge innerhalb der Sicherheitspolitik aussehen müssen und was sie bedeuten. In dem formalen Modell muss die Syntax der eingesetzten Sprache festgelegt werden. Einträge in die Sicherheitspolitik müssen den Syntaxregeln entsprechen, damit sie verstanden und durchgesetzt werden können.

In dem Modell müssen weitere Eigenschaften bzw. Bedingungen definiert werden, wie z.B.:

- Ausführungen, die in der Menge enthalten sind, werden akzeptiert
- Ausführungen, die nicht in der Menge sind, werden nicht akzeptiert

Wichtig ist hier, wie bestimmte Ausführungen beschrieben werden können, um innerhalb der Sicherheitspolitik adressiert werden zu können. Die Menge der Ausführungen wird in Regeln definiert. Die Regeln bilden dabei eine Liste von Subjekten auf eine Liste von Objekten auf eine bestimmte Anzahl von Aktionen ab. Mithilfe dieser Regeln können dann z.B. Restriktionen der Aktionen von Subjekten auf Objekte ausgedrückt werden.

Die Beschreibung der Sicherheitspolitik anhand individueller Subjekte und Objekte ist im Allgemeinen nicht praktikabel. Zumeist ist die Anzahl der beteiligten Entitäten zu groß bzw. noch unbestimmt und kann daher nicht erfasst werden. Andererseits ist dadurch auch der Verwaltungsaufwand zu hoch, da jedes spezifische Subjekt innerhalb der Sicherheitspolitik benannt werden muss. Durch Gruppierung und Zuordnung von Personen zu Aufgaben (Rollenverteilung) wird die Komplexität stark verringert.

2.6 Komponenten der Zugriffssteuerung

Die Zugriffssteuerung ist verantwortlich für die Informationssicherheit des Systems. Sie ist zuständig für die Identifikation und Authentifikation der Subjekte, die Rechtevergabe, die Rechtedurchsetzung und u.U. auch für die Protokollierung und Auswertung der Zugriffsversuche. Ein wichtiges Kriterium der Zugriffssteuerung ist die Granularität, also der Feinheitsgrad und die Differenzierbarkeit des Zugriffsschutzes.

Folgend soll der Ablauf eines Zugriffs erläutert werden. Anhand der vergebenen Zugriffsrechte werden während der Interaktion zwischen einem Subjekt und einem Objekt die getätigten Zugriffe auf ihre Berechtigung hin überprüft (Rechteprüfung (engl. Access Control)).

1. Die Entität bekommt zuvor Zugriffsrechte zugeteilt (Autorisation)
2. Bei einer Anfrage wird die Entität authentifiziert (Authentifikation)
3. Daraufhin kann die Zulässigkeit der Operation überprüft werden (Zugriffskontrolle)

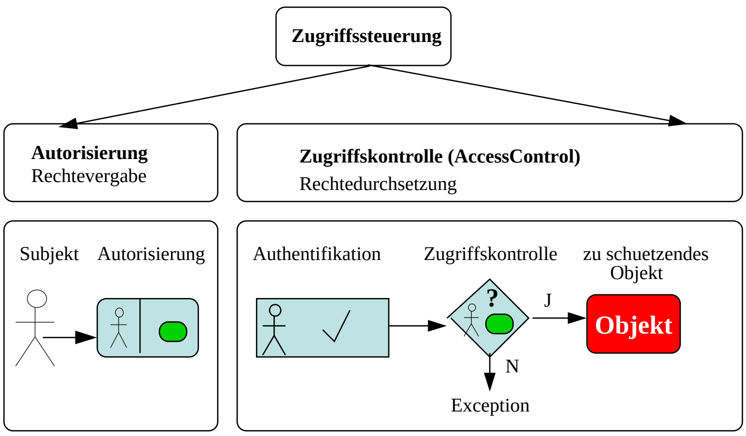


Abbildung 4: Ablaufzyklus einer Zugriffskontrolle

Betrachtet man die einzelnen Punkte, so sind der Zugriffssteuerung noch weitere Aufgaben und Prozesse zuzuordnen (in Anlehnung an [CLS 94]).

Autorisation (Rechtevergabe)

1. Zuteilung der Rechte (Autorisation)
   - wer bekommt Rechte?

- welches Aussehen haben diese Rechte?
- welche Zugriffsform (Operationen) geben diese Rechte?
- soll anhand statischer/dynamischer Kriterien entschieden werden?
- spielt der Zustand eines Objekts eine Rolle?

2. Administration der Rechte
   - wer darf autorisieren?
   - welche Zugriffe dürfen autorisiert werden?
   - Aufteilung der Autorisation (Aufgabenteilung)

3. Rücknahme der Rechte

Authentifikation

1. Identifikation und Authentifikation des Nutzers durch
   - Wissen (Passwort)
   - Besitz (Ticket)
   - Signatur und Zertifikate
   - Ursprung (Code-Base)

Zugriffskontrolle (Rechtedurchsetzung)

1. Vermeidung unautorisierter Zugriffe
   - Schutz vor unautorisierten Zugriffen
   - Schutz vor Missbrauch der Information
   - Schutz vor z.B. Trojanischen Pferden, Back Doors, ...

2. Monitoring der Zugriffe, da
   - Zugriffskontrollen z.T. weggelassen werden
   - Identifikations- und Authentifikationsprozesse angreifbar sind
   - Ressourcen-Limitierungen kontrollierbar sein sollen
   - bei einem Zwischenfall Aufzeichnungen hilfreich sind
   - Anomalien feststellbar sind

2.6.1 Identifikation und Authentifikation

Eine Entität muss innerhalb eines Systems eindeutig identifizierbar sein. Anhand der Entität können dem Subjekt Zugriffsrechte zugeteilt werden.
Die Identifizierung reicht alleine jedoch nicht aus. Da die Informationssicherheit von dieser Identifizierung abhängt, ist es erforderlich die ermittelte Identität zu verifizieren. Nachdem sichergestellt wurde, dass es sich wirklich um die angegebene Identität handelt, handelt es sich um eine authentifizierte Identität.
Die Authentifikation eines Benutzers oder eines Programms, welches einem Benutzer zugeordnet ist, erfolgt vor der ersten Interaktion mit dem System. Durch die eindeutige Identifikation und Verifikation des zugreifenden Subjekts kann die Berechtigung des Subjekts ermittelt werden.
Die Aufgabe der Identifikation und Authentifikation übernimmt dabei die Zugangskontrolle. In Netzwerken kann dabei zwischen der zentralen und der lokalen Zugangskontrolle unterschieden werden. Die lokale Zugangskontrolle bezeichnet dabei den Zugang zu dem Rechensystem. Die zentrale Zugriffskontrolle wird von i. Allg. von dem Netzwerkbetriebssystem vorgenommen, wobei die Nutzung bestimmter Netzdienste z.B. durch die Eingabe der Benutzerkennung und eines Passworts ermöglicht wird [VOS 95, OPP 97].

In der Zugangskontrolle ist das Passwortkonzept am weitesten verbreitet. Sicherheitshardware, Memory- und Smart-Cards kommen nur in speziellen Anwendungsbereichen wie Home-Banking zum Einsatz. Zur Verifikation der Identität und der Integrität kommen klassische Techniken zum Einsatz. So z.B. digitale Signaturen und Zertifikate [ECK 98].

2.6.2 Zugriffsrechte

Korrekt definierte Zugriffsrechte sind unabdingbar für die Sicherheit. Durch die Zugriffsrechte wird festgelegt wie ein System genutzt werden kann. Zugriffsrechte müssen definiert und überprüft werden.

Es existieren verschiedene Formen von Zugriffsrechten. Die Wahl der zu verwendenden Zugriffsrechte hat einen großen Einfluss auf die Datenintegritätseigenschaften. Es gibt sowohl universelle Zugriffsrechte als auch objektspezifische Zugriffsrechte [ECK 98]. Typische universelle Zugriffsrechte sind Read (Lese-), Write (Schreibe-), Execute (Ausführ-) und Delete (Lösch-) Rechte. Universelle Zugriffsrechte beinhalten nur eine Aktion und nicht das Objekt auf das Zugegriffen wird. Sie können daher keine objektspezifische Operationen bezeichnen. Durch die Vergabe von universellen Zugriffsrechte werden den Subjekten große Freiheitsgrade gewährt.
Ein Pendant dazu stellen die spezifischen Zugriffsrechte dar. Innerhalb der spezifischen Zugriffsrechte werden ebenso die Objekte adressiert. Durch objektspezifische Zugriffsrechte ist es möglich, Zugriffsrechte zugeschnitten auf die spezifische Aufgabe des Subjekts zu vergeben.

Die Zugriffsrechte entsprechen den Operationen, die Subjekte auf Objekte (allgemein oder spezifisch) ausführen.

Betrachtet man das Schreibrecht näher, so sind noch weitere feingranularere Zugriffsrechte denkbar. An diesem Beispiel wird auch deutlich, dass je feingranularer Zugriffsrechte sind, desto eher eine anwendungsspezifische Vergabe von Zugriffsrechten möglich ist [GAL 87].

Write Append
: Hier darf nur etwas an das Objekt angehängt werden.

Write Change
: Objekte dürfen verändert, aber nicht gelesen oder erweitert werden.

Write Update
: Hier darf der Inhalt geändert werden. Das hinzufügen, löschen oder ansehen ist nicht erlaubt.

Write
: Hier darf der Inhalt des Objekts modifiziert, gelöscht oder etwas hinzugefügt werden. Der Inhalt kann nicht angesehen werden.

Administrationsrechte
Ein besonderes Zugriffsrecht ist das Administrationsrecht. Administrationsrechte existieren zumeist bei benutzerbestimmten Zugriffskontrollpolitiken (DAC), wobei hier der Eigentümer des Objekts das Administrationsrecht auf sein Objekt besitzt. Nur der Besitzer des Administrationsrechts auf das Objekt kann Zugriffe auf das Objekt autorisieren bzw. widerrufen.

In einer rollenbasierten Zugriffskontrolle könnte das Administrationsrecht in einer Administrationsrolle wiedergespiegelt sein. Für unterschiedliche Administrationsaufgaben könnten auch unterschiedliche Administrationsrollen existieren, die jeweils die für die Erledigung der Aufgabe notwendigen Berechtigungen beinhalten. Dadurch ist eine feinere Einteilung in Administrationsumgebungen möglich. Eine eventuell vorhanden Rollenhierarchie integriert dabei die Vorteile einer systembestimmten Zugriffskontrolle (MAC).

2.6.3 Rechtevergabe

Die Rechtevergabe hat die Aufgabe der Vergabe und der Rücknahme von Zugriffsrechten. Wird einem Subjekt ein Zugriffsrecht erteilt, so spricht man hier von einer Autorisation des Subjekts.
Die Rechtevergabe (Autorisation) sollte ein semantisches Konzept darstellen, welches von systemspezifischen Mechanismen getrennt ist. Zur Repräsentation eines unabhängigen Konzepts

wird eine Sprache benötigt, die den Autorisationsanforderungen genügen. Die Sprache muss eine formale Semantik besitzen, so dass die Bedeutung einer Autorisationsanforderung präzise festgelegt und überprüft werden kann [WOL 92].

Die Rechtevergabe hat enorme Auswirkungen auf die Sicherheit eines Systems. Um die Sicherheit nicht zu gefährden müssen bei der Autorisation folgende Punkte beachtet werden [FAL 00]:

- **Verständlichkeit**
  Um den Administrationsaufwand möglichst gering zu halten und Fehlkonfigurationen möglichst auszuschließen, ist die Verständlichkeit (Umfang, Darstellung) der Autorisation wichtig.

- **Integrierte Verwaltung**
  Eine komplexe heterogene Umgebung mit unterschiedlichen Zugriffskontrollkomponenten bedarf einer integrierten Verwaltung aller Komponenten, um eine hohen Administrationsaufwand zu mindern. Alle Komponenten sollten in einem Sicherheitsmodell untergebracht werden.

- **Aufgabenteilung**
  Die Administration sollte dennoch auf mehrere Administratoren aufgeteilt werden können. Verteilte und autonome Administrationsstrukturen sollten realisierbar sein.

- **Überprüfbarkeit**
  Die Umsetzung der Sicherheitsmaßnahmen müssen regelmäßig überprüft werden. Wobei hier eine Gesamtsicht der Autorisationen erforderlich ist.

- **Durchsetzbarkeit**
  Die Autorisationsanforderungen müssen durchsetzbar sein.

In der Literatur finden sich noch weitere Anforderungen die erfüllt werden müssen, um bei den Autorisationsvorgänge die Informationssicherheit nicht zu gefährden [BRO 92]:

- **Vollständigkeit**
  Alle relevanten Subjekte und Objekte müssen erfasst werden.

- **Überschaubarkeit**
  Inkonsistenzen und Widersprüche sind zu vermeiden.

- **Verfügbarkeit der Entscheidungsdaten**
  Das fehlen von Entscheidungsdaten darf nicht zu undefinierten Zuständen oder zu einem Umgehen der Zugriffsrechte führen.

- **Schutz vor Modifikation der Benutzerprofile**

- **Bindung der Zugriffsrechte an Inhalte**

Je nach Anforderung können dabei auch unterschiedliche Strategien zum Einsatz kommen. Die Vergabe von Zugriffsrechten kann entweder die Strategie des expliziten Verbietens verfolgen, d.h. alle Zugriffsarten bis auf die verbotenen sind erlaubt oder aber auch die Strategie des expliziten Erlaubens, d.h. alle Zugriffe bis auf explizit erlaubten werden unterbunden [ECK 98].

Subjekt- ,objekt- und anwendungsspezifische Zugriffsrechte
Die Rechtevergabe vergibt Zugriffsrechte im Idealfall nach dem Least-Privilege Konzept. Dies beinhaltet die objekt- und die subjektspezifische Rechtevergabe. Unter einer subjektspezifischen Rechtevergabe versteht man dabei die Möglichkeit der Vergabe von Zugriffsrechten an ein einzelnes spezifisches Subjekt. Können einzelne Subjekte individuell autorisiert werden, wird durch den Einsatz von objektspezifischen Zugriffsrechten (vgl. Kapitel 2.6.2) eine anwendungsspezifische

Autorisation ermöglicht. Bei einer anwendungsspezifischen Autorisation werden den spezifischen Subjekten nur die Zugriffsrechte zugeteilt, die sie für die Bearbeitung der Aufgabe benötigen. Diese Minimalzuweisung von Zugriffsrechten ist jedoch praktisch schwer zu realisieren da sich die dafür notwendigen Informationen in der Regel dynamisch ändern. Die meisten Konzepte sind jedoch statischer Art.

Autorisationsmethoden
Es existieren unterschiedliche Methoden der Autorisation. Subjekt- oder objektbasierte Methoden sind vor allem bei Betriebssysteme verbreitet. Die sprachbasierte-Methode ist eine Methode, welche die Autorisation von Subjekten über eine Zugriffskontrollpolitik realisiert.

Objektbasierte Mechanismen Objektbasiert bedeutet im Kontext der Zugriffsmatrix, dass die Zugriffsmatrix Spaltenweise repräsentiert wird. Dabei haften die Rechte an dem Objekt. Beispiele hierfür sind:

- **Zugriffskontrolllisten (Access Control Lists)** *(Objekt/Spalten) jedes Objekt besitzt eine Liste mit Subjekten*
  Eine Zugriffskontrollliste wird an das Objekt angeheftet. Durch die Verwendung von Wildcards kann dabei die Größe der Zugriffskontrolllisten beeinflusst werden. Folgende ACL-Features existieren zusätzlich zur Unterstützung der Benutzer [GAL 87]:

  - Default-ACLs
    Die Default-ACL wird beim Erstellen eines neuen Objekts kopiert, bzw. es zeigt ein Zeiger auf eine Default-ACL. Die Default-Einträge erteilen dabei zumeist rudimentäre Berechtigungen.
  - Named-ACLs
    Die ACL besitzen spezifische Namen, wodurch bei der Vergabe einer ACL an ein Objekt der Name der ACL herangezogen werden kann. Die benannte ACL wird entweder kopiert, oder ein Pointer zeigt auf die Named-ACL. Einträge müssen somit nicht jedesmal neu vorgenommen werden.

  Die ACLs sind der flexibelste und gebräuchlichste DAC-Mechanismus. Die Implementierung der ACLs kann so erfolgen, dass jede DAC-Anforderung (vgl. [GAL 87]) erfüllt werden kann. Durch Zugriffskontrolllisten kann jeder Zugriff eines spezifischen Benutzers auf ein spezifisches Objekt kontrolliert werden. Zugriffskontrolllisten kommen unter anderem in neueren UNIX-Systemen, in Microsoft Windows NT und in MULTICS zum Einsatz [SUM 97]. Die Verwendung von Zugriffskontrolllisten in verteilten Systemen ist aufgrund der hohen Dynamik der Subjekte jedoch problematisch, da hier der Verwaltungsaufwand hoch und eine mangelnde Skalierbarkeit vorhanden ist [ECK 98].

- **Protection Bits** *(Objekt/Spalten) jedes Objekt hat gesetzte Bits für Subjekt-Gruppen*
  Die Protection Bits sind am Objekt angeheftet. Anstatt einer Liste von Nutzern werden Protection Bits für jedes Objekt gesetzt. In UNIX kann man z.B. Protection Bits für alle (all), Gruppen (Groups) und Besitzer (Owner) setzen. Änderungen der Protection Bits kann nur der Besitzer (Owner) oder der Superuser vornehmen [GAL 87].

  Bei den Protection Bits handelt es sich nicht um eine vollständige Implementierung der Zugriffsmatrix. Ein Zugriffsrecht kann nicht einem einzelnen Benutzer zugeteilt oder entzogen werden.

Subjektbasierte Mechanismen Im Kontext der Zugriffsmatrix bedeutet dies eine spaltenweise Repräsentation der Zugriffsmatrix. Die Rechte haften an dem Subjekt. Subjektbasierte Mechanismen besitzen kein effizientes Mittel, um Aussagen zu können, welcher Nutzer Zugriff auf ein bestimmtes Objekt besitzt. Desweiteren bestehen noch Implementierungsschwierigkeiten bei der Rücknahme von vergebenen Zugriffsrechten und die Implementierung von Gruppenrechte.

- **Capabilities** *(Subjekt/Reihen) jedes Subjekt besitzt Capabilities*
  Capabilities spezifizieren die Zugriffsrechte deren Besitzer auf ein innerhalb der Capabilities spezifizierten Objekte. Dabei sind Capabilities innerhalb einer bestimmten Domain gültig. Damit ausgeschlossen werden kann, dass Capabilities unrechtmäßig kopiert oder erzeugt werden, kann nur eine Kontrollinstanz Capabilities erzeugen oder ändern [GAL 87].

  Einzelne Programme können Capabilities besitzen. Capabilities können aber auch z.B. in einer Datei gespeichert sein. Sie werden dabei durch Hardware- oder Softwaremechanismen bzw. durch Verschlüsselungsmechanismen geschützt. Ein Capability kann unter Umständen auch weitergereicht werden und so anderen Subjekten den Zugriff auf spezifische Objekte ermöglichen. Die Delegation von Zugriffsrechten ist somit möglich. Durch Capabilities kann das Least-Privilege Prinzip erfüllt werden, wobei zudem eine sich dynamisch ändernde Umgebung unterstützt wird [SUM 97].

  Durch die Möglichkeit, Capabilities weitergeben zu können, ist es jedoch sehr schwer bzw. unmöglich bestimmen zu können, wer gerade Zugriff auf ein spezifisches Objekt hat. Dies macht die Implementierung einer vollständig benutzerbestimmbaren Zugriffskontrollpolitik (DAC-Implementierung) nicht einfach. Untersuchungen zur Implementierung eines vollständigen DAC-Mechanismus mittels Capabilities finden sich in "An augmented capability architecture to support lattice security and traceability of access" [HEK 84]. Die Rücknahme von Zugriffsrechten ist ein weiteres Problem bei der Verwendung von Capabilities. Dennoch gibt es auch hier Lösungsansätze, so behandelt der Artikel von D. Redell "Revokable Capabilities" [RED 74]. Schwierigkeiten bestehen auch bei der präzisen Definition und Implementierung von Gruppen [GHW 86]. Eine reine Capability-basierte Methode ist aufgrund dieser Nachteile in der Regel ungeeignet.

- **Profiles** *(Subjekt/Reihen) jedes Subjekt besitzt Profiles*
  Jeder Benutzer besitzt eine Liste. In der Liste sind die Objekte enthalten, auf die der Benutzer bestimmte Zugriffsrechte besitzt. Damit der Objektname nur ein spezifisches Objekt adressiert, müssen alle Objektnamen eindeutig sein. Auch hier gestaltet sich die Rücknahme von Zugriffsrechten schwierig, falls das Benutzerprofil nicht automatisch bei jedem Zugriff auf ein Objekt überprüft wird. Besitzt ein Benutzer Zugriffsrechte auf viele Objekte, kann auch hier das Benutzerprofil sehr groß werden [GAL 87].

- **Passwords** *(Subjekt/Reihen) jedes Objekt sichert den Zugriff mit einer Passwortabfrage*
  Jeder Benutzer hat ein Passwort für jedes Objekt. Dieser Mechanismus ist dem der Capabilities ähnlich. Hier ist das Passwort das Ticket zum Objekt [GAL 87]. Beispielimplementierung finden sich mit RACF (Ressource Access Control Facility) für das Betriebssystem MVS von IBM [IBM 83, IBM 99a] und in CDC's NOS [CDC 83]. In der Steuer-Datei von RACF stehen die Daten der zu schützenden Ressourcen und Nutzerbeschreibungen, die die Art und den Umfang der Zugriffsberechtigung einzelner Nutzer spezifizieren. Dazu gehört das Nutzerkennzeichen, ein verschlüsselt geführtes Kennwort sowie Nutzungsattribute. Autorisierte und unautorisierte Zugriffe werden geeignet aufgezeichnet. Mittels RACF kann eine strenge Trennung der Zugriffsrechte auf z.B. Datenbanken realisiert werden, so dass jeder Nutzer nur auf die für ihn benötigten Daten zugreifen kann.

  Durch die große Anzahl von Objekten gestaltet sich jedoch die Passwortverwaltung schwierig, zudem muss für jedes spezifisches Objekt, auf das zugegriffen werden soll, ein neues Passwort verwendet werden, womit ein geeignetes Konzept der Passwortverwaltung benötigt wird.

  SECURE, eine für IBM-Betriebssysteme entwickelte Software, implementiert ebenso einen Passwortschutz für jede Datei in sogenannten Dateiprofilen. Dabei sind für jede Datei mehrere Passwörter zulässig, an denen jeweils abgestufte Zugriffsrechte geknüpft werden können [WAE 93]. Passwörter müssen hier nicht per Hand eingegeben werden und sind dem

Benutzer sogar unbekannt. Sie werden aus Elementen der Ordnungsmerkmale der Anwendung bzw. des Jobs, wie Auftragsnummer, Programm-, Programmier- und Dateiname vom Sicherungssystem selbst gebildet. Trojaner haben hier keine Chance mehr.

- **Attribut Zertifikate (Privilege Management Extensions (PKI))**
  Eine weitere Autorisationsmethode ist die Verwendung von Attribut Zertifikaten (Privilege Management Extensions (PKI)). Durch die Verwendung von Attribut Zertifikaten (attribute certificates) wird die Public Key Infrastruktur erweitert. Benutzer können sich durch die Übertragung von Attribut Zertifikaten bei Anwendungen autorisieren.

  Die durch existierende PKI (z.B. PGP, MISSI, PEM, etc.) authentifizierten Entitäten übertragen ihre Autorität durch ein mit ihrem privaten Schlüssel signierten, speziell designten Attribut Zertifikat. Das Attribut Zertifikat enthält kein Schlüsselmaterial, also beispielsweise keinen öffentlichen Schlüssel (public key), sondern einen Fingerabdruck (Message Digest) der Anwendung, der Privilegien und der Sicherheitspolitik. ANSI X9 und die IETF haben hier Versuche zur Standardisierung gestartet.

  Wie drückt man die Privilegien und die Policies im Zertifikat aus? Die Syntax muss durch Maschinen verarbeitbar sein, gut genug um reale Welt Privilegien und Policies aufnehmen zu können, jedoch benutzerfreundlich. Privilegien können z.B. durch **privilege= attribute set** notiert werden. Sicherheitspolitiken sind hingegen schwieriger zu notieren, da der Schutz, den eine Anwendung bei der Ausübung ihrer Aktivitäten besitzen soll, ausgedrückt werden muss.

Nachfolgend werden Vor- bzw. Nachteile der subjekt- bzw. objektbasierten Methoden betrachtet. Die Aussagen sind als Diskussionsgrundlage gedacht, da je nach Betrachtungsweise ein Nachteil auch ein Vorteil sein kann.

Die Vorteile der subjektbasierten Methoden liegt in der feineren Adressierungsmöglichkeit der Subjekte. So kann hier z.B. explizit einer spezifischen Person ein Zugriffsrecht erteilt werden. Auch bei einer sich ständig dynamisch änderten Anzahl von Subjekten ist die subjektbasierte Methode vorzuziehen. Bei einem Zugriff auf ein Objekt ist in der Regel auch die Kontrollzeit geringer, da i. Allg. die Zugriffsrechte eines Subjekte stark begrenzt sind und somit keine große Liste oder ähnliches durchsucht werden muss. Durch subjektbasierte Methoden können weiter auch dynamische Domains unterstützt werden, so dass Zugriffsrechte nur für bestimmte Domänen gültig sind. Betrachtet man die Umgebung, so unterstützen subjektbasierte Methoden eine sich dynamisch ändernde Umgebung. Dadurch dass die Zugriffsrechte an den Subjekten haften, ist in der Regel auch eine Delegation der Zugriffsrechte leichter zu realisieren.

Die Nachteile einer subjektbasierten Methode liegen in der Problematik der Rücknahme von Zugriffsrechten. Desweiteren müssen auch die Objektnamen eindeutig sein. Nach einer Änderungen der Objektnamen müssen so Subjekte oft neu autorisiert werden. Sind außerdem sehr viele Objekte vorhanden, sind subjektbasierte Methoden oft unhandlich. Ein Defizit stellt auch die Vergabe von Zugriffsrechten an Gruppen dar. Aus der Administrationssicht sind subjektbasierte Methoden weniger überschaubar. Aussagen wie "wer hat auf das Objekt X Zugriff", sind nahezu unmöglich. Die Vergabe, das Löschen und das Ändern von Zugriffsrechten ist ebenso schwierig. Betrachtet man z.B. Capability basierte Systeme, so existieren noch der Kritikpunkt des teuren Aufrufs. Die Systemperformance und das Einsperren der Privilegien sind problematisch. Entweder muss die Hardware Tagged Memory bereitstellen, oder der Aufruf läuft über einen Umweg zwischen dem Prozess und den Capabilities.

Betrachtet man objektbasierte Methoden, so liegen die Vorteile in der feingranularen Adressierungsmöglichkeit der Objekte. Da die Zugriffsrechte direkt an den Objekten haften, ist so die Vergabe der Zugriffsrechte für einzelne Objekte möglich. Ändert sich die Anzahl der Objekte bzw. ändern sich die Objekte häufig, so ist eine objektbasierte Methode gegenüber einer subjektbasierten Methode vorzuziehen. Auch die Rücknahme der Zugriffsrechte gestaltet sich hier einfacher. Die Vergabe von Zugriffsrechten an Gruppen von Subjekten ist leicht möglich, zudem können jederzeit Aussagen gemacht werden, wer gerade Zugriffsrechte auf dieses Objekt besitzt. Sollen Zugriffsrechte vergeben, geändert oder gelöscht werden, stellt dies bei einer objektbasierten Methode in der Regel kein Problem dar, da die Zugriffsrechte zentral zugänglich sind.

Nachteilig wirkt sich eine objektbasierte Methode oft bei der Granularität der Adressierung der Subjekte aus. So kann hier in der Regel nicht explizit nur eine Person berechtigt werden. Findet ein Zugriff statt, so sind die Kontrollzeiten oft sehr groß, da viele Subjekte Zugriff auf ein Objekt haben und somit eine große Anzahl von Subjekteinträgen durchsucht werden muss. Zudem können bei der objektbasierten Methode durch komplexe Wechselwirkungen zwischen den Objekten Widersprüche bezüglich der Rechtevergabe entstehen.

Zugriffskontrollpolitik Die Zugriffskontrollpolitik ist ein Regelwerk, welches die Zugriffe auf Objekte regelt. Innerhalb der Policy werden spezifischen Subjekten durch den Einsatz einer wohldefinierten Policysprache Zugriffsrechte zugeteilt.
Der Vorteil der Autorisation anhand einer Policy liegt darin, dass Zugriffskontrollentscheidungen anhand vorhandener Kriterien getroffen werden können. Werden Entscheidungen anhand vorhandener Kriterien getroffen, müssen keine neue Kriterien (wie z.B. Protection Bits, ...) eingeführt werden.
Erfolgt die Spezifizierung anhand einer geeigneten Policysprache, gestaltet sich die Rechtevergabe flexibler, unabhängiger und ausdrucksstärker. Subjekte und Objekte können je nach Policysprache feingranular adressiert werden.
Mischformen der Autorisation durch einen sprachbasierten Ansatz und einer subjekt- bzw. objektbasierten Methode vereinen die Vorteile beider Seiten. So könnte ein Kriterium einer Policy ein Capability oder ein Eintrag in einer Zugriffskontrollliste (ACL) sein.

Delegation von Zugriffsrechten Die Delegation von Zugriffsrechten ist eine weitere Methode der Autorisation. Durch die Integration der Delegation von Zugriffsrechten sind Subjekte in der Lage, die ihnen zugeteilten Berechtigungen an weitere Subjekte zu übertragen und damit zu autorisieren. Vorteile der Delegation von Zugriffsrechten ist, dass ein Subjekt seine Aufgabe an weitere Subjekte delegieren kann, die dann auch die benötigten Berechtigungen besitzen, um die Aufgaben abarbeiten zu können. Die Delegation von Zugriffsrechten setzt gewissermassen ein Zugriffskontrollmodell voraus, welches die gewünschte Weitergabe von Berechtigungen allgemein zulässt.

Die Delegation kann erfolgen durch [FAL 00]:

1. **explizite Weitergabe von Privilegien an Subjekte** (bei subjektbasierten Methoden)
   Das Subjekt gibt explizite Privilegien an Subjekte weiter. Diese können z.B. in Form von Capabilities oder Tickets vorliegen. Das Subjekt kann nur diese Privilegien weitergeben, die es selbst besitzt. Erhält ein Subjekt ein Privileg, ist der Zugriff dem Privileg entsprechend erlaubt.

2. **Eintragungen in den für die Verwaltung der Zugriffsrechte vorgesehenen Mechanismen** (bei objektbasierten Methoden)
   Die Delegation müsste durch Operationen erfolgen die Einträge z.B. in einer Access Control List (ACL) oder einer Zugriffsmatrix vornehmen. Diese Operationen dürften wiederum nur berechtigte Subjekte ausführen.

3. **Vergabe von Administrationsrechten auf die betreffenden Objekte**
   Bekommt ein Subjekt die Administrationsrechte eines Objekts zugesprochen, kann das Subjekt die nötigen Veränderungen selbstständig vornehmen, damit weitere spezifische Subjekte die nötigen Berechtigungen erhalten.

Administration der Rechtevergabe
Die Notwendigkeit der Administration der einzelnen Modell führte zu unterschiedlichen Kontroll Modellen [GAL 87].

- **Hierarchisches Modell**
  Die Kontrolle ist hier hierarchisch angeordnet. Ein Beispiel wäre der Administrator als oberste Instanz, der alle Rechte setzen darf. Er hat, da er sich in der höchsten Ebene befindet, alle Kontrollrechte auf alle Objekte. Das Kontrollrecht beinhaltet auch die Möglichkeit der Weitergabe (Control with passing ability). Der Administrator ist in der Lage verschiedene Untergruppen (z.B. nach Arbeitsbereichen) zu bilden. Die default-Zugriffskontrolle gibt dem Kopf einer Untergruppe die Kontrollrechte über die darunterliegenden Directories. Die Benutzer die in der letzten Hierarchiestufe stehen, besitzen keine Kontrollrechte.
  Verwaltungstechnisch gesehen stehen vertrauenswürdigere Parteien weiter oben in der Hierarchie. Wird das hierarchisches Modell verwendet, so spiegelt es zumeist die organisatorische Umgebung wieder. Ein Nachteil des hierarchischen Modells ist die strikte Einhaltung der Kontrollrechte, wobei eine in der Hierarchie weiter oben stehende Instanz immer Zugriff auf darunterliegende Objekte besitzt.

- **Konzept des Eigentümers**
  Dieses Modell verbindet mit jedem Objekt einen Eigentümer. Dies ist normalerweise der Erzeuger des Objekts, der als einziger, mit Ausnahme des Administrators, Kontrollrechte auf seine Objekt besitzt. Er hat keine Möglichkeit dieses Kontrollrecht weiterzugeben. Dies entspricht einem hierarchischem System mit zwei Ebenen. Implementiert werden kann das Konzept der Kontrollrechte der Eigentümer indem man eine benutzerbestimmbare Zugriffskontrollpolitik implementiert. Der Erzeuger eines Objekts ist dem System bekannt. Er kann Zugriffsrechte vergeben, die Kontrolle der Rechte jedoch nicht weitergeben, da nur er der Eigentümer ist. UNIX ist ein Beispiel eines Betriebssystems, bei dem nur der Eigentümer eines Objekts die Möglichkeit hat, die Zugriffsrechte zu ändern (mit Ausnahme des Administrators).

- **Laissez-Faire Konzept**
  Jeder Benutzer mit Kontrollrechten und der Möglichkeit diese weiterzugeben, darf diese an jeden anderen Benutzer weitergeben. Das Prinzip des Eigentümers existiert hier nicht. In einer Zugriffskontrollliste (ACL) stehen alle Nutzer, die in der Lage sind die Zugriffskontrollliste eines Objekts zu ändern. Wird die Zugriffskontrolle einmal aus der Hand gegeben, ist die Kontrolle der Zugriffe auf das Objekt schwierig. Solange die Weitergabe des Kontrollrechts unabhängig von der Vergabe des Zugriffsrechts ist, erfüllt das Laissez-Faire Prinzip die Anforderungen der benutzerbestimmten Zugriffskontrolle (DAC) aller Sicherheitsstufen.

- **Zentralisiert (Centralized)**
  Dieses Prinzip gibt einem Nutzer, i.d.R. dem Systemadministrator, Kontrollrechte auf alle Objekte. Der Nutzer kann dabei die Kontrollrechte nicht weitergeben. Alle Kontrollrechte sind also bei einem Benutzer, dem Administrator. Sollen Rechte für Objekte geändert werden, kann dies nur der Administrator bewerkstelligen. Dies bedeutet eine erheblich Einschränkung der Flexibilität des Systems.

2.6.4 Gruppen- und Rollenkonzepte

Autorisation anhand einzelner Subjekten und Objekten ist aufgrund der Menge und der Komplexität z.B. der dadurch entstehenden Zugriffskontrolllisten nicht praktikabel. Gruppierungen oder der Einsatz von Rollen mindert hier die Komplexität. Ein weiterer Ansatz ist die Einteilung von Objekten in Domänen, wobei bei der Entscheidungsfindung die Ursprungsdomäne und die Zieldomäne betrachtet wird.

Eine Gruppe besteht aus einzelnen Subjekten, die etwas gemeinsam haben (z.B. das Arbeiten am gleiche Projekt). Problematisch ist die Vergabe von Zugriffsrechte an Gruppen dann, wenn

ein Benutzer in mehreren Gruppen vertreten ist und sich dadurch die einzelnen Zugriffsrechte aufaddieren. Eine anwendungsspezifische Vergabe von Zugriffsrechten ist dann nicht mehr möglich [ECK 98].
Ein Rolle hingegen drückt die Beziehung Subjekt-Auftrag aus. Je nach Auftrag sind unter Umständen unterschiedliche Rechte notwendig. Verschiedene Subjekte können einer Rolle zugewiesen bekommen.

Bei Systemen, welche Role Based Access Control und Domain and Type Enforcement unterstützen, können Policies einfacher durch gruppierte Subjekte oder Objekte ausgedrückt werden.

Role Based Access Control (RBAC) Die rollenbasierten Zugriffskontrolle handelt es sich um eine von dem Subjekt unabhängige und funktionsorientierte Zuweisung von Rechten an Rollen. Den einzelnen Benutzern werden Rollen zugewiesen. Ein Benutzer erhält seine Rechte ausschließlich über die Rolle. Ein Subjekt kann in mehreren Rollen auftreten, wobei die Rechte dann addiert werden [KUF 00].
Eine Rolle modelliert dabei die Aktion, die das Subjekt ausführen muss und die Information, auf die ein Subjekt zugreifen darf [ECK 98]. Es stehen also nicht die Objekte und Informationen im Mittelpunkt.
Wird ein rollenbasierter Mechanismus eingesetzt, muss z.B. nicht mehr für jedes Subjekt eine eigene Zugriffskontrollliste (ACL) angelegt werden. Dadurch wird die Komplexität und die damit anfallenden Kosten der Sicherheitsadministration in großen Netzwerken erheblich reduziert. Die Administration und das Management der Privilegien ist vereinfacht: Rollen können erneuert werden, ohne die Privilegien jedes einzelnen Nutzers auf einer individuellen Basis zu erneuern.
Werden Zugriffsrechte an Rollen vergeben, bleibt die Rechtevergabe unberührt von einer eventuell vorhandenen starken Dynamik der Subjekt-Erzeugung und -Terminierung [ECK 98].
RBAC ist eine Alternative zu den traditionellen DAC- und MAC-Policies. Sicherheitspolitiken werden anhand der natürlichen Organisationsstruktur spezifiziert und durchgesetzt. Jeder Nutzer wird einer oder mehreren Rollen zugeteilt und jeder Rolle werden eine Menge von Privilegien zugeteilt. Die Realisation einer rollenbasierten Autorisation besteht i. Allg. aus:

1. einem Benutzer wird eine Rolle zugewiesen
2. die Rollenhierarchie wird definiert
3. der Rolle bekommt Zugriffsrechte zugewiesen

Die Analyse, wie eine Organisation operiert, gibt Auskunft über die bestmöglichste Definition der Rollen. Durch die Rollen-Hierarchie können gemeinsame Attribute zusammengefasst werden. Der Vorteil des Rollenkonzepts liegt in der geringeren Veränderung der Zugriffsrechte, die eine Rolle zugeteilt bekommt. Die Änderung der Zugriffsrechte von einzelnen Benutzern ist hier weitaus dynamischer.
Die Verwendung von Rollen kann sich u.U. jedoch nachteilig auf die Flexibilität in Bezug auf die Differenzierung der Rechtevergabe auswirken [ECK 98].

2.6.5 Fazit (Rechtevergabe)

Das Ziel einer anwendungsspezifischen Rechtevergabe ist die Vergabe von benutzerbestimmten Zugriffsrechten auf spezifische Objekte. D.h. es sollte die Möglichkeit geben, Zugriffsrechte an einzelne spezifische Subjekte vergeben zu können. Betrachtet man die Objekte, auf die durch ein Zugriffsrecht zugegriffen werden darf, so sollte auch hier eine möglichst anwendungsspezifische feingranulare Adressierung ermöglicht werden. Erst dadurch kann sichergestellt werden, dass ein Subjekt nur die Zugriffsrechte erlangt, die es notwendigerweise für die Erledigung der Aufgabe benötigt (Least-Privilege Prinzip).

Subjektbasierte Methoden haben die Problematik, Objekte möglichst feingranular adressieren zu können. Betrachtet man die objektbasierten Methoden so liegt die Problematik in der feingranularen Adressierung der Subjekte.

Die sprachbasierte Methode ist i. Allg. ausdrucksstärker und flexibler. Durch die sprachbasierte Methode lassen sich i. Allg. alle Entitäten feingranular adressieren. Die Problematik liegt hier in der Suche nach einem geeigneten Mechanismus, der die Zugriffskontrollpolitik durchsetzen kann.

Um Vorteile von subjekt-, objekt- und sprachbasierten Methoden zu vereinen, sind auch Mischformen denkbar.

2.6.6 Rechtedurchsetzung

Bei einem Zugriffsversuch muss ein Mechanismus überprüfen, ob das zugreifende Subjekt die notwendige Autorisation besitzt. Die Zugriffskontrolle erfolgt jeweils vor einem Zugriff auf ein Objekt durch einen geeigneten Zugriffsmechanismus. Dabei kann eine Zugriffskontrolle noch feiner unterteilt werden in eine Entscheidungsfunktion (Decision Function) und eine Durchsetzungsfunktion (Enforcement Function), die auch getrennt realisiert sein können [SUM 97]. Auch hier existieren unterschiedliche Konzepte, Mechanismen und Techniken, um die Zugriffskontrolle durchzuführen.

Der Schutz der Information sollte auf der Ebene der Funktionen ansetzen, die für den Zugriff und die Bearbeitung der Informationen genutzt werden. Nach Lieb [LIE 90] lassen sich fünf Schutzebenen unterscheiden. Das Wirkungsfeld wird mit absteigender Ebene tendenziell begrenzter und spezifischer.

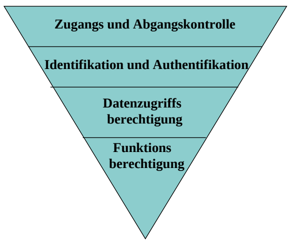


Abbildung 5: Schutzebenen in Anlehnung an [LIE 90]

Hat ein Subjekt die ersten zwei Sicherheitsebenen passiert, ist sicherzustellen, dass der Zugriff auf Daten und Funktionen im erwünschten Umfang begrenzt bleibt. Relevant dafür ist die Datenzugriffs- und Funktionsberechtigung, welche sich durch spezifische Schutzkonzepte, wie z.B. das Konzept der Zugriffsmatrix (vgl. Kapitel 2.4.1), realisieren lassen. Jeder Zugriff ist somit einer Zugriffskontrolle zu unterziehen. Zu beachten ist dabei, dass die Rechtsicherheit eines Systems von der Fähigkeit der Zugriffskontrolle abhängt, sicher zu stellen, dass die Zugriffskontrolle nicht umgangen werden kann.

Die Zugriffskontrolle ist die Durchsetzungsinstanz der Zugriffskontrollpolitik. Sie beantwortet Fragen über die geltende Zugriffskontrollpolitik und sucht, falls nötig, nach einer Entscheidung. Was gespeichert wird und wie die Suche abläuft, hängt dabei von der spezifisch definierten Abstraktion des Systems ab. Aus Performancegründen kann die Zugriffskontrolle in der Regel Zugriffskontrollentscheidungen zwischenspeichern.

Am Beispiel UNIX sind die Zugriffsrechte (Protection Bits) der Directories und der Dateien die Entscheidungskriterien. Sie sind auf der Festplatte oder Diskette gespeichert. Die Rolle der Zugriffskontrolle übernimmt der Kernelcode, der anhand der Protection Bits die Entscheidung trifft, ob der Zugriff zulässig ist oder nicht.

Referenz Monitor Prinzip Die Realisation der Zugriffskontrolle erfolgt zumeist nach dem Referenz Monitor Prinzip. Jeder Zugriff fällt dabei vom Referenz Monitors überwacht, wodurch sichergestellt ist, das jeder Zugriff kontrolliert wird.

Ein Referenz Monitor besitzt folgende Eigenschaften:

- er kann nicht umgangen werden
- er verwaltet alle Zugriffe
- er ist vor Attacken geschützt (manipuliersicher (tamper-proof))
- er ist schmal genug, um analysiert und getestet zu werden

Um diese Prinzip durchzusetzen und so die Zugriffskontrolle durchzuführen können hardwar- und softwarebasierte Schutzmechanismen verwendet werden. Die hardwarbasierten Mechanismen werden jedoch zunehmend von softwarebasierten Mechanismen verdrängt. Softwarebasierte Mechanismen sind in der Regel flexibler, leichter zu realisieren, haben Performance Vorteile und eignen sich vor allem zur Implementierung einer plattformunabhängigen Zugriffskontrolle. Hardwarebasierte Kontrollen beschränken sich zumeist auf grobgranulare und unflexible Speicherschutzmaßnahmen und sollen daher nicht weiter betrachtet werden [BAF 00]. Die Abb. 6 gibt einen Überblick über bestehende Schutzmechanismen.

Softwarebasierte Schutzmechanismen Softwarebasierter Schutz basiert einerseits auf programmiersprachliche Konzepte, aber auch auf Schutzmechanismen der Übersetzer und der Binder. Zu den programmiersprachlichen Konzepten zählen die Typisierung, die Ausnahmebehandlung und die Verhinderung von direkten Speicherzugriffen durch den Programmierer [ECK 98].
Der Hauptvorteil eines softwarebasierten Schutzes ist die Portabilität. Würde ein hardwarebasierter Schutz angewandt, müssten die Systeme Zugriff auf die Page Tables und Systemaufrufe unterstützen. Diese Mechanismen sind jedoch nicht einheitlich auf allen Plattformen vorhanden.

Der softwarebasierte Schutz kann in zwei Teile aufgeteilt werden [BAF 00]:

- **Speicherschutz** (*memory protection*)
- **Sicherheitsdienste** (*secure services*)

Der Speicherschutz verhindert unautorisierte Zugriffe auf den Speicher oder auf den ausführbaren Code. Dies reicht jedoch nicht immer aus. Müssen auch andere Ressourcen (z.B. Files, Displays, ...) geschützt werden, werden weitere Sicherheitsdienste benötigt. Die Sicherheitsdienste ergänzen dann den Speicherschutz.
Während z.B. der Speicherschutz das unberechtigte Lesen der als `private` deklarierten Methoden verhindert, überprüft ein Sicherheitsdienst aktiv, ob das Subjekt hinreichend autorisiert ist.

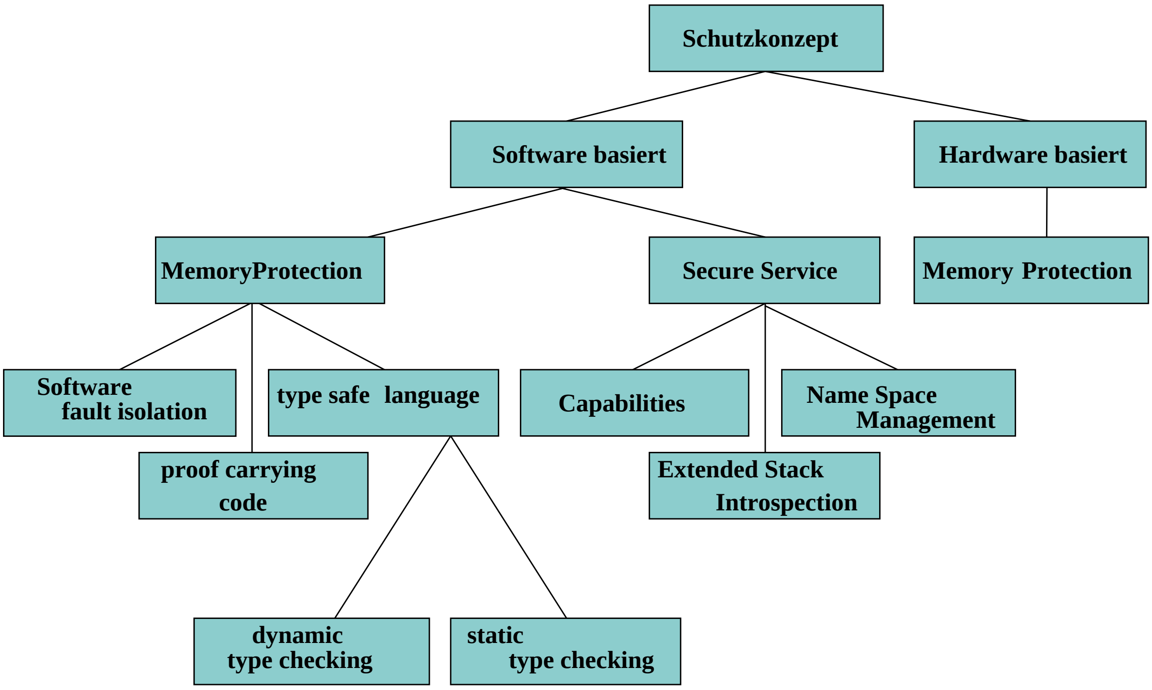


Abbildung 6: Schutzmechanismen

Speicherschutz (Memory Protection) Der Speicherschutz lässt sich weiter unterteilen in:

- **Software Fault Isolation**
  Potentiell gefährliche Operationen werden vor ihrer Ausführung dynamisch überprüft.

- **Proof-Carrying Code**
  Die Prüfung findet schon beim Laden statt. Das Programm kann so in voller Geschwindigkeit ablaufen.

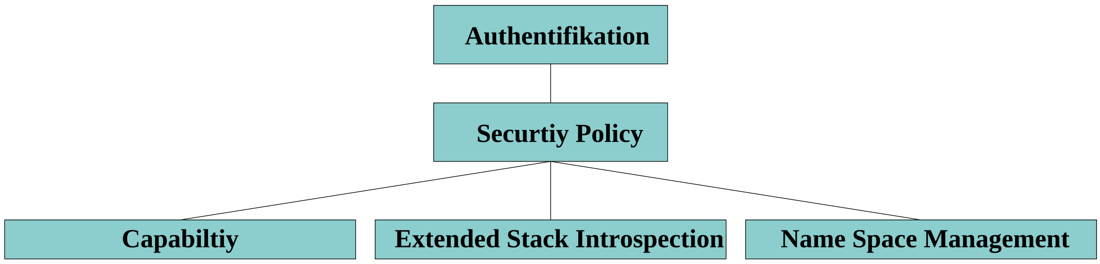
- **Type-Safe Languages**

  - ***Dynamic Type Checking***
    Bei jedem einzelnen Schritt findet eine lokale Überprüfung statt. Dabei werden Typ-Fehler (Type-Errors) vor der Ausführung erkannt und können somit verhindert werden. Dynamic Type Checking ist einfacher als Static Type Checking.

  - ***Static Type Checking***
    Beim Static Type Checking findet eine Überprüfung vor dem Programmstart statt. Der Vorteil ist die schnellere Ausführung des überprüften Codes, jedoch ist die Überprüfung an sich viel komplizierter. Java verwendet zumeist Static Type Checking.

Sicherheitsdienste (Secure Services) Gilt es eine Ressource zu schützen, so wird ein Sicherheitsdienst benötigt. Die Aufgabe des Dienstes besteht i. Allg. darin, definierte Regeln (Sicherheitspolitik), welche die jeweilige Ressource betreffen, durchzusetzen. Es existieren unterschiedliche Methoden zur Implementierung von Zugriffskontrollpolitiken (Security Policies) [BAF 00].
Zumeist werden Ausnahmebehandlungen verwendet, um die Zugriffskontrolle durchzusetzen. Durch die Verwendung von Ausnahmebehandlungen kann während der Laufzeit eine Einhaltung von aufgestellten Regeln durchgesetzt werden. Ein Programmierer ist z.B. in der Lage, gezielt Zugriffe auf Kontrollkomponenten umzulenken.

Realisationskonzepte Folgend werden Beispiel-Implementierungsstrategien vorgestellt, die die Sicherheitspolitiken in softwarebasierten Sicherheitssystemen durchsetzen. Alle Mechanismen nutzten dabei Digitale Signaturen zur Authentifikation.
Die am häufigsten antreffenden Modelle sind Name Space Management, Capabilities und Extended Stack Introspection. Es gibt jedoch auch noch andere Modelle bzw. Mischformen. So z.B. die Verwendung von Proxy-Objekten oder die Verwendung von Prozessen zur Durchsetzung der Zugriffskontrollpolitik.



Capabilities
Grundsätzlich ist eine Capability ein fälschungssicherer Zeiger auf eine zu kontrollierende System Ressource. Eine Capability muss explizit erteilt werden. Dies kann bei der Initialisierung oder beim Aufruf einer anderen Capability geschehen. Hat ein Programm einmal eine Capability, kann dieses Programm die Capability einmal oder mehrmals nutzen und unter Umständen an andere Programme weiterreichen. Besitzt ein Programm ein Capability, so muss es ihm vorher zugewiesen worden sein [BAF 00]

Extended Stack Introspection
Das Prinzip ist denkbar einfach. Bevor ein Zugriff erlaubt wird, wird überprüft, ob der Programmcode innerhalb des Aufrufstacks die notwendige Berechtigung besitzt, die Operation auszuführen [BAF 00].
Da oftmals auch unautorisierte Programme Zugriffe auf geschützte Informationen ausführen müssen, um ihre Aufgabe zu erfüllen, besteht die Möglichkeit den Programmcode kurzzeitig durch einen expliziten Aufruf zu autorisieren.
Ein Problem ist jedoch, dass ein Prozess Zugriff auf verschiedene Ressourcen hat. Jemand, der diesen Prozess ausführt, sollte nicht auf alle Ressourcen Zugriff haben. Dieses Problem ist unter dem Namen Confused-Deputy Problem bekannt und es gab diese Problematik auch schon innerhalb von Betriebssystemen. Nicht ausreichend gesicherte Funktionen ermöglichten den Zugriff auf Ressourcen und stellten somit eine Sicherheitslücke dar. In UNIX sind hier z.B. die S(ecure)-Bit Programme [HER 99] zu nennen, die oftmals eine Schwachstelle darstellen und des öfteren dazu führten, dass unberechtigte Nutzer den Root Status erlangten [ANO 99].

Name Space Management
Dieser Mechanismus wird benutzt, um Klassen zu verstecken oder auszutauschen. Durch die Verwendung des Name Space Managements kann eine Zugriffskontrollpolitik durchgesetzt werden, indem kontrolliert wird, wie die Klassennamen eines Programms aufgelöst werden. Dabei kann eine Klasse aus der Name Space Umgebung entfernt werden (Klassenaufruf misslingt), oder der Name referenziert eine zur Original Klasse kompatiblen Klasse [BAF 00].
Diese Technik wird z.B. in Safe-Tcl [BOR 94] und in Plan 9 [PTP 96] verwendet. Bei Safe-Tcl werden beispielsweise Kommandos vor einem nicht vertrauenswürdigen Interpreter versteckt.
In einer objektorientierten Programmiersprache repräsentieren Klassen Informationen, die geschützt werden müssen. So repräsentiert die File-Klasse das Dateisystem und die Socket-Klasse die Netzwerkoperationen. Ist die File-Klasse und alle andere Klassen, die das Dateisystem beanspruchen für den Aufrufer (Subjekt) nicht sichtbar, so steht das Dateisystem dem Programm

nicht zur Verfügung und kann somit auch nicht angegriffen werden. Ein Zugriff auf diese Klasse ist gleichzusetzten mit einem Zugriffsversuch auf eine nicht existente Klasse.
Um nun komplexere Zugriffskontrollpolitiken durchzusetzen, kann man Umgebungen erstellen, die die zu beschützenden Klassen durch kompatible stellvertretende Klassen ersetzen. Die stellvertretende Klasse kann einen Aufruf beispielsweise überprüfen und gegebenenfalls die Original Klasse aufrufen. Damit die Funktionalität erhalten bleibt, muss die stellvertretende Klasse kompatibel zu der Klasse sein, die sie vertritt.
Für ein flexibles und generelles Zugriffskontrollschema ist ein kontrollierbarer Einsatz der stellvertretenden Klassen notwendig. Die Kontrolle bezieht sich dabei z.B. auf privilegierte Nutzer oder Gruppen.
Entscheidungen sind hier statisch und werden vor der Ausführung des Codes getroffen. Ist in einem Name Space eine Klasse versteckt bzw. ausgetauscht worden, so gibt es keine Möglichkeit mehr, den ursprünglichen Zustand wieder herzustellen.

Prozessbasiert Methode
Ein Sicherheitsdienst, welcher die Zugriffskontrollpolitik durchzusetzen hat, kann auch basierend auf Prozesse realisiert werden. Jede Anwendung läuft in einer separaten Instanz einer virtuellen Maschine. Um die Prozess noch weiter aufzuteilen, kann eine separate physikalische Maschine reserviert werden, um nicht vertrauenswürdigen Code auszuführen.
Der Vorteil eines Prozessmodells, welches auch oft bei Betriebssystemen zum Einsatz kommt, ist die Möglichkeit, jede einzelne Anwendung exakt aufzeichnen zu können. Wird eine Anwendung beendet, so können seine Threads unverzüglich beendet und der Speicherplatz freigegeben werden. Zwei virtuelle Maschinen haben keine direkten Referenzen auf die jeweiligen Daten, stattdessen werden indirekte Referenzen, wie bei Aufrufen von physikalisch getrennten Systemen, verwendet. Wird eine potentiell gefährliche Operation aufgerufen, so wird eine Nachricht von der nichtvertrauenswürdigen virtuellen Maschine zu einer vertrauenswürdigen virtuellen Maschine gesendet, die daraufhin den Zugriff überprüfen kann. Nachteilig wirkt sich hierbei der große Overhead aus.

Verwendung von Proxy-Objekten
Die Durchsetzung der Zugriffskontrollpolitik kann auch durch den Einsatz von Proxy-Objekten realisiert werden. Bei einer Anfrage einer Anwendung wird ein Proxy-Objekt unter Berücksichtigung der Zugriffskontrollpolitik erzeugt. Das erzeugte Proxy-Objekt besitzt abgesicherte Schnittstellen zu den einzelnen Ressourcen.

Proxy-Objekte haben die Nachteile, dass für jede Agenten Instanz, die auf Ressourcen zugreift, jeweils ein Proxy-Objekt erstellt werden muss. Dadurch unterstützen sie jedoch auch die Möglichkeit, spezifisch für jede Subjekt Entscheidungen zu treffen. Zudem können sie dynamisch für jedes Subjekt erzeugt werden. Wird für einen Agenten ein Proxy-Objekt erstmals erstellt, so ist die Zugriffskontrolle an sich mit wenig Rechenaufwand verbunden.

Wrapper-Objekte
Das Subjekt, welches auf ein Objekt zugreifen will, besitzt nur eine Referenz auf ein Wrapper-Objekt und kann nicht direkt auf das Objekt zugreifen. Der Wrapper besitzt eine Zugriffskontrollliste, um erkennen zu können, ob es sich um einen erlaubten Zugriff handelt.
Diese Technik ist, da sie auf einer Zugriffskontrollliste basiert, nur für eine begrenzte Anzahl von Subjekten geeignet [KAT 98].

Wrapper-Objekte sind zwar leicht zu implementieren und sind auch transparent gegenüber dem Implementierer, zudem existiert für jede Ressource ein Wrapper-Objekt, jedoch ist dieser Mechanismus unflexibel. Jeder Agent muß zum gleichen Zugriffskontrollmechanismus zeigen, der bei jedem Zugriff durchgeführt wird.

Hybrid-Systeme
Haben die genannte Methoden verschieden Vor- und Nachteile, so wird durch den Einsatz von

Mischformen versucht, die Vorteile einzelner Methoden zu vereinen.

Name Space Management und Capabilities
Name Space Management ist hervorragend für das Verstecken von Klassen geeignet und könnte deshalb eine gute Basis für eine Implementierung eines starken Capability basierten Systems dienen. Nur die Klassen, welche die Capabilities implementieren, können die Original System Klassen sehen.

Stack Inspection und Capabilities
Stack Inspection kann teuer sein hat jedoch hervorragende Sicherheitseigenschaften. Capability Systeme sind schnell, erfordern jedoch einen besonderen Programmierstil. Die Kombination bringt die Vorteile beider Seiten zur Geltung.
Stack Inspection schützt wichtige Aufrufe, so z.B. das Öffnen von Dateien oder Netzwerkverbindungen, während die Objekte, welche die Capabilities zur Nutzung bestimmter Targets darstellen, zurückgegeben werden. Sicherheit ist jedoch nur dann gegeben, wenn Capabilities nicht gestohlen oder anderweitig in unautorisierte Hände gelangen.

Für ein strenges Capability basiertes System, wobei die einzige Möglichkeit zum Öffnen einer Datei darin besteht, ein Capability zu besitzen, könnte Stack Inspection genutzt werden, um alle bis vom System aufgerufene Datei Interfaces zu blockieren, während den Capability-Klassen der Zugriff gewährt wird. Die Vorteile im Vergleich zu einer rein Capability-basierten Methode liegen darin, dass jeder Zugriff auf jedes Objekt überprüft wird, in der Authentifikation der Subjekte und der Abrechenbarkeit der Zugriffe.

2.7 Anforderungen an die Zugriffssteuerung der unterschiedlichen Architekturen

Die Entwicklungsgeschichte der Zugriffssteuerung begann schon sehr früh mit den ersten Systemen, auf die mehrere Benutzer Zugriff hatten. Mit zunehmender Komplexität der Funktionalität und der Architekturen stiegen auch die Anforderungen an die Zugriffssteuerung. Vorhandene Konzepte und Mechanismen entwickelten sich den Anforderungen der einzelnen Architekturen entsprechend weiter.

2.7.1 Einzelsysteme

Betrachtet man Einzelsysteme, sind Zugriffe auf Informationen des Systems relativ beschränkt. Es gab nur eine gewisse Anzahl von Benutzern, die Zugriff auf das System hatten. Zudem erfolgte der Zugang immer zentral. Um auf die Information zugreifen zu können, musste man sich zuerst bei dem System anmelden. Die Subjekte und Objekte waren dem System bekannt und in einer einheitlichen Form vorhanden. Die Aufgaben der Zugriffssteuerung übernahm dabei das Betriebssystem.

Unterscheidungen der Zugriffssteuerung können anhand der Nutzerverwaltung getroffen werden. Es existieren Betriebssysteme, die nur einen Nutzer kennen (Windows, DOS) oder eine Multiuser-Umgebung unterstützen (Windows NT, Unix, Linux), wobei eine Zugriffskontrolle zumeist nur auf Multiuser Betriebssystemen realisiert wurde [SUM 97].

Multiuser Betriebssysteme unterstützen den Nutzer im Hinblick auf die Sicherung der Daten und der Programme während der Ausführung. Dabei stützt sich das Betriebssystem auf unterschiedliche Funktionen ab [SUM 97, OPP 97, ECK 98, BRO 92, VOS 95, POW 93]:

- Identifikation des Nutzers (durch z.B. einer Passwortabfrage)
- Überprüfung der Autorisation und Gewährleistung der benötigten und berechtigten Zugriffe auf Daten und Ressourcen

- Löschung der im Hauptspeicher zurückgelassenen Daten nach Beendigung des Verarbeitungsprozesses
- Behandlung von Fehlern aller Art
- Abwehrreaktionen bei unautorisierten Zugriffsversuchen bis hin zum Systemstop
- Aufzeichnung bestimmter Ereignisse
- usw.

Betriebssystemschutz wird i. Allg. durch strikte hierarchische Strukturen und Privilegien von Systemprogrammen gewährleistet. Sicherheitskonzepte mit strikter Trennung sowohl der Anwenderprogramme untereinander, als auch der Daten und der Trennung der Anwenderprogramme von den Betriebssystemkomponenten gelten als sicherer [WAE 93].

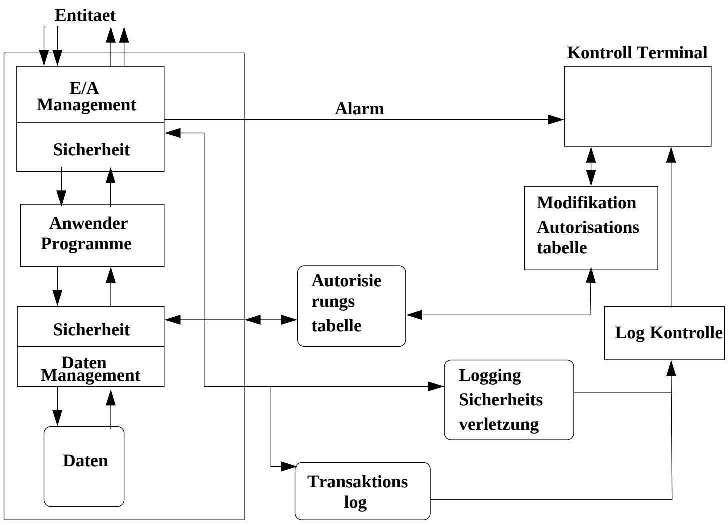


Abbildung 7: Softwarebasiertes Sicherheitskonzept [WAE 93]

In der Abb. 7 wird der Zugriffschutz der einzelnen Anwendungen eines Betriebssystems verdeutlicht. Den Schutz übernimmt dabei die Ein- und Ausgabe des Betriebssystems. Eine weitere Aufgabe, die das Betriebssystem hier erfüllt, ist unter anderem die Verwaltung der Identifikations- und Autorisationstabellen [WAE 93].

Unautorisierter Zugriff auf Anwendungsprogramme sowie auf Daten sind aufgrund des Schutzes durch das Betriebssystem nicht möglich. Dabei verwaltet das Betriebssystem die Identifikations- und Autorisationstabellen, zudem auch das Logging und die Kommunikation mit dem Nutzer bzw. dem Sicherheitsbeauftragten.

Prozessebasierte Zugriffskontrolle

Bei Betriebssystemen kommt dabei zumeist der prozessbasierte Zugriffskontrollmechanismus zum Einsatz. In einem Prozess-Strukturiertem System setzt der Kernel die Zugriffskontrollpolitiken durch. Der Kernel überprüft die Argumente bei jedem Kernel-Aufruf und trifft die Entscheidung, ob der Aufruf erlaubt ist. Dabei kann ein Prozess nur durch einen Shared Memory Buffer oder mithilfe eines System Aufrufs mit der Außenwelt kommunizieren, welche unter der Kontrolle des Kernels stehen.

Beispielsweise besitzen Betriebssysteme für IBM Mainframes Systemprogramme mit einem Supervisor-Status, die sie von den Anwendungsprogrammen abgrenzt. Unterschiedliche Prozessmodi existieren auch z.B. bei dem Betriebssystem VAX-11 (Systemkernmodus, Ausführungsmodus, Überwachungsmodus, und Benutzermodus). Hier wurde jedem Modus eine genau definierte Teilmenge an Befehlsvorraten zugewiesen [WAE 93].
In der UNIX-Systemarchitektur besitzt jedes Objekt (z.B. die Datei passwort.txt) Zugriffsrechte. Diese werden durch einen Autorisationsvorgang erteilt. Jeder Prozess besitzt eine User ID und eine Gruppen ID. Mithilfe dieser IDs wertet der Kernel aus, ob der Prozess genügend Privilegien besitzt, um auf die Ressource zugreifen zu können. Beim traditionellen UNIX System wird die Autorisation mittels Permission Bits realisiert. Diese (Neun) Bits geben Auskunft über Lese-, Schreib- und Ausführungsrechte [SUM 97].

Durch die Integration von Zugriffskontrolllisten wurden dabei eine erhöhte Granularität bei der Autorisation ermöglicht. Zugriffskontrolllisten werden z.B. von neueren UNIX-Systemen, MULTICS und Microsoft Windows NT eingesetzt [SUM 97].

Ringschutz
Um die hohen Sicherheitsanforderung bezüglich des Informationsflusses zu erfüllen, wurden weitere Mechanismen wie z.B. der Ringschutz zur Realisation einer systembestimmten Zugriffskontrollpolitik entwickelt. Der Ringschutz realisiert dabei eine systembestimmte Zugriffskontrollpolitik (MAC). Die Struktur ähnelt den Baumringen. Je näher ein Code in der Mitte plaziert ist, desto mehr Privilegien besitzt er. Aufrufe an weiter innen liegenden Informationen sind eingeschränkt. Der Ringschutz kann auch mit den UNIX-Systemaufrufen verglichen werden [HER 99].

MULTICS besitzt diesen Ringschutz und unterstützt den Zugriffschutz auch bei der Anwendung von Formen der gestreuten Adressierung, also bei der Arbeit mit Segment- und Seitentabellen beim virtuellen Speicher [WAE 93].

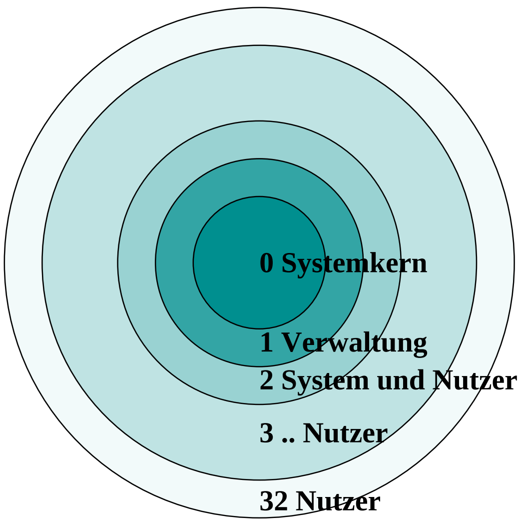


Abbildung 8: Schutzringkonzept bei MULTICS aus [WAE 93]

Die Ringe 0,1 und 2 bei MULTICS bleiben dem Betriebssystem vorbehalten. Die Ringe 3 bis 32 sind für Nutzerdaten angedacht. Ein zugreifender Prozess weist sich beim Zugriff durch eine Operandenadresse aus, die unter anderem die Schutzringnummern enthält [WAE 93].

Stimmt die Schutzringnummer überein, erhält der Prozess Zugriff laut Zugriffsattribut. Stimmen die Schutzringnummern nicht überein, überprüft eine Unterbrechungsroutine, ob die Schutzringnummer größer oder kleiner ist, und erlaubt bzw. verweigert den Zugriff.

Capability Systeme
Hardware- oder softwarebasierte Capabilities wurden in traditionellen Maschinen in einem markierten Speicher (Tagged Memories) gespeichert, wobei die Hardware das Verändern des Wertes des Zeigers durch nicht vertrauenswürdige Prozesse schützt. Eine andere Möglichkeit bestand in der Unterstützung von Handles oder Descriptors des Kernels, was einem nicht vertrauenswürdigen Prozess erlaubte, seine Capabilities zu verwenden [LEV 84]. Ein Benutzerprogramm konnte Capabilities laden, speichern und ausführen, jedoch konnte nur der Kernel Capabilities erstellen [LEV 84].

2.7.2 Verteilte Systeme

Die Regelung der Zugriffssteuerung für verteilte Systeme ist komplizierter als die der Einzelsysteme. Die Zugriffssteuerung steht hier neuen Anforderungen gegenüber, da beispielsweise die Objekte verteilt sind. Zudem kommen bei verteilten Systemen neue Objekte (Rechner, Verbindungen, ...) hinzu. Ein Hauptproblem verteilter Systeme ist die systemweite Realisation einer einheitlichen Adressierung und Authentifikation der Entitäten. Die Zugriffskontrolle muss Sicherheitspolitiken unterschiedlich administrierten Domains durchsetzen [SUM 97]. Betrachtet man verteilte Systeme, so gestaltet sich die Sicherstellung der Informationssicherheit aufgrund der verteilten Komponenten schwieriger als bei Einzelsystemen.
Durch die Verteilung der einzelnen Komponenten, welche über ein spezifisches Transportsystem verbunden werden, entstehen durch neue Techniken neue Problematiken bezüglich der Zugriffssteuerung.

Beispiel neuer Techniken:

- verteiltes Rechnen (Distributed Processing)
- verteilte Anwendungen (Distributed Applications)
- verteilte Dateisysteme (Distributed File System)
- Netzwerk Betriebsysteme (Network Operating System NOS)
- verteilte Architekturen (Client-Server, CORBA, ...)
- einheitliche Protokolle (DNS, RPC, NIS, NFS)

Die möglichen Bedrohungsarten steigen durch die Möglichkeit auch entfernter unautorisierter Zugriffe auf Informationen und Serverdienste. Auch hier gilt es jedoch die Informationssicherheit zu verwirklichen. Neue Konzepte hatten so neue Mechanismen hervorgebracht, welche die Zugriffskontrollpolitiken systemweit durchsetzen. Dazu gehört z.B. das Network File System (NFS) [STE 95], der Network Information Service (NIS) [STE 95] und das Andrew File System (AFS) [SUM 97]. NIS beispielsweise setzt auf eine verteilte Datenbank auf, welche z.B. die Verwaltung der Kennwortdateien erleichtert [STE 95, SUM 97].

Aus Sicht der Zugriffsteuerung müssen die verteilten Objekte allgemein adressiert werden können. Nur so können spezifische Zugriffsrechte auf Objekte, die sich auch auf einem anderen System befinden können, vergeben und auch durchgesetzt werden. Etwas problematischer wirkt sich jedoch auch das Vorhandensein unterschiedlicher Zugriffskontrollpolitiken der unterschiedlichen Systeme aus, die sich zudem in unterschiedlichen Organisationen und somit Administrationsbereichen befinden können. Hier ist zumindest eine Unterscheidung der Abstraktionsebenen von internen und externen Zugriffsmöglichkeiten notwendig. So finden beispielsweise Zugriffskontrollentscheidungen nicht nur anhand des authentifizierten Subjekts und des Objekts statt, sondern auch anhand weiteren spezifischen Eigenschaften wie beispielsweise der Zielort des Zugriffs oder eines Passworts.

Betrachtet man typische Anwendungen von verteilten Systemen (Telnet, FTP, WWW), basiert der Zugriffsschutz auf der Authentifikation eines Benutzers anhand eines Passworts. Zudem kann der Zugriff auf Basis der Herkunft der Anfrage eingeschränkt sein.

Ein System innerhalb eines Verbundes kann dabei Bereiche freigeben, auf die alle zugreifen dürfen, Einzelbereiche können z.B. durch eine Passwortabfrage geschützt werden oder die Berechtigung richtet sich u.a. nach der Identität des Kommunikationspartners.

Durch ein heterogenes Umfeld wird die Realisation einer Zugriffssteuerung zusätzlich erschwert. Hier ist eine geeignete Abstraktionsebene gefragt, die sich nicht auf spezifische Sicherheitsmechanismen abstützt. Letztendlich muss in der Regel die Zusammenarbeit unterschiedlicher Sicherheitsinfrastruktur und Sicherheitsdienste gewährleistet werden.
In [WOL 92] wird auf die Zugriffskontrolle auf verteilte Objekte in heterogenen Netzwerkmanagementsystemen eingegangen. Das Sandhu‘s Typed Access Matrix (TAM) Modell [ASA 92, SAN 92] dient als Rahmenwerk, um die Zugriffskontrollpolitiken in der oben genannten Umgebung zu untersuchen und zu erstellen. Ein TAM-Modell besteht dabei aus einer Zugriffmatrix kombiniert mit einer begrenzten Anzahl von Kommandos, welche die Erneuerung der Matrix kontrollieren. Dieses Modell erzeugt eine Sprache mit einer Semantik, welche unabhängig von einem spezifischem Implementierungsmechanismus ist, und erlaubt eine einfache Aussage von unterschiedlichen Zugriffskontrollpolitiken. Das TAM-Modell definiert eine Anzahl von Operationen, um die gemeinsame Zugriffskontroll-Informationen und die Funktionen für die Zugriffskontroll-Entscheidung festzulegen.
Die Zugriffskontrollschemata von CMIP und SNMPv2 setzten ein Identitäts-basierendes ACL Schema mit Objektgruppen ein. Durch das gegensätzliche Design ist die Integration der Objekte, die durch diese Zugriffskontrollpolitiken geschützt werden, schwierig.
Letztendlich soll das TAM-Modell als Basis dienen, um die unterschiedlichen Konzepte der Zugriffskontrollinformationen in ein gemeinsames Konzept unterzubringen. Desweiteren lockern die erweiterten Definitionen der Zugriffskontroll-Policies im TAM-Modell die Aufgabe des Sicherheitsmanagement. Die TAM-Spezifikationen unterstützen den Systemadministrator durch die allgemeine Sicht der zu managenden Ressourcen, und das unabhängig von dem eingesetzten Management Protokoll.

Die Problematik der globalen Authentifikation von Entitäten bei verteilten Systemen innerhalb einer Domäne können durch z.B. Authentifikations- und Autorisations-Server realisiert werden. Im Vordergrund steht hier vor allem die Trennung der Authentifikation von den eingesetzten Durchsetzungsmechanismen. Zugriffskontrollentscheidungen werden in vernetzten Systemen auch von Netzelementen, wie z.B. von Routern getroffen. Konzepte, die die Zugriffskontrollpolitiken der einzelnen Domänen verteilter Systeme durchsetzen, sind z.B. Firewalls bzw. Paketfilter.

Firewalls
Der einfachste Weg ein Cluster, also einen begrenzten Teilbereich des Netzes zu schützen, ist den Zugriff auf das Cluster durch eine Firewall zu schützen. Dies ist momentan auch der weitverbreiteste Schutz von Rechnernetzen. Eine Firewall lässt auf Grundlage einer Filterpolitik Anfragen bestimmter Nachrichten nicht durch. Dabei richtet sich die Sicherheitspolitik in der Regel nach der Herkunft des zugreifenden Subjekts.

Verteilte Capabilities
Capabilities erstrecken sich auch über Netzwerke hinweg. Bestehen entfernte Objektreferenzen aus Bits, die nicht erraten werden können, so hat eine entfernte Capability-Referenz alle Sicherheitseigenschaften einer lokalen Capability [GON 89]. Es erschwert sich jedoch die ohnehin schon problematische Rücknahme der Capabilities. Zudem ist das Risiko, dass eine dritte Partei unberechtigt an eine Capability gelangt, weitaus höher [SUM 97].

Verteilte ACLs

Zugriffskontrollentscheidungen können auch auf der Basis der Identität eines Aufrufers gefällt werden. Zwei Maschinen können eine verschlüsselte Verbindung aufbauen, wobei auch ein Handshake stattfinden kann, um die Teilnehmer zu identifizieren. Viele Architekturen die die Kommunikationsinfrastruktur für verteilte Objekte zur Verfügung stellen (z.B. CORBA und DCE), erlauben bei einem Objekt Aufruf die Befragung einer Access Control List.

Proxy Objekte

Auch Proxy Objekte könne zur Autorisation und Accounting Zwecken eingesetzt werden [NEU 93].

2.7.3 Agentensystemarchitekturen

Ist die Realisation der Zugriffsteuerung in einem verteilten System erschwert, kommen in einer Agentensystemarchitekur noch zusätzliche Problematiken hinzu. Zusätzlich zu den verteilten Objekten kommt noch hinzu, dass die Objekte und Subjekte teilweise in Form eines Agenten von System zu System wandern können. Die Informationssicherheit des Agentensystems und der Agenten muss jedoch auch hier sichergestellt werden.

Betrachtet man die herkömmlichen Schutzkonzepte, so reichen diese nicht aus. Ist der Einsatz von Firewalls beispielsweise bei einem Cluster ausreichend, um bestimmte Kommunikationsverbindungen zu unterbinden, um so z.B. nur mit vertrauenswürdigen Rechnern kommunizieren zu können, bieten Firewalls keinen Schutz gegenüber mobilen Code. Passiert ein Mobiler Agent die Firewall, so entspricht dies einer Textdatei. Diese Datei wird erst bei ihrer Ausführung gefährlich, falls es sich um einen bösartigen Agenten handelt. Zur Zeit der Ausführung hat eine Firewall jedoch keinen Einfluß mehr auf den Mobilen Agenten. Eine Firewall verhindert den Zugriff von außerhalb auf das geschützte Cluster, der Zugriff von innerhalb des Cluster nach außen ist in der Regel jedoch gestattet. Ein bösartiger mobiler Agent kann vom Internet heruntergeladen werden, durchdringt ungehindert die Firewall, und kann bei der Ausführung sein Unwesen treiben, falls keine geeigneten Sicherheitsvorkehrungen vorhanden sind.

Aufgrund der Mobilität von Objekten müssen Zugriffskontrollpolitiken dieser Objekte auch auf anderen, z.T. fremden Systemen durchgesetzt werden können. Unautorisierte Zugriffe und der Missbrauch sind hier ebenfalls zu verhindern.

Folgende Sicherheitsfragen fallen in den Zuständigkeitsbereich der Zugriffskontrolle bei Agentensystemen:

- Wie kann ein Agentensystem seine Informationen schützen?
- Wie kann ein Agent sein Informationen schützen?

Agenten, die auf einem Agentensystem ausgeführt werden, müssen vor unbefugten Zugriffen geschützt sein. Der Schutz eines Agenten vor böswilligen Agentensystemen gestaltet sich äußerst schwierig, da der Agent dem Agentensystem in erster Linie vollkommen ausgeliefert ist. Jedoch existieren auch hier Überlegungen, die den Agenten vor böswilligen Agentensystemen schützen sollen [VIG 98, SAT 98a, HOH 98]

Betrachtet man die Kontrolle des Informationsflusses, so ist sie ebenfalls von besonderer Bedeutung für die Informationssicherheit. Beispielsweise kann durch die Informationsflusskontrolle sichergestellt werden, dass spezifische sicherheitskritische Informationen das Agentensystem nicht verlassen können.

Zugriffe auf Objekte wie z.B. Dateien, sollten nach Möglichkeit ebenso nachvollziehbar und einer Person zuordenbar sein. Dies erfordert eine globale Authentifikationsmöglichkeit der einzelnen Subjekte.

Auch das Auditing sicherheitskritischer Zugriffe ist für ein Agentensystem unerlässlich.

Bei Agentensystemen werden in der Regel Zugriffskontrollpolitiken durch Überprüfungsmechanismen der Programmiersprache oder der Laufzeitumgebung durchgesetzt. Die Zugriffskontrolle erfolgt dabei generell über Kommunikations-Sicherheitsmechanismen wie z.B. einer Zugriffskontrollliste, oder durch die Agenten-Programmiersprache. Policies, Credentials, Zugriffskontrolllisten (ACLs) bis hin zu Tickets und Capabilities werden in vorhandenen Agentensystemen eingesetzt.

Solange nur einige Services des Systems genutzt werden (z.B. Java Applets im Sandbox Modell), reicht es aus, den Zugriffsschutz durch Isolation der unterschiedlichen Codes untereinander und durch Zugriffskontrollen, die die Ausführungsumgebung schützen zu realisieren. Dies entspricht dem Sandbox-Modell. Für Applikationen im Allgemeinen aber werden jedoch mehr Services des Systems benötigt.

Betrachtung der Agenteninstanz History
Jede Migration von einem Agentensystem zu einem anderen Agentensystem wird Hop genannt. Agenten, die mehrere Agentensysteme besuchen, sind Multi-Hop Agenten. Ein One-Hop Agent migriert nur auf ein Agentensystem und reist dann nicht mehr weiter. Ein Beispiel für einen One-Hop Agenten ist ein Applets. Ein Two-Hop Boomerang Agent migriert von seinem Home-Agentensystem auf ein weiteres Agentensystem und dann wieder zurück. Involviert ist also nur ein weiteres Agentensystem [ORD 98].

Bei One-Hop Agenten setzt sich das sendende Agentensystem keiner Gefahr aus. Bei einem Two-Hop Boomerang Agenten besteht die Gefahr eines modifizierten Agenten nur durch ein Agentensystem. Die Herstellung einer Vertrauensbeziehung zu einem Agentensystem kann durch ein Public-Key Verfahren erfolgen. Hier können die beiden Agentensysteme jeweils die Daten des Agenten verschlüsseln. Durch die Verschlüsselung mit jeweils dem öffentlichen Schlüssel des Partners. Bei Multi-Hop Agenten ist die Herstellung der Sicherheitsanforderungen erschwert, da mehrere Agentensysteme involviert sein können, wodurch die Gefahr eines Angriffs steigt.

- **One-Hop Agenten**
  Das Home-Agentensystem hat nichts zu befürchten. Das Agentensystem B muss sich gegen böswillige Agenten bzw. nicht autorisierte Operationen und Zugriffen schützen.

- **Two-Hop Boomerang Agent**
  Das Agentensystem "B" muss sich geeignet schützen. Kommt der Agent auf das Home-Agentensystem, so kann er jetzt unter Umständen ebenso böswillig sein, da das besuchte Agentensystem "B" ihn unter seiner totalen Kontrolle hatte, und somit evtl. auch in der Funktion ändern konnte. Die Problematik ist hier jedoch aufgrund der beteiligten Teilnehmer stark eingeschränkt und somit kontrollierbarer. Durch eine Vertrauensbeziehung z.B. anhand von Zertifikaten ist es möglich, einem Agenten trauen zu können.

- **Multi-Hop Agenten**
  Vertraut ein Agentensystem "A" einem Agentensystem "B" und vertraut ein Agentensystem "B" einem Agentensystem "C", so ist der Schluss, dass das Agentensystem "A" dem Agentensystem "C" traut, nicht unbedingt richtig. Durch die Anzahl der möglichen beteiligten Entitäten ist eine erhöhte Vorsicht geboten. Die History des Agenten spielt eine Rolle. Durch die History des Agenten ist bekannt, welches Agentensystem der Agent in seiner Vergangenheit besucht hat. Dabei ist darauf zu achten, dass die History nicht verfälscht werden kann.

2.8 Modelle zur Bewertung und Prüfung der Sicherheit

Um Sicherheit zu gewährleisten, müssen Sicherheitsstandards auf Grundlage der Gefährdung entwickelt und eingehalten werden. Als einheitlicher Maßstab zur Bewertung und Prüfung von Systemen dienen spezifische Sicherheitskriterien.

Allgemein betrachtet ist es schwer, die Sicherheit eines Systems einzustufen. Unterschiedliche Modelle, wie z.B. das Orange Book, zeigen Richtlinien auf, nach denen Systeme auf ihre Sicherheit hin eingestuft werden können. Dieser Abschnitt geht allerdings nur auf das formale Modell des Orange Book ein. Die Betrachtung der formellen Prüfung (formal proofs) oder formelle Spezifikationssprachen würden den Rahmen dieser Arbeit sprengen. Standards wurden erstmals für Einzelsysteme entwickelt. Durch die Architekturen vernetzter Rechner wurden diese Standards unter Einbeziehung der Eigenschaften der verteilten Systeme erweitert und angepasst. Standards, die die Eigenschaften mobiler Agentensysteme berücksichtigen, existieren noch nicht.

"Die Sicherheit eines Rechensystems ist seine Fähigkeit, eine festgelegte Sicherheitspolitik zu realisieren. Die Frage nach der Sicherheit eines Systems ist also relativ zu den für das System (oder einer spezifischen Anwendung) festgelegten Sicherheitsanforderungen zu beantworten" [ECK 98]

2.8.1 TCSE85 (Orange Book)

Im Jahr 1983 wurde durch das National Computer Security Center des Departments of Defence [NCS 85] ein Katalog von Bewertungskriterien für vertrauenswürdige Computersysteme (TCS) herausgegeben. Die Farbe des Kataloges gab dem Buch den Namen "Orange Book".
Dieser Katalog dient dazu Kriterien bereitzustellen, anhand derer Betreiber und Hersteller von Rechensystemen Sicherheitsanforderungen und deren Realisation bewerten können [POW 93, WAE 93].

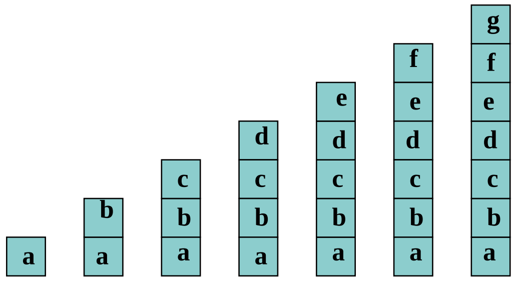
Das Orange Book beschreibt dabei Eigenschaften, die der Einstufung bezüglich der Sicherheit eines Systems dienen. Die Eigenschaften beziehen sich auch auf den jeweiligen Zugriffsschutz. Eine Einstufung kann beispielsweise anhand der unterschiedlichen MAC- und DAC-Zugriffskontrollpolitiken, die im Kapitel 2.5.1 vorgestellt wurden, erfolgen. Diese Eigenschaften werden nachfolgend aufgezeigt, da sie für die weiteren Abschnitte dieser Arbeit relevant sind.

Die Bewertung der Betriebssicherheit wird anhand sechs Hauptkriterien geregelt:
1. Security Policy
2. Marketing
3. Identifikation
4. Accountability
5. Assurance
6. Continuous Protection

Alle für die Betriebssicherheit wichtigen Komponenten müssen permanent und zuverlässig vor unautorisierten Zugriffen geschützt sein. Zur Handhabung des Kriterienkataloges werden sieben Sicherheitsklassen definiert [WAE 93, SUM 97, CLS 94]:

D	: Minimaler Schutz
C1	: Benutzergesteuerter Zugriffsschutz
C2	: Kontrollierter Zugriffsschutz
B1	: Schutz durch Kennzeichnung
B2	: Schutz durch Strukturierung
B3	: Sicherheitsbereiche
A1	: Verifizierter Entwurf

Systeme der Gruppe C gestatten dem Nutzer die Vorgabe der Schutzgrade für Objekte und der Zugriffsrechte der Subjekte auf diese Objekte. Systeme der Gruppe B sind durch festgelegte, unveränderliche Schutzgrade und Zugriffsrechte gekennzeichnet, die jeweils bis zu einer bestimmten Sicherheitsstufe innerhalb eines hierarchisch strukturierten System reichen und die Rechte der darunter liegenden Sicherheitsstufen einschließen. Für Systeme der Gruppe A muss der Nachweis erbracht werden, dass keine Lücken im System enthalten sind.



a) Minimale Sicherheit

b) Benutzerbestimmbarer Zugriffssschutz
Identifizierung
Benutzerhandbuch ueber Sicherheitsmerkmale

c) Aufzeichnungen ueber Sicherheitsverletzungen
Wiederverwendung von Objekten

d) Festgelegter Zugriffsschutz
Kennzeichnung von Objekten

e) Normale Sicherheitspolitik
Kennzeichnung von Subjekten und Objekten

f) Kennung fuer Sicherheitsbeauftragten mit speziellen Rechten
Modularer Aufbau des Sicherheitssystems

g) Foramle Verifikation des Sicherheitssystems

Abbildung 9: Zuordnung von Sicherheitsanforderungen zu Sicherheitsklassen nach [LIE 90]

Die Beispielanalyse von Sicherheitsschwachstellen des Betriebssystems UNIX anhand des Orange Book findet sich ebenfalls in [LIE 90].

2.8.2 Weitere Modelle

Weitere Standardisierungsbeiträge sind die IT-Sicherheitskriterien und die Ergänzungen des Orange Book durch die TNI Trusted Network Interpretation, wobei das Orange Book auf den Geltungsbereich der Netzebene ausgeweitet wurde. Danach stellt ein Rechnernetz entweder ein sicheres, abgeschlossenes System (Single Trusted System View) oder ein offenes System aus einzelnen sicheren abgeschlossenen Systemen dar (Interconnected Accredited Systems View). Der Standard betrachtet somit auch die Eigenschaften von verteilten Systemen.

Ein weiterer Beitrag leistet ECMA mit dem Standard Security in Open Systems - A Security Framework (TC32/TG9). Er definiert Anforderungen an den Security Service. Diese sollen dem Anwender OSI-gerechter Anwendungen in der Schicht 7 bei der Erfüllung von Sicherheitsanforderungen auf der logischen Ebene seiner Subjekte und seiner Objekte unterstützen.

Der Vollständigkeit halber, sei noch das Bell-LaPadula-Modell erwähnt, wobei das Problem der Kontrolle des Zugriffs beschrieben ist. Das Orange Book ist eng mit diesem Modell verbunden. [CLS 94, SUM 97]. Sicherheitsprinzipien werden hier formal beschrieben. Es wird versucht, die Regeln der Informationsflusskontrolle bei eingestuften Informationen formal zu erfassen [POW 93, SUM 97].

Es existieren unterschiedliche Standards, die untereinander jedoch nicht schlüssig sind, da Rücksicht auf die Ausdruckskraft genommen wird [KAR 95].

- X.812
  Das Zugriffsmodell des OSI-Management basiert auf diesem Modell der Zugriffskontrolle.
- POSIX 1003.6
- X.501
- X.741
  ACL-Schema, Zugriffskontrollinformation ist als Managed Objekt (MO) spezifiziert.
- RFC2196
  Das "Site Security Policy"-Handbuch gibt erste Hilfestellungen und Richtlinien für Sicherheitspolitiken und Prozeduren deren Systeme mit dem Internet verbunden sind. Dabei dient es lediglich als Rahmenwerk.
- RFC1445
  Zugriffskontrolle für SNMPv2. Die Entscheidungsfunktion (Access Decision Function) basiert auf einem Subjekt-Paar (Principals), wobei der Kontext von dem Initiator mit dem Kontext des Targets übereinstimmen muss. Dieser Kontext kann eine Gruppe, eine Zugehörigkeit oder eine Rolle sein.

Die Anzahl der Sicherheitsstandards für Sicherheitspolitiken wachsen stetig an.

2.9 Java Sicherheit

Dieser Abschnitt geht detailliert auf vorhandene Konzepte und Mechanismen ein, die innerhalb von Java bezüglich der Sicherheit verwendet werden. Der Schutz der Informationen wird dabei aus der Sichtweise eines Java Programms betrachtet. Es werden vor allem die Konzepte und Schutzmechanismen betrachtet, welche eingesetzt werden, um ein fremdes, heruntergeladenes Programm sicher auf einem System auszuführen. Um Java spezifische Begriffe hervorzuheben, werden sie im weiteren Verlauf zusammen geschrieben (z.B. JavaClassLoader).

2.9.1 Die Java Sicherheitsarchitektur

Betrachtet man eine Java Applikation, kommen die in der Abb. 10 dargestellten Komponenten zum Einsatz. Eine Komponente eines Java Programms ist der Quellcode des Programms. Übersetzt man den Quellcode, bekommt man einen maschinenunabhängigen Byte-Code als Ergebnis. Der Byte-Code ist in heterogenen Umgebungen einsetzbar. Ein Java Runtime System kann diesen Byte-Code dann entweder direkt interpretieren oder ihn compilieren und dann ausführen.
Soll eine Klasse geladen werden, so muss sie den Bytecode Verifier passieren. Dabei kann es sich um lokale Klassen des Systems, entfernte Klassen von anderen Systemen, und um signierte Klassen handeln. Der ClassLoader ist für das Laden der Klassen zuständig.

Der SecurityManager bildet die Schnittstelle zwischen der Core API und dem Betriebssystem. Der SecurityManager verwaltet dabei die gesamten Zugriffe auf die Ressourcen des Endsystems. In Java 2 werden Zugriffskontrollentscheidungen zumeist an den AccessController weitergeleitet. Die SecurityPackages bilden die Basis für die Authentifikation signierter Java Klassen. Die benötigten Daten für die Verifikation können in einer Datenbank abgelegt werden [OAK 00, GON 00].

Die Java Virtual Machine (JVM) stellt eine Umgebung für Java-Programme dar und führt den Bytecode aus [YEL 96b]. Dabei kommen die in der JVM implementierten (Low-Level)

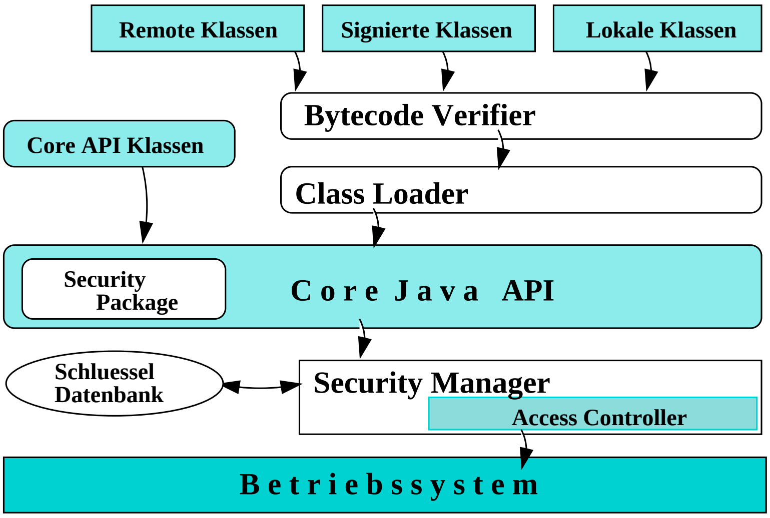


Abbildung 10: Java Applikationsmodell aus [OAK 00]

Schutzmechanismen zum Einsatz. Vom Betriebssystem aus betrachtet ist die JVM ein einzelner Prozess und alle Systemaufrufe der JVM laufen unter der Autorität des Eigentümers der JVM ab.

Die Möglichkeit fremde Programme auf ein System herunterzuladen und auszuführen stellt dabei besondere Anforderungen an die Sicherheitskonzepte. Die in Java eingesetzten Sicherheitskonzepte basieren auf Sicherheitsvorkehrungen der Sprache an sich und des Compilers. Durch den Einsatz des ClassLoaders und dem SecurityManager werden Sicherheitsvorkehrungen auf der Applikationsebene bereitgestellt. Das Java Sicherheitsmodell basiert hauptsächlich auf dem Bytecode Verifier, dem ClassLoader, dem SecurityManager und seit Java 2 auch auf dem AccessController [MOM 99].

Die vier Sicherheitskomponenten sind voneinander abhängig [IBM 99b]. Bei einer Sicherheitslücke eines dieser Elemente ist demnach das ganze System bedroht:

Der SecurityManager und der AccessController müssen sich auf den ClassLoader verlassen. Er kann entfernte von lokal geladenen Klassen unterscheiden und die den Klassen zugeordneten Protection-Domains zuordnen. Die Trennung der Klassen in Namensräumen verhindert Namenskomplikationen, und die Trennung in spezifisch geschützten Domains helfen die lokalen vertrauenswürdigen Klassen vor einem Überschreiben durch andere Klassen zu verhindern.
Der ClassLoader muss sich allerdings auch auf den SecurityManager verlassen, da dieser zum Beispiel verhindert, dass ein Programm einen eigenen ClassLoader ausführt. Wäre dies möglich, so könnten die durch den ClassLoader realisierten Schutzkonzepte umgangen werden.
Die Basis der Vertrauensbeziehungen bildet der ClassFile Verifier. Er stellt sicher, dass die innerhalb eines Programms getroffenen Schutzvorkehrungen auch durchgesetzt werden.

Low-Level Security (Sprache und Compiler)
Wird ein Java Programm ausgeführt, so kommen zuerst die Low-Level Sicherheitsmechanismen von Java zum tragen. Bevor ein Code ausgeführt wird, wird er überprüft und verifiziert. Dadurch wird sichergestellt, dass der Code den Spezifikationen der virtuellen Maschine entspricht. Unzulässige Operationen (Stack Overflow, etc.) werden erkannt und verhindert [YEL 96a].

Die Programmiersprache Java und ihre Ausführungsumgebung bietet Sicherheit durch die Überprüfung der Typ Einhaltung (Strong Type Safety). Das Static Byte Checking in Form von Byte

Code Verification überprüft den Code vor der Ausführung. Unterstützt wird zudem eine Überprüfung zur Laufzeit (Dynamic Checking).
Einem nicht vertrauenswürdiger Code steht ein distinkter Namensraum zur Verfügung. Referenzen zwischen einzelnen Modulen von verschiedenen Namensräumen sind nur bei Methoden die als `public` deklariert worden sind möglich [OAK 00].

Folgend werden die Sicherheitseigenschaften von Java erläutert:

- Sicherheit durch Veröffentlichung des Java Interpreters und Compilers zur Überprüfung
- durch eindeutige Definitionen
  (Zwei korrekte Java Compiler erzeugen nie unterschiedliche Ergebnisse)
  - alle primitive Typen haben eine feste Größe
  - alle Operationen laufen in einer spezifizierten Reihenfolge ab
- Sicherheit durch das Fehlen von Zeigerarithmetik (Lack of Pointer Arithmetic)
- durch die Speicherbereinigung (Garbage Collection)
- Sicherheit durch strikte Überprüfung während der Übersetzung
  (strict Compile-Time Checking)
  - Objekte können ohne einen expliziten Runtime Check keine Typumwandlung vornehmen (nach einer Subklasse)
  - Alle Referenzen auf Methoden und Variablen werden überprüft, auch in Hinsicht der Sicherheits-Barrieren (z.B. einer als privat deklarierten Methode)
  - Integers können nicht nach Objekte konvertiert werden und umgekehrt
  - strikte Objektorientierung
    wobei unterschiedliche Zugriffsebenen existieren
    (private, protected, public und default bzw. package)
  - keine Zeigerstrukturen und somit kein direkter Speicherzugriff
    daher keine Implementierungsfehler in Bezug auf Speicherprogrammierung und Unterbindung von Zugriffen auf einen fremden Speicher
  - Möglichkeit der final-Deklaration
    Final-Deklarierte Methoden und Klassen können nicht mehr überschrieben oder vererbt werden
  - explizite Initialisierung
    Objekte und Variablen besitzen immer einen gültigen Default-Wert
  - Unterstützung atomarer Zugriffe
    Zugriffe werden somit thread-sicher und stabil

Die Sicherheitskonzepte wurden in den einzelnen Java-Versionen weiterentwickelt (vgl. Abb. 11, 12 und 13), und für bestimmte Sicherheitsanforderungen optimiert. Stand bei JDK 1.0 das Sandbox-Security Modell im Vordergrund, so kam bei JDK 1.1 das Konzept des Trusted Code zum Tragen. In JDK 1.2 (Java 2) ermöglicht das Sicherheitsmodell in Java eine feingranulare Zugriffskontrolle anhand von als trusted oder untrusted eingestuften Code ohne großen Implementierungsaufwand. Die Java 2 Security Architecture erlaubt die Umsetzung der Sicherheitsanforderungen durch z.B. Zuweisung einer spezifischen Schutzumgebung an einen fremden Code. Durch diese Zuweisung kann eine Zugriffskontrolle des fremden Codes auf Systemressourcen durchgesetzt werden [OAK 00, GON 98, GON 00, IBM 99b].

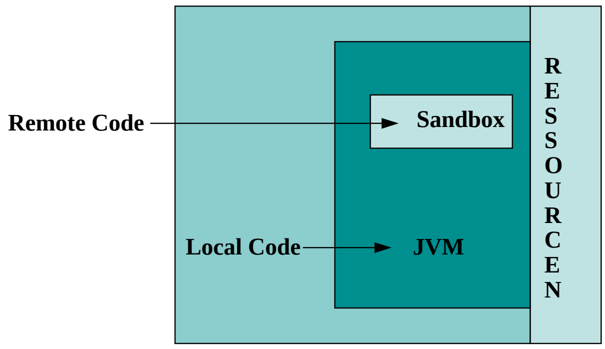


Abbildung 11: Java Sandboxmodell nach [IBM 99b]

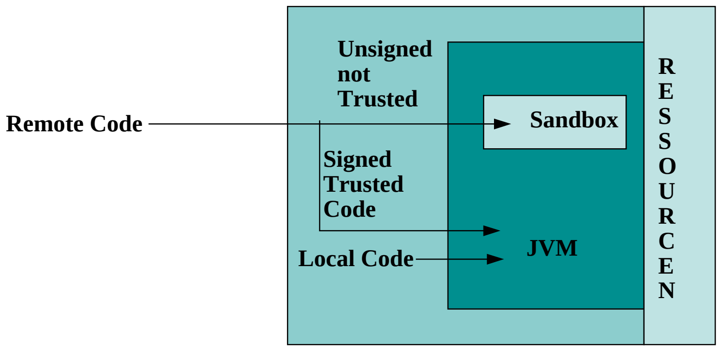


Abbildung 12: Java Trusted/Untrusted Code in Java nach [IBM 99b]

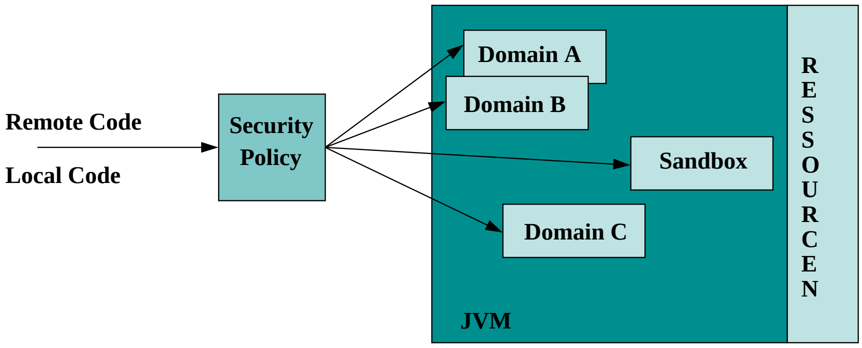


Abbildung 13: Zugriffskontrollmechanismen in Java 2 nach [IBM 99b]

ClassLoader
Der ClassLoader lädt den Bytecode, der den Bytecode Verifier passiert hat, als Klasse in die Java Virtual Machine (JVM) und übernimmt damit die Funktion des Binders. Er ist verantwortlich für das Zusammenfügen der verschiedenen Klassen, so dass das Programm ausgeführt werden kann. Weitere Aufgaben des ClassLoader bezüglich Sicherheitsaspekte sind [OAK 00, IBM 99b, GON 98]:

- Die Zuordnung von Klassen zu Namensräumen (Namespace)
- Die Zusammenarbeit mit dem SecurityManager
- Bevorzugung von lokalen gegenüber remote Klassen

Der ClassLoader ist zudem in der Lage, Klassen dynamisch zu laden, d.h. Programme können zur Laufzeit eingebunden werden. Identifiziert werden kann eine Klasse anhand des Namens und des ClassLoaders, der die Klasse geladen hat. Bezieht sich eine geladene Klasse auf eine noch nicht geladene Klasse, ruft die JVM dieselbe ClassLoader-Instanz auf, die die Klasse geladen hat, um die neue Klasse zu laden. Alle Klassen, die in einer Applikation genutzt werden, werden daher von ein und demselben ClassLoader geladen. Auf Basis dieser Technik kann jeder Applikation ein eindeutiger Namensraum zugeordnet werden. Dadurch kann es nicht zu Namensüberschneidungen der Klassen unterschiedlicher Applikationen kommen [OAK 00].
Durch die Verwendung des ClassLoaders sind Programme mit gleichlautendem Klassennamen ausführbar, und die Programme sind untereinander abgeschottet, wobei Zugriffe auf Klassen von fremden Programmen verhindert werden.

Applets nutzen diese Technik, indem jedes heruntergeladene Applet eine eigene ClassLoader-Instanz bekommt. Der dem Applet zur Verfügung gestellte Namensraum dient der Sicherheit des Systems, aber auch des Applets [OAK 00, GON 98].

Die Möglichkeit des dynamischen Ladens von Klassen ist jedoch schwer in Einklang zu bringen mit den Sicherheitsanforderungen der Type-Safety. Wie kann man wissen, dass der Code Typsicher ist, wenn er noch gar nicht vorhanden ist? Dieser Frage geht D. Dean in seiner Dissertation nach [DEA 98]. Letztendlich führt eine unkontrollierte dynamische Auflösung offener Referenzen zu Sicherheitsproblemen. Der Binder ist daher eine sicherheitskritische Komponente. Beispielsweise enthielten alte Java Versionen Sicherheitsmängel im ClassLoader, wodurch Schutzkonzepte umgangen werden konnten [ECK 98].
Der ClassLoader ist ein wichtiger Bestandteil des Sicherheitmodells. Er arbeitet eng mit dem SecurityManager und AccessController zusammen. Der ClassLoader besitzt als einziger bestimmte Informationen über die Klassen, welche in die JVM geladen worden sind. Nur der ClassLoader kann erkennen, ob eine Klasse lokal oder von einem entfernten Rechner geladen worden ist, und ob es sich um eine signierte Klasse handelt [IBM 99b].

Der in Java 2 vorhandene SecureClassLoader dient als Basis zur Entwicklung eines ClassLoaders mit dem Unterschied, dass der SecureClassLoader mit jeder geladenen Klasse eine ProtectionDomain verbindet. ProtectionDomains dienen dabei als Basis zum Einsatz des AccessControllers. Wird der AccessController verwendet, um die System Security Policy durchzusetzen, so ist die Verwendung des SecureClassLoaders unabdingbar [OAK 00, IBM 99b, GON 00].

Der Einsatz eines SecureClassLoader sieht dann wie folgt aus:

1. Aufteilung der Namensräume
2. Schutz der Core Java-Klassen Packages
3. Realisation der ProtectionDomain, welche die Berechtigungen einer geladenen Klasse festlegt, und die als Basis einer Zugriffskontrolle auf Ressourcen zur Laufzeit genutzt werden kann.

4. Durchsetzung der Such-Hierarchie einer angeforderten Klasse

SecurityManager
Die Aufgabe des SecurityManager besteht darin, Operationen zur Laufzeit einer Zugriffskontrolle zu unterziehen. Der SecurityManager agiert als zentrale Instanz, wobei er die für die Kontrolle benötigten sicherheitsrelevanten Informationen und Mechanismen bereitstellt und die Zugangskontrolle durchführt. Die Zugriffskontrolle wird durch die Verwendung von Ausnahmebehandlungen realisiert.
Der SecurityManager entscheidet z.B., ob eine Datei gelesen werden darf oder nicht, ob sich der Zustand eines Threads ändern darf, ob der Zugriff auf ein Package erlaubt ist bzw. ob eine Verbindung zu einem anderen Rechner erlaubt wird. Der SecurityManager kann zur Entscheidungsfindung auch den ClassLoader heranziehen. So kann er beispielsweise den ClassLoader fragen, von woher die Klasse stammte. Daher kann eine differenzierte Zugriffskontrolle anhand dem Wissen, dass der ClassLoader besitzt, erfolgen. Hierfür muss der SecurityManager nur über die speziell implementierte Schnittstelle des ClassLoaders informiert sein [OAK 00].

Die Voraussetzung damit der SecurityManagers hinzugezogen wird ist die explizite Umlenkung eines zu schützenden Aufrufs auf den SecurityManager [OAK 00].
Die von dem SecurityManager verwendete Zugriffskontrollpolitik ist hart codiert. Durch einen entsprechenden Aufruf einer geschützten Funktion wird der SecurityManager angesprungen, der daraufhin die innerhalb des SecurityManagers codierte Zugriffskontrollpolitik durchsetzt [OAK 00].
In der Abb. 14 ist der Ablauf der Zugriffskontrolle durch den SecurityManager ersichtlich [IBM 99b].

1. Eine Klasse, welche eine zu schützende Operation enthält, wird aufgerufen.
2. Der Aufruf der Operation wird zum SecurityManager umgelenkt.
3. Der SecurityManager überprüft den Aufrufstack, wobei jede Klasse auf ihre Vertrauenswürdigkeit hin überprüft wird. Generell werden Klassen auf dem eigenen Rechner als vertrauenswürdig angesehen, wogegen Klassen von entfernten Rechnern als nicht vertrauenswürdig angesehen werden.
4. Sind alle sich auf dem Stack befindlichen Klassen mit den notwendigen Berechtigungen vorhanden, erfolgt keine Ausnahmebehandlung, und der Aufruf kehrt zurück.
5. Jetzt kann die Klasse die geschützte Operation durchführen.

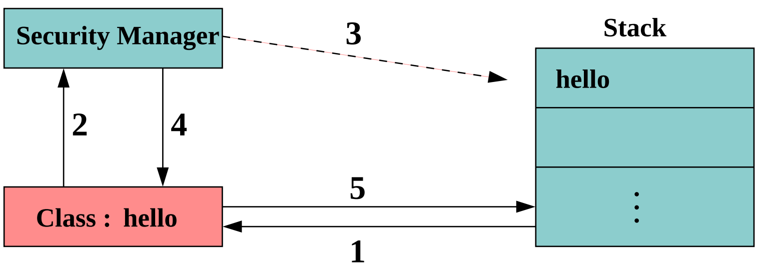


Abbildung 14: SecurityManager

Existiert eine nicht vertrauenswürdige Klasse auf dem Stack, erfolgt eine Ausnahmebehandlung, welche den Zugriff auf das Objekt verhindert.

Betrachtet man die Vorgehensweise des SecurityManagers, wird das Konzept eines Referenz Monitors implementiert, da jeder, als sicherheitskritisch eingestufter Aufruf, die zentrale Kontrollinstanz durchlaufen muss. Ist der Zugriff nicht zulässig, erfolgt eine Ausnahmebehandlung

(Exception). Ist der Zugriff zulässig, ist die Zugriffskontrolle für den Nutzer transparent.

Kommt der Security Manager in einer unmodifizierten Form zum Einsatz, so gibt es folgende Nachteile:

- Beide Klassen, die Klasse, die die Aufrufe macht, und die aufgerufene geschützte Klasse, müssen geladen sein, bevor der Security Manager aufgerufen wird.
- Um eine Klasse zu schützen, muß sie von dem Autor als solche implementiert worden sein. Dabei muß jeder zu schützende Methodenaufruf der Klasse explizit den Security Manager zur Verifikation aufrufen. Dies passiert alles zur Laufzeit, was einen zusätzlichen Overhead zur Folge hat.
- Kommt eine neu zu schützende Operation hinzu, so muß der Security Manager modifiziert und danach an alle Systeme verteilt werden, die diese neue Operation nutzen sollen.
- Ruft eine vertrauenswürdige Klasse eine geschützte Operation auf, so kann sie eventuell abgewiesen werden, falls sie von einer nicht vertrauenswürdigen Klasse aus aufgerufen worden ist.
- Durch den Einsatz von digitalen Signaturen wird ermöglicht, dass aufgrund der CodeSource als nicht vertrauenswürdig eingestufter Code als vertrauenswürdiger Code angesehen werden kann.

Seit Java 2 kann jedoch auch zusätzlich der AccessController zur Entscheidungsfindung herangezogen werden, der im Gegensatz zum SecurityManager eine klare Trennung der Zugriffskontrollpolitik von dem Durchsetzungsmechanismus besitzt. Die Zugriffskontrolle erfolgt aus Kompatibilitätsgründen durch den SecurityManager, jedoch leitet der SecurityManager die Zugriffskontrolle an den AccessController weiter.

Access Controller
Der AccessController existiert erst ab Version 1.2 und ersetzt einige Funktionalitäten des SecurityManagers [OAK 00].
Der Vorteil des AccessController besteht darin, dass für die Änderung der Sicherheitspolitik nicht mehr der Programm Code modifiziert werden muss [OAK 00]. Die Änderungen können nun benutzerfreundlich durch Einträge in einer Policy-Datei vorgenommen werden, welche von dem AccessController durchgesetzt wird [MOM 99]. Dabei sind die zu vergebenden Zugriffsrechte weitaus flexibler und feingranularer. Lokale Klassen können nun auch einer Beschränkungen unterworfen werden.

Der AccessController basiert auf Konzepten, die durch spezielle Klassen implementiert werden und folgend vorgestellt werden:

1) **Code Source**
Objekte gleicher Herkunft werden zusammengefasst. Die Zusammenfassung basiert in der Standard System Security Policy von Java 2 auf Basis der CodeBase (URL) und eventuell vorhandener Signaturen.

2) **Permissions**
Permissions sind Rechte, die ausdrücken, welche Operation durchgeführt werden dürfen.

3) **System Security Policy**
In der Standard System Security Policy von Java 2 werden Zugriffsrechte (Permissions) auf Basis der CodeSourcen vergeben.

4) **Protection Domain**
Die ProtectionDomains stellen einen Schutzbereich dar, in dem CodeSourcen mit gleichen Permissions zusammengefasst werden.

Der AccessController verwendet Stack Inspection, kontrolliert also ebenfalls den Stackinhalt. Im Gegensatz zum SecurityManager ist hier jedoch nicht mehr die Klassen-Tiefe (Class-Depth), sondern die in den sich auf dem Stack befindlichen ProtectionDomains enthaltenen Permissions-Objekte relevant. Da die ProtectionDomains, die für die jeweilige Klasse zugeteilten Berechtigungen enthalten, kann entschieden werden, ob alle Klassen des aktuellen Aufrufstacks, die für den Zugriff notwendige Berechtigungen besitzen.

1) CodeSource
Die Code Herkunft wird durch ein `java.net.URL`-Objekt repräsentiert. Die Liste der Signierer wird durch ein Array von `java.security.cert.Certificates`-Objekte repräsentiert. Der Konstruktor der CodeSource-Klasse sind folgendermassen aus:

```
CodeSource(URL url, Certificate[] certs)
```

Die URL-Herkunft bekommt man durch die `getLocation()`-Methode und die Zertifikat durch die `getCertificates()`-Methode.

2) Permissions
Die Permission-Klassen repräsentieren den Zugriff zu den System Ressourcen. Das `java.security`-Package stellt die abstrakte Permission-Klasse bereit, um die spezifischen Zugriffe zu repräsentieren. Es sind einige Subklassen in der Java Core API vorhanden. Man kann auch seine eigene Permission-Klassen spezifizieren, indem man eine Subklasse der betreffenden Klasse erstellt oder vorhandene Subklassen (z.B. `java.security.BasicPermission`) nutzt [OAK 00].

Spezifische Zugriffe werden von Permission-Klassen repräsentiert, die zumeist im dafür zuständigen Package enthalten sind. So ist die Permission-Klasse FilePermission ein Teil des `java.io`-Package.

Die vorhanden Permission-Klassen im `java.security`-Package sind [OAK 00]:
AllPermission, BasicPermission, SecurityPermission, UnresolvedPermission

Desweiteren gibt es noch abstrakte Klassen:
`java.security.PermissionCollection`
(Repräsentation einer Sammlung homogener Berechtigungen)
und `java.security.Permissions`
(Sammlung heterogener Permission-Objekte)

3) System Security Policy
Die in der Java 2 Security Architecture eingesetzte Policy-Klasse spezifiziert, welche Zugriffsrechte einer CodeSource zugesprochen werden soll.

Das `java.security`-Package enthält die abstrakte Policy-Klasse, die die Sicherheitspolitik der Java Applikationsumgebung (System Security Policy) repräsentiert. Das Ziel dieser Klasse ist zu erkennen, welche Berechtigungen (Permissions) einer CodeSource in der Policy-Datei zugeteilt worden sind.
Die Sicherheitspolitik wird durch eine Subklassen-Policy repräsentiert, die eine Implementierung einer abstrakten Methode in der Policy-Klasse enthält.

Die Herkunft der Policy-Informationen, um das Policy-Objekt zu generieren, hängt von der Implementierung ab. So kann die Policy-Konfiguration in einer ASCII-Datei, in einer serialisierten Binary-Datei der Policy-Klasse oder in einer Datenbank enthalten sein [POL 00].

Die Standardimplementierung der `java.security.Policy`-Klasse wertet die Zugriffskontroll-Policies, die aus einer Menge an Policy-Einträgen besteht, aus. Die Standardkonfiguration wertet eine systemspezifische und eine benutzerbestimmten Zugriffskontrollpolitik aus. In einer

Umgebung mit mehreren Rechnern und Nutzern, kann bzw. muss für jeden Rechner und jeden Nutzer eine Zugriffskontroll-Policy definiert werden. Zur Vereinfachung kann die Definition der Zugriffskontroll-Policy nicht als Einheit spezifiziert werden, sondern als eine Einheit zusammengesetzter Teil-Policies [FAL 00].

Die Default Policy-Klasse wird durch die `PolicyFile`-Klasse von Sun implementiert.

```
policy.provider= sun.security.provider.PolicyFlile
```

Die Permissions werden auf Basis einer Policy-Datei erstellt, wodurch die `PolicyFile`-Klasse die Policy-Datei zuvor parst.
Anhand der Policy-Klasse wird letztendlich dem CodeSource Berechtigungen zugeteilt.

Per Default Einstellung werden zwei Policies durchgesetzt. Dies wird durch die System Security-Datei (Security Property File) festgelegt und kann auch geändert werden. Hier werden einige Eigenschaften bezüglich der Sicherheit festgelegt.

Innerhalb der System Security-Datei werden für die JVM global gültige Eigenschaften und Parameter definiert. Die Datei enthält in der Default Einstellung unter anderem den Eintrag: `policy.provider=security.provider.PolicyFile`. Hier wird die Klasse spezifiziert, welche als System-Policy instanziiert wird. Der Default Eintrag ist die Klasse `PolicyFile` im Package `sun.security.provider`. Diese Klasse definiert daher die default Java 2 SDK Policy Implementierung, welche die Policy-Dateien zur Sicherheitskonfiguration der JVM verwendet.
Durch den Eintrag `policy.url.nummer` kann der Ort der Policy-Datei festgelegt werden. Die Datei kann sich auch auf einem entfernten System befinden, wodurch sich z.B. auch Systemweite Policy-Dateien auf einem Policy-Server befinden können [IBM 99b, OAK 00, GON 00].

4) Protection Domain
Eine ProtectionDomain wird verwendet, um einer CodeSource mit der ihr zugewiesenen Berechtigungen zu gruppieren [OAK 00].
Die ProtectionDomain-Klasse repräsentiert eine Schutzeinheit innerhalb einer Java-Applikationsumgebung. Die Argumente des Konstruktors sind CodeSource-Objekte und PermissionCollection-Objekte. Die PermissionCollection-Objekte stellten die Berechtigungen dar, die dem CodeSource-Objekt angehören.

```
ProtectionDomain(CodeSource codesource, PermissionCollection permissions)
```

- Klassen eines anderen CodeSource mit den selben Permissions gehören verschiedener ProtectionDomains an.
- Jede Klasse gehört nur einer ProtectionDomain an. Diese hängt vom CodeSource und den zugestandenen Permissions ab.
- Alle Klassen desselben CodeSource gehören der selben ProtectionDomain an.

Jede Klasse, die in die JVM geladen wird, wird einer einzigen ProtectionDomain zugeordnet. Diese Zuordnung basiert auf den CodeSource der Klasse. Mehrere Klassen können einer ProtectionDomain zugeordnet sein. Eine ProtectionDomain kann eine oder mehrere Berechtigungen beinhalten, wobei die einzelnen Berechtigungen in jeder ProtectionDomain enthalten sein können.

Weitere Sicherheitsdienste (JCA, JSSE, JAAS)
Folgende Funktionalitäten runden die Sicherheitskonzepte in Java ab. Java Cryptography Architecture (JCA) ist eine Architektur, die Java Cryptography Extension 1.2 (JCE) in JDK 1.2 implementiert. Um eine sichere Kommunikation zu bewerkstelligen, gibt es die Java Secure Socket Extension (JSSE). Außerdem existiert noch ein Rahmenwerk, das Java Authentication and Authorization Service (JAAS) [SUN 00a], welches Authentifikation des Users und eine

Zugriffskontrolle realisiert. Zuletzt können die Applikationen selbst eigene spezifizierte Sicherheitsmechanismen implementieren, wobei die Sicherheitfeatures der Java Plattform verwendet werden können.

Java Authentication and Authorization Service (JAAS)
JavaTM Authentication and Authorization Service (JAAS) erlaubt die Authentifikation und Durchsetzung der Zugriffskontrolle anhand von Benutzern. JAAS implementiert dabei das Standard Pluggable Authentication Module (PAM), welches ein Rahmenwerk darstellt und die Zugriffskontrollarchitektur der Java 2 Security Architecture um die benutzerbasierte Rechtevergabe erweitert [SUN 00b].

Durch JAAS ist eine Zugriffskontrolle auf Basis der CodeSource und der Entität (Principal), die das Programm ausführt möglich [SUN 00c]. Hierfür muss die Authentifikation des Benutzers erfolgen und eine Entscheidung, anhand der Identität des Benutzers, getroffen und durchgesetzt werden. Ein Principal in Java repräsentiert dabei eine Entität (z.B. einen individuellen Benutzer). Um JAAS nutzen zu können wird die Standard Edition Version 1.3 benötigt.

Die Politik-Sprache wurde in JAAS wie folgt erweitert:

```
grant CodeBase ["URL"],
      Signedby ["signers"],
      Principal [Principal_Class] ["Principal_Name"],
      Principal ... {

   permission Permission_Class ["Target_Name"]
                               [, "Permission_Actions"]
                               [, signedBy "SignerName"];
   ...
};
```

Dadurch sind folgende Beispieleinträge innerhalb der Policy-Datei möglich:
Beispiel 1: JAAS Benutzer basierte Policy

```
grant Codebase "http://www.haendler.de, Signedby "Haendler",
      Principal bar.Principal "Messe" {
   permission java.io.FilePermission "/temp", "read";
}
```

Beispiel 2: Der Rolle Administrator werden Rechte zugeteilt

```
grant Principal provider.Role "administrator" {
   permission java.io.FilePermission "/results/-", "read, write";
}
```

Beispiel 3: Einem Entwicklungsgruppe (Group) werden Rechte zugeteilt

```
grant Principal hersteller.Team "Messen" {
   permission java.io.FilePermission "/team/-", "read";
}
```

Die JAAS Policy-Datei erlaubt auch, dass die Principal-Klasse ein PrincipalComparator ist. (Die Klasse implementiert das PrincipalComparator Interface). Die Berechtigung wird jedem Subjekt erteilt, welches der PrincipalComparator impliziert, wodurch Gruppierungen von Benutzern unterstützt werden.

```
public interface PrincipalComparator {
    boolean implies(Subject subject);
}
```

Beispiel 4: Benutzer “user” können auf das Temp-Verzeichnis lesend und schreibend zugreifen.

```
grant Principal bar.Role "user" {
   permission java.io.FilePermission "/temp/-", "read, write";
}
```

Finden Überprüfungen während der Ausführung statt, durchquert der Java 2 SecurityManager die JAAS-Politik, und erneuert daraufhin den momentanen AccessControlContext mit den zusätzlichen Berechtigungen, die dem Subjekt und dem CodeSource gewährt wurden. Wurde die Aktion beendet, wird das Subjekt von dem aktuellen AccessControlContext entfernt und das Ergebnis wird dem Aufrufer übergeben [SUN 00c].

Um die Verbindung des Subjekts mit dem aktuellen AccessControlContext zu realisieren, wird eine interne JAAS Implementierung des `java.security.DomainCombiner` Interfaces verwendet.

Zusätzliche Sicherheitsfunktionen in Java 2

In Java 2 stehen noch weitere Sicherheitsfunktionen zur Verfügung. So werden in Java 2 multiple Signaturen unterstützt. Die SignedObject-Klasse ermöglicht die Erstellung authentischer Objekte die sich in Ausführung befinden und deren Integrität sichergestellt ist. Ein SignedObject enthält ein serialisierbares Objekt und seine Signatur.
Es kamen zwei neue Subpackages zum `java.security.package` hinzu (`java.security.cert` und `java.security.spec`). Diese Pakete dienen der Handhabung der X.509 Zertifikate (Certificate Revocation Lists (CRLs) und Certificate Signing Requests (CSRs)). Es werden nun auch X.509-Zertifikate V3 unterstützt. Ein zusätzliches Zertifikat Interface, das X509 Extension Interface, ermöglicht zusätzliche Attribute mit privaten oder öffentlichen Schlüssel, um die Zertifikathierarchie und die CRL Verteilung zu managen.
Sicherheitskritische Zugriffe können in der Java 2 Security Architecture aufgezeichnet werden, indem die `java.security.debug` Systemeigenschaft gesetzt wird.
Anstatt die Klassentiefe auf dem Stack (Class-Depth) einer Klasse angeben zu müssen, um privilegierten Code ausführen zu können, kann die `doPrivileged()`-Methode eingesetzt werden, wodurch sensitiver Programmcode an einem bestimmten Platz abgelegt und unter einem privilegiertem Modus ablaufen (Namen Lexical Scoping of Privilege Modification) kann. Damit wird unter anderem eine flexible Zugriffskontrolle zur Laufzeit ermöglicht [IBM 99b, S.91: Run-Time Access Controls].
Java bietet auch Konzepte, welche zur Adressierung von Subjekten herangezogen werden können. Ein Principal in Java repräsentiert eine Entität (z.B. einen individuellen Nutzer). Um dies zu ermöglichen, definiert das `java.security`-Package ein Principal-Interface. Eine Gruppe von Principals wird durch ein Group-Interface repräsentiert (vgl. `java.security.acl`-Package).
Das Guard Interface wird zur Erzeugung eines Objekts benötigt, welches eine bestimmte Ressource schützt. Derjenige, der eine Ressource zur Verfügung stellt, kann ein Objekt erstellen, welches diese Ressource repräsentiert und in ein GuardedObject, also einem beschützten Objekt, einkapselt. Der Zugriff auf eine so geschützte Ressource ist nur möglich, wenn die Sicherheitsüberprüfungen im Guard-Objekt erfolgreich abgeschlossen sind.

Es gibt unterschiedliche Sicherheits-Tools, die den Umgang mit spezifischen Mechanismen erleichtern.

- JAR (Java Archive)
  Einpacken einer Klasse in ein Archive

- keytool [-genkey][-export][-import]
  Das keytool-Tool kreiert public- und privat- Keys

- jarsigner
  Signieren und Verifizieren der JAR-Dateien

- policytool
  Änderung der Policy-Einträge, vermeidet unkorrekte Einträge in der Policy-Datei.

2.9.2 Rechtevergabe in Java 2

Die Autorisation von Zugriffen auf Systemressourcen erfolgt durch Einträge innerhalb der System Security Policy Konfiguration. Die Autorisation erfolgt auf Basis der CodeSource. Die CodeSource beinhaltet die CodeBase, also den Ort von wo aus die Klasse heruntergeladen worden ist und eine eventuell vorhandene Signatur. Es können einerseits Standard Zugriffsrechte (Permission-Objekte) der Java API oder aber auch selbst erzeugte Zugriffsrechte (Permission-Objekte) vergeben werden. Betrachtet man die Granularität der in der Security Policy adressierbaren Entitäten, so sind Objekte feingranular adressierbar. Die Subjektadressierung ist jedoch auf die CodeSource beschränkt. Um eine spezifischere Adressierung zu ermöglichen, existiert ab Java 1.3, der Java Authentication and Authorization Service (JAAS), wodurch ein Subjekt innerhalb der Security Policy zusätzlich, anhand eines spezifizierten Principals (Benutzer), adressiert werden kann (vgl. Kapitel 2.9.1).

Die Autorisation erfolgt nach dem Fail-Safe Defaults Prinzip (vgl. Kapitel 2.3), wobei jeder, nicht explizit autorisierte Zugriff, unterbunden wird. Jedoch muss ein sicherheitskritischer Zugriff explizit durch den Implementierer als solchen deklariert und einer Zugriffkontrolle unterworfen werden. In Java 2 sind vertrauenswürdige Klassen nur noch die des Boot Class Paths, welcher intern in der JVM spezifiziert ist (`sun.boot.class.path` Property). Alle anderen Klassen müssen in der Regel den Verifier und die Sicherheitspolitik passieren.

Wird die Sicherheitspolitik nur durch den SecurityManager implementiert, ist das sogenannte Seperation of Policy and Mechanism Prinzip (vgl. Kapitel 2.3) verletzt. Die fehlende Trennung zwischen dem Entscheidungs- und Durchsetzungsmechanismus und der Zugriffskontrollpolitik führt zu einer unflexiblen Zugriffskontrolle [ECK 98]. Hier ist die Verwendung der Policy-Klasse von Java 2 vorzuziehen.

Applikationen können ihre eigene Policy-Subklassen unter der Verwendung der abstrakte Methode der `java.security.Policy`-Klasse implementieren. Es können mehrere Instanzen dieser Klasse vorhanden sein, wobei jedoch immer nur eine Policy durchgesetzt werden kann. Das gegenwärtig installierte Policy-Objekt kann durch die `getPolicy()`-Methode erhalten werden. Programme mit der nötigen Berechtigung können das gegenwärtig installierte Policy-Objekt durch den Aufruf der `setPolicy()`-Methode, ändern.

Bei einem Multiuser-System kann der Systemadministrator eines Systems eine systemweite Default Policy-Datei festlegen, wobei jeder einzelne Nutzer diese um eine eigene Policy-Datei erweitern kann.

Die Autorisationen erfolgen auf Basis spezifischer Zugriffsrechte (vgl. Kapitel 2.6.2), die neben der Aktion auch das Objekt adressieren. Es existieren spezifische Zugriffsrechte, die um weitere anwendungsspezifische Zugriffsrechte erweitert werden können.

Beispiel: Zugriffsrecht-Eintrag in die Policy-Datei

```
grant Codebase "http://foo.com", Signedby "foo" {
  permission java.io.FilePermission "/temp/-", "read";
  }
```

Die Beispiel Zugriffskontrollpolitik erlaubt dem von http://foo.com heruntergeladenen und von foo signiertem Code, alle Dateien unterhalb des Temp-Ordners zu lesen.

Durch die Erweiterung mit JAAS (vgl. Kapitel 2.9.1) ist es möglich zusätzlich noch das Kriterium des Ausführers des Codes hinzuzuziehen.

Beispiel: Zugriffsrecht-Eintrag JAAS

```
grant Codebase "http://www.mnm.com, Signedby "mnm",
  Principal foo.Principal "hanisch" {
      permission java.io.FilePermission "/temp/hanisch/-", "read";
}
```

Dieser Eintrag, der durch JAAS ermöglicht wird, erlaubt nur den von "mnm" signierten Code von der Web-Seite "www.mnm.com" den Lesezugriff auf das Verzeichnis /temp/hanisch/ und allen weiteren Unterverzeichnissen.

Der Nutzer kann also in seinen Sicherheitspolitiken exakt definieren, auf welche Ressource ein bestimmter Code, einer bestimmten URL und (oder) einer bestimmten Signatur zugreifen darf.

Durch die Verwendung spezieller Überprüfungsmechanismen kann zudem die Benutzung von nicht autorisierten Methoden, die jedoch zur Aufgabenerfüllung spezifischer Methoden benötigt werden, erlaubt werden. So kann durch die Methode `AccessController.doPrivileged()` eine Klasse auf dem Stack als privilegiert markiert werden. Dies hat zur Folge, dass ab dieser Klasse keine weiteren ProtectionDomains kontrolliert werden. Zusätzlich besteht auch noch die Möglichkeit einen Snapshot eines Aufrufkontextes zur Entscheidungsfindung heranzuziehen. Dies ist dann notwendig, wenn die Überprüfung auf Basis eines speziellen Aufrufkontextes erfolgen muss.

Betrachtet man die Standardimplementierung der Java 2 Security Policy als Matrix-Modell, so erfolgt die Speicherung der Sicherheitsmatrix zeilenweise. Die zeilenweise Abspeicherung ist bei Mobilen Agenten Architekturen vorzuziehen, da sich die Subjekte häufiger ändern. Desweiteren wird durch die Standardimplementierung ein DAC-Kontrollmechansimus realisert.

2.9.3 Rechtedurchsetzung in Java

Die Rechtedurchsetzung in Java erfolgt ausnahmslos über Ausnahmebehandlungen. Methoden und Operationsaufrufe werden überprüft und falls spezifische Bedingungen nicht zutreffen, wird durch eine Ausnahmebehandlung der weitere Verlauf blockiert, wodurch der Zugriff unterbunden wird. Dies erfordert ein explizites Anspringen einer Überprüfungsmethode vor dem Ausführen der eigentlichen Operation [ECK 98].

Der SecurityManager ist verantwortlich für die Rechtedurchsetzung. In Java 2 werden die grundsätzlichen Überprüfungen der Zugriffe auf Ressourcen des Endsystems von dem Security-Manager an den AccessController weitergeleitet. Der AccessController ist in diesem Fall für die Rechtedurchsetzung verantwortlich. Der SecurityManager, als auch der AccessController betrachten den Stack, bzw. den Stack Context, um eine Entscheidung bzgl. des Zugriffs treffen zu können.

In Java 2 sieht die Überprüfung durch den AccessController wie folgt aus:
Um Zugriffskontrollentscheidungen zu treffen, wird die Methode `getPermissions(codesource)` einer konkreten Implementierung der abstrakten Klasse `java.security.Policy` aufgerufen. Die Methode bestimmt auf Basis der durchgesetzten Policy die Menge der Berechtigungen (Permissions), die die aufrufende Klasse besitzen soll. Die konkrete Implementierung der Policy-Klasse parst die Policy-Datei und erstellt auf Basis der Einträge ein Permissions-Objekt. Die Policy enthält die durchzusetzenden Zugriffskontrollregeln in Form von Grant-Einträgen. Die Policy

Einträge werden der Reihe nach ausgewertet.

Für einen erfolgreichen Zugriff muss jede ProtectionDomain einer Klasse des überprüften AccessControllContexts, der die ProtectionDomains des Stacks enthält, die notwendigen Berechtigungen in Form von Permission-Objekten beinhalten.

2.9.4 Limitierung der Systemressourcen in Java

Durch die Verwendung der interpretierbaren Sprache Java ist die Ressourcenkontrolle bestimmter Ressourcen oft gar nicht oder nur mit sehr viel Aufwand möglich, da der Java-Interpreter die dafür benötigten Funktionen, zum Zugriff auf Laufzeitdaten, nicht zur Verfügung stellt. Es ist in Java momentan nicht möglich, sich einen Überblick über den aktuellen Speicherbedarf eines Objekts zu verschaffen. Wird das Agentensystem und der Agent vom selben Interpreter verwaltet, kann man nicht erkennen, wie sich die Ressourcen auf die verschiedenen Agenten und das Agentensystem aufteilen [KUH 96].

In Java sind daher keine Mechanismen vorhanden um Systemressourcen bezüglich der Nutzungsdauer und Nutzungslast zu limitieren. Dies rührt auch daher, weil die traditionellen Java Scheduler einfache Round Robin Systeme darstellen. Können zum Beispiel noch einzelne Netzwerkverbindungen anhand der Anzahl von Netzwerkverbindungen beschränkt werden, ist die Übertragung von Datenmengen nicht limitierbar.

Es gibt jedoch auch Entwicklungen, die sich auf die Limitierung der Ressourcen konzentrieren. Das Java Betriebssystem, welches von dem Flux Projekt in Utah erstellt wurde, ist momentan eines der seltenen Systeme, welches Ressourcen Limitierung in Java realisiert. Dabei arbeitet man mit traditionellen Prozess-Modellen.

2.9.5 Einschränkungen von Java

- Sicherheitsprobleme
  Java ist nicht immun vor Sicherheitslücken. Die Implementierung eines sprachbasierten Sicherheitsmodells ist nicht einfach und es existieren viele Möglichkeiten Fehler zu machen.

- Ressourcenkontrolle
  Die Ressourcenkontrolle wird nicht vollständig unterstützt. So bietet Java keine Möglichkeit an, Prozessor und Speicher Ressourcen, die von einem Objekt oder Thread benutzt werden, einzuschränken.

- Accountability
  Es gibt keine Möglichkeiten die verbrauchten Ressourcen einem bestimmten Nutzer zuzuordnen.

- Es werden keine Nutzergruppen unterstützt.

- Zugriffseinschränkungen auf Objekte
  Es ist zur Zeit nicht möglich, den vollen Ausführungsstatus eines Objekts abzufragen. Informationen, die den Status eines Programmezählers und den Frame Stack betreffen, sind Java Programmen nicht zugänglich.

- Java-Klassen können dekompiliert werden [CHE 00].

Wie bei jeder Software ist jedoch auch Java nicht 100% sicher! So wurde beispielsweise der Java 2 Bytecode Verifier erfolgreich attackiert. Unter bestimmten Umständen untersucht die Java Virtual Machine nicht den gesamten Bytecode, der geladen wird. So kann ein Angreifer Anweisungen einschmuggeln, welche die für Java typischen Sicherheitsmechanismen unterlaufen. Karsten Sohr ist es gelungen zu zeigen, dass sich tatsächlich auf diese Weise über das Internet in einen fremden Computer eindringen lässt. (Das Problem ist relativ einfach durch Hinzufügen einer einzigen Programmierzeile zu lösen [SOM 00]).

2.9.6 Zugriffskontrollkonzepte in Java

Neben der Default Implementierung der Zugriffskontrolle in Java haben sich erweiterte Konzepte etabliert. Folgend sollen die verbreitesten Konzepte vorgestellt werden. Darunter gehört das Capability Konzept, das Name Space Management und das Extended Stack Introspection, welches beispielsweise in den gängigen Browsern zum Einsatz kommt.

Capabilities
In Java stellen Capabilities eine simple Referenz auf ein Objekt dar. Java Type Safety bewahrt dabei die Objektreferenzen vor Missbrauch. Dabei werden Zugriffe auf Methoden oder Variablen blockiert, die nicht als `public` deklariert worden sind [BAF 00].

Die momentanen Java Class Libraries benutzen bereits ein Capability basiertes Interface, um offene Dateien und Netzwerkverbindungen zu repräsentieren. Mithilfe statischer Methodenaufrufe werden die entsprechenden Capabilities erworben. In einem weiter ausgebauten Capability-basiertem System werden alle System Ressourcen (auch um eine Datei erst zu öffnen) mithilfe Capabilities repräsentiert. In einem solchen System wird einem Applet ein Array mit Capabilities bei der Initialisierung übergeben.

Bei einem flexiblen Sicherheitsmodell wird das System die Sicherheitspolitiken (Security Policies) vor dem Appletstart auswerten und dem Applet daraufhin die Capabilities überreichen, die dem Applet die in der Security Policy autorisierten Zugriffe ermöglicht. Alternativ kann auch ein sogenannter Broker einer authentifizierten Identität Capabilities übergeben.

Die Java Capabilities stellen einen Vermittler als exklusives Interface zu den System Ressourcen dar. Ein Programm muss z.B. die File System Capabilities, die es bei der Initialisierung oder durch einen Broker erworben hat benutzen, um auf Dateien zugreifen zu können. Die herkömmliche File-Klasse oder der Public Konstruktor FileInputStream darf somit nicht mehr genutzt werden. Damit die Sicherheit gewährt ist, sollte der Zugriff nur stattfinden können, falls ein Aufrufer das entsprechende Capability erhalten hat.
Die Schwierigkeit besteht darin, keine andere Zugriffe zu ermöglichen, die die Sicherheitsvorkehrungen umgehen.

Da eine Capability nur eine Referenz auf ein Java-Objekt darstellt, kann das referenzierte Objekt seine eigene Sicherheitspolitik (Security Policy) implementieren und gegebenenfalls die Argumente, bevor sie an privaten, internen Methoden übergeben werden, überprüfen. Letztendlich kann eine Capability eine Referenz auf eine weitere Capability beinhalten. Solange beide Objekte dasselbe Java Interface implementieren, können die Capabilities voneinander unterschieden werden.

Mit den momentanen Java Class Libraries kann ein Programm zum Öffnen einer Datei seinen eigenen FileInputStream konstruieren. Um dies zu verhindern, darf der FileInputStream keinen `public`-deklarierten Konstruktor haben oder die FileInputStream-Klasse muss vor Programmen versteckt werden. Diese Restriktionen betreffen natürlich auch andere Datei-relevante Java-Klassen, wie z.B. die RandomAccessFile-Klasse. Während Java die Capabilities unterstützt, sind signifikante Änderungen der Java Runtime Libraries und der, von den Libraries abhängigen, Programme notwendig, um sicherstellen zu können, das der Schutzmechanismus nicht mehr umgangen werden kann.

Extended Stack Introspection in Java
Variationen dieser Mechanismen finden sich in NS4.0, MSIE4.0 und Sun JDK1.2. Ein vereinfachtes Modell des Stack Inspection kommt im NS3.0 zum Einsatz [NET 00, DFW 84, ROS 00, BAF 00]. Extended Stack Introspection in den ersten Versionen von Java besaß ebenfalls die in Kapitel 2.6.6 aufgezeigte Confused Deputy-Problematik:
Das einfache Stack Inspection Modell erlaubte dem System Code optional die Privilegien freizugeben. Bevor eine Datei geöffnet oder eine andere Operation ausgeführt wird, überprüfte das System, ob diese Privilegien freigegeben worden sind. Wurde vom System Code ein Privileg freigegeben und danach ein nicht vertrauenswürdiges Applet ausgeführt, konnte das Applet unautorisiert auf eine Ressource zugreifen. Diese Attacke ist auch unter dem Namen "Luring Attack" bekannt. Netscape löste dieses Problem, indem die Privilegien aller Stack Frames bei der Suche berücksichtigt werden. Findet sich in einem Stack Frame ein nicht vertrauenswürdiger Code, so wird das Privileg verweigert.

Der momentan in Java eingesetzte Stack Inspection Algorithmus betrachtet nicht nur den nicht vertrauenswürdigen Code und den vertrauenswürdigen Systemcode. Das System unterstützt Code einer unbegrenzten Anzahl von Principals, wobei jedem Principal verschiedene Levels des Vertrauens ausgesprochen werden kann. Es können Targets definiert werden, die verschiedene spezifische Privilegien darstellen.

Extended Stack Introspection (vgl. Abb. 15) kommt z.B. bei den Browsern von Netscape und Microsoft zum Einsatz. JavaSoft arbeitet auch am Einsatz von Extended Stack Introspection. Für den Einsatz benötigt man die Operationen:

- enablePrivilege(target)
- disablePrivilege(target)
- checkPrivileg(target)

Für jede geschützte Ressource wird ein Target erzeugt. Will das System auf die geschützte Ressource zugreifen, so muss es zuerst die Methode `checkPrivileg()` aufrufen. Will ein Code (Im Beispiel Code "A") diese Ressource nutzen, muss er zuerst die Methode `enablePrivileg()` aufrufen. Die Policy Engine wird daraufhin kontrollieren, ob dieser signierte Code (und somit ein Principal) die Erlaubnis besitzt, diese Ressource zu nutzen. Falls dies der Fall ist, wird ein `enabled Privilege P` erstellt und auf dem Stack des Codes abgelegt. `checkPrivileg(target)` sucht im Stack Frame des Aufrufers nach dem Privileg für das entsprechende Target. Wird es nicht gefunden, so hat dies eine Ausnahmebehandlung (Exception) zur Folge. Wird in dem Stack ein Stack Frame gefunden, das keine Berechtigung auf diese Ressource hat, wird der Zugriff verweigert (Verhinderung von Luring Attacks).

Netscape und Explorer bieten eine Reihe vordefinierter Targets, die Ressourcen repräsentieren, an. Microsoft erlaubt nur voll vertrauenswürdigen Codes neue Targets zu definieren, während Netscape jedem erlaubt, ein Target zu erstellen. Somit können Entwickler von Libraries ihre eigenen geschützten Ressourcen definieren.

Name Space Management in Java
Beim Name Space Management handelt es sich um eine Modifikation des Java's Dynamic Linking Mechanismus. Dieser Mechanismus wird benutzt, um für Applets sichtbare Klassen zu verstecken oder auszutauschen. Name Space Management in Java wird realisiert, indem der Java ClassLoader modifiziert wird. Klassen von vertrauenswürdigen Subsystemen, die in nicht-vertrauenswürdigen Codes enthalten sind, haben unterschiedliche Principals. Sie können somit Klassen, die nur für vertrauenswürdigen Code sichtbar sind, einsehen [BAF 00].

Schwierig ist der Einsatz von Name Spaces bei geteilten Klassen. So nutzt das Dateisystem und das Netzwerksystem die Klassen InputStream und OutputStream. Diese können nicht einfach

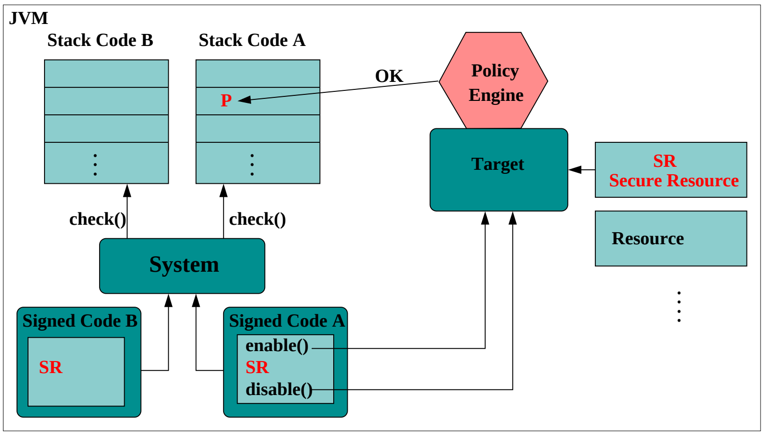


Abbildung 15: Extended Stack Introspection nach [ROS 00]

umbenannt bzw. ersetzt werden. Zudem können neue Java Features, wie z.B. das Reflection API [GRE 00], den Einsatz von Name Space Management unmöglich machen. Reflection und Klassen Libraries könnten jedoch neu designed werden, damit der Einsatz von Name Space Management möglich ist.

Beispielimplementierungen existieren als Zusatz für den Microsoft Explorer 3.0, wobei die JVM nicht modifiziert worden ist, jedoch Klassen in der Java Libraries geändert worden sind [BAF 00].

2.10 Fazit

Die Problematik der Realisation der Informationssicherheit gibt es schon seitdem unterschiedliche Benutzer Zugriff auf ein und dasselbe System haben müssen. Durch den Einsatz von verteilten Systemen und Anwendungen wurde die Problematik noch verschärft. Neue Konzepte, Techniken und Mechanismen wurden entwickelt, wobei der softwarebasierte Schutz immer mehr im Vordergrund stand.
Ein Hauptanteil der Entwicklung von Sicherheitskonzepten und -mechanismen wurde auf der Suche nach einem sicheren Betriebssystem geleistet. Durch die Möglichkeit der Vernetzung von Komponenten und deren Zusammenarbeit mussten neue Konzepte und Mechanismen zum Schutz der Informationssicherheit gefunden werden.
Betrachtet man traditionelle Sicherheitskonzepte bezüglich Agentensystemarchitekturen, so kann man sagen, dass sie weder in der Lage sind den Sicherheitskontext einer Anfrage geeignet einzufangen, noch geeignete, ausdrucksfähige Politiken, zur Bestimmung ob die Anfrage zulässig ist, besitzen.
Eine Agentensystemarchitektur stellt neue Anforderungen an die Schutzkonzepte, wobei auch hier eine anwendungsspezifische, feingranulare Zugriffskontrollpolitik benötigt wird [REV 00]. Durch die Mobilität der Agenten wird ein kompletter Schutz der Informationssicherheit nicht nur erheblich erschwert, sondern deren Verwaltung wird zunehmend komplexer. Ebenso gilt es die Informationen eines Agenten zu sichern.
Die Realisation des Informationsschutzes der Agentensysteme ist ebenso eine schwierige und komplizierte Aufgabe. Agenten müssen authentifiziert werden und einer geeigneten Zugriffskontrolle unterworfen werden. Es müssen spezifische Eigenschaften der Agenten beachtet werden, die sich im Laufe der Lebenszeit eines Agenten dynamisch ändern können.

Betrachtet man die vorhandenen Konzepte und Mechanismen, die z.B. die Programmiersprache Java bereitstellt, so sind auch hier enorme Fortschritte zu verzeichnen. In Java wird softwarebasierter Schutz durch die Verwendung von Ausnahmebehandlungen (Exceptions) realisiert.

Java realisiert unterschiedliche Schutzmechanismen, welche beim Laden des Programms zum Einsatz kommen (Übersetzer und Binder). Der Schutz vor direkten Speicherzugriffen erhöht die Sicherheit. Durch die Java 2 Security Architecture werden so grundlegende Sicherheitsanforderungen eines Sicherheitskonzepts erfüllt. Beispielsweise ist die Zugriffskontrollpolitik von ihrem Durchsetzungsmechanismus getrennt worden.

Dem Einsatz der Java 2 Security Architecture in unterschiedlichen Anwendungsgebieten sind jedoch auch Grenzen bezüglich der Sicherheit gesetzt. Dies ist hauptsächlich darauf zurückzuführen, dass die Sicherheitskonzepte größtenteils zur Lösung der durch die Applets verursachten Risiken konzipiert worden sind. Betrachtet man die Sicherheitsanforderungen von Agentensystemarchitekturen, sind grundlegende Sicherheitsanforderungen nicht erfüllbar.
Beispielsweise kann in Java zur selben Zeit nur eine systemglobale Sicherheitspolitik zum Einsatz kommen. Dabei steht vor allem der Schutz der Ausführungsumgebung im Vordergrund. Dynamische Änderungen der Politiken sind ebenso wenig vorgesehen, wie dynamische Änderungen von Zugriffsrechten aufgrund von sich dynamisch ändernden Eigenschaften der Subjekte. Anwendungsspezifische Sicherheitspolitiken die eine feingranulare Adressierung der Subjekte, Objekte und der Zugriffsrechte erfordern, können in der Standard Implementierung der Policy-Klasse nicht formuliert bzw. durchgesetzt werden. Desweiteren können Zugriffe auf Ressourcen nicht limitiert werden.

3 Anforderungsanalyse

In diesem Kapitel werden hierfür die speziellen Anforderungen der einzelnen Bestandteile eines Rechtekonzepts für MASA analysiert. Dazu wird zuerst ein einfaches Modell von MASA betrachtet. Anhand des Modells werden mögliche Zugriffe beschrieben. Die folgenden Abschnitte gehen dann auf Anforderungen der einzelnen Komponenten des Rechtekonzepts ein. Im Speziellen handelt es sich dabei um Anforderungen an die Zugriffskontrollpolitiken der Agenten und der Agentensysteme. Dann werden die Anforderungen an die Authentifikation der Subjekte und der Rechtevergabe behandelt. Schließlich werden noch Anforderungen der Rechtedurchsetzung betrachtet.

3.1 Modell der Agentensystemarchitektur MASA

Um die unterschiedlichen Kommunikationsvorgänge und Zugriffsmöglichkeiten darzustellen, soll hier ein abstraktes Modell einer Agentensystemarchitektur betrachtet werden.
Das Modell besteht lediglich aus vier Komponenten: dem Agenten, dem Agentensystem, dem Endsystem und einer Person. Der Agent ist ein mobiler Agent (MA), der sich von Agentensystem zu Agentensystem bewegen kann. Ein Agentensystem (AS) stellt eine Umgebung zur Ausführung des Agenten bereit. Ein Agent besitzt immer ein Home-Agentensystem. Das Home-Agentensystem ist dabei dasjenige Agentensystem, auf dem der Agent erstmals gestartet wurde. Das Agentensystem wird auf einem Endsystem (ES) zur Ausführung gebracht. Zugriffe auf Informationen des Endsystems sind lediglich über das Agentensystem möglich. Die Kommunikation der einzelnen Entitäten erfolgt über Kanäle.

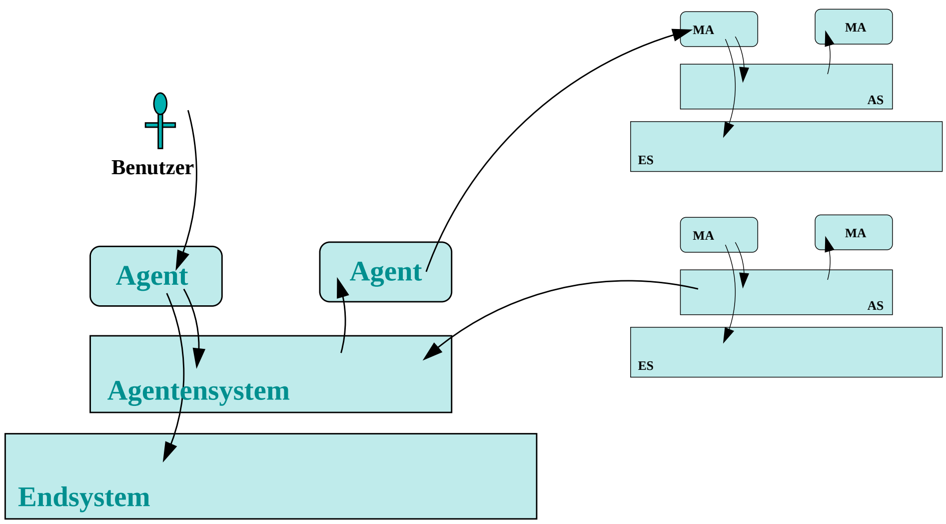


Abbildung 16: Einfaches Agentensystem Modell nach [ROE 99]

Das Endsystem, das Agentensystem und die Agenten stellen spezifische Schnittstellen bereit, die über wohldefinierte Kanäle angesprochen werden können. Die Kommunikationsmöglichkeiten werden in der Abb. 16 als Pfeile dargestellt.

Die in dem Modell vorkommenden Komponenten können anhand der in Kapitel 2.1 definierten Begriffe zugeordnet werden:

- Ein **Subjekt** kann ein Agent, ein Agentensystem oder eine Person sein. Ein Subjekt kann einer Gruppe, einer Organisation oder einer bestimmten Rolle angehören.

- Ein **Objekt** kann ein Agent, ein Agentensystem oder eine Ressource des Endsystems sein.

- Die **Aktion** beschreibt dabei die Art und Weise, wie ein Subjekt auf ein Objekt zugreift.

Betrachtet man die einzelnen Subjekte, so sind verschiedene Zugriffe möglich. Bei Zugriffen von einem Agent auf einen Agenten, von einem Agent auf ein Agentensystem und von einem Agentensystem auf ein Agentensystem kann unterschieden werden, ob es sich um einen lokalen oder einen entfernten Zugriff handelt. Ausschlaggebend ist, wo sich das Subjekt und das Objekt zum Zeitpunkt des Zugriffs befinden. Befinden sich das Subjekt und das Objekt auf demselben Agentensystem, so handelt es sich um einen lokalen Zugriff. Handelt es sich um einen entfernten Zugriff, so befinden sich das Subjekt und das Objekt nicht auf demselben Agentensystem.

Eine Person kann auf folgende Objekte zugreifen:
(1.1) Agenten-Objekt
(1.2) Agentensystem-Objekt
(1.3) Endsystem-Objekt

Ein Agent kann auf folgende Objekte zugreifen:
(2.1) Agenten-Objekt lokal
(2.2) Agenten-Objekt entfernt
(2.3) Agentensystem-Objekt lokal
(2.4) Endsystem-Objekt lokal

Ein Agentensystem kann auf folgende Objekte zugreifen:
(3.1) Agenten-Objekt lokal
(3.2) Agenten-Objekt entfernt
(3.3) Agentensystem-Objekt lokal
(3.4) Agentensystem-Objekt entfernt
(3.5) Endsystem-Objekt lokal

Betrachtet man die Objekte, auf die zugegriffen werden kann, so handelt es sich um Objekte von Agenten, Objekte von Agentensystemen und um Objekte des Endsystems.

Betrachtet man die Agenten, Agentensysteme und Endsysteme, so müssen alle drei in der Lage sein, Zugriffe auf ihre Objekte zu beschränken. Hierfür müssen sie jeweils die Möglichkeit haben, ausdrücken zu können, wer auf Objekte zugreifen darf und wer nicht. Somit sind Zugriffskontrollpolitiken der Endsysteme, der Agentensysteme und der Agenten erforderlich.
Dabei ist darauf zu achten, dass die Durchsetzung der Zugriffskontrollpolitiken nicht die Informationssicherheit der einzelnen Objekte gefährdet.

Betrachtet man mobile Agenten im Speziellen, so ist es erforderlich, dass ein Agent jeweils an die lokale Umgebung gebunden, autorisiert und einer Zugriffskontrolle unterzogen werden muss. Dabei ist zu regeln, wer für Zugriffe, die dieser Agent durchführt, verantwortlich ist.

Soll einem Subjekt ein Zugriff erlaubt werden oder explizit verboten werden, so müssen zur Durchsetzung dieser Forderung diese Subjekte authentifiziert werden können.

3.2 Authentifikation

Um zu überprüfen, ob es sich bei einem Zugriff auf ein Objekt um einen autorisierten Zugriff handelt, ist es notwendig, das zugreifende Subjekt identifizieren zu können. Ist das Subjekt identifiziert, muss die Identität des Subjekts noch verifiziert werden. Erst durch eine erfolgreiche Verifikation ist ein Subjekt authentifiziert.

Die in MASA vorkommenden Subjekte (Personen, Agenten, Agentensysteme) müssen demnach eindeutig authentifiziert werden können. Die Identifikation der Subjekte muss nach wohldefinierten Eigenschaften der Subjekte erfolgen. Die Eigenschaften sind dabei so zu wählen, dass sie über Systemgrenzen hinaus ein Subjekt eindeutig identifizieren.

Da ein Agentensystem in einer verteilten Umgebung eingesetzt wird, wird ein geeignetes Konzept benötigt, welches die Authentifikation in einer verteilten Umgebung unterstützt.

Anhand der authentifizierten Identität können die Zugriffsrechte ermittelt werden, welche dem Subjekt durch eine Zugriffskontrollpolitik zugeteilt worden sind.

3.3 Zugriffsrechte

Ein Zugriffsrecht berechtigt einem Subjekt den Zugriff auf Informationen. Dabei adressiert ein Zugriffsrecht die Aktion bzw. die Aktion und das Objekt. Zugriffsrechte sollten die Aktionen möglichst feingranular beschreiben können, damit die Vergabe der Zugriffsrechte anwendungsspezifisch erfolgen kann.

Hier sind spezifische Zugriffsrechte den universellen Zugriffsrechten vorzuziehen (vgl. Def. in Kapitel 2.1). Durch die Verwendung von spezifischen Zugriffsrechten wird die Einschränkung der Zugriffe auf einzelne Informationen eines Objekts ermöglicht. Ein spezifisches Zugriffsrecht steht eng in Verbindung mit der Funktionalität des Objekts.

Die Zugriffsrechte sollten desweiteren eindeutig und daher aussagekräftig benannt sein, damit bei der Vergabe der Zugriffsrechte Fehler aufgrund falscher Interpretation ausgeschlossen sind. Durch die Namenvergabe muss eindeutig die Berechtigung erkennbar sein, die das Zugriffsrecht beinhaltet.

3.4 Rechtevergabe (Autorisation)

Bei der Vergabe der Zugriffsrechte müssen alle relevanten Subjekte und Objekte erfasst werden können. Dabei sollten unter anderem die Prinzipien Economy of Mechanism, Psychological Acceptability, Fail-Safe Defaults, Resource Limits und das Least-Privileg Prinzip aus Kapitel 2.3 beachtet und umgesetzt werden.
Autorisationsvorgänge müssen benutzerfreundlich, übersichtlich und möglichst einfach und intuitiv erfolgen können. Dabei ist die Eindeutigkeit und Widerspruchsfreiheit der Rechtevergabe sicherzustellen. Subjekte sollten nur Zugriffsrechte zugeteilt bekommen, die sie unbedingt für die Erledigung der Aufgabe benötigen (Least-Privilege Prinzip). Dazu gehört auch die Limitierung der Systemressourcen.

Greifen räumlich verteilte Subjekte auf räumlich verteilte Objekte zu, was in MASA der Fall ist, wird eine Rechtevergabe benötigt, die die große Anzahl der Subjekte und Objekte ebenso handhaben kann, wie auch die dynamische Erzeugung und Terminierung der Subjekte. Da Zugriffsrechte unter Umständen wieder zurückgenommen werden müssen, sollte das Rechtekonzept auch die Rechterücknahme berücksichtigen.

Durch das Vorhandensein unterschiedlicher Administrationsdomänen sind Zugriffskontrollpolitiken unterschiedlicher Interessengemeinschaften vorhanden. Sie machen ein Zusammenwirken der Zugriffskontrollpolitiken notwendig, welche die systematische Integration unterschiedlicher Zugriffskontrollpolitiken unterstützen muss.
Es müssen Zugriffskontrollpolitiken unterschiedlicher Administrationsbereiche durchgesetzt werden, dazu gehören auch die benutzerspezifischen Zugriffskontrollpolitiken der einzelnen Agenten.

Werden innerhalb der verschiedenen Zugriffskontrollpolitiken unterschiedliche Zugriffsrechte auf z.T. ein und demselben Objekt vergeben, müssen bei der Durchsetzung der beteiligten Zugriffskontrollpolitiken die Autonomie-Anforderungen der unterschiedlich beteiligten Objekte erfüllt werden. Setzt ein Agentensystem beispielsweise zusätzlich zu seiner Zugriffskontrollpolitik die Zugriffskontrollpolitik eines Agenten durch, sollte durch die Zugriffskontrollpolitik des Agenten die Zugriffskontrollpolitik des Agentensystems nicht verletzt werden. D.h. Zugriffe, die nach der Zugriffskontrollpolitik des Agentensystems erlaubt sind, müssen weiterhin erlaubt sein. Die Autonomie des Agentensystems muss also gewährt bleiben. Zusätzlich darf die Informationssicherheit nicht verletzt werden, d.h. Zugriffe die explizit durch die Zugriffskontrollpolitik des Agentensystems verboten sind, müssen nach der Komposition mit der Agenten-Zugriffskontrollpolitik immer noch verboten sein.

Um eine anwendungsspezifische Vergabe von Zugriffsrechten zu ermöglichen, müssen Objekte beliebig feingranular adressiert werden können. Die Granularität der adressierbaren Objekte ist dabei so zu wählen, dass die Vergabe von Berechtigungen nach dem Least-Privilege Prinzip (vgl Kapitel 2.3) ermöglicht wird. Um auch benutzerspezifische Zugriffsrechte vergeben zu können, ist ebenso eine feingranulare Adressierung der Subjekte notwendig.

Einfache Zugriffsbeschränkungen, d.h. Zugriffsbeschränkungen gebunden an ein Subjekt, reichen für eine objektspezifische Vergabe von Zugriffsrechten z.T. nicht aus. Es müssen auch komplexe Zugriffsbeschränkungen, das sind Zugriffsrechte geknüpft an Bedingungen, ermöglicht werden (vgl. [ECK 98]). So muss innerhalb von MASA ermöglicht werden, dass Zugriffsrechte an einen Agenten in Abhängigkeit seiner Vergangenheit (History) vergeben werden können. Dabei müssen Bedingungen ausgedrückt werden können, die z.B. die von dem Agenten zuvor besuchten Agentensysteme berücksichtigen.

Damit Agenten ihre Aufgabe an andere Agenten verteilen und diese Agenten die Aufgabe auch erledigen können, ist es notwendig, dass Agenten ihre Zugriffsrechte an weitere Agenten delegieren können. Die Weitergabe der Zugriffsrechte ist dabei zu kontrollieren, um die Informationssicherheit nicht zu gefährden.

3.4.1 Betrachtung der Agentenhistory

Die History einer Agenteninstanz spielt für die Sicherheit eine enorme Rolle. Daher sollten in einem Rechtekonzept ebenfalls die Vergabe von Zugriffsrechten von den zuvor besuchten Agentensystemen abhängig gemacht werden können. Da eine Agenteninstanz einem Agentensystem in jeder Hinsicht ausgeliefert ist, können so sicherheitskritische Operationen nur Agenteninstanzen erlaubt werden, welche lediglich vertrauenswürdige Agentensysteme besucht haben. Ein Agentensystem, dem nicht vertraut wird, kann so keinen Schaden durch eine böswillige Modifikation einer Agenteninstanz anrichten.

3.4.2 Rechtedelegation

Mobile Agenten handeln autonom und wandern von Agentensystem zu Agentensystem. Will ein Agent, um die Aufgabenstellung effizienter zu lösen, einen anderen Agenten beauftragen, eine bestimmte Tätigkeit auszuführen, so ist dies in vielen Fällen nur möglich, wenn der beauftragte Agent dieselben Zugriffsrechte besitzt. Agenten müssen daher in der Lage sein, ihre Rechte weitergeben zu können. Durch die Möglichkeit, Zugriffsrechte zu delegieren, ist eine autonome Aufteilung der Aufgaben möglich, wodurch eine höhere Flexibilität erreicht wird.

Die Delegation der Zugriffsrechte darf jedoch nicht die Informationssicherheit gefährden.
Der gekennzeichnete Agent der Abb. 17 besitzt zuerst als einziger das Zugriffsrecht, um auf das Objekt zuzugreifen. Nach der Anforderung der Rechtedelegation sollte er jedoch in der Lage sein, das Zugriffsrecht auch an einen weiteren Agenten delegieren zu können. Dabei kann sich der zweite Agent auf dem gleichen Agentensystem, aber auch auf einem entfernten Agentensystem, befinden.

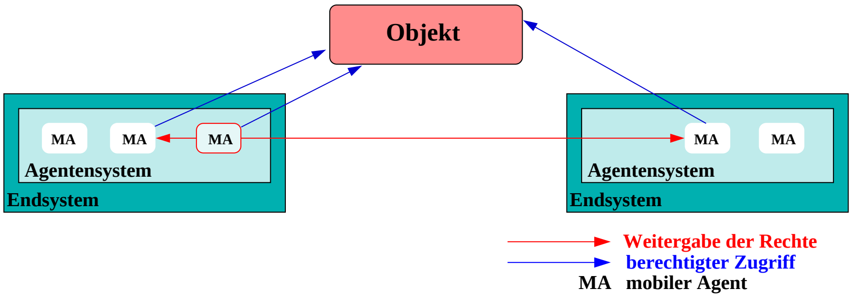


Abbildung 17: Problematiken der Rechtedelegation

Betrachtet man dieses Beispiel, so müssen folgende Anforderungen bezüglich der Rechtedelegation erfüllt werden, um die Informationssicherheit nicht zu gefährden:

1. Es muss geregelt werden, an welche Subjekte das Zugriffsrecht delegiert werden darf. Die Regeln müssen dabei von dem Eigentümer des Objekts aufgestellt werden können.

2. Die Rechtedelegation über Systemgrenzen hinaus muss geregelt werden.

3. Das Zugriffsrecht darf nicht erschlichen werden können. Ist ein Agent nicht autorisiert und wird ihm der Zugriff verweigert, so darf es keine Möglichkeit geben, die Zugriffskontrolle zu umgehen. Es muss verhindert werden, dass der Agent das notwendige Zugriffsrecht erschleicht bzw. über ein anderes Subjekt Zugriff auf dieses Objekt erlangt.

4. Es muss bei delegierbaren Zugriffsrechten darauf geachtet werden, dass die Möglichkeit besteht, dass ein Agent ein unsicheres Agentensystem besucht. Durch ein geeignetes Konzept ist sicherzustellen, dass die Delegation der Zugriffsrechte auch hier nicht die Informationssicherheit gefährdet.

Jeglicher Missbrauch der zur Rechtedelegation eingesetzten Mechanismen und Techniken muss ausgeschlossen werden. Dabei muss sichergestellt sein, dass kein Agent unberechtigterweise an ein Zugriffsrecht gelangen kann. So ist auch ein Konzept gefragt, welches ausschließt, dass ein bösartiges Agentensystem und ein bösartiger Agent durch Kooperation erreichen, dass der Agent unautorisierten Zugriff auf ein Objekt erlangt.
Es ist notwendig, die Delegation der Zugriffsrechte durch einen geeigneten Mechanismus kontrollieren zu können. Beispielsweise sollten Zugriffsrechte nur dann weitergegeben werden können, wenn ein dafür notwendiges Zugriffsrecht vorhanden ist.

Auch hier ist darauf zu achten, dass die Zugriffskontrollpolitik des Eigentümers nicht verletzt wird. Zugriffe, die nach der Zugriffskontrollpolitik des Eigentümers erlaubt sind, müssen weiterhin erlaubt bleiben. Die Informationssicherheit sollte ebenso nicht verletzt werden, d.h. Zugriffe, die explizit durch die Zugriffskontrollpolitik des Eigentümers verboten sind, dürfen nicht gewährt werden.

Dient die Rechtevergabe als Beschreibungsmittel für zulässige und unzulässige Zugriffe, ist die vollständige Durchsetzung der Rechtevergabe letztendlich für die Informationssicherheit verantwortlich.

3.5 Rechtedurchsetzung

Die Rechtedurchsetzung ist verantwortlich für die Durchsetzung der einzelnen Zugriffskontrollpolitiken. Da die Informationssicherheit von der Rechtedurchsetzung abhängt, sind Anforderungen bzgl. der Rechtedurchsetzung besonders kritisch.

Folgende Anforderungen müssen beachtet und realisiert werden:
Es darf keine Möglichkeit geben, die Rechtedurchsetzung zu umgehen. Es ist erforderlich, dass alle Zugriffe der Subjekte auf Objekte überprüft werden (Complete-Mediation Prinzip). Die Informationen, welche zur Entscheidungsfindung herangezogen werden, müssen ebenfalls vor unautorisierten Zugriffen geschützt werden. Manipulationen dieser Informationen sind auszuschließen.

Bei einem Zugriff auf ein Objekt eines Agentensystems muss die Zugriffskontrollpolitik des Agentensystems durchgesetzt werden. Nur so kann die Informationssicherheit des Agentensystems sichergestellt werden. Bei einem Zugriff auf ein Objekt eines Agenten muss die Zugriffskontrollpolitik des Agenten durchgesetzt werden. Dabei ist die Zusammenarbeit mit der Zugriffskontrolle des Agentensystems notwendig. Die Zugriffskontrollpolitik des Endsystems ist bei Zugriffen auf die Informationen des Endsystems durchzusetzen.

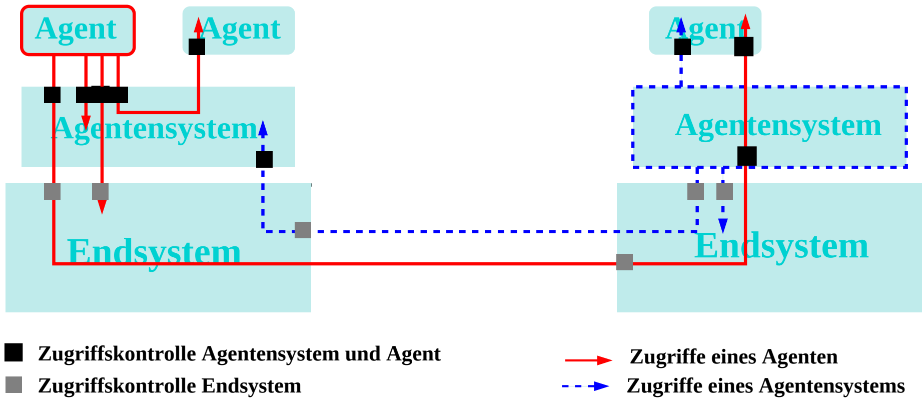


Abbildung 18: Zugriffskontrollen

Die Abb. 18 soll verdeutlichen, welche Entität in MASA bei einem Zugriff jeweils eine Zugriffskontrolle durchführen muss. Ein Pfeil bedeutet hier jeweils einen Zugriff auf ein Objekt. Das Subjekt ist aus dem Pfeilursprung ersichtlich und das Ende des Pfeils bezeichnet das Subjekt, auf welchem sich das Objekt, auf das zugegriffen wird, befindet.

Die Zugriffskontrolle muss kontrollieren, ob das Subjekt das notwendige Zugriffsrecht besitzt, um auf das jeweilige Objekt zuzugreifen. Dabei sollte jeder nicht explizit autorisierte Zugriff verboten werden (Fail-Safe Defaults Prinzip).
Bei der Wahl des Konzepts und dessen Mechanismen zur Rechtedurchsetzung spielt die Funktionalität und Robustheit ebenso eine Rolle, wie die Effizienz (Performance), wobei auf Kompatibilitätseigenschaften (Compatibility) geachtet werden sollte. Die Sicherheit der eingesetzten Mechanismen und Techniken sollte dabei nicht von der Geheimhaltung der Verfahren abhängig sein (Open-Design Prinzip).

3.6 Programmiersprachen Konzepte

Bei der Verwendung von Sicherheitsmechanismen die auf Programmiersprachen basieren, müssen die Spezifikationen der Sicherheitsmechanismen eingehalten bzw. umgesetzt werden. Da es sich um softwarebasierte Mechanismen handelt, sind Fehler bzgl. der Programmierung auszuschließen. Die Kontrolle des Programms sollte daher effizient ermöglicht werden bzw. überprüfbar sein (Code Auditability). Die fehlerfreie Funktion der Mechanismen und Techniken ist eine Voraussetzung, um die Rechtssicherheit der Informationen garantieren zu können.

3.7 Zusammenfassung

Bei einem Agentensystem muss bei jeglichen Zugriffen auf Objekte überprüft werden, ob dieser Zugriff zuvor autorisiert wurde. Dabei müssen die Subjekte in der verteilten Umgebung authentifizierbar sein.
Jedes Objekt, auf das zugegriffen wird, muss in der Lage sein, seine Sicherheitspolitik durchzusetzen. Nur so kann die Informationssicherheit garantiert werden.

Mobile Agenten müssen an die lokale Umgebung gebunden, autorisiert und einer Zugriffskontrolle unterzogen werden. Wie bei jedem Zugriff muss auch hier der Zugriff einer bestimmten Person zugeordnet werden. Ein mobiler Agent handelt dabei als ein Abgesandter des Eigentümers des Agenten und besitzt die Ausführungsrechte des Anwenders des Agenten.

Je nach Anwendungsgebiet von MASA ist eine Klassifizierung der Anforderungen unterschiedlich. Dabei sollte MASA in der Lage sein, auch Anforderungen einer hohen Integritätsstufe und einer hohen Vertraulichkeitsstufe zu erfüllen.

Die einzelnen Sicherheitspolitiken der Endsysteme, Agentensysteme und Agenten müssen durchgesetzt werden. Dabei darf die Informationssicherheit der einzelnen Entitäten nicht gefährdet werden. Die Sicherheitspolitik muss weiter durch eine Kontrollinstanz verwaltet werden (Administrator), welche die Rechtevergabe kontrolliert.

Betrachtet man die Beschränkung der Zugriffe näher, so sind auch Zugriffe auf Systemressourcen zu limitieren. Beispielsweise sollte der CPU-Verbrauch kontrolliert werden und nicht unbeschränkt genutzt werden können.

Die folgende Zusammenfassung gibt nochmals einen Überblick über die Sicherheitsanforderungen bezüglich des Rechtekonzepts:

Anforderungen der **Authentifikation**:

- Subjekte müssen authentifizierbar sein
- Die Authentifikation muss die verteilte Umgebung unterstützen

Bezüglich der **Zugriffsrechte** sind folgende Anforderungen zu erfüllen:

- Zugriffsrechte sollten die Aktionen feingranular adressieren können
- Zugriffsrechte sollten für den Benutzer eindeutige, verwechslungsfreie Berechtigungen darstellen
- Zugriffsrechte sollten ebenfalls eine verteilte Umgebung unterstützen

Die **Rechtevergabe** sollte unter anderem die Prinzipien aus Kapitel 2.3 berücksichtigen. Die folgenden Prinzipien müssen ebenfalls erfüllt werden:

- Alle relevanten Subjekte und Objekte müssen erfasst werden können

- Die Vergabe von Zugriffsrechten muss anwendungsspezifisch erfolgen können (feingranulare Adressierung der Subjekte und Objekte innerhalb der Zugriffskontrollpolitik)
- Zugriffskontrollpolitiken von Agenten und Agentensystemen müssen durchgesetzt werden
  - Zusammenwirken unterschiedlicher Zugriffskontrollpolitiken und deren Durchsetzung
  - Die Autonomie des Agentensystems und des Endsystems muss gewahrt bleiben
  - Die Informationssicherheit nach einer Komposition von Zugriffskontrollpolitiken muss gewahrt bleiben
- Die Unterstützung dynamischer, modifizierbarer Politiken ist erforderlich
- Die Rechtevergabe muss über Systemgrenzen hinaus mit einer großen Anzahl sich dynamisch ändernden Subjekten und Objekten geregelt sein
- Autorisationen müssen gegebenenfalls an Bedingungen geknüpft werden können (z.B. der Vorgeschichte eines Agenten (History))
- Die Widerspruchsfreiheit und Eindeutigkeit der Autorisation muss sichergestellt sein
- Zugriffsrechte müssen gegebenfalls zurückgenommen werden können
- Eigentümer von Objekten müssen Zugriffsrechte auf ihre Objekte bestimmen und einschränken können
- Die Delegation der Zugriffsrechte muss ermöglicht werden
  - Zugriffsrechte der Agenten sollten delegierbar sein
  - Die Delegation muss kontrolliert und eingeschränkt werden können
  - Der Eigentümer muss bestimmen, wer Zugriffe auf seine Objekte delegieren darf

Die Anforderungen der **Rechtedurchsetzung** in MASA sind:

- Es darf keine Möglichkeit geben, die Rechtedurchsetzung zu umgehen (Complete-Mediation Prinzip)
- Schutz der für die Rechtedurchsetzung relevanten Information
- Durchsetzung der beteiligten Zugriffskontrollpolitiken
- Jeder Zugriff, der nicht explizit autorisiert worden ist, muss verboten werden (Fail-Safe Default Prinzip)

In dieser Arbeit gilt es nun ein für MASA passendes Rechtekonzept zu entwickeln, dass diese Anforderungen erfüllt. Um herauszufinden, inwieweit die oben genannten Anforderungen schon erfüllt bzw. beachtet werden, wird im nächsten Kapitel MASA bezüglich dieser Anforderungen genauer untersucht und analysiert.

4 Risikoanalyse von MASA

In diesem Kapitel wird die momentane Situation in MASA, in Bezug auf die Gewährung der Informationssicherheit in MASA, betrachtet. Es werden Stärken und Schwächen der momentanen Implementierung aufgezeigt. Die Risikoanalyse stützt sich dabei auf die in der Anforderungsanalyse spezifizierten Anforderungen.

Analysiert werden die Authentifikation, die Rechtevergabe, die eingesetzten Zugriffsrechte und die eingesetzten Zugriffskontrollmechanismen.

4.1 Erweitertes Modell von MASA

Um das Rechtekonzept von MASA analysieren zu können, müssen die spezifischen Komponenten von MASA betrachtet werden. Hierfür wird das einfache Agentensystem Modell aus Kapitel 3.1 um die spezifischen Eigenschaften von MASA erweitert.

Agentengattung - Agenteninstanz
Betrachtet man Agenten in MASA, so zerfällt der Begriff Agent einerseits in die Agentengattung und andererseits in die Agenteninstanz (vgl. [ROE 99] Kapitel 3.1).

Die Agentengattung entspricht dem statischen Teil eines Agenten. Der statische Teil beinhaltet dabei alle Teile, welche alle Agenten derselben Gattung besitzen, wie z.B. der Programmcode der Agentengattung.

Die Agenteninstanz stellt den dynamisch Teil eines Agenten dar. Um einen Agenten auf einem Agentensystem auszuführen, wird der Programmcode (Agentengattung) instanziiert, d.h. auf dem Agentensystem zur Ausführung gebracht. Sobald der Programmcode auf dem Agentensystem ausgeführt wird, spricht man von einer Agenteninstanz. Die Agenteninstanz besitzt daher, im Gegensatz zur Agentengattung, dynamische Teile, die sich von Agent zu Agent derselben Gattung unterscheiden können. Beispiele dynamischer Teile sind die aktuelle Belegung der Attribute.

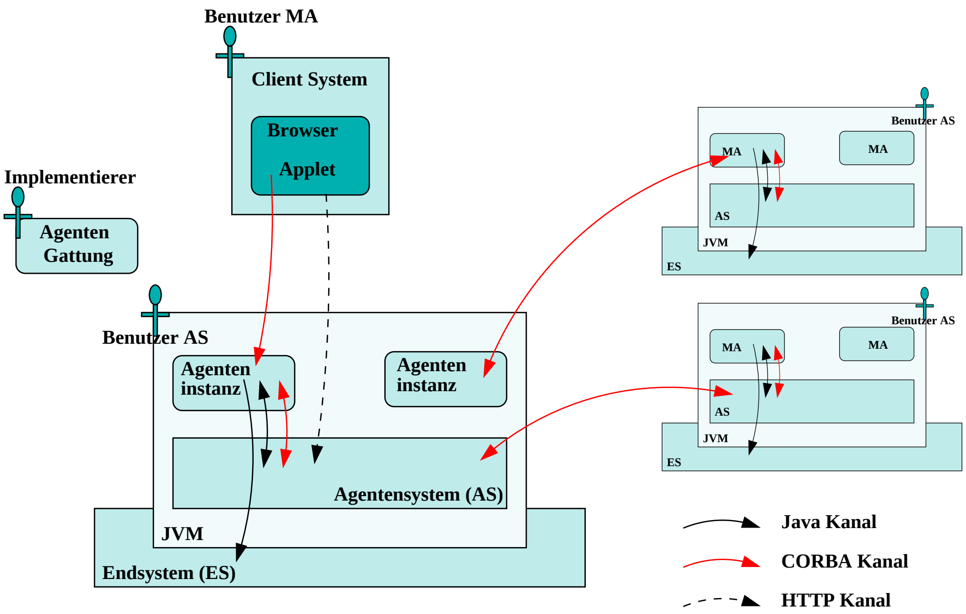


Abbildung 19: Agentenmodell MASA in Anlehnung an [ROE 99])

Die Abb. 19 zeigt folgende Entitäten: Agentengattungen, Applets, Agenteninstanzen, Agentensysteme, Endsysteme, Client-Systeme mit einem Browser, Benutzer von Agentensystemen, Benutzer von Agenteninstanzen und Implementierer von Agentengattungen. Eine von einem Benutzer implementierte Agentengattung kann instanziiert werden. Bei einer instanziierten Agentengattung handelt es sich dann um eine Agenteninstanz. Die einzelnen Entitäten kommunizieren über Java-, CORBA- und HTTP-Kanäle (vgl. [ROE 99] Kapitel 3.1).

Da die Agentengattung u.a. den Programmcode des Agenten darstellt, kann ihr der Implementierer zugeordnet werden. Der Implementierer ist die Person, die die Agentengattung implementiert hat. Eine Agenteninstanz kann einerseits dem Eigentümer, die Entität, die die Agenteninstanz gestartet hat und andererseits dem Anwender, die Person, die den Agenten verwendet, zugeordnet werden.

Ein Agentensystem hat ebenso einen Eigentümer. Das ist die Person, die das Agentensystem gestartet hat. Entitäten, die das Agentensystem benutzen, sind die Anwender des Agentensystems.

Hier wird jeweils die Funktionalität eines Benutzers betrachtet. In MASA existieren demnach die unten genannten Benutzer, deren Oberklasse folgend als Person bezeichnet wird (vgl. [ROE 99] Kapitel 5.1.1).

- Anwender eines Agentensystems
- Authority (Eigentümer) eines Agentensystems
- Anwender eines mobilen Agenten
- Authority (Eigentümer) eines mobilen Agenten

Betrachtet man die Aufgaben der Personen, so ist der Eigentümer einer Agenteninstanz nur für die jeweilige Agenteninstanz verantwortlich.
Der Eigentümer eines Agentensystems ist verantwortlich für das Agentensystem und übernimmt zusätzlich noch administrative Aufgaben. Die administrativen Aufgaben können sich dabei auch auf die Agenteninstanzen beziehen, die auf dem Agentensystem ausgeführt werden bzw. zur Ausführung gebracht werden sollen.

Eine weitere Unterscheidung, die man vornehmen kann, ist die Unterscheidung, ob eine Entität ein Subjekt oder ein Objekt ist (vgl. Kapitel 2). In MASA können Applets, Agenteninstanzen, Agentensysteme, Client-Systeme und Benutzer Subjekte darstellen, die auf Objekte zugreifen. Objekte in MASA können Agentengattungen, Agenteninstanzen, Agentensysteme und Endsysteme sein. Auf diese Objekte kann zugegriffen werden. Agentensysteme und Agenteninstanzen können demnach die Stellung eines Subjekts und die Stellung eines Objekts einnehmen. Die Subjekte können jeweils Aktionen an den von den Objekten bereitgestellten Schnittstellen ausführen.

Ein Benutzer kann beispielsweise durch ein Applet, welches auf seinem Browser zur Ausführung gebracht wird, auf eine Agenteninstanz zugreifen und Aktionen ausführen (vgl. Abb. 19). Die durch den Benutzer getätigten Aktionswünsche werden von den Applets über CORBA-Kanäle an die Agenteninstanzen weitergeleitet. Die Agenteninstanzen kommunizieren über einen Java-Kanal und einen CORBA-Kanal mit dem Agentensystem, auf dem sie ausgeführt werden. Agenteninstanzen können durch Java-API Aufrufe auch Aktionen auf dem Endsystem ausführen. Agenteninstanzen sind auch in der Lage über einen CORBA-Kanal andere Agenteninstanzen zur Ausführung bestimmter Aktionen zu veranlassen. Die Agentensysteme untereinander verständigen sich über CORBA-Kanäle (vgl. [ROE 99] Kapitel 3.1).

4.2 Basis des Sicherheitmodells von MASA

Folgend soll das von [ROE 99] entwickelte Sicherheitsmodell von MASA betrachtet werden.

Ausgangsbasis des Sicherheitmodells von MASA ist die Annahme, dass für alle Entitäten eine getrennte Ausführungsumgebung bereitgestellt wird. Dies gilt für die Trennung zwischen Agentensystemen, zwischen Agentensystem und Agenteninstanzen und zwischen Agenteninstanzen (vgl. [ROE 99] Kapitel 5.3).

Realisiert wird die Trennung durch die Verwendung von Thread-Groups für Agenteninstanzen und durch die Trennung der Namensräume der Entitäten durch den Java ClassLoader (vgl. [ROE 99] Kapitel 6.3.2-3). So bekommt jede Agenteninstanz, bevor die Hauptklasse der Agenteninstanz geladen wird, eine eigene ClassLoader-Instanz, die dann den Programmcode der Hauptklasse aus dem Code Repository lädt. Die Trennung der Namensräume bleibt auch dann bestehen, wenn eine Agenteninstanz weitere Klassen nutzt, da hier derselbe ClassLoader verwendet wird.

Architektonisch bedingt gilt aufgrund der Einbettungsbeziehung, dass ein Agentensystem seinem Endsystem und ein Agent seinem Home-Agentensystem vertrauen muss (vgl. [ROE 99] Kapitel 5.2).

Sicherungsmaßnahmen werden durch das Agentensystem überwacht und durchgesetzt. Dabei wird eine negativistische Grundannahme durchgesetzt, d.h. dass alles verboten wird, was nicht explizit erlaubt ist. Da das Agentensystem die Sicherungsmaßnahmen überwacht und durchsetzt, kann es auch die Sicherung zwischen den Agenteninstanzen übernehmen, wodurch die Agenteninstanzen u.a. voreinander geschützt werden können. Die Endsysteme und die Agenteninstanzen müssen sich daher aber auch auf das Agentensystem verlassen können (vgl. [ROE 99] Kapitel 3.7). Durch die Verwendung des plattformunabhängigen Agentensystems als Kontrollinstanz, steht eine vereinheitlichte Schnittstelle für das Management der Sicherheit zur Verfügung.

Der Einsatz des Code Repositories in MASA trägt zum Schutz der Informationssicherheit bei. Bei einem Zugriff einer Agenteninstanz können zwei Vertrauensbeziehungen unterschieden werden:

1. Dem Implementierer der Agentengattung wird getraut
2. Den Daten der Agenteninstanz wird getraut

Durch die Bereitstellung des Code Repositories kann das Vertrauen zu Agentengattungen abhängig vom Speicherort der Gattungsdaten getroffen werden.

Auch hier trifft das Agentensystem die Entscheidung bezüglich des zu verwendendem Code Repository, von dem die Agentengattung geladen werden soll. Zusätzlich entscheidet das Agentensystem noch, ob es sich bei einem Zertifikat eines Subjekts um ein vertrauenswürdiges Zertifikat handelt. (Die Entscheidung, ob ein Zertifikat vertrauenswürdig ist, muss ebenso eine Agenteninstanz treffen können.)

Die Kommunikationskanäle werden durch geeignete Mechanismen geschützt. Dabei wird das Secure Socket Layer (SSL) Protokoll V3.0 für die in MASA eingesetzten CORBA/IIOP- und HTTP-Kanäle verwendet. Das SSL-Protokoll unterstützt dabei zusätzlich die Authentisierung durch den sicheren Austausch von Zertifikaten (vgl. [ROE 99] Kapitel 6.1.2). Der HTTP-Kanal verwendet das unter dem Namen HTTPS bekannte SSL-Verfahren. Der CORBA-Kanal wird ebenfalls durch das SSL-Protokoll geschützt. In MASA kommt hierfür ein SSL-fähiger ORB zum Einsatz.

Eine Agenteninstanz kann nur über den CORBA-Kanal mit einer weiteren Agenteninstanz oder dem Agentensystem kommunizieren. Dadurch kann in diesem Fall die Sicherung des Java-Kanals entfallen.

Betrachtet man das Sicherheitsmodell, so sind alle handelnden Entitäten (Personen, Agenteninstanzen, Agentensysteme) in der Lage, über gesicherte Kanäle zu kommunizieren, womit eine Basis eines sicheren Agentensystems gegeben ist. Bezüglich der Informationssicherheit muss desweiteren die Authentifikation der Entitäten möglich sein.

4.3 Identifikation und Authentifikation

Die Authentifikation der einzelnen Entitäten in MASA basiert auf den in [ROE 99] eingeführten Zertifikaten und Zertifikathierarchien. Entitäten entsprechen innerhalb der Implementierung den Principals. Betrachtet man die Anforderungen aus Kapitel 3, so müssen alle Entitäten eindeutig identifizierbar sein.

Die eindeutige Identifizierung der Entitäten ist in MASA sichergestellt. Jede Entität (Principal) ist mit einem eindeutigen Identifikator versehen, der eine verteilte Umgebung unterstützt (vgl. [ROE 99] Kapitel 5.5).

Die Agentengattung wird beispielsweise über eine Kombination des Namens und eine organisatorische Klassifizierung identifiziert. Der eindeutige Identifikator einer Agentengattung `FOO`-Agent wird durch den Namen der Hauptklasse erzeugt:

[`de.unimuenchen.informatik.mnm.FOO.FOOMobileAgent`]

Der Name wird aus dem DNS-Namen *informatik.uni-muenchen.de*, der Organisation *MNM*, der Gattung *FOO* und dem Typen *mobiler Agent* gebildet. Ausgehend davon, dass jede Organisation, die einen Agenten implementiert, im DNS-Namensraum vertreten ist, ist die Bildung des Gattungsnamen ausreichend für die eindeutige Identifizierung. Zudem wird jedem Agentensystem über den NamingWebServer beim Start eine in der Region eindeutige ID-Nummer zugewiesen.

Beim Erzeugen einer Agenteninstanz auf einem Agentensystem bekommt die Agenteninstanz eine vom Agentensystem erzeugte, fortlaufende Nummer zugewiesen. Dadurch können Agenteninstanzen derselben Gattung auf einem Agentensystem unterschieden werden. Agenteninstanzen und Agentensysteme können eindeutig über die Datenstrukturen, die in MASA eingesetzt werden, identifiziert werden. Die Identifizierung der Endsysteme können über den Identifikator des Agentensystems eindeutig identifiziert werden (auch hier wird für die Identifizierung der DNS-Namensraum herangezogen). Die Client Systeme werden im Namensraum der IP-Adressen identifiziert. Personen können über ihre Zertifikate identifiziert werden. Die Eindeutigkeit wird hier über die Kombination von Namen und einer organisatorischen Klassifizierung erreicht.

Betrachtet man die Entitäten als Klassen, existiert die Oberklasse Principal und eine Unterklasse für die spezifischen Entitäten. Neben einem gemeinsamen Attribut (Identifikator) aller Entitäten (Principals) werden die Subtypen Agentensystem und Agenteninstanz mit weiteren Attributen versehen, damit eine Beziehung zu den Personen des Modells hergestellt werden kann. Die Abb. 20 zeigt das Klassendiagramm.
Für Agenteninstanzen und Agentensysteme werden jeweils die Authority (Eigentümer) und der Implementierer betrachtet. Unterscheidungen bezüglich des Implementierers und des Eigentümers sind damit möglich.

Die Authority, also der Eigentümer einer Agenteninstanz, kann eine Agenteninstanz, ein Agentensystem oder eine weitere Entität (Principal) sein. Der Eigentümer (Authority) eines Agentensystems kann nur eine Person sein. Ein Implementierer einer Agentengattung oder eines Agentensystems kann ebenfalls nur eine Person sein.

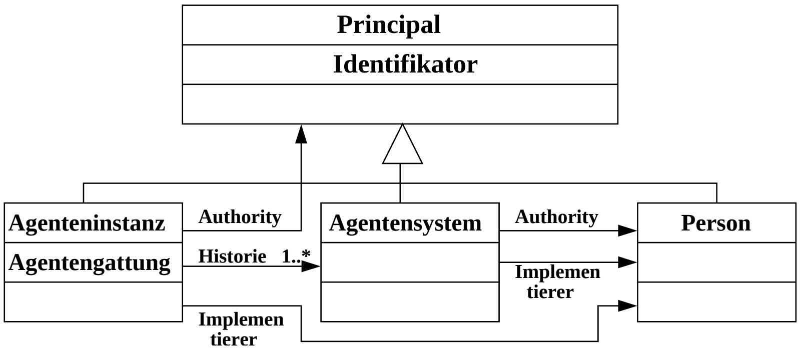


Abbildung 20: Klassendiagramm Principals (aus [ROE 99])

Entitäten können somit anhand von wohldefinierten Eigenschaften identifiziert werden (vgl. [ROE 99] Tabelle 5.2):

- **Agentengattungen** werden durch das Java-Package und den Namen der Hauptklasse identifiziert
- **Agenteninstanzen** werden durch die Datenstruktur CfMAF.Name identifiziert
- **Agentensysteme** werden ebenfalls durch die Datenstruktur CfMAF.Name identifiziert
- **Endsysteme** können nur indirekt über Agentensysteme identifiziert werden
- **Client-Systeme** können anhand ihrer IP-Adresse identifiziert werden
- **Der Benutzer des Agentensystems** kann über sein Zertifikat identifiziert werden
- **Der Benutzer von Agenteninstanzen** kann ebenfalls über sein Zertifikat identifiziert werden
- **Der Implementierer einer Agentengattung** kann anhand der Signatur und seinem Zertifikat identifiziert werden

Werden die Identitäten der Entitäten verifiziert, sind sie gegenüber des Agentensystems authentifiziert.

Die Verifikation der Identität eines Principals erfolgt in MASA durch ein Public-Key-Verfahren. Ein Principal ist mit einem Schlüsselpaar ausgestattet. Jedes Principal kann eindeutig identifiziert werden und ist Eigentümer eines Schlüsselpaars, wodurch die Erstellung eines Zertifikats ermöglicht wird. Hierfür unterzeichnet die Entität ihren Identifikator mit dem geheimen Schlüssel, woraus eine Signatur entsteht. Das 3er-Tupel: Identifikator, Signatur und öffentlicher Schlüssel stellen das Zertifikat der Entität dar. Durch die Zertifikate lassen sich unmittelbar handelnde Entitäten (Principals) authentisieren (vgl. [ROE 99] Kapitel 5.5.2).

Damit auch die Zugehörigkeit des Schlüsselpaares zu dem jeweiligen Principal überprüft werden kann, werden Zertifikatketten eingesetzt.
Ein Zertifikat kann von einem weiteren Principal signiert werden, der für die Echtheit des Zertifikats bürgt. Dieses Principal kann gegebenenfalls wiederum von einem Principal signiert worden sein (usw.). Anhand der Überprüfung der Zertifikatkette, kann festgestellt werden, ob ein Zertifikat als vertrauenswürdig eingestuft werden kann. Ist dies der Fall, kann das Schlüsselpaar eindeutig dem Principal zugeordnet werden und das Principal kann als authentifiziert betrachtet werden.

Um alle Principals authentisieren zu können, ist:

- in der Klasse Principal ein Attribut Schlüsselpaar vorhanden
- in der Klasse Principal das Attribut Zertifikat vorhanden (welches u.a. den Identifikator enthält)
- das Verfahren zur Erstellung der Zertifikatkette festgelegt

Das Zertifikat für die Authority eines Agentensystems wird *out of band* erteilt, also nicht durch eine Entität die dem Agentensystem angehört. Die Erteilung eines Zertifikats kann z.B. durch eine vorhandene Zertifizierungshierarchie erstellt werden.
Das Zertifikat für ein Agentensystem kann mit dem Zertifikat für die Authority eines Agentensystems erstellt werden.

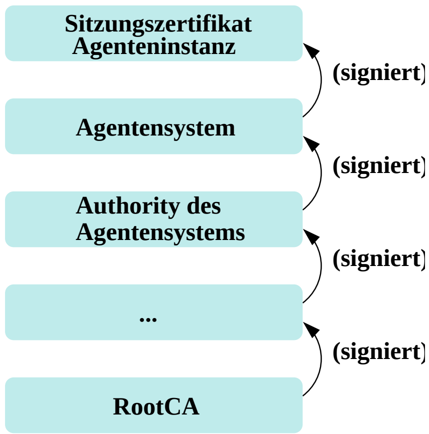


Abbildung 21: Zertifikatkette (aus [ROE 99])

Ablauf der Zertifikaterstellung eines Agentensystems (aus [ROE 99] Kapitel 5.5.4):

1. Das Agentensystem erzeugt beim Start selbständig ein Schlüsselpaar
2. Das Agentensystem generiert ein Zertifikat (enthält Eigenschaften des Agentensystems wie z.B. Name des ES, Version, ...)
3. Das vom Agentensystem generierte Zertifikat wird von der Authority des Agentensystems signiert

Aufgrund dieser Vorgehensweise wird die Authority eines Agentensystems immer durch das vorletzte Zertifikat in der Zertifikatkette definiert (Das vorletzte Zertifikat steht in der Abb. 21 an zweiter Stelle von oben).

Ein Sitzungszertifikat für Agenteninstanzen wird jeweils bei der Erzeugung einer Agenteninstanz von einem Agentensystem erstellt (aus [ROE 99] Kapitel 5.5.4):

1. Das Agentensystem erzeugt ein neues Schlüsselpaar
2. Das Agentensystem erzeugt ein Zertifikat für die Agenteninstanzen
3. Das Agentensystem signiert das Zertifikat

Aus diesem Sitzungszertifikat der Agenteninstanz lassen sich folgende Identitäten ableiten:

- Identität der Agenteninstanz (Zertifikatstelle 0)
- Identität des Agentensystems (Zertifikatstelle 1)
- Authority des Agentensystems (Zertifikatstelle 2)

Die Zertifikate für Implementierer und Anwender von Agenten werden in der Regel ebenfalls *out of band* von unabhängigen Instanzen generiert.

Folgend eine Übersicht der Eigentümer eigener Zertifikate (vgl. [ROE 99] Tabelle 5.2):

- **Agentengattung:** digital signiert (gespeichert als Teil der Daten)
- **Agenteninstanz:** eigenes Sitzungs-Zertifikat (gespeichert als eigenes Attribut der Agenteninstanz)
- **Agentensystem:** eigenes Zertifikat (gespeichert als eigenes Attribut des Agentensystems)
- **Implementierer der Agentengattung:** eigenes Zertifikat (aus dig. Signatur der Agentengattung)
- **Authority der Agenteninstanz:** eigenes Zertifikat (aus Attribut der Agenteninstanz)
- **Anwender Agenteninstanz:** eigenes Zertifikat (wird direkt vom Anwender übermittelt)
- **Authority Agentensystem:** eigenes Zertifikat (aus Zertifikatkette AS)
- **Anwender Agentensystem:** eigenes Zertifikat (wird direkt vom Anwender übermittelt)

Die Authentifikation aller Attribute einer Agenteninstanz kann auf eine Signatur des erzeugenden Agentensystems zurückgeführt werden. Dies betrifft z.B. auch die History der Agenteninstanz.

Realisiert wird die Verwendung von Zertifikaten und Schlüsseln durch die Verwendung der Java Cryptography Extension (JCE). Ein Zertifikat Manager (CertificateManager) erstellt und speichert das Agentensystemzertifikat und die Sitzungszertifikate der Agenteninstanzen. Außerdem überprüft er die Integrität der Zertifikatketten und kann anhand einer Authentication-Policy die Vertrauenswürdigkeit der Zertifikate bestimmen (vgl. [ROE 99] Kapitel 6.6.1).
Durch die Verwendung von Code Repositories, die die Gattungsdaten einer Agentengattung bereitstellen, ist es möglich, die Agentengattung und die Agenteninstanz unabhängig voneinander zu autorisieren. Die in einem Code Repository befindliche Agentengattung ist dort als JAR-Datei ([OAK 00]) abgelegt. Die Zuordnung der Agentengattung zu einem Implementierer sowie die Sicherstellung der Integrität wird durch die digitale Signatur des Implementierers realisiert. Hierfür legt der Implementierer sein Zertifikat innerhalb der JAR-Datei ab und signiert daraufhin die JAR-Datei (vgl. [ROE 99] Kapitel 6.5.1).

Die für die Authentisierung eingeführten Methoden werden zudem zur Sicherstellung der Integrität von Informationen und zur gesicherten Kenntlichmachung des Erstellers der Information herangezogen.

Fazit
In MASA können somit alle handelnden Entitäten authentifiziert werden und auch mit einer Person in Verbindung gebracht werden, die für die handelnde Entität verantwortlich ist. Zudem ist durch die Verwendung von Zertifikaten und der Wahl der eindeutigen Namen auch die globale Authentifikation sichergestellt. Die Authentisierungsanforderungen aus Kapitel 3 werden somit erfüllt. Durch den Authentifikationsmechanismus ist die Grundlage vorhanden, Zugriffsrechte an Subjekte innerhalb einer Zugriffskontrollpolitik gezielt vergeben zu können.

4.4 Rechtevergabe

Bei der Rechtevergabe muss zwischen der Rechtevergabe des Endsystems, des Agentensystems und des Agenten differenziert werden. Jede einzelne dieser Entitäten muss nach der Anforderungsanalyse die Informationssicherheit ihrer Objekte sicherstellen können.

4.4.1 Autorisationsmöglichkeit des Endsystems

Endsysteme realisieren den Schutz ihrer Informationen durch die Autorisation der Zugriffe anhand ihrer Zugriffskontrollpolitik.
Die Sicherungsmaßnahmen betreffen hier nur Objekte des Endsystems. Die in der Zugriffskontrollpolitik des Endsystems vorgenommenen Autorisationen sind über die gesamte Dauer der Ausführung eines Agentensystems gültig. Der Schutz durch die Zugriffskontrollpolitik des Endsystems wird daher dann eingesetzt, wenn auf Informationen des Endsystems nie oder immer von einem Agentensystem bzw. einer Agenteninstanz aus zugegriffen werden soll.

Die Zugriffskontrollpolitik des Endsystems ist von dem Betriebssystem abhängig, auf dem das Agentensystem zur Ausführung gebracht wird.
Wird ein Multiuser Betriebssystem eingesetzt, kann ein Endsystem zwischen verschiedenen Eigentümern eines Agentensystems unterscheiden. Unterschiedliche Personen könnten so Agentensysteme auf dem Endsystem, mit jeweils unterschiedlichen Zugriffskontrollpolitiken ausführen. Zugriffe durch ein Agentensystem auf das Endsystem können so individuell vom Endsystem autorisiert werden. Die Autorisation ist dabei von dem Administrator des Endsystems vorzunehmen.

Wird MASA beispielsweise verwendet, nur um statistische Daten eines Endsystems auszulesen und zu verwerten, wobei keine Schreibberechtigung benötigt wird, reicht es aus, dass die Personen, welche die Agentensysteme starten, nur Leseberechtigungen besitzen. Durch die Zugriffskontrollpolitik des Endsystems kann nun der schreibende Zugriff eines Agentensystems (und der Agenteninstanzen, die auf dem Agentensystem ausgeführt werden) auf das Endsystem gänzlich unterbunden werden.

Im Normallfall soll das Agentensystem jedoch das darunterliegende Endsystem vollständig nutzen können. Nur so kann die volle Funktionalität eines Agentensystems genutzt werden. Dabei ist davon auszugehen, dass der Benutzer, der das Agentensystem startet, eine Berechtigung besitzt, die einem Administrator des Endsystems gleichkommen. Die Sicherheitsanforderungen, die von dem Agentensystem erfüllt werden müssen, sind dann dementsprechend hoch.

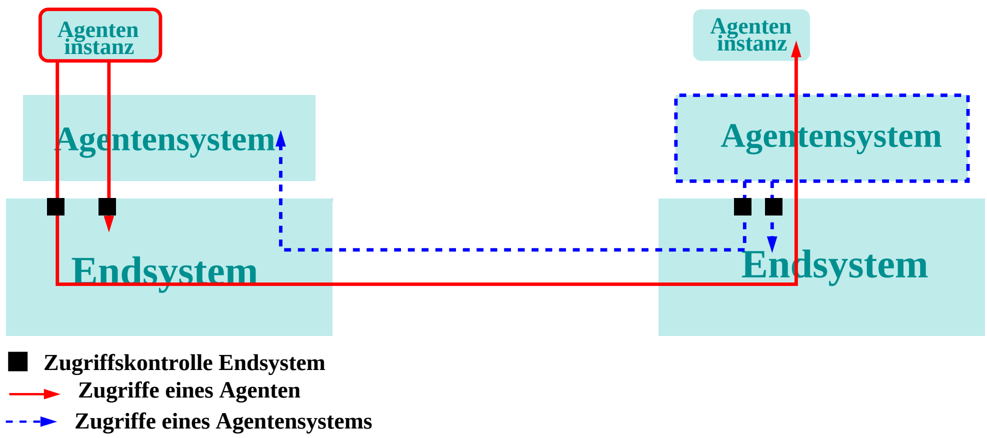


Abbildung 22: Zugriffskontrolle Endsystem

In der Abb. 22 sind die Zugriffe ersichtlich, die durch die Zugriffskontrolle des Endsystems beschränkt werden können. Dabei wird die Autorisation durch die Zugriffskontrollpolitik des Endsystems vorgenommen. Zu beachten ist jedoch, dass die in der Zugriffskontrollpolitik beschränkten Zugriffe über die ganze Lebenszeit eines Agentensystems bestehen bleiben und nicht geändert werden können. Zudem können die Zugriffsbeschränkungen nicht anwendungsspezifisch erfolgen. Aus diesem Grund sind, auch um die Funktionalität des Agentensystems nicht einzuschränken, die Zugriffsbeschränkungen der angegebenen Zugriffe möglichst nur vom Agentensystem aus vorzunehmen.

4.4.2 Autorisationsmöglichkeit des Agentensystems

Betrachtet man die Zugriffskontrollpolitik des Agentensystems, so kommt ihr eine besondere Bedeutung zu. Dies ist auf die Stellung des Agentensystems als zentrale Kontrollinstanz für die Agentensystemarchitektur zurückzuführen.

Das Agentensystem muss Zugriffe auf seine Schnittstellen und die Schnittstellen des Endsystems autorisieren. Zusätzlich müssen evtl. noch Zugriffe auf Schnittstellen der Agenteninstanzen autorisiert werden.
Dies ist dann notwendig, falls Agenteninstanzen eine von der Agenteninstanz nicht autorisierte Schnittstelle anbieten, die auf dem Agentensystem oder dem Endsystem kritische Aktionen ausführt. Durch die Autorisationsmöglichkeit des Agentensystems müssen hier kritische Aktionen, die durch die Agenteninstanz auf dem Agentensystem oder auf dem Endsystem ausgeführt werden, beschränkt werden. Das Agentensystem autorisiert in diesem Fall die Benutzung der von der Agenteninstanz angebotenen Schnittstelle.

In MASA können momentan Agenteninstanzen durch die Vergabe von Zugriffsrechten in der Java System Policy autorisiert werden. Die innerhalb der Java System Policy vergebenen Zugriffsrechte werden einer Agenteninstanz beim Ladevorgang zugeteilt. Es handelt sich daher nur um statische Zugriffsrechte, wobei die Autorisation auf der CodeSource der jeweiligen Agenteninstanz basiert.
Desweiteren existiert auf jedem Agentensystem eine MASA Policy-Datei (`masa.boot.policy`). Innerhalb dieser Datei werden für die Ausführung des Agentensystems relevante, spezifische Zugriffsrechte vergeben. Zusätzlich werden in der `masa.boot.policy`-Datei noch Zugriffsrechte für Agenteninstanzen vergeben. Dabei handelt es sich um rudimentäre Zugriffsrechte, die zur Ausführung einer Agenteninstanz benötigt werden. Zudem werden hier Zugriffsrechte vergeben, anhand derer bestimmt werden kann, ob eine Agenteninstanz berechtigt ist, Klassen zu definieren bzw. Klassen zu benutzen (vgl. [ROE 99] Kapitel 7.7.1).

Adressierung der zugreifenden Subjekte
Momentan können nur Agenteninstanzen autorisiert werden. Die Autorisation einer Agenteninstanz erfolgt auf Basis der Agentengattung. Dies ist darauf zurückzuführen, dass innerhalb der Standardpolicy zugreifende Subjekte nur anhand des Herkunftorts (URL) und einer Signatur adressiert werden können (vgl. Kapitel 2.9).
Eine Agentengattung wird in MASA in ein Code Repository in Form einer JAR-Datei abgelegt. Dabei entspricht der Name der JAR-Datei dem Namen der Agentengattung. Dadurch ist durch den Herkunftsort der Agentengattung die Agentengattung erkennbar. Durch die zusätzliche Möglichkeit, die JAR-Datei signieren zu können, kann hier die Signatur herangezogen werden, um den Implementierer der Agentengattung zu bestimmen. Der Implementierer einer Agentengattung legt sein Zertifikat innerhalb der JAR-Datei ab und signiert daraufhin die JAR-Datei. Autorisationen innerhalb der Standard Java Policy auf Basis einer Signatur repräsentieren daher den Implementierer der Agentengattung.

Betrachtet man den folgenden Eintrag der System Java Policy, wird der Agentengattung [messe], welche vom Implementierer [Lorenz] implementiert worden ist, erlaubt, den Namen des Betriebssystems auszulesen:

```
grant signedBy "Lorenz",
      codeBase "systemresource://de/unimuenchen/informatik/mnm/masa/agent/messe"
{
  permission java.util.PropertyPermission "os.name", "read";
}
```

Zugriffsrechte
Bei den in der Standard Java Policy eingesetzten Zugriffsrechten handelt es sich um spezifische Zugriffsrechte (vgl. Kapitel 2.6.2). D.h., dass das Zugriffsrecht die Berechtigung und das spezifische Objekt adressiert. Das Zugriffsrecht hängt somit eng mit der Funktionalität des Objekts zusammen.

Adressierung der Objekte
Die Adressierung der Objekte erfolgt demnach innerhalb des Zugriffsrechts. Dabei existieren für die Objekte des Endsystems, auf die zugegriffen werden kann, standardisierte Permission-Objekte. Ein Zugriff auf ein Endsystem Objekt erfolgt durch einen Java-API Aufruf. Innerhalb der Java-API existieren spezifische Permission Objekte, die verwendet werden können. Diese Permission Objekte können in der Java Policy zugeteilt werden. Die Granularität der Objekte des Endsystems sind vorgegeben, i.Allg. jedoch ausreichend.
In MASA können innerhalb der Standard Java Policy jedoch keine Zugriffsrechte für Objekte der Agentensysteme und Agenteninstanzen vergeben werden, da hier keine Permission Objekte zur Verfügung gestellt werden.
Die Möglichkeit, spezifische Permission Objekte zu erstellen, ist zwar gegeben, jedoch fehlt hier eine Spezifikation. Werden solche Permission Objekte erzeugt, müsste ein quasi Standard für MASA existieren, der die Struktur der Permission Objekte festlegt. Nur so könnte sichergestellt werden, dass jeder Administrator die zu vergebenden Permission Objekte kennt, eindeutig interpretieren und vergeben kann.

Eine Beispielimplementierung von MASA spezifischen Permission Objekten findet sich in [ROE 99] Kapitel 7.7.1.

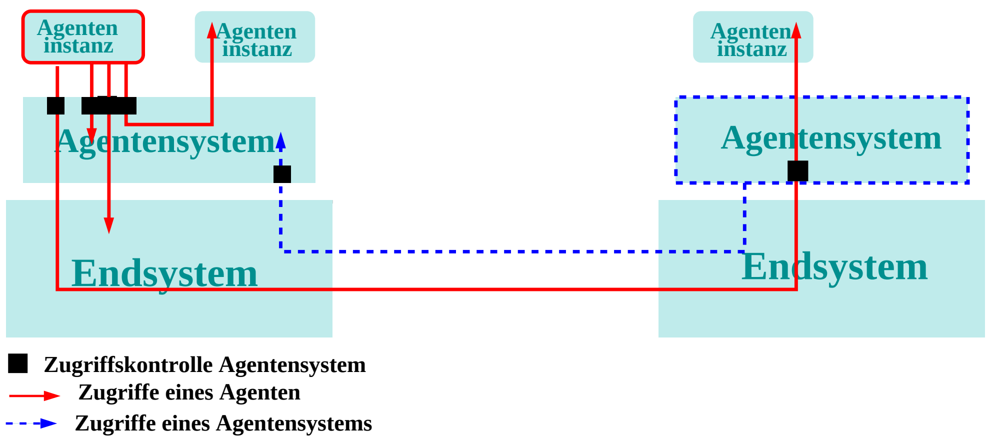


Abbildung 23: Zugriffskontrolle Agentensystem

In der Abb. 23 werden alle Zugriffsmöglichkeiten aufgezeigt, die durch die Zugriffskontrolle des Agentensystems kontrolliert werden müssen. In MASA können momentan nur Zugriffe von Agenteninstanzen auf Objekte des Endsystems und Zugriffe bezüglich der Definition und der Nutzung von Klassen autorisiert werden.

Bewertung
Betrachtet man die Autorisationsmöglichkeit des Agentensystems, so sind hier keine anwendungsspezifischen Autorisationen möglich, da keine subjektspezifische Vergabe von Zugriffsrechten möglich ist. Erst durch eine subjekt- und objektspezifische Autorisation ist es möglich, ein Subjekt nur für eine spezielle Aufgabe und damit anwendungsspezifisch zu autorisieren.

Agenteninstanzen können nur auf Basis ihrer Agentengattung autorisiert werden. Die Autorisation einer Agenteninstanz auf Basis der Agentengattung ist jedoch unzureichend. Zugriffe von Agentensystemen auf Agentensysteme und Agenteninstanzen können überhaupt nicht autorisiert werden. Letztendlich können innerhalb der Java System Policy nur Zugriffe von Agenteninstanzen auf Informationen des Endsystems autorisiert werden und das nur auf Basis der Agentengattung der Agenteninstanz. Zudem ist die Autorisation nur statisch. D.h. die Autorisation kann nicht von bestimmten Bedingungen abhängig gemacht werden.

Durch die Verwendung von spezifischen Zugriffsrechten können Zugriffsrechte für Objekte anwendungsspezifisch erzeugt werden. Eine objektspezifische Autorisation ist durch die Möglichkeit der Erzeugung eigener spezifischer Zugriffsrechte gegeben. Jedoch wird kein spezielles Konzept zur Erzeugung und Benennung von MASA spezifischen Permission-Objekten vorgegeben. Daher ist die Eindeutigkeit und Verwechslungsfreiheit der Zugriffsrechte nicht gegeben. Zugriffe von Agenteninstanzen auf Objekte des Agentensystems bzw. einer weiteren Agenteninstanz können somit ebenfalls nicht geeignet autorisiert werden.

4.4.3 Autorisationsmöglichkeiten der Agenten

Betrachtet man Agenteninstanzen, so sind hier Zugriffe von anderen Agenteninstanzen und dem Agentensystem, auf dem sich die Agenteninstanz befindet, zu kontrollieren (siehe Abb. 24).

Eine Agenteninstanz ist momentan in MASA nicht in der Lage, Zugriffe auf ihre Informationen zu beschränken. Die Informationssicherheit der Agenteninstanzen ist daher nicht sichergestellt.

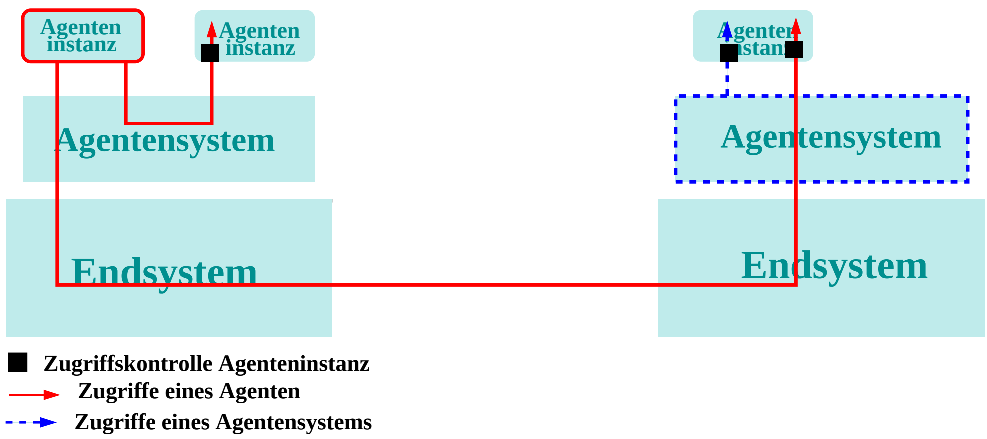


Abbildung 24: Zugriffskontrolle Agenteninstanz

4.5 Rechtedurchsetzung

Die Rechtedurchsetzung der Zugriffskontrollpolitik des Agentensystems wird durch das Agentensystem durchgesetzt. Die Zugriffskontrolle (Access Control) wird, da die Standard Implementierung von Java 2 verwendet wird, durch den eingebundenen SecurityManager und den AccessController realisiert (vgl. Kapitel 2.9.1).

Jede Agenteninstanz wird von einem eigenen ClassLoader geladen, der jeweils seinen eigenen separaten Namensraum definiert. Dieser ClassLoader ist für die Lokalisierung und das Laden von Klassen verantwortlich, die die Agenteninstanz bei der Ausführung benötigt. Durch den ClassLoader Mechanismus wird der Agenteninstanz zudem eine spezifische Protection Domain zugewiesen. Eine Protection Domain beinhaltet dabei die Zugriffsrechte, die der Agenteninstanz auf Basis ihrer Agentengattung innerhalb der Java Policy Datei zugeteilt worden sind. Genau diese Zugriffsrechte werden nun von dem Agentensystem durch die Verwendung des SecurityManagers bzw. des AccessControllers durchgesetzt.

Greift nun eine Agenteninstanz auf Informationen des Endsystems zu, wird dieser Zugriff von dem SecurityManager des Agentensystems kontrolliert.

Die Durchsetzung der Zugriffskontrollpolitik des Endsystems erfolgt betriebssystemspezifisch. Da Agenteninstanzen keine Zugriffskontrollpolitik haben, ist deren Durchsetzung auch nicht möglich.

4.6 Zusammenfassung

Eine eindeutige Identifizierung von Agenten und Agentensystemen ist durch die eindeutige Namenstruktur gegeben. Mit der Hilfe von Zertifikaten und Signaturen wird auch die Authentifikation der Subjekte sichergestellt. Durch die Authentifikation sind Subjekte einer verantwortlichen Person zuordenbar.

Die Unterscheidung zwischen Agentengattungen und Agenteninstanzen ermöglicht eine differenzierte Autorisation anhand einerseits der Agentengattung und deren Implementierer und andererseits der Attribute einer Agenteninstanz.
Die Kanäle, welche zur Kommunikation der Entitäten genutzt werden, sind gänzlich durch die Verwendung von Public-Key-Verfahren gesichert.
Durch die Bereitstellung von abgeschotteten Ausführungsumgebungen sind Agenteninstanzen voreinander geschützt. Die Bereitstellung der abgeschotteten Ausführungsumgebungen durch das Agentensystem schützt zudem das Agentensystem vor Agenteninstanzen.

Betrachtet man den Schutz der Daten einer Agenteninstanz oder eines Agentensystems, so kommen zuerst die Visibilitätsmodifikatoren und noch einige weitere Modifikatoren (z.B. der `final` Modifikator) von Java zum Einsatz (vgl. Kapitel 2.9). In MASA werden die von Java bereitgestellten Schutzmechanismen konsequent eingesetzt und dienen der Implementierung statischer Zugriffstrukturen, wodurch Klassen, Attribute und Methoden geschützt werden.

Zum Schutz der Endsystemschnittstelle verwendet MASA die Java Platform 2 Security Architecture (vgl. Kapitel 2.9). Dabei kommt der Mechanismus der ProtectionDomain und des SecurityManagers in Kombination mit dem AccessController zum Einsatz.
Durch die Verwendung der Security Policy wird einer Agenteninstanz eine spezifische Protection Domain zugewiesen, die die innerhalb der Policy zugeteilten Zugriffsrechte in Form von Permission Objekten enthält. Die Permission Objekte sind für Aktionen, welche auf dem Endsystem ausgeführt werden können, durch die Java Sicherheitsarchitektur vorgegeben.

Die zur Sicherung der Informationssicherheit eingesetzten Mechanismen erfüllen jedoch nicht die Anforderungen aus Kapitel 3. So wird nicht jeder Zugriff auf ein Objekt kontrolliert (complete mediation). Damit kann auch nicht garantiert werden, dass jeder Zugriff, der nicht explizit erlaubt ist, verboten ist (fail-safe defaults). Zugriffe von Agentensystemen können innerhalb der Policy Datei nicht autorisiert werden. Agenteninstanzen können Zugriffe auf ihre Schnittstellen überhaupt nicht autorisieren. Nur der statische Code einer Agentengattung ist durch die vorhandene Signatur der JAR-Datei vor unautorisierten Modifikationen durch die Signatur geschützt. Dynamische Informationen, die eine Agenteninstanz während der Lebenszeit besitzt, sind ebenfalls nicht vor unautorisierten Zugriffen geschützt.

Es kann demnach nur das Agentensystem Zugriffe auf seine Schnittstellen und auf Schnittstellen von Agenteninstanzen beschränken. Betrachtet man die Autorisation der Zugriffe der Agenteninstanzen, so sind jedoch auch hier nicht alle Anforderungen erfüllt.
Die Adressierung der Agenteninstanzen in der Java Policy Datei kann nicht benutzerspezifisch erfolgen. Die Vergabe von Berechtigungen nur auf Basis der Gattung und des Implementierers der Gattung ist als unzureichend anzusehen. Hier wird eine benutzerspezifische Vergabe von Zugriffsrechten benötigt.
Desweiteren kann die Autorisation von Agenteninstanzen auch nicht von bestimmten Bedingungen abhängig gemacht werden. So ist eine Autorisation von Agenteninstanzen, die von der Vergangenheit (History) der Agenteninstanz abhängt, nicht möglich.

Betrachtet man explizit die Zugriffsrechte, so handelt es sich um spezifische Zugriffsrechte. Diese Zugriffsrechte adressieren somit auch die Objekte und nicht nur die Aktion. Die objektspezifische Adressierung ist hier möglich, jedoch fehlt eine Standardisierung, um die Eindeutigkeit der Zugriffsrechte zu garantieren.
Sind die durch die Java-API standardisierten Zugriffsrechte für Objekte des Endsystems i.Allg. ausreichend, fehlen MASA spezifische Zugriffsrechte. Dadurch können Zugriffe auf das Agentensystem bzw. auf Agenteninstanzen nicht autorisiert und daher auch nicht durchgesetzt werden.

Die Limitierung von Systemressourcen ist in MASA nicht möglich. Ist ein Zugriff auf eine Systemressource erlaubt, kann diese unbeschränkt genutzt werden. Hier besteht die Gefahr von Denial-Of-Service Attacken.

Es können keine Entscheidungen anhand dynamischer Attribute getroffen werden. Dadurch ist die Autorisation der Zugriffe auf statische Eigenschaften eingeschränkt. Eine Agenteninstanz bekommt beim Ausführungsbeginn Zugriffsrechte zugeteilt, die sich während der Ausführungszeit nicht mehr ändern können, d.h. es können keine neuen Zugriffsrechte hinzukommen und es können auch keine Zugriffsrechte entfernt werden.

Die in MASA als Politik-Sprache eingesetzte Standard Politik Syntax von Java 2 stellt sich als ungeeignet heraus. Um die zugreifenden Subjekte in MASA individuell adressieren zu können, fehlt eine Spezifikation einer geeigneten Politik-Sprache.

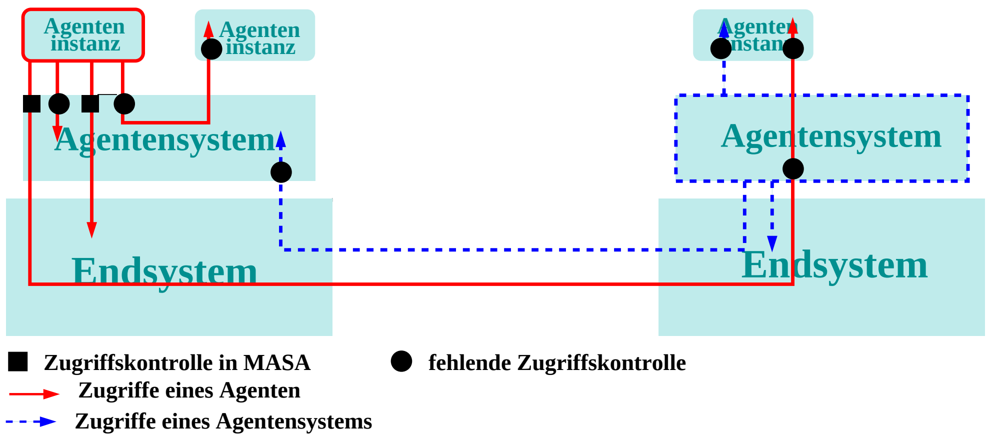


Abbildung 25: Autorisation in MASA

Betrachtet man die Abb. 25, so sind hier einerseits Zugriffe erkenntlich, welche in MASA autorisiert und durchgesetzt werden können und andererseits Zugriffe, weder autorisiert noch durchgesetzt werden. Die Zugriffskontrollen des Endsystems wurden dabei vernachlässigt.

5 Standards, Konzepte und eingesetzte Rechtekonzepte

Bei der individuellen Entwicklung verschiedener Agentensysteme stand zumeist die Funktionalität im Vordergrund. Die Sicherheitstechniken bestehender Agentensysteme wurden speziell für das jeweilige Agentensystem entwickelt. Dadurch kommt es zu Defiziten bezüglich der Interoperabilität der Agentensysteme. Betrachtet man die Standardisierungsvorgänge, so wird ein Rahmenwerk gesucht, das kompatible Sicherheitstechniken in ein effektives Sicherheitsmodell integriert [WAK 00].

Nachdem kurz auf die Standardisierungsvorgänge eingegangen wird, werden Konzepte vorgestellt, die zum Schutz der Agentensysteme und der Agenten herangezogen werden können. Abschließend werden Agentensysteme bezüglich der realisierten Schutzmechanismen und Techniken betrachtet.

5.1 Standardisierung bei MAFS, MASIF und FIPA

Das Standardisierungsgremium Mobile Agent Facility Specification (MAFS) macht über die Realisationskonzepte keine Angaben. Die Standardisierung betrifft Anforderungen an eine Agentensystemarchitektur. Sind alle Anforderungen erfüllt, ist die Rechtssicherheit einer Agentensystemarchitektur gegeben. Sicherheitspolitiken bei MAFS besitzen folgende Eigenschaften [MAF 00]:

- **Einschränkung oder Gewährung von Agenten Fähigkeiten**
  - erstellen eines neuen Agenten
  - reisen des Agenten
  Die Regeln werden durch Überprüfungsmechanismen der Programmiersprache oder der Laufzeitumgebung durchgesetzt.

- **Limitieren des Ressourcenverbrauchs**
  - Administrator kann die Ressourcen limitieren
  - Anzahl neuer Agenten
  - Anzahl der Netzwerkverbindungen
  Die Durchsetzung der Regeln werden durch Überprüfungsmechanismen der Programmiersprache oder der Laufzeitumgebung durchgeführt.

- **Einschränkung oder Gewährung von Zugriffen**
  - der Administrator kontrolliert die Zugriffe
  - Operationen, die ein Agent aufrufen kann, unterstehen einer Zugriffskontrolle
  - Daten, auf die ein Agent Zugriff hat, unterstehen einer Zugriffskontrolle
  Die Durchsetzung einer Zugriffskontrolle erfolgt dabei generell über Kommunikations-Sicherheitsmechanismen wie z.B. einer Zugriffskontrollliste oder durch die Agenten-Programmiersprache.

Der Mobile Agent System Interoperability (MASIF) Standard der Object Management Group (OMG) hingegen macht eine klare Aussage über das Thema Sicherheit. Diese stützt sich auf die CORBA Security Services Architektur ab. Leider betrachten die Security Services beim CORBA Modell nur die Sicherheitsfragen der Agenten Plattform hinreichend. Die unabhängigen Sicherheits Services, die ein Agent benötigt, werden ignoriert [WAK 00].

Schwerpunkt bei der Foundation for Intelligent Physical Agents (FIPA) ist die Standardisierung der Agenten Kommunikations Sprachen, die kooperierende Agenten nutzen. Viele Details, die die Architektur der Agenten Plattform betreffen, benötigen noch Entwicklungszeit, bevor man sich dem Thema der Sicherheit widmen kann [WAK 00].

5.2 Konzepte zum Schutz von Agentensystemarchitekturen

Der Schutz einer Agentensystemarchitektur kann in den Schutz des Agentensystems, den Schutz der Agenten und den Schutz eines Verbundes von Agentensystemen aufgeteilt werden.

Die Autorisation der Agenten wird über verschiedene Mechanismen realisiert. Es kommen Sicherheitspolitiken, Credentials, Zugriffskontrolllisten (ACLs),Tickets und Capabilities zum Einsatz.

5.2.1 Schutz des Agentensystems

Agentensysteme schützen sich vor unautorisierten Zugriffen, indem die zugreifende Entität identifiziert und die Identität verifiziert wird. Anhand der Authentifizierten Entität kann so eine Zugriffskontrolle die Autorisation der Entität überprüfen.

Dadurch ergibt sich der folgende Ablauf:

1. Kryptographische Verifikation der Identität des Owners eines Agenten
2. Erteilung von Zugriffsrestriktionen des Agenten aufgrund der Identität des Owners
3. Ausführung des Agenten in einer sicheren Umgebung, welche die Restriktionen durchsetzen kann.

Die Zugriffskontrolle basiert zumeist auf dem Referenz-Monitoring Prinzip, so dass jeder Zugriff der Zugriffskontrolle unterworfen wird. Die Realisation des Monitoring-Konzepts basiert auf konventionellen Mechanismen und Techniken wie zum Beispiel dem Einsatz von:

- Mechanismen zur Isolation von Prozessen untereinander und dem Kontroll-Prozess
- Mechanismen zur Zugriffskontrolle der Ressourcen
- Kryptographische Methoden zum Entziffern des Informationsaustausches
- Kryptographische Methoden zur Identifizierung und Authentifikation von Benutzern, Agenten und Agentensystemen
- Mechanismen zur Aufzeichnung (logging) von Sicherheitsrelevanten Ereignissen

Folgend werden Konzepte vorgestellt, die für den Schutz der Agentensysteme eingesetzt werden können [WAK 00].

1. **Software basierte Fehler Isolation (Software-Based Fault Isolation)**
   Software-Based Fault Isolation ist eine Methode, Applikations Module in einer abgegrenzten, virtuellen Umgebung zu isolieren. Diese Technik erlaubt, nicht vertrauenswürdige Programme, geschrieben in unsicheren Sprachen (bspw. C) innerhalb eines virtuellen, abgegrenzten Adressbereiches sicher auszuführen. Zugriff auf die Systemressourcen können auch kontrolliert werden. Diese Technik ist auch unter dem Namen Sandboxing bekannt.
2. **Sichere Code Interpretation (Safe Code Interpretation)**
   Agentensysteme benützen in der Regel zu interpretierende Sprachen. Hauptanliegen dabei ist, eine heterogene Umgebung zu unterstützen. Die Idee von Safe Code Interpretation ist, dass bestimmte Kommandos vor Programmen gesichert bzw. verboten werden können.

   Die wohl meistbenützte Interpreter Sprache ist Java. Die Programmiersprache Java und ihre Ausführungsumgebung bietet Sicherheit aufgrund von Strong-Type Safety. Java folgt dabei dem Sandbox-Modell, das wie oben beschrieben den Speicher und die Zugriffsmethoden isoliert und zur Ausführung eine exklusive Ausführungsumgebung bereitstellt.

Innerhalb von Java wird z.B. das Static-Byte-Checking in Form von Byte-Code-Verification angewendet, um die Sicherheit des Programms zu kontrollieren. Dabei wird auch teils Dynamic-Checking während der Ausführungszeit unterstützt. Ein separater Namensraum steht dem nicht vertrauenswürdigen Code zur Verfügung. Referenzen zwischen Modulen in verschiedenen Namensräumen ist nur bei als `public` deklarierten Methoden möglich.

3. **Signierter Code (Signed Code)**
   Eine weitere Technik Agentensysteme zu schützen sind digital signierte Programme oder Objekte [OAK 00]. Dabei wird durch die digitale Signatur das Objekt authentifiziert und dessen Integrität und Originalität sichergestellt.

   Durch die Signatur eines Agenten kann so z.B. der Auftraggeber, unter dessen Namen der Agent unterwegs ist, bestimmt werden. Denkbar ist somit auch, den Agenten die Rechte, die der Auftraggeber besitzt, einzuräumen (Attribute Certificate). Ein Objekt kann je nach Modell auch multiple Signaturen besitzen, wobei es dann mehreren Principals angehören kann.

   Beim Signieren kommt die Public Key Kryptographie zum Einsatz. Eingesetzt wird ein Schlüsselpaar, ein privater und ein öffentlicher Schlüssel. Die weite Verbreitung und der Einsatz der Public Key Infrastruktur (PKI) spielt eine große Rolle, um diesen Sicherheitsmechanismus geeignet nutzen zu können.

   Durch eine digitale Unterschrift kann das Agentensystem die Herkunft des Agenten eindeutig bestimmen. Durch Überprüfen der digitalen Signatur des Programmcodes kann außerdem festgestellt werden, ob der Agent von anderen Agentensystemen manipuliert worden ist. Der Status bzw. die mehrfach zu verändernden Daten eines Agenten können nicht signiert werden und stellen deshalb ein Risiko dar. In diesem Fall könnte der Programmcode des Agenten eine Routine enthalten, die den Status überprüfen kann, wodurch das Agentensystem sicherstellen kann, dass nur Agenten mit einwandfreiem Status gestartet werden [WAK 00].

4. **Zustandsschätzung (State Appraisal)**
   Das Ziel von State Appraisal ist es, Gewissheit darüber zu erlangen, dass der Agentenstatus nicht verfälscht wurde. Bösartige Veränderungen der Zustände werden erkannt, da z.B. bestimmte Zustände nicht möglich sind [WAK 00].

5. **Pfad Aufzeichnung (Path Histories)**
   Der zurückgelegte Pfad des Agenten soll nachvollziehbar sein. Es ist nötig, dass jedes Agentensystem einen signierten Eintrag beim Agenten hinterlässt, der die Identität des Agentensystems und die des nächsten Agentensystems enthält. Ein Agentensystem kann durch dieses Wissen entscheiden, ob es den vorangegangenen Agentensystemen trauen will oder nicht [WAK 00].

6. **Code mit mitgeführtem Sicherheitsbeweis (Proof Carrying Code)**
   Der Programmierer beweist, dass das Programm sichere Eigenschaften nach den Auflagen des Benutzers besitzt. Diese Technik verhindert Attacken, während signierter Code nur zur Authentifikation, Identifikation und Unverfälschtheit des Codes dient. Die Sicherheitsaussage, die die Semantik des Programms repräsentiert, wird direkt vom native Code generiert, um sicherzustellen, dass der mit dem Code mitgeführte Beweis mit diesem zusammenhängt. Verfälschung des Sicherheitsbeweises oder des Codes hat einen Verifikationsfehler, bzw. falls die Verifikation geklappt hat, eine Sicherheitscodetransformation zur Folge. Die Proof Carrying Code Methode basiert auf den Prinzipien der Logik, der Type-Theorie und der formellen Verifikation (Formal Verification) [WAK 00].

7. **Ein Security Assistant überwacht Ereignisse**
   Ein etwas anderer Ansatz ist in Rasmusson und Jansson (1996) zu finden. Sie unterscheiden zwischen Hard Security und Soft Security.
   Bei Soft-Security wird ein Agent während der Ausführung überwacht. Falls eine Operation ausgeführt werden soll, für die keine Berechtigung vorliegt bzw. die dem System schadet, wird der Agent angehalten und aus der Ausführungsumgebung entfernt.
   Bei Hard-Security werden Agenten und Programme einem bestimmten Typ zugeordnet. Bei jedem Typ wird festgelegt, welche Ressourcen benötigt werden bzw. welches Verhalten erwartet wird. In der Ausführungsumgebung für Agenten ist ein Security Assistant dafür zuständig die Aktivitäten jedes Programms zu überwachen und dessen Verhaltensmuster mit für seinen Typ normalem Verhalten zu vergleichen. Dabei beobachtet der Security Assistant folgende Kriterien:

   - Intensität von Aktivitäten (z.B. plötzliche Ausführung vieler I/O-Operationen).
   - Zugriffszähler (z.B. Anzahl der gelesenen Files).
   - Kategorische Messungen (z.B. Verzeichnisse werden Kategorien zugeordnet; es wird gemessen, auf welche Kategorien häufiger als auf andere zugegriffen wird).
   - Statistische Verteilungen (z.B. Standardabweichung und Durchschnitt der Dauer von Datenbankzugriffen).

   Für die Daten zu diesen Kriterien werden dann entweder Abweichungen von normalen Werten untersucht, oder sie werden mit Mustern bösartigen Verhaltens verglichen.

8. **Autorisation mit Attribut Zertifikaten (Authorization and attribute certificates)**
   Attribut-Zertifikate erweitern die Puplic Key Infrastruktur. Dem Zertifikat, welches zur Authentifikation herangezogen wird, wird ein Attribut mitgegeben (z.B eine Policy, Privilegien, usw.), die mit zum Zielsystem übertragen werden. Aufgrund des Zertifikats und den mitgelieferten Attributen kann nun der Besitzer authentifiziert werden und es kann z.B. bestimmt werden, welche Operationen ausgeführt werden dürfen und welche nicht.

5.2.2 Schutz des Agenten

Ein Agentensystem sollte nicht in der Lage sein, unautorisiert auf einen Agenten zuzugreifen. Der Schutz des Agenten gestaltet sich jedoch enorm schwer, da ein Agent dem Agentensystem, auf dem er ausgeführt wird, in jeder Hinsicht ausgeliefert ist.

Während Gegenmaßnahmen zum Schutz des Agentensystems zum Teil mit traditionellen Schutzmechanismen getroffen werden können, basieren Gegenmaßnahmen zum Schutz der Agenten in der Regel auf der Erkennung eines Angriffs.

Folgend sollen Mechanismen vorgestellt werden, die zum Schutz eines Agenten bzw. dessen Information eingesetzt werden können:

1. **Verschlüsselung** (Encrypted Algorithms)
   Ein Agent sollte sensitive Informationen nur verschlüsselt bei sich tragen. Das Programm und seine Eingaben an sich können dabei auch verschlüsselt sein [SAN 96]. Das verschlüsselte Programm ist direkt ausführbar, wobei die Funktionalität erhalten bleibt. Die Ausgabe ist ebenso verschlüsselt und kann nur von demjenigen entschlüsselt werden, der das Programm verschlüsselt hat.

   In "Towards Mobile Cryptography" [SAT 98b] wird versucht, die Sicherheitsproblematik durch den Einsatz der Kryptographie zu lösen. Dabei wird auch der Fall, dass sich der Agent in einer nicht vertrauenswürdigen Ausführungsumgebung befindet, betrachtet.

2. **Bedeutungslosigkeit einzelner Komponenten** (Components)
   Die Information ist ohne die Zusammenarbeit mit einem vertrauenswürdigen Agentensystem

bedeutungslos. Eine andere Möglichkeit besteht darin, die sensitive Information soweit aufzusplitten, dass sie einzeln bedeutungslos ist. Bei der Übertragung der einzelnen Pakete wird dann jeweils überprüft, ob ein Paket attackiert wurde. Dies setzt geeignete Aufzeichnungsmechanismen (Logging) des Agenten voraus. Jede Komponente kann dabei mit unterschiedlichen Schlüsseln verschlüsselt und signiert werden. Eine erfolgreiche Attacke bringt somit nur einen unbrauchbaren Teil ans Licht.

3. **Weiterreichung nur an vertrauenswürdige Agentensysteme**
   Sensitive Informationen werden nur über vertrauenswürdige Agentensysteme weitergeleitet und kommen daher nie in Gefahr von einem nicht vertrauenswürdigen oder unbekannten Agentensystem attackiert zu werden. Die Sicherheit basiert hier auf dem Vertrauensverhältnis.

4. **Signatur** (Signature)
   Um Daten vor dem Ausspionieren durch Agentensysteme zu schützen, können die Agenten auf einem Agentensystem gesammelte Daten mit dem Public Key ihres Eigentümers verschlüsseln. Diese verschlüsselten Daten können sie dann entweder direkt an den Eigentümer zurücksenden oder sie werden an den Agenten angehängt. Nur der Eigentümer kann die Daten mit dem passenden Schlüssel wieder entschlüsseln.
   Eine Sabotage des Verhaltens des Agenten kann verhindert werden, indem der Code und unveränderliche Daten durch eine digitale Unterschrift des Eigentümers gesichert werden. Ein Agentensystem untersucht dann vor dem Start des Agenten, ob die Unterschrift noch intakt ist. Wurde der ursprüngliche Inhalt geändert, so wird dies hier erkannt und der Agent kommt gar nicht zur Ausführung. Es wird verhindert, dass ein Agentensystem das zukünftige Verhalten des Agenten zu seinen Gunsten beeinflussen kann.

   Probleme beim Signieren eines Agenten auf dem Home-Agentensystem entstehen, sobald der Agent von einem zweiten auf einen dritten Agentensystem reist (multi-hop Agent). Es kann nur der Teil des Agenten durch die Signatur kontrolliert werden, der der Home-Agentensystem Signatur entspricht. Änderungen der Daten oder des Status des zweiten Agentensystems können natürlich nicht mit dieser Signatur kontrolliert werden. Durch eine Verständigung mit dem Home-Agentensystem nach jedem besuchten Agentensystem kann gegebenenfalls der Schaden verringert werden. Dies bedeutet aber auch eine höhere Belastung des Systems. (Das Jumping Beans Agentensystem implementiert z.B. eine Client/Server Architektur, wobei ein Agent jedesmal zu einem sicheren zentralen Agentensystem zurückkehrt, bevor er ein neues Agentensystem besucht.). Der Vorteil ist, dass es sich hier nur um Two-Hop Boomerang Agenten handelt. Nachteilig wirkt sich hier der große Overhead aus.

5. **Einkapseln von Teilergebnissen** (Partial Result Encapsulation)
   Die Ergebnisse der Aktionen eines Agenten werden jeweils für jedes besuchte Agentensystem gekapselt. Später können so die einzelnen Vorgänge jedes Agentensystems verifiziert werden. Dabei dient die Einkapselung mehreren Zielen, die mithilfe unterschiedlicher Mechanismen umgesetzt werden können:

   - Vertraulichkeit durch das Verschlüsseln der Daten
   - Integrität und die Abrechnungsmöglichkeit (Accountability) durch digitale Signaturen

   Die eingekapselte Information hängt dabei von der Implementierung des Agenten ab, wobei es drei alternative Wege gibt die Ergebnisse einzukapseln:

   - Der Agent kann die Informationen einkapseln
   - Das Agentensystem ist in der Lage die Informationen geeignet einzukapseln
   - Eine vertrauenswürdige dritte Partei berechnet den digitalen Fingerabdruck der Ergebnisse

Keiner dieser Mechanismen verhindert bösartiges Handeln des Agentensystems, jedoch kann bösartiges Handeln erkannt werden.

Eine weitere Methode zum Einkapseln der Informationen ist der Gebrauch des Partial Result Authentication Codes (PARC). Hierbei handelt es sich um kryptographische Prüfsummen, erstellt mithilfe der Secret Key-Kryptographie (ähnlich dem message authentication code). Anfangs existieren mehrere Schüssel. Verwendet ein Agent nun einen dieser Schlüssel, so wird dieser aus der Liste des Agenten gelöscht, bevor er zum nächsten Agentensystem zieht. Der Vorteil ist, dass die Authentifikation der Ergebnisse auf jedem besuchten Agentensystem möglich ist, und das mit vorwärts Integrität.

Werden jedoch von einem Agentensystem Kopien der noch nicht genutzten Schlüssel gemacht bzw. ist die Schlüsselgenerierungsfunktion eines Agenten bekannt, sind diese beim erneuten Besuch des Agenten bzw. auch auf dem Agentensystem die unter Umständen mit diesem Agentensystem kooperieren, wertlos. Schon verschlüsselte Daten können so modifiziert werden, indem man Daten mit den bekannten Schlüsseln verschlüsselt. Um die Vorwärtsintegrität zu bewahren, besteht die Möglichkeit, Agenten, nachdem sie ein unbekanntes Agentensystem besucht hatten, wieder auf ein vertrauenswürdiges Agentensystem migrieren zu lassen, bei dem der Agent einen digitalen Zeitstempel bekommt. Auch hier ist der zusätzliche Overhead als Nachteil zu nennen.

6. **Gegenseitiges Aufzeichnen der Reiseroute** (Mutual Itinerary Recording)
   Eine interessante Variation des Path Histories ist die Aufzeichnung der Spur eines Agenten mithilfe eines kooperierenden Agenten und umgekehrt. Während dem Bewegen zwischen Agentensystemen sendet der Agent die Information der zuletzt besuchten, der momentanen und des nächsten Agentensystems zum kooperierenden Partner über einen authentifizierten Kanal. Der Partner kann dann gegebenenfalls auf Ungereimtheiten reagieren. Dadurch, dass sich zumindest ein Partner auf einem vertrauenswürdigen Agentensystem befindet, können so böswillige Agentensysteme entdeckt werden.

   Dieses Schema kann erweitert werden, indem es z.B. mehr als zwei kooperierende Agenten gibt. Es besteht aber auch die Möglichkeit einen Agenten statisch auf dem Home-Agentensystem zu belassen.

7. **Aufzeichnung der Reiseroute mithilfe von Kopien und Abstimmung** (Itinerary Recording with Replication and Voting)
   Werden mehrere Kopien eines Agenten losgesendet, so ist die Wahrscheinlichkeit größer, dass die in Auftrag gegebene Arbeit durchgeführt wird, da nicht alle Agenten wohl das Pech haben, auf ein böswilliges Agentensystem zu gelangen. Desweiteren können böswillig veränderte Agenten erkannt werden, falls die Eigenschaften der meisten anderen Agenten nicht stimmig sind. Diese Technik kann aber nur da eingesetzt werden, wo eine Mehrfachausführung eines Agenten möglich bzw. sinnvoll ist.
   Muss ein Agent beispielsweise für seine Ausführung bezahlen, könnte die eigentlich unnötige Mehrfachausführung zu hohe Kosten verursachen.

8. **Ausführung des Agenten aufzeichnen** (Execution Tracing, Audit Logs)
   Diese Technik dient dem Erkennen unautorisierter Modifikationen eines Agenten während der Ausführung des Agenten auf einem Agentensystem. Dabei muss jedes Agentensystem diese Aufzeichnung tätigen und geeignet vor Modifikationen schützen (fingerprint, etc.). Die Signatur des Agentensystems ist dabei nur für die Instruktionen, die mit dem Agentensystem zusammenhängen, nötig. Diese Technik beinhaltet dabei auch Sicherheits-Protokolle, um die Sicherheitsinformationen untereinander sicher austauschen zu können. Ein vertrauenswürdiges Agentensystem kann dann zur Auswertung der Traces herangezogen werden.

9. **Selbst Authentifikation** (Self-Authentification)
Es ist möglich eine Authentifikations-Routine zu erstellen, die die Statusinformationen auf offensichtlich inkonsistente oder nicht mögliche Werte untersucht. Die Routine wird dabei mit dem Schlüssel des Owners signiert. Jeder Agentenserver führt diese Routine aus, beendet den Agenten und benachrichtigt bei einem Vorfall das Home-Agentensystem des Agenten [PES 97].

10. **Umgebungsrelevante Schlüsselgenerierung** (Environmental Key Generation)
Ein Agent darf vordefinierte Aktionen ausführen, wenn bestimmte Umgebungsbedingungen zutreffen. Dabei generiert der Agent, falls bestimmte Bedingungen übereinstimmen, einen Schlüssel, mit dem ein verschlüsselter ausführbarer Code entschlüsselt werden kann. Die erwartete Umgebungbedingung ist dabei durch eine Einweg-Hash Funktion bzw. durch eine Puplic Key Verschlüsselung geschützt. Dabei darf kein Agentensystem bzw. kein Bewacher des Agenten die erwartete Umgebungsbedingung erkennen bzw. auslesen können.

    Ähnlichkeit hat diese Technik z.B. mit der Passwortabfrage, wobei das Passwort verschlüsselt vorliegt und verglichen wird. Die Sicherheitslücke besteht aber wieder darin, dass das Agentensystem den Agenten komplett kontrolliert und den Agenten z.B. dazu bringen kann, den Code auszudrucken und nicht auszuführen, nachdem die erwartete Umgebungsbedingung eingetroffen ist.

11. **Rechnen mit verschlüsselten Funktionen (Computing with Encrypted Functions)**
Das Ziel ist eine Methode zu bestimmen, in der Mobiler Code sicher kryptographische Primitive berechnen kann, wie z.B. digitale Signaturen, und das genauso, falls der Code in nicht vertrauenswürdiger Umgebung ausgeführt wird. Zudem soll er autonom und ohne Interaktion mit dem Home-Agentensystem operieren.

    Das Agentensystem, das eine verschlüsselte Funktion enthält, führt ein Programm aus, ohne die Möglichkeit zu haben, die original Funktion zu erkennen. Dieser Ansatz erfordert die Unterscheidung zwischen einer Funktion und dem Programm, das diese Funktion implementiert.

12. **Verwirrender Code** (Obfuscated Code (Time Limited Blackbox [HOH 97], Encrypted algorithms)
Fritz Hohl gibt eine detailierte Übersicht der Bedrohungen, die von einem Agenten stammen, der mit einem bösartigen Agentensystem zusammentrifft, als Motivation zum Einsatz der Blackbox Security.

    Die Strategie ist simpel:
    Zerstückle den Code, so dass es niemandem möglich ist, die komplette Funktion zu verstehen (auch der Daten und der Spezifikationen) oder den resultierenden Code unerkannt modifizieren zu können. Leider ist noch kein Algorithmus bekannt, der diesen Blackbox Schutz möglich macht. Die verschlüsselten Funktionen sind jedoch Beispiele einer Annäherung an die Blackbox Kategorie.

13. **Zertifikate**
Ein anderer Ansatz zum Schutz von Agenten bietet die Zertifizierung vertrauenswürdiger Agentensysteme. Besucht ein Agent nur zertifizierte Agentensysteme, kann von einer sabotagefreien Ausführung ausgegangen werden. Dadurch kann erreicht werden, dass ein Agent nur vertrauenswürdige Agentensysteme besucht.

    Hat ein bösartiges Agentensystem jedoch ein solches Zertifikat erhalten bzw. ist es erst nach diesem Erhalt bösartig geworden, gefährdet dies das ganze System.

14. **Aufsuchen eines vertrauenswürdigen Agentensystems**
Ein Agent sollte kritische Entscheidungen nur auf einem neutralen oder einen absolut vertrauenswürdigen Agentensystem fällen weil das Agentensystem auch bei unverändertem Code die Befehle des Agenten überwachen kann und statt dieser andere ausführen kann.

15. **Migrations Vergangenheit** (Migration History)
Durch die Aufzeichnung der Route und deren Einbettung, die geschützt werden kann, kann man Rerouting-Attacken aufspüren. In Kombination mit digitalen Signaturen kann ein bösartiges Agentensystem keine Komponenten des Agenten entfernen [RMS 96].

16. **Agent als Aufgabenübermittler**
Der Schutz der sensible Informationen eines Agenten erfolgt dadurch, dass ein Agent die zu erfüllenden Aufgaben nur anstößt. Er wirkt sozusagen als Katalysator. Der mobile Agent migriert auf die Agentensysteme und gibt den Auftrag an das Agentensystem weiter. Bei der weiteren Migration trägt er somit keinerlei sensiblen Informationen mit sich. Wurde ein solcher Auftrag durch einen Agenten erteilt, veranlasst das Agentensystem einen mobilen Agenten, welcher die Aufgabe erfüllen kann, zu starten.

    Der Vorteil ist hier, dass der Agentencode vertrauenswürdig ist. Wurde die Aufgabenstellung erfüllt, kann dieser auf das anfragende Agentensystem migrieren und die Ergebnisse kundtun. Da der mobile Agent vertrauenswürdig ist und nur ein One-Hop Agenten darstellt, sind ebenfalls die möglicherweise sensiblen Informationen besser geschützt.

    Vorteile gegenüber der Methode der Verschlüsselung mit dem Privat Key des Agentensystems liegen in dem Erkennen der Fehlerstelle, falls nicht alle Informationen eintreffen. Die sensible Information durchwandert auch nicht unterschiedlichste Agentensysteme und ist somit weniger Angriffen ausgesetzt. Der mobile Agent, der letztendlich die Aufgabe ausführt, ist vom eigenen Agentensystem. Somit sind Angriffe durch modifizierte mobile Agenten anderer Agentensysteme ausgeschlossen. Die Gefahren von Trojanischen Pferden und Back Door Angriffen wird somit ausgeschlossen. Durch den Einsatz eigener, vertrauenswürdiger Agenten ist die Notwendigkeit der Zugriffskontrolle entschärft.

    Nachteilig ist die Notwendigkeit, dass das Agentensystem einen Agenten, welcher die Aufgabe bearbeiten kann, benötigt. Hier sind somit Einbußen in der Flexibilität und Dynamik vorhanden.

    Ein weiterer Nachteil besteht darin, dass das anfragende Agentensystem eine linear zu den befragten Agentensystemen ansteigende Anzahl von ankommenden mobilen Agenten verarbeiten muss. Da diese mobilen Agenten jedoch nur die resultierenden Ergebnisse mit sich tragen, ist dieser Aufwand i.Allg. gering einzuschätzen. Zusätzlich muss auf jedem Agentensystem ein eigener mobiler Agent gestartet werden, und das anfragende Agentensystem bekommt pro Anfrage an ein Agentensystem jeweils einen Agenten als Antwort, wodurch auch hier eine größere Belastung entsteht.

5.2.3 Schutz einer Gruppe von Agentensystemen

Hat ein mobiler Agent ein bestimmtes Ressourcenlimit innerhalb einer bestimmten Domäne, sind in der Regel mehrere Agentensysteme betroffen.

Ein Agent kann sehr viele Ressourcen verbrauchen, indem er z.B. bei vielen Agentensystemen so viele Ressourcen wie möglich beansprucht. Dies kann auch der Fall sein, falls ein Agent unendlich lange lebt und immer weitere Ressourcen auf verschiedenen Agentensystemen beansprucht, oder dass ein Agent neue Agenten erzeugt, die wiederum neue Agenten erzeugen. Hier besteht die

Gefahr eines Denial-of-Service Angriffs. Kann z.B. verhindert werden, dass unzählige Agenten auf dem Agentensystem ausgeführt werden, ist eine große Auslastung eines bösartigen Agenten, welcher auf jedem Agentensystem möglichst viele Agenten startet, nur schwer zu kontrollieren. Ebenso schwer ist die Kontrollmöglichkeit der Benutzung anhand der Ressourcenauslastungen.

5.3 Schutzmechanismen bestehender Agentensystemarchitekturen

Innerhalb der vorgestellten Agentensystemarchitekturen sind zum Teil unterschiedliche Sicherheitsmodelle realisiert worden. Die in Java realisierten Agentensystemarchitekturen signieren zumeist den Agenten und setzten mithilfe der Sicherheitsmechanismen in Java (ClassLoader, SecurityManager) die Zugriffsrestriktionen durch. (Odyssey [GEN 01], Voyager [OBJ 01], Concordia und IBM Aglets [GLO 97]). Das Tacoma Too Projekt experimentiert mit der Technik der Software Fault Isolation und dem Sicherheitsautomat (Security Automata) zur Durchsetzung der Zugriffsrestriktionen. D'Agents realisiert eine benutzerbestimmbare Sicherheitspolitik (DAC), wobei jede Ressource eine Zugriffsliste besitzt, welche erlaubte Aktionen für jeden Owner eines Agenten spezifiziert. Die Agentensystemarchitektur Ara setzt Restriktionen betreffend der CPU Zeit und des Speichergebrauchs durch, unterstützt momentan jedoch noch keinen Schutz des Dateisystems und des Netzwerks und wird nicht näher betrachtet [PES 97].

5.3.1 Aglets

Die Agentensystemarchitektur Aglets von IBM unterstützt Mobile Agenten, implementiert in Java [KOO 98, GLO 97].

ASDK 1.1 beta 1 realisiert eine feine granulare Zugriffskontrolle durch den Einsatz der Java 2 Security Architecture. Jedes Aglet wird durch einen eigenen ClassLoader geladen. Die Transaktion von Bytecode und die Zugriffe auf Ressourcen des Endsystems eines Aglets werden durch den SecurityManager auf die Zulässigkeit hin überprüft. Empfängt ein Aglet Bytecode, wird beispielsweise kontrolliert, ob das Aglet die Berechtigung besitzt, die Klasse zu definieren. Es wird eine benutzerbestimmte Zugriffskontrolle (DAC) implementiert, so dass jedes individuelle Aglet seinen eigenen Schutz gegen Nachrichten anderer Aglets selbst bestimmen kann. Das Rechtekonzept basiert auf Domains, wobei Agentensysteme innerhalb einer Domain als vertrauenswürdig angesehen werden.

Authentifikation:
In Aglets ist eine Serverauthentifikation implementiert. Es können Aglets und Domänen authentifiziert werden.

Alle Server einer Domain teilen sich einen geheimen Schlüssel, mit dem sie sich gegenseitig authentifizieren können. Agentensysteme innerhalb derselben Domain gelten als vertrauenswürdig. Nach erfolgreicher Authentifikation wird ein Credential des Aglets mit samt dem Aglet an das Ziel-Agentensystem gesendet. Ist das Aglet angekommen, so kann anhand des Credentials bestimmt werden, inwieweit man diesem Aglet vertraut.

Migriert ein Aglet auf ein anderes Agentensystem, so wird dem Aglet ein Credential mitgegeben, womit das Ziel-Agentensystem die Vertrauenswürdigkeit des Aglets einstufen kann. So kann anhand des Credentials z.B. durch die Verwendung des geheimen Schlüssels festgestellt werden, ob das Aglet von einem Agentensystem derselben Domain stammt.

Innerhalb einer Domain benötigt jeder User bzw. die Administratoren der Agentensysteme dieser Domain den geheimen Schlüssel. Fällt der geheime Schlüssel in falsche Hände, kann keinem mehr vertraut werden.

Autorisation:
Die Autorisation erfolgt ähnlich nach den Mechanismen der Java 2 Security Architecture. Es werden Permission-Klassen verwendet, wobei auch ein Zugriffschutz der Aglets realisiert wird.

Ein Domain-basierte Policy wird nicht unterstützt. Der Implementierer kann einfach durch die Signatur des Codes ermittelt werden. Ein User wird über Credentials autorisiert, wobei einem Aglets auf Basis des Owners und der Codebase Zugriffsrechte zugeteilt werden.

Durch Einträge in eine Policy-Datenbank oder mithilfe eines GUI-Interfaces können Berechtigungen erteilt werden. Es können unter anderem die in JDK1.2 vorkommenden Berechtigungen (Permissions) spezifiziert werden.

```
java.io.FilePermission             : File read/write/execute
java.net.SocketPermission          : Socket resolve/connect/listen/accept
java.awt.AWTPermission             : showWindowWithoutWarningBanner, accessClipboard
java.util.PropertyPermission       : Java property
java.lang.RuntimePermission        : queuePrintJob, load library
java.security.SecurityPermission   : getPolicy, setSystemScope
java.security.AllPermission        : all other permissions

com.ibm.aglets.security.ContextPermission  : context property, start, shutdown
com.ibm.aglets.security.AgletPermission    : dispatch, deactivate, etc.
com.ibm.aglets.security.MessagePermission  : messaging
```

ContextPermission ist die Berechtigung, auf die Kontext-Properties zugreifen zu können. Mögliche Parameter sind z.B. shutdown (Herunterfahren), start, retract, create.codebas@classname, listener.add, listener.remove oder property.-key- sein.

Die AgletPermission-Klasse repräsentiert den Zugriff auf ein Aglet. Ein AgletPermission besteht aus einem Principal-Name eines Ziel-Aglets und den verschiedenen Operationsnamen für dieses Aglet. Der Principal-Name ist der User-Name des Ziel-Aglets (z.B. Guenter, anonymous, ...), und die Operationsnamen sind die Methodennamen der Aglet-Klasse (deactivate, activate, clone, retract, dispatch, dispose).

Die MessagePermission-Klasse repräsentiert die Berechtigung, Nachrichten an ein Aglet zu senden. Ein MessagePermission enthält den Principal-Name des Ziel-Aglets und die Art der Nachricht. Beispielnamen sind z.B. message.show und message.getResult.

Rechteüberprüfung:
Die Rechteüberprüfung basiert auf den der Aglets zugeteilten ProtectionDomains und den der ProtectionDomain zugeordneten Zugriffsrechten.

Rechtedurchsetzung:
Die Rechtedurchsetzung erfolgt entsprechend der Java 2 Security Architecture.

Weitere Sicherheitsmerkmale:
Die Kommunikationspfade werden zwischen Servern innerhalb einer Domain durch Integritätschecks gesichert. Der Integritätscheck erfolgt mithilfe des geheimen Schlüssels.

Schutz des Aglets In Aglet wird ein Schutz der Aglets gegenüber anderen Aglets realisiert. Aglets können nicht vor bösartigen Agentensystemen geschützt werden.

- System-Nachrichten
  (Berechtigung nach User-Name des Empfängers der System-Nachricht und Aktion)

- Nachrichten
  (Berechtigung nach User-Name des Empfängers und Aktion)

- der Kontext
  (Berechtigung nach Ziel-Namen und Aktion)
  Die Berechtigung (*, create) erlaubt z.B. die Erstellung (create) eines beliebigen Aglets einer beliebigen Codebase.

5.3.2 Ajanta

Die Agentensystemarchitektur Ajanta der University of Minnesota [UMN 01] unterstützt die Sprache Java.

Ajanta erlaubt durch sein Sicherheitskonzept einen sicheren Zugriff auf Server-Ressourcen. Zudem kann ein Agent einen tamper-proof Container, um Daten unterschiedlicher Places, die verschlüsselt werden können, mit sich tragen. Besitzer eines Agenten besitzen eine sichere entfernte Kontrolle. Ein Agent kann mit Exceptions umgehen und ein sicherer Name-Service unterstützt standortunabhängige Namen.

Zur Realisation der Zugriffskontrolle von Agenten auf Server-Ressourcen und zur Erstellung geschützter Domains wird der ClassLoader und der SecurityManager eingesetzt.

Authentifikation:
Wird ein Mobiler Agent auf einem System ausgeführt, so wird anhand des Credentials der Mobile Agent authentifiziert.

Autorisation:
Der Server verwendet ACLs. Autorisiert werden Agenten durch Credentials, die sie mit sich tragen. Jeder Agent besitzt unterschiedliche Credentials, welche die Agenten Identität mit dem Owner und dem Creator in Verbindung bringt. Das Credential beinhaltet dabei auch das Public Key Zertifikat des Owner. Der Creator kann dem Agenten unterschiedliche aber auch nur begrenzte Privilegien zuteilen. Damit befinden sich also Zugriffsrestriktionen in den Credentials.

Restriktionen können auch vom Erschaffer des Agenten erteilt werden. Hierbei handelt es sich um negative Privilegien. So mag ein Owner eines Agenten z.B. nicht, dass sein Agent einen bestimmten Server aufsucht oder nichts einkaufen darf etc. .

Credentials können mit einer Verfallszeit ausgestattet werden, um den Missbrauch bei einem Diebstahl einzuschränken. Auf Basis einer Erweiterung können Zugriffs-Privilegien selektiv zurückgenommen werden. Desweiteren besteht bei Ajanta die Möglichkeit, Zugriffsrechte zu delegieren.

Rechteüberprüfung:
Der Agenten Server enthält eine Domain Datenbank. In dieser steht für jeden Agenten unter anderem die Thread Gruppe, der Owner, der Creator und die URL der Herkunftsseite. Hier befinden sich auch die Autorisation der Zugriffe, Limitierungen und momentaner Gebrauch unterschiedlicher Ressourcen des Servers. Hat ein Agent momentan Zugriff, so stehen noch zusätzliche Informationen in der Datenbank zur Verfügung. Die Datenbank kann nur von einem Thread, der von der Server ProtectionDomain ausgeführt wird, upgedatet werden.

Rechtedurchsetzung:
Der Server verwendet zur Rechtedurchsetzung einen Proxy Mechanismus. Die Sicherheit des

Proxy-Mechanismus basiert dabei auf dem Java Sicherheitsmodell [KAT 98].

Der Proxy basierte Mechanismus erlaubt einzelnen Ressourcen eine eigene Policy zu implementieren [TNK 99]. Durch den Proxy Mechanismus ist eine Inaktivierung einer oder mehrerer Interface Methoden möglich. Diese Inaktivierung basiert bei Ajanta auf der Grundlage der Security Policy und den Credentials des Client Agenten.

Der Ajanta SecurityManager wird durch einen erweiterten RMI Security Manager implementiert, welcher die Zugriffe auf Ressourcen schützt. Der SecurityManager verwendet dafür eine ACL um Agenten den Zugriff auf lokale Dateien und Ressourcen zu gewähren. Diese ACLs Definitionen basieren auf URNs und daher auf Basis der Identität des Agenten Owners.

Eine mögliche Erweiterung erlaubt das Accounting des Ressourcenverbrauchs.

Schutz der Agenteninstanz
Ein Agent ist in der Lage, Daten von verschiedenen Hosts in einem manipulationssicheren Logfile, wobei Daten nur jeweils angehängt werden können, zu sammeln. Es ist auch möglich, verschlüsselte Daten für einen bestimmten Host zu einem spezifischen Host weiterzuleiten [KAT 00].

Privilegien und Restriktionen
Die Privilegien und Restriktionen von Agenten werden für folgende Operationen definiert:

- Eine Liste von Servern, die der Agent besuchen darf
- Wie oft ein Agent einen Server besuchen darf (Schutz vor Replay Attacken)
- Auf welche Ressourcen der Agent bei jedem einzelnen Server zugreifen darf und welche Methoden er aufrufen darf.
- Eine Time-to-Live Periode, die die maximale Zeit spezifiziert, wie lange ein Agent auf einem Server ausgeführt werden darf.
- Namen von entfernten Hosts, mit welchen der Agent eine Netzwerkverbindung von dem jeweiligen Server aus aufbauen darf.
- Namen der Agenten, mit welchen der Agent eine RMI basierte Kommunikation von dem jeweiligen Server aus aufbauen darf.
- der Namen des Agenten, welcher die Erlaubnis besitzt, die Methoden des Agenten mithilfe von RMI aufzurufen .

Privilegien und die Delegation von Privilegien
Der Erzeuger eines Agenten bei Ajanta vergibt Agenten Privilegien und Agenten Restriktionen. Die folgende Abbildung zeigt die Verbindung der Privilegien und Restriktionen mit den Credentials des Agenten.
Hier sind nicht alle Felder des Credentials dargestellt. Bevor ein Agent fertiggestellt wird, wird er noch vom Owner des Agenten signiert. Jede Manipulation des Objekts, also auch der Privilegien und der Restriktionen, kann erkannt werden, indem man die Hash Werte und die Signatur des Credentials Objekts verifiziert.

Die Eingaben von Privilegien für einen bestimmten Server können mittels dem Public Key des Servers verschlüsselt werden.

Der Mechanismus zur Weitergabe der Privilegien und der Restriktionen von einem Server zu einem anderen sieht folgendermaßen aus:
Der Server “A”, der die Anfrage an einen anderen Server “B” weiterleitet, erstellt ein Ticket zusammen mit dem Objekt, das die Spezifikationen der zusätzlichen Privilegien und Restriktionen

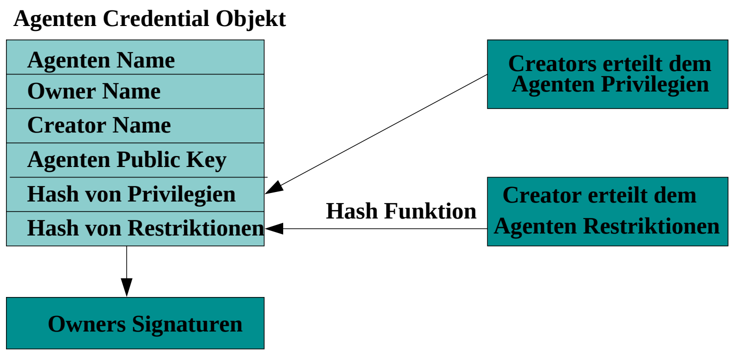


Abbildung 26: Agenten Credentials und Privilegien [KAT 00]

für den Agenten enthält. Mithilfe dieser Privilegien kann der Agent Dienste im Auftrag des Servers "A" ausführen. Das Ticket enthält den Agenten Name, Server Name, Hash Werte der Server "A" weitergegebenen Rechte und Restriktionen, Verfallsdatum und die Signatur des Agenten Owners am Credential-Objekt. Der Server signiert das Ticket. Das Ticket steht in Verbindung mit Agenten, dessen Name und Credentials Signatur im Ticket enthalten sind. Die Verfallszeit stellt sicher, das der Agent diese Privilegien nicht unendlich lange nutzen kann.

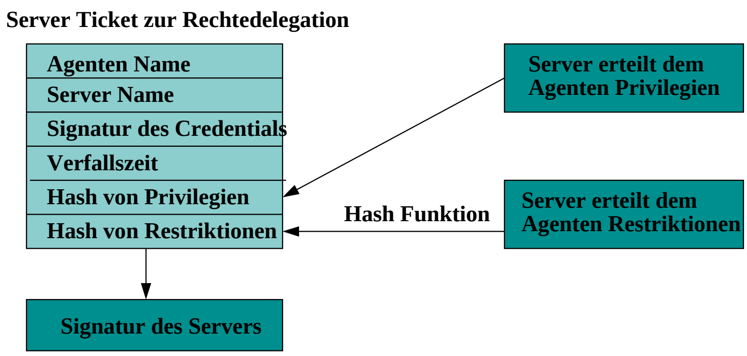


Abbildung 27: Agenten Credentials und Privilegien [KAT 00]

Die Manipulation des Tickets kann erkannt werden. Zuerst wird die Signatur des Tickets verifiziert und dann werden die Hash Werte der vom Server zugewiesenen Privilegien und Restriktionen neu berechnet und mit den Werten des Tickets verifiziert. Am Ende wird noch die Signatur des Owners des Agenten Credentials Objekts verifiziert.

Ablauf der Durchsetzung der Privilegien und Restriktionen

Ein Agent besitzt beim Server eine eigene ProtectionDomain, wobei der Server Informationen über die Agenten Schutz Domain in einer Datenbank schreibt. Die Thread Gruppen ID, die Credentials und Tickets Objekte werden zusammen mit den Privilegien und Restriktionen gespeichert.

Jeder Server hat eine ACL worin die Security Policies definiert sind. Ein Server setzt Agenten Privilegien und Restriktionen durch den Kontext seiner eigenen Security Policies durch, welche als Dominant betrachtet werden. Alle sicherheitssensitiven Operationen auf System Ressourcen rufen den Ajanta SecurityManager des Servers auf. Der SecurityManager entscheidet anhand der Security Policies des Servers, ob der Zugriff erlaubt ist. Ist er erlaubt, werden die Privilegien und Restriktionen, die in der Datenbank gespeichert sind, überprüft. Die Operation wird, falls keine

Security Exception auftrat, erlaubt.

Fragt ein Agent nach einer Server Ressource mithilfe der getResource Primitive des Servers, so überprüft der Server die Privilegien und Restriktionen des Agenten um zu sehen, ob der Zugriff erlaubt ist. Falls ja, geht ein Aufruf mit dem Credential des Agenten an das Ressourcen Objekt. Das Ressourcen Objekt erstellte ein passendes Proxy, das an den Agenten zurückgegeben wird.

Das Problem der Rechtedelegation ist das Erstellen von signierten Credentials für ein Child-Agenten, das von einem Agenten auf einem entfernten Knoten erschaffen wurde. In Ajanta ist es einem Agenten erlaubt, Child-Agenten zu erzeugen, jedoch sind die neuen Agenten Credentials von dem jeweiligen Server signiert, wo das Child-Agent erschaffen wurde. Solche Agenten werden nur mit hoch limitierten Privilegien ausgeführt.

Proxy Objekte werden beispielsweise in Ajanta eingesetzt, um Nachteile bei der alleinigen Verwendung des SecurityManagers zu unterbinden. Nur der Agent besitzt eine Referenz auf dieses Proxy Objekt [SHA 86].

In Ajanta wurde ein solches Modell wie folgt implementiert. Dabei hält sich die Beschreibung an den Bericht von A. Tripathi und N. Karnik [KAT 98]. Das folgend abgebildete Skeleton kommt zum Einsatz.

Eine generische Ressource:

```
public interface Resource {
  // generic methods, common to all resources
  // e.q. queries for name/id, ownership, etc.
}

public class ResourceImpl implements Resource {
  // implementations of the above methods
}
```

Das System definiert also ein Ressourcen Interface und unterstützt eine ResourceImpl-Klasse, welches das Interface implementiert, wobei die Methoden dieser Klasse die generellen Funktionalitäten für alle Ressourcen unterstützt.

Das Resource Access Protocol, welches die Mobile Agenten an die lokalen Ressourcen bindet, ist definiert in terms generic Resource Objekte, um es auch unabhängig von Applikations definierten Typen beizubehalten. Alle Applikations definierten Ressourcen-Klassen müssen das Resource Interface implementieren, wobei das typischerweise durch Vererbung von der ResourceImpl-Klasse gemacht wird.

Als Beispiel kann eine Applikation eine Buffer Resource Interface unterstützen, welche durch eine BufferImpl-Klasse wie folgt implementiert wurde.

Eine gebundene Buffer Ressource:

```
public interface Buffer extends Resource {
  // An application-defined bounded buffer

  public synchronized BufItem get();
  public synchronized void put (BufItem);

  //etc.
```

```
}

public class BufferImpl extends ResourceImpl
  implements Buffer, AccessProtocol{

  // implementation of the Buffer and
  // AccessProtocol methods
}
```

Eine Proxy-Klasse der gebundenen Buffer Ressource:

```
public class BufferProxy implements Buffer {

  private Buffer ref;
  // ref is a reference to the underlying
  // resource, wich is initialized by the
  // constructor (below)

  BufferProxy(Buffer b) {ref = b;}

  public synchronized BufItem get() {
    if (isEnabled("get"))
       return ref.get();
    else
       // throw a security exception
  }

// etc.

private boolean isEnabled(String method) {
  // if the named method is disabled,
  // it throws a security exception.
  }
}
```

Agenten bekommen also nicht die Referenz zu der Ressource, stattdessen bekommen sie eine Instanz einer Proxy-Klasse geliefert. Dafür muss für jede Applikations definierte Ressource-Klasse eine entsprechende Proxy-Klasse definiert werden.

Ein Proxy enthält eine Referenz zu der aktuellen Ressource, welche nicht außerhalb des Proxy sichtbar sein sollte. Die Proxy-Klasse implementiert das Interface, das die Ressource präsentiert. Jede Interface Methode reicht den Aufruf zu dem Ressource Objekt weiter, nachdem festgestellt wurde, dass die Ressource freigegeben ist.

Die Autorisation wird von der Ressource-Klasse realisiert, welche das AccessProtocol Interface implementiert

Das Access Protocol Interface:

```
public interface AccessProtocol {
  // Defines the generic resource access
  // interface. The getProxy method returns
  // a proxy object (typecasted to Resource).
```

```
  public Resource getProxy();
}
```

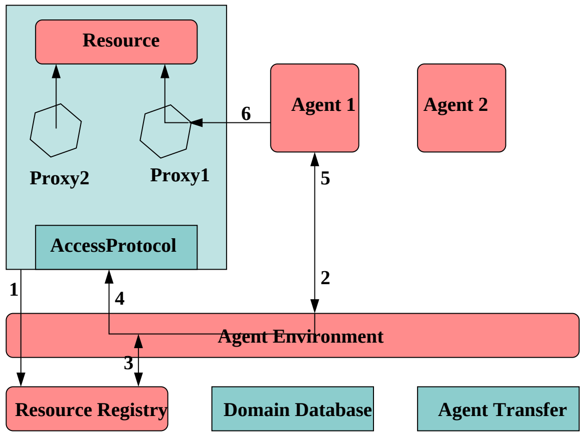


Abbildung 28: Dynamisches binden einer Ressource [KAT 98]

1. die Ressource registriert sich selbst
2. der Agent stellt eine Anfrage an eine Ressource
3. der Server schaut in der Registry nach der Ressource
4. die `getProxy`-Methode wird aufgerufen
5. der Agent bekommt das Proxy Objekt geliefert
6. der Agent greift über das Proxy Objekt auf die Ressource zu

Die Abb. 28 zeigt weiter zwei Agenten, die jeweils eine eigene geschützte Umgebung besitzen. Ressourcen werden den Agenten durch den Aufruf der jeweiligen registerResource Primitive der Agenten, welche die Ressourcen Namen und die Referenz zu dem Ressource Objekt in der Ressource Registry speichert, zugreifbar gemacht. Dabei enthält jeder Eintrag Besitzerinformationen (Owner-Informationen), welche genutzt werden, um unautorisierte Modifikationen der Registry Einträge zu unterbinden.

Um einen Zugriff auf eine Ressource zu bekommen, muss ein Agent die getResource Primitive aufrufen und einen globalen Namen der gewünschten Ressource bereitstellen. Daraufhin sucht die Agentenumgebung die Resource Registry auf, um dass entsprechende Resource Objekt zu lokalisieren, um dann die `getProxy`-Methode aufrufen zu können (Schritte 1-4). Die `getProxy`-Methode erhält falls nötig, die Agenten Credentials durch Anfragen der Server-Domain Datenbank. Ist nach der Sicherheits Policy der Aufruf zulässig, so kreiert das Resource Objekt einen zu dem Agenten zugeordneten Proxy, welches dem Agenten durch die Agentenumgebung übertragen wird. Nun kann der Agent, durch das Proxy Objekt, freigegebene Methoden des Resource Interfaces aufrufen. (Schritte 5 und 6)

5.3.3 Bond

Die Agentensystemarchitektur Bond des Computer Sciences Department der Purdue Universität unterstützt die Sprachen Java, Python und Jess [CSD 00, JHM 99].

Authentifikation:
Bond Objekte können authentifiziert werden.

Autorisation:
Der Zugriffskontrollmechanismus basiert auf Zugriffsrechten, die einem Objekt mittels einem Bond Ticket gewährt werden und die der Bond Security Agent durchsetzt. Das Bond Ticket wird bei der Erstellung des Bond Shadows generiert und signiert.

Ein Bond Ticket definiert die Zugriffsrechte als Möglichkeit, KQML-Nachrichten mit spezifischen Sub-Protokollen (PropertyAccess, AgentControl), Performative (ask-one, achieve), Content (get, set, start-agent) und Parameter (alpha, WorkSpace) senden zu können. Die Zugriffsrechte auf Dienste betreffen die Leistung, das spezifische Sub-Protokoll, Restriktionen der Inhalte der Leistung, und Restriktionen der Parameter.

Rechteüberprüfung:
Überprüft werden die in der Policy erfolgten Autorisationen der Bond Objekte.

Rechtedurchsetzung:
Ein Objekt kann seinen eigenen Security Agenten erstellen um seine eigene Security Policy durchzusetzen. Eine Gruppe von Objekten kann aber auch einen Security Agenten gemeinsam benutzen.

Ein Security Agent hat die Aufgabe, die Berechtigung der zugreifenden Subjekte zu überprüfen. Nur ein Secure Bond Objekt kann Subjekte authentifizieren und die Zugriffskontrolle durchsetzen. Der Security Agent ist fähig, Bond Tickets für ein oder mehrere Objekte zu generieren und zu speichern und die Zugriffskontrolle durchzusetzen. Der Security Agent stellt dabei einen Proxy auf der Empfängerseite dar.

Kommunikationsablauf:
Vor der Kommunikation erstellt der Initiator ein Bond Shadow, welches aus Informationen des Targets erstellt werden kann und unidirektionale Kommunikation unterstützt. Die Kommunikation erfolgt dann immer über dieses Shadow bis hin zum Target. Das Shadow ist sozusagen ein lokaler Proxy für das entfernte Objekt. Der Kommunikationablauf sieht dann folgendermaßen aus:

- Sende Nachricht über das Shadow an das Target-Objekt
  Es werden Sicherheitsüberprüfungen und Formatverwandlungen vorgenommen. Die Sicherheitsüberprüfung basiert dabei auf ausgetauschten Tickets, die erlaubte Zugriffsmöglichkeiten beinhalten. Dies verhindert schon das Versenden von Nachrichten, die beim Target aufgrund fehlender Rechte gar nicht verarbeitet werden.
- Transport der Nachricht
- Die Nachricht durchläuft einen Security Agenten, wobei auch hier Sicherheitsprüfungen vorgenommen werden (Zugriffskontrolle, ...).
- Die Nachricht wird beim Empfänger einer Syntaxanalyse unterzogen und an das Ziel-Objekt übergeben.

Jedes Bond Objekt unterstützt eine Menge an Diensten, welche für die anderen anhand von Message Patterns, in Bond sub-protocols nutzbar sind. So kann ein Objekt seine Properties offenlegen und anderen Objekten erlauben, diese Properties anhand dem *property access sub-protocol* zu modifizieren.

Um die Zugriffskontrolle zu realisieren, werden für jeden KQML-Nachrichtentyp die Rechte definiert und überprüft. Jede Nachricht besteht dabei aus einem sogenannten Performative Feld, einem Content-Feld und einem oder mehreren Attribut-Feldern.

Um das Zugriffsmodell zu implementieren, muss das Bond Shadow Objekt um das Zugriffkontroll-Objekt (Bond Ticket) und dem Public Key des Ziel Objekts erweitert werden. Das Bond Ticket des Shadows ist dabei eine Kopie des Bond Tickets des Ziel Objekts.

Der Shadow benötigt eine Kopie des Bond Tickets, welches von dem Ziel-Objekt gewährt wurde, um die Nachrichten filtern zu können. Es werden nur Nachrichten an das Ziel-Objekt weitergeleitet, welche mit den Zugriffsrechten des Bond Tickets übereinstimmen.

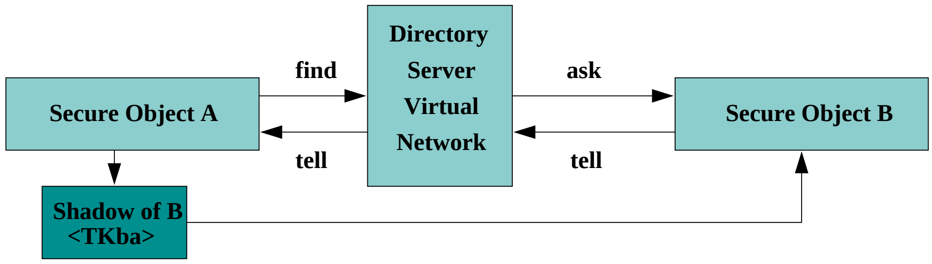


Abbildung 29: Zugriffskontrolle bei Bond [JHM 99]

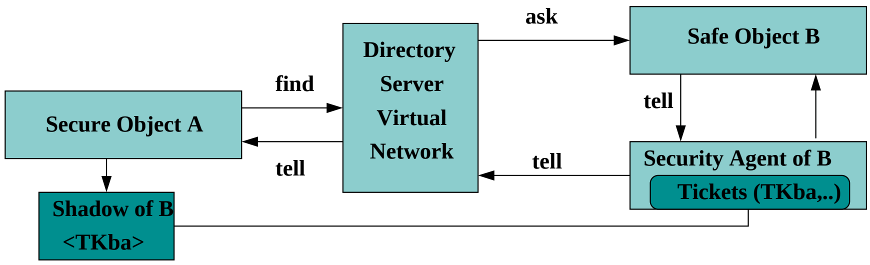


Abbildung 30: Zugriffskontrolle für ein Safe Bond Objekt [JHM 99]

Die Shadows auf der Client und der Server Seite implementieren die Zugriffskontrolle:

Ein Bond SecurityContext beinhaltet sicherheitsrelevante Objekte, die unterschiedliche Sicherheitsfunktionen implementieren. Jedes sicherheitsrelevante Objekt implementiert eine von Bond definierte Sicherheitsschnittstelle. Folgende Schnittstellen sind enthalten:

- bondCredentialInterface
  Diese Schnittstelle definiert die Methode, auf das Credential zuzugreifen, welches im aktuellen bondSecurityContext enthalten ist. Ein Credential kann ein User/Passwort, Zugriffs-Ticket oder ein Public/Privat Schlüsselpaar sein.

- bondAuthenticatorInterface
  Die bondAuthenticatorInterface Schnittstelle definiert die Methode, die verwendet wird, um jede empfangene Nachricht zu authentifizieren. Der Programmierer oder der Administrator kann verschiedene Authentifikationsmechanismen definieren. Beispielmechanismen sind Benutzername/Passwort basierte Authentifikation, Ticket basierte Authentifikation oder zertifikatbasierte Authentifikation. Die einzige Methode dieser Schnittstelle ist die `authenticatClient()`-Methode, welche die authentifizierte Benutzer Identität als Rückgabewert liefert.

- bondAccessControlInterface
  Diese Schnittstelle definiert die Methoden, welche die Zugriffskontrolle durchsetzen. Der Pro-

grammierer oder Administrator kann verschiedene Methoden definieren. So z.B. Firewalls, basierend auf IP-Adressen oder Access Control Lists (ACL) basierend auf den Benutzernamen. Die Schnittstelle stellt die `initACL()`-Methode zur Initialisierung des Zugriffskontroll-Objekts und die `checkRigth()`-Methode, um die Zugriffsrechte einer authentifizierten Identität zu überprüfen, zur Verfügung.

Jedes Bond Objekt besitzt die Möglichkeit, seine eigene Security Policy auszuwählen. Interfaces für Sicherheitsüberprüfungen müssen jedoch einheitlich sein. Es wurden für alle Sicherheitsfunktionen abstrakte Java Klassen definiert. Dazu gehören die Sicherheitsüberprüfungen und die Ticket Generierung. Jedes Objekt hat die Wahlmöglichkeiten, diese abstrakten Security-Klassen zu implementieren. Es gibt default Security-Klassen, falls man keine eigenen implementieren will.

5.3.4 D‘Agents

Die Agentensystemarchitektur D‘Agents des Dartmouth College [DAM 01] unterstützt Mobile Agenten implementiert in Agent Tcl, Agent Java und Agent Scheme.
Es wird eine benutzerbestimmbare Zugriffskontrolle (DAC) implementiert, wobei jede Ressource eine Zugriffsliste besitzt, welche die erlaubten Aktionen eines jeden Owners, das ist der Anwender des Agenten, spezifiziert. Dadurch können Zugriffe auf das Agentensystem beschränkt werden. Sobald ein Agent einen nicht vertrauenswürdigen Rechner besucht, wird er als anonymer Agent angesehen. Ihm kann nicht mehr vertraut werden. Ein aktiver Schutz des Agenten ist also nicht vorhanden.

Authentifikation:
Jeder D‘Agent Server unterscheidet zwischen anonymen Agenten und Agenten, denen ein Anwender zugeordnet werden kann. Der Owner des Agenten muss hierfür authentifiziert werden, wobei sich der authentifizierte Anwender als autorisierter Nutzer in einer Liste des Servers aufgeführt werden muss.

Autorisation:
Ist die Identität des Owners sichergestellt, werden dem Agenten die betreffenden Zugriffsrestriktionen erteilt (Autorisation). Die Autorisation erfolgt durch konfigurierbare Policies, unterstützt durch ein Sprachenunabhängiges Policymodul für jede System Ressource. Ein erlaubter Zugriff kann zusätzlich limitiert werden.
Die meisten Ressourcen-Manager sind mit einfachen Security Policies implementiert. Jeder Ressourcen-Manager hat eine Konfigurationsdatei, welche die Zugriffsrechte für einen einzelnen Nutzer spezifiziert. Der Manager initialisiert diese Zugriffsliste beim Start und überprüft den Zugriff auf Basis des Owners jeder Anfrage mit den Einträgen in der Liste. Der Manager weiß, ob der Owner authentifiziert werden konnte, bzw. ob der Agent auf derselben Maschine ausgeführt wird.

Rechteüberprüfung:
Jede Ressource erhält einen eigenen Ressourcen-Manager, der kontrolliert, ob der Zugriff erlaubt ist oder nicht. Beispielsweise kann der CPU Ressourcen-Manager entscheiden, ob der Agent die CPU nutzen darf und wenn, dann wie lange bzw. mit welcher Priorität [LWO 97].

Rechtedurchsetzung:
Das System überwacht die Einhaltung der Zugriffsrestriktionen (Enforcement). Ein sprachabhängiges Rechtedurchsetzungsmodul sendet Anfragen auf Ressourcen zu einem stationären Agenten, den Ressourcen-Manager, der die Sicherheitspolicy für diese Ressource bestimmt. Jede Sprache besitzt ihr eigenes Durchsetzungs Modul. Einige Entscheidungen bzgl. Ressourcen werden auch durch den Code des Agenten durchgesetzt.
Zugriffsrestriktionen werden mithilfe von Safe Tcl, dem SecurityManager oder dem Scheme 48 Modul durchgesetzt.

Die Rechtedurchsetzung übernimmt die Safe-TCL Infrastruktur. Der Safe-Tcl Kernel Interpreter fängt dabei sensitive Prozeduraufrufe ab. Beim ersten Zugriffsversuch wird der Ressourcen-Manager nach der Zugriffsberechtigung befragt. Dabei wird die Antwort für erneute Anfragen zwischengespeichert. Diese Architektur trennt dabei klar den Mechanismus (Safe-Tcl) von der Policy (Ressourcen-Manager), was unterschiedliche Policies zulässt. Der Ressourcen-Manager ist somit aber auch unabhängig von der Programmiersprache.

Java Durchsetzungsmodul
Explizit soll nun das Durchsetzungsmodul in D'Agents für Java betrachtet werden. Dieses Modul ist durch den Java SecurityManager implementiert. Die Java SecurityManager-Klasse beinhaltet einige Zugriffskontrollmethoden. Zum Beispiel checkExec, checkRead und checkExit. Die Java System Klassen rufen diese Methoden auf, um prüfen zu können, ob diese Operationen erlaubt sind. Jede Check-Methode überprüft dabei die interne Zugriffsliste und kontaktiert, falls nötig den Ressourcen-Manager.

- **Decision caching**
  Der Performance wegen können Entscheidungen zwischengespeichert werden. Schließlich muss nicht bei jedem gelesenen Zeichen einer Datei die Anfrage erneut überprüft werden. Die Entscheidungen des Ressourcen-Manager werden in einer internen Zugriffsliste gespeichert.

- **Require**
  Ein Agent kann den Ressourcen-Manager explizit nach einer Zugriffsberechtigung fragen. Dies geschieht mittels dem require-Kommando, welches den Namen des Ressourcen-Managers (z.B. Dateisystem) enthält und eine Liste mit paarweisen Einträgen, die die benötigten Zugriffsrechte enthält (z.B. /home/datei.txt, read).

- **restrict**
  Ein Agent kann sich selbst Zugriffsrestriktionen erteilen. Dies geschieht mit dem restrict-Kommando. Bei den Consumable-Ressourcen werden diese Restriktionen auch bei anderen Maschinen (Domänenweit) durchgesetzt.

Ressourcen:
Ressourcen werden in zwei Typen eingeteilt:

- Indirekte Ressourcen
  Auf Indirekte Ressourcen kann nur durch einen anderen Agenten zugegriffen werden. Der Agent, der die Ressource kontrolliert setzt, dabei seine eigenen Zugriffsrestriktionen durch. Der Agent überprüft jede Anfrage anhand einer Zugriffsliste (ACL). Es können auch Capabilities eingesetzt werden. Capabilities werden nur an Agenten, die sich auf demselben Rechner wie die Ressource befinden überreicht. Die Capability enthält dabei genügend Identifikations Informationen, wobei die Capabilities ungültig werden, sobald das Agentensystem verlassen wird. Realisierbar wird dies durch einen Zeitstempel, den jeder Agent bei der Ankunft bekommt.

- Builtin Ressourcen
  Builtin Ressourcen sind direkt durch eine Library bzw. einem Sprachen Primitiv zugreifbar. Dies ist z.B. der Speicher, das Dateisystem, die CPU Zeit usw. Der Server setzt die Zugriffspolicies durch. Limitierungen und Zugriffsrestriktionen sind in einer Server Konfigurationsdatei spezifiziert. Für alle anderen Ressourcen wird die Sicherheit durch den Ressourcen-Manager geregelt. Durch die Trennung des Durchsetzungsmechanismus und der Policy kann der System Administrator die Sicherheits-Policy leicht ändern, indem er eine andere Ressourcen-Manager Implementierung einsetzt.

Momentan existieren 6 Ressourcen-Manager:

1. Consumables
   Dieser Ressourcen-Manager ist für konsumierbare Ressourcen zuständig (CPU Zeit, Anzahl der Agenten Kinder, Anzahle der Migrationen, ...). Hier spielt die Zugriffskontrolle keine Rolle, sondern nur das Kontingent. Der Agent bekommt ein kleines Kontingent, worauf er, falls er mehr benötigt, erneut Ansprüche stellen muss.

2. File System
   Lese- und Schreibrechte auf Dateien und Directories werden kontrolliert. Eine Maximalanzahl von schreibbaren Dateien schützt vor einer Auslastung des Dateisystems. Der Dateisystemmanager sollte dabei auch einen Schutz des Dateisystems bieten, indem auch die zulässigen Zugriffe auf das Dateisystem eingeschränkt werden können.

3. Libraries
   Dieser Manager entscheidet, welche Libraries der Tcl Funktionen, Scheme Funktionen oder Java Klassen ein Agent laden kann.

4. Programs
   Der Programm-Manager kann entscheiden, welche externen Programme ausgeführt werden dürfen. Dies ist nötig, da externe Programme nicht Subjekt derselben Sicherheitsüberprüfungen sind.

5. Network
   Es werden Entscheidungen auf direktem Zugriff auf TCP/IP und UDP Netzwerk Services getroffen. Es kann z.B. ein kompletter Zugriff bzw. überhaupt kein Zugriff erlaubt werden. Erweiterungen zur Unterscheidung der Hosts, der Domains, Ports oder Protokolle sind geplant. Die Zugriffe pro Sekunde können hier ebenso nicht kontrolliert bzw. eingeschränkt werden.

6. Screen
   Der Zugriff kann erlaubt oder nicht erlaubt werden. Feinere Kontrollen könnten den Zugriff erlauben, jedoch das Aufzeichnen des gesamten Bildschirminhaltes verhindern. Desweiteren könnte z.B. noch die Fensteranzahl und die Fenstergröße eingeschränkt werden. Wünschenswert wäre eine Einstellungsmöglichkeit mittels einer Policy durch den Nutzer.

Schutz einer Gruppe von Agentensystemen:
Entweder befinden sich die Rechner unter einer administrativen Kontrolle (LAN einer Organisation) oder nicht (Internet). Die Maschinen innerhalb einer administrativen Kontrolle vertrauen sich in der Regel, misstrauen jedoch den meisten Maschinen im Internet.

Innerhalb einer Umgebung bei denen man von vertrauenswürdigen Rechnern ausgeht, bekommt ein Agent eine maximale Menge an Ressourcen zugeteilt, solange er sich in dieser Umgebung befindet. D'Agents unterstützt dies durch das Einbinden eines "usage vector" in den Agenten, der den maximalen und den momentanen Ressourcenverbrauch auflistet.

Internetweit ist eine Ressourcenlimitierung jedoch nicht einsetzbar. Momentan wird an einer Market-based Methode gearbeitet, bei dem ein Agent für seinen Ressourcenverbrauch bezahlen muss [BRK 98].
Es existieren Beispielimplementierungen eines Währungs-Modells, eines Banking Systems und eine Anzahl von währungsabhängigen Ressourcen-Managern. Dabei erlaubt diese Infrastruktur den Ressourcen-Managern, Preise für Ressourcen zu setzen und Agenten dynamisch an die Ressource-Preis Umgebung anzubinden. [BRK 98]. Aktivitäten von Mobilen Agenten, die elektronische Zahlungsverfahren, ein Banking System und eine Anzahl von Ressource Managern nutzen, können somit kontrolliert werden.

5.3.5 EC‘s Trust Manager for Java

ECTM/J implementiert eine benutzerbestimmte Zugriffskontrolle (DAC). Um den Schutz der einzelnen Klassen zu realisieren, wird ein eigener ClassLoader eingesetzt. Der User kann über eine Policy Datei, das Trust Boot File (TBF), Zugriffe autorisieren [ECO 00].

Der Ablauf gestaltet sich wie folgt (vgl. Abb. 31):

1. Soll eine neue Klasse geladen werden, so ruft der ClassLoader den Trust Manager auf.
2. Der Trust Manager befragt die Policy in der TBF, um zu sehen, welche Klassen geschützte Klassen sind.
3. Beinhaltet die neu zu ladende Klasse nun Referenzen zu eingeschränkten Klassen, ruft der Trust Manager die Zertifikat-Datenbank auf. Dort finden sich die Policy Äußerungen, die gegebenenfalls der neuen Klasse das Recht zur Importierung der eingeschränkten Klasse erteilt. Diese Zertifikate müssen auch bei den Policy Äußerungen in der TBF vorkommen, oder durch andere Zertifikate, die wiederum überprüft werden müssen, autorisiert werden.
4. Wurde der Klasse die von ihr geforderte Mächtigkeit gestattet, kann sie geladen und gestartet werden.

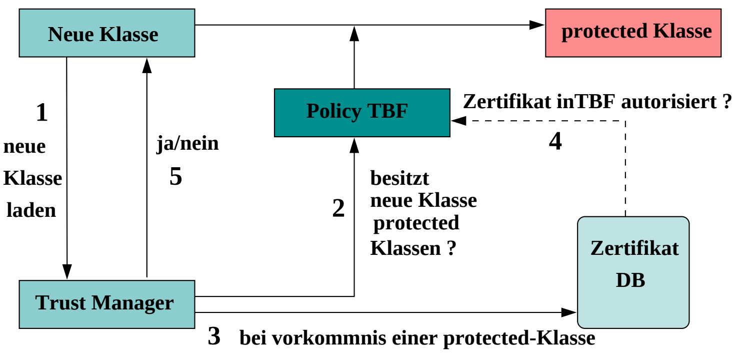


Abbildung 31: EC‘s Trust Manager [ECO 00]

Die Policy wird in einer Datei TBF (Trust Boot File) auf der Festplatte des Nutzers abgelegt. Sie beinhaltet Zertifikate oder die Public Keys. Die Signaturen innerhalb des Trust Boot File repräsentieren Parteien, denen unterschiedliche Rechte weitergegeben werden können. Zertifikate können auch mithilfe von URLs angegeben werden, die den Ort des Zertifikats definieren. Die Zertifikate können dann mithilfe der entsprechenden Public Keys verifiziert werden. Rechte können durch den Einsatz von Zertifikatketten delegiert werden.

Vorteile:

- Die Sicherheit wird vor dem Laden der Klassen überprüft.
- Es sind nur minimale Modifikationen des ClassLoaders nötig. So muss nur die Zertifikatüberprüfung hinzugebunden werden.
- Die Herkunft der Programme spielt keine Rolle mehr. Die Sicherheit basiert allein auf den signierten Zertifikaten.
- Eine dezentrale Überprüfung wird unterstützt. Der Implementierer muss sich nicht mehr um Aufrufe des SecurityManagers kümmern, bzw. Klassen als geschützt (protected) deklarieren. Soll eine Klasse eingeschränkt werden, so muss der Implementierer sie nicht umschreiben.

- Die Klassen können über unsicherer Kanäle versendet werden, da Authentisierungmöglichkeiten existieren.

5.3.6 Grasshopper

Die Agentensystemarchitektur der IKV++ GmbH [GRA 00] unterstützt die Sprache Java

Bei Grasshopper wird zwischen externer Sicherheit und interner Sicherheit unterschieden. Externe Sicherheit beschreibt die Kommunikation über die Systemgrenzen hinweg, entfernte Interaktionen und die Migration eines Agenten. Interne Sicherheit bei Grasshopper bezieht sich auf den Schutz der Schnittstellen der Agenten, Agentensysteme und der Ressourcen auf einem Agentensystem vor unautorisierten Zugriffen durch Agenten.

- Externe Sicherheit
  Schützt alle entfernten Interaktionen. Hierfür kommen X.509 Zertifikate und Secure Socket Layer (SSL) zum Einsatz.
- Interne Sicherheit
  Interface der Agenten und Agentensysteme, auch Agentensystem Ressourcen (Dateisystem, ..) werden vor unautorisierten Zugriffen mittels Agenten geschützt. Die Zugriffskontrolle leistet eine Authentifikation und Autorisation des Nutzers des zugreifenden Agenten. Hier werden Zugriffskontrollpolicies aktiv. Der Schutz basiert auf den Sicherheitsmechanismen von Java.

Um ein Sicherheitsprotokoll für die entfernten Interaktionen zu nutzen, muss der Agent bei der Erstellung eines Server-Proxies einen Kommunikations-Empfänger spezifizieren:

```
serverProxy =
(IAsyncServerAgent)ProxyGenerator.newInstance
(IAsyncServerAgent.class, serverID, "socketssl://myHost:7000/myAgency");
```

Für das Sicherheitsprotokoll der Migration muss ebenfalls ein Kommunikations-Empfänger als Parameter der `move()`-Methode spezifiziert werden:

```
move(new GrasshopperAddress(
"socketssl://myHost:7000/myAgency"));
```

Authentifikation:
Der Agent wird durch seine Owner authentifiziert. Die Authentifikation wird mittels Signaturen realisiert.

Autorisation:
Durch die Authentisierung wird die Autorisation mittels einer Policy ermöglicht. Der Agentensystem- Administrator kann diese Policy konfigurieren.

Rechteüberprüfung:
Grasshopper verwendet den SecurityManager zur Rechteüberprüfung. Dabei werden folgende Berechtigungen überprüft:

- akzeptiere Verbindung von einem spezifischen Host
- akzeptiere spezifischen Thread oder Threadgruppe
- akzeptiere AWT Event Queue
- verbinde mit einem spezifischen Host / Port
- verbinde mit einem spezifischen Objekt auf einem spezifischen Host / Port

- Erstelle neuen ClassLoader
- Lösche spezifische Datei
- Führe ein spezifisches Kommando aus
- Beende die Virtuelle Maschine
- Linke eine native Bibliothek
- Höre bestimmten Port ab
- Zugriffsmitglieder einer spezifischen Klasse
- Nutze Multicast Kommunikation
- Zugriff auf ein spezifisches Package
- Definition von Klassen in einem spezifischen Package
- Start eines Drucker-Jobs
- Zugriff auf Systemeigenschaften
- Zugriff auf eine spezifische Systemeigenschaft
- Lesen von einem Dateidescriptor
- Lesen von einer spezifischen Datei
- Definition des Sicherheitssystems und Auslösen der spezifischen Aktion
- Setzen einer Socket oder Stream Handler Factory
- Zugriff auf das System Clipboard
- Top-Level Window
- Schreiben in ein Dateidescriptor
- Schreiben in eine spezifische Datei

5.3.7 TuX (Tacoma UniX)

Die Agentensystemarchitektur der Universität von Tromso [TUX 01] unterstützt momentan die Sprachen C und Binaries. Jedoch ist das System leicht auf weitere Sprachen zu erweitern. TuX 1.2 unterstützt Tcl, C Phyton und Scheme.

Das Agentensystem Tacoma implementiert ein Replication and Voting-Mechanismus. Durch den Security Automata bei Tacoma Too hängen die momentan erlaubten Aktionen von der History des Agenten ab.

Autorisation:
Die Autorisation erfolgt über konfigurierbare Access Control Lists (ACLs)

Rechteüberprüfung und Rechtedurchsetzung:
Das Tacoma Too Projekt experimentiert mit der Technik der Software Fault Isolation and Security Automata zur Durchsetzung der Zugriffsrestriktionen. Ein Security Automata ist eine Zustandsmaschine, wobei jeder Übergang einen erlaubten Zugriff auf eine Ressource entspricht. Tacoma enthält eine Rear-Guard-Agenten, der verschwundene Agenten neu startet.

6 Rechtekonzept für MASA

In diesem Kapitel wird ein Rechtekonzept für MASA entwickelt. Das Rechtekonzept basiert auf der Authentifikation der einzelnen Entitäten durch die in der Arbeit von Harald Rölle [ROE 99] eingeführten Zertifikate und Zertifikathierarchien. Der Schwerpunkt liegt daher auf der Realisation einer geeigneten Rechtevergabe und Rechtedurchsetzung. Das Ziel des Rechtekonzepts ist die Erfüllung der in Kapitel 4 aufgeführten Sicherheitsanforderungen.

Nach einer kurzen Einführung (Kapitel 6.1), wird die für das Rechtekonzept essentielle zentrale Kontrollinstanz (ZKI) und ihre Funktionsweise vorgestellt (Kapitel 6.2). Im Kapitel 6.3 wird die Rechtevergabe und im darauffolgenden Kapitel die Adressierung der einzelnen Entitäten von MASA beschrieben.
Um die Anforderungen einer Agentensystemarchitektur bezüglich der Autorisation zu erfüllen, wird im Kapitel 6.3.2 eine geeignete Politik-Sprache entwickelt. Mit Hilfe der Politik-Sprache kann dann eine anwendungsspezifische Autorisation der Zugriffe erfolgen. Die Autorisation erfolgt dabei anhand den vorgestellten Zugriffskontrollpolitiken.
Um Zugriffsszenarien analysieren zu können, wird im Kapitel 6.4 eine entsprechende Modellierungssprache vorgestellt. Die Modellierungssprache wird im weiteren Verlauf verwendet, um Beispielszenarien vorzustellen und daran die Rechtevergabe zu erläutern. Anhand der Modellierungssprache können einzelne Zugriffe beschrieben werden. Dadurch kann exakt bestimmt werden, welche Zugriffsrechte das jeweilige Subjekt benötigt, um einen spezifischen Zugriff erfolgreich ausführen zu können.
Das anschließende Kapitel 6.5 erläutert die Einsatzmöglichkeiten der Zugriffskontrollpolitiken. Dabei werden die Zuständigkeitsbereiche der einzelnen Politiken und ihre Funktionalität verdeutlicht.
Die im Kapitel 6.6 vorgestellte Verknüpfung der Zugriffskontrollpolitiken ist ein Hauptbestandteil des Rechtekonzepts. Durch die Verknüpfung der Politiken ist sichergestellt, dass alle Zugriffskontrollpolitiken unter der Berücksichtigung der Autonomieanforderungen durchgesetzt werden.
Kapitel 6.8 betrachtet die Rücknahme der Zugriffsrechte.
Die Realisation der in der Anforderungsanalyse geforderten Delegation der Zugriffsrechte der Agenteninstanzen wird im Kapitel 6.9 vorgestellt.
Kapitel 6.10 behandelt die Verwaltungsstruktur und deren Handhabung.
Das Kapitel 6.11 beschäftigt sich mit der Rechtedurchsetzung der Zugriffskontrollpolitiken.
Abschließend werden im Kapitel 6.12 Beispielszenarien vorgestellt.
Im Kapitel 7 wird das Rechtekonzept in die MASA Umgebung eingebettet.

6.1 Einführung in das Rechtekonzept

Das Ziel des Rechtekonzepts ist die Erfüllung der in der Anforderungsanalyse ermittelten Anforderungen.

Durch das Rechtekonzept ist ein Zugriff auf ein Objekt erst dann möglich, wenn das Subjekt zuvor explizit autorisiert worden ist. Damit wird das in Kapitel 3.5 geforderte Fail-Safe Defaults Prinzip erfüllt.

Unterschiedliche Subjekte können über die Zugriffskontrollpolitik des Endsystems, des Agentensystems und der Agenteninstanzen autorisiert werden. Die Zugriffsrechte innerhalb der Zugriffskontrollpolitik einer Agenteninstanz entsprechen Wunsch-Zugriffsrechte, die die jeweilige Agenteninstanz auf einem Agentensystem durchsetzen will. Realisiert wird eine objekt- und subjektspezifische Autorisation der Subjekte innerhalb der Zugriffskontrollpolitiken des Agentensystems und der Agenteninstanz (vgl. Kapitel 2.6.3). Damit kann eine anwendungsspezifische Zugriffskontrollpolitik formuliert werden, wodurch die Autorisation das im Kapitel 3.4 geforderte Least-Privilege Prinzip erfüllt, d.h. jedem Subjekt können nur die Zugriffsrechte eingeräumt werden, die für die Bearbeitung der jeweiligen Aufgabe notwendig sind.

Die vollständige Informationssicherheit aller Objekte garantiert die Durchsetzung der Zugriffskontrollpolitiken der Agentensysteme und die Integrationsmöglichkeit, den von dem Agentensystem akzeptierten Zugriffsrechte der Agenteninstanzen. Die zentrale Kontrollinstanz (ZKI) setzt die Zugriffskontrollpolitik des Agentensystems und die von dem Agentensystem akzeptierten Zugriffsrechte der Zugriffskontrollpolitiken der Agenteninstanzen durch. Die Durchsetzung der Zugriffskontrollpolitik des Endsystems liegt in den Händen des Betriebssystems.
Durch die zentrale Kontrollinstanz (ZKI) werden alle Zugriffsmöglichkeiten, des in Kapitel 4.1 vorgestellten Modells der Agentensystemarchitektur MASA, einer Zugriffskontrolle unterzogen. Damit wird das in der Anforderungsanalyse (Kapitel 3.5) geforderte Complete-Mediation Prinzip erfüllt wird.

Das Rechtekonzept wird durch die Möglichkeit der Aufzeichnung von sicherheitsrelevanten Operationen und Zugriffen vervollständigt. Anhand der Aufzeichnungen ist es möglich, Autorisationsvorgänge und Zugriffe auf Objekte, einzelner Subjekte zuzuordnen. Das Protokollieren der sicherheitsrelevanten Zugriffe dient der Entdeckung und Beweisführung von Manipulationsversuchen.

6.2 Zentrale Kontrollinstanz (ZKI)

Das Rechtekonzept basiert auf einer zentralen Kontrollinstanz (ZKI). Die zentrale Kontrollinstanz (ZKI) kontrolliert jeden Zugriff, der getätigt wird und ist auf jedem Agentensystem vorhanden. Findet ein Zugriff statt, überprüft die Kontrollinstanz (ZKI) des Agentensystems, auf dem sich das Objekt befindet, ob der Zugriff gewährt oder unterbunden wird.
Objekte, auf die zugegriffen werden kann, sind von der Außenwelt abgekapselt und nur über Schnittstellen erreichbar, die durch die zentrale Kontrollinstanz (ZKI) überwacht werden. Dadurch kann auf ein Objekt nur zugegriffen werden, wenn die Kontrollinstanz (ZKI) den Zugriff gewährt. Die Überprüfung basiert auf den in den Zugriffskontrollpolitiken vergebenen und von dem Agentensystem akzeptierten Zugriffsrechten. Die von der zentralen Kontrollinstanz (ZKI) akzeptierten Zugriffsrechte bilden die ZKI-Policy.

Betrachtet man die Objekte der Agenteninstanzen, der Agentensysteme und der Endsysteme (die durch das Agentensystem erreichbar sind), so bieten sie spezifische Zugriffsschnittstellen an, die vollständig von der zentralen Kontrollinstanz (ZKI) erfasst und überprüft werden.

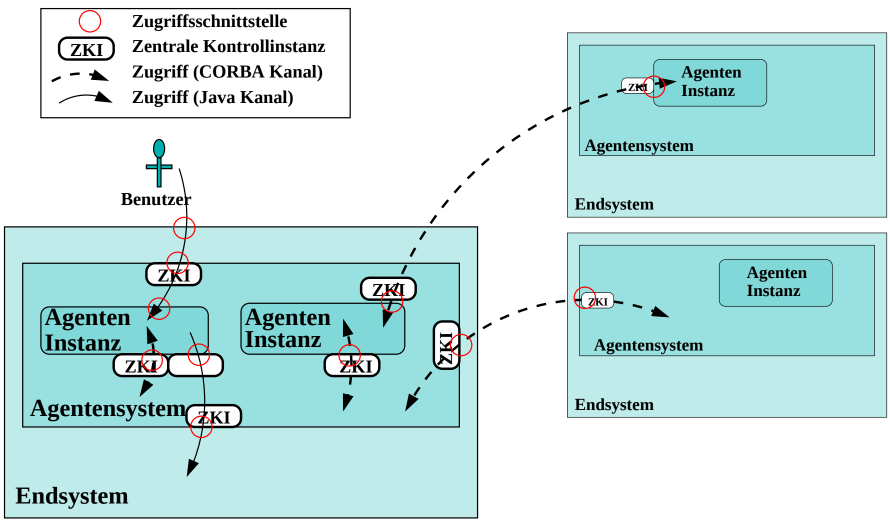


Abbildung 32: Einkapselung der Entitäten

Die Abb. 32 verdeutlicht die Lage der zentralen Kontrollinstanz (ZKI). Die Zugriffsschnittstellen eines Zugriffes sind ebenfalls gekennzeichnet. Die Abb. 32 zeigt weiter, ob eine Zugriffsschnittstelle über einen Java-Kanal oder über einen CORBA-Kanal angesprochen werden kann.

6.3 Rechtevergabe durch Zugriffskontrollpolitiken

Die Autorisation der Zugriffe erfolgt sprachbasiert über Zugriffskontrollpolitiken der Agentensysteme und über die Wunsch-Zugriffskontrollpolitiken der Agenteninstanzen. Die Autorisation der Objekte des Endsystems erfolgt zudem durch die Zugriffskontrollpolitik des Endsystems. Die zentrale Kontrollinstanz (ZKI) kontrolliert Zugriffe der Subjekte, indem sie auf Basis der ZKI-Policy überprüft, ob die Berechtigung für den Zugriff vorhanden ist. Die ZKI-Policy ist die vom Agentensystem durchzusetzende Zugriffskontrollpolitik, welche die Zugriffsrechte der AS-Policy und die von dem Agentensystem akzeptierten Zugriffsrechte der Wunsch-MA-Policy enthält. Der Zugriff wird erst erlaubt, wenn innerhalb der ZKI-Policy ein passendes, für den Zugriff notwendiges Zugriffsrecht vorhanden ist.

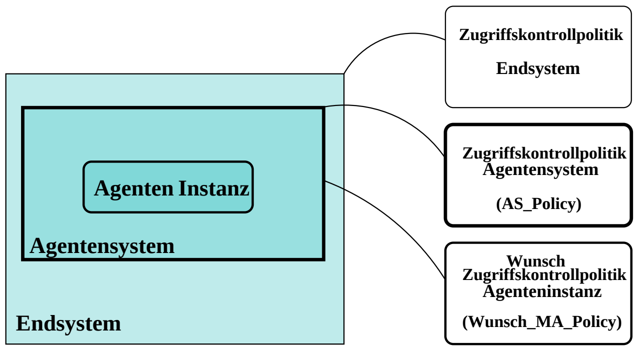


Abbildung 33: Existierende Zugriffskontrollpolitiken

Die Zugriffskontrollpolitik des Endsystems beschränkt Zugriffe auf Objekte des Endsystems. Die Zugriffsbeschränkungen sind durch die Zugriffskontrollpolitik des Endsystems nur von statischer Art und über den gesamten Ausführungszeitraum des Agentensystems nicht veränderbar. Zudem können keine anwendungsspezifischen Zugriffsbeschränkungen formuliert werden, wodurch in der Regel die Zugriffsbeschränkung durch die Zugriffskontrollpolitik des Agentensystems zu bevorzugen ist. Die Zugriffskontrollpolitik des Endsystems eignet sich jedoch für Beschränkungen von Zugriffen, die niemals stattfinden dürfen. Beispielsweise könnte hier der Zugriff auf ein Verzeichnis unterbunden werden, auf dem ein Agentensystem (und daher auch die Agenteninstanzen) niemals Zugriff haben soll.

Das Agentensystem ist in der Lage Zugriffe auf seine Schnittstellen über eine Zugriffskontrollpolitik (AS-Policy) zu autorisieren. Hierunter fallen dann auch Zugriffe auf Objekte des Endsystems, die von dem Agentensystem aus möglich sind. Innerhalb der Zugriffskontrollpolitik des Agentensystems kann eine anwendungsspezifische Autorisation erfolgen.

Agenteninstanzen können Zugriffe ebenfalls durch eine Wunsch-Zugriffskontrollpolitik (Wunsch-MA-Policy) beschränken. Die Zugriffskontrollpolitik beschränkt dabei die Zugriffe auf Objekte der Agenteninstanz und gegebenenfalls der Objekte der Agentensysteme und der Endsysteme, die von dem Agentensystem aus erreichbar sind.
Die Zugriffskontrollpolitik einer Agenteninstanz ist eine Wunschpolicy, da die in der Zugriffskontrollpolitik enthaltenen Zugriffsrechte vor ihrer Durchsetzung vom Agentensystem überprüft werden und nur die vom Agentensystem akzeptierten Zugriffsrechte von dem Agentensystem durch-

gesetzt werden.
Das Agentensystem überprüft vor der Durchsetzung eines Zugriffsrechts einer Wunsch-MA-Policy, auf Basis von Filter-Politiken, ob das Zugriffsrecht durchgesetzt werden darf. Dies betrifft vor allem Zugriffsrechte, die Zugriffe auf Objekte des Agentensystems bzw. des Endsystems oder einer anderen Agenteninstanz autorisieren. Dabei wird geprüft, ob das Zugriffsrecht innerhalb der Filter-Policy des Agentensystems oder der jeweiligen Agenteninstanz als unzulässig deklariert worden ist.

Durchgesetzt werden die vom Agentensystem akzeptierten Zugriffsrechte der Wunsch-MA-Policy durch die zentrale Kontrollinstanz (ZKI) des Agentensystems und nicht durch die Agenteninstanz selbst. Dadurch basiert die Informationssicherheit auf einer Vertrauensbeziehung zwischen der Agenteninstanz und dem Agentensystem. Die Agenteninstanz muss sich darauf verlassen, dass die Zugriffsrechte der Wunsch-MA-Policy durch das Agentensystem durchgesetzt wird.

Die akzeptierten Zugriffsrechte der Wunsch-MA-Policies der Agenteninstanzen und die Zugriffsrechte der AS-Policy des Agentensystems bilden die von der zentralen Kontrollinstanz (ZKI) durchzusetzenden ZKI-Policy.

Somit bestehen folgende Beziehungen zwischen den Schnittstellen und den für die Autorisation zuständigen Entitäten:

Schnittstelle	**Autorisierende Entität**
Endsystem vom Agentensystem nicht erreichbar	**Endsystem**
Endsystem vom Agentensystem erreichbar	**Agentensystem**
Schnittstellen des **Agentensystems**	**Agentensystem**
Schnittstellen einer **Agenteninstanz**	**Agenteninstanz**

Tabelle 1: Beziehung: Schnittstelle - autorisierende Entität

Ein Agentensystem und eine Agenteninstanz besitzen neben den Zugriffskontrollpolitiken auch jeweils einen Filter-Mechanismus, welcher durch eine Filter-Policy konfigurierbar ist. Durch die Verwendung eines Filters ist es einem Agentensystem und einer Agenteninstanz möglich, Regelungen bezüglich den zu akzeptierenden Zugriffsrechten einer Wunsch-MA-Policy einer Agenteninstanz festzulegen. Die innerhalb der Filter-Policy enthaltenen Einträge spezifizieren, welche Zugriffsrechte der Wunsch-MA-Policy von dem Agentensystem nicht durchgesetzt werden. Damit können systemglobale Bestimmungen des jeweiligen Agentensystems und der jeweiligen Agenteninstanz durchgesetzt werden. Der Filter kommt bei der Verknüpfung der Zugriffsrechte der Wunsch-MA-Policy mit den Zugriffsrechten der AS-Policy (s. Kapitel 6.6) zum Einsatz.

Die Kontrolle über die Zugriffsbeschränkung der einzelnen Objekte besitzt jeweils der Eigentümer. Der Eigentümer einer Agenteninstanz und eines Agentensystems ist diejenige Person, die die Entität gestartet hat. Der Eigentümer der Ressourcen des Endsystems, auf die das Agentensystem Zugriff hat, ist das Agentensystem, welches auf dem Endsystem ausgeführt wird.

Die Autorisation der Subjekte erfolgt durch Einträge innerhalb der Zugriffskontrollpolitik, welche den Subjekten objektspezifische Zugriffsrechte zuteilen. Bei jedem Zugriffsversuch prüft die zentrale Kontrollinstanz (ZKI) des Agentensystems, auf dem sich das Objekt befindet, ob das Subjekt über die notwendige Autorisation verfügt. Dies geschieht auf Basis der in der AS-Policy des Agentensystems und der von dem Agentensystem akzeptierten Zugriffsrechte der Wunsch-MA-Policies. Zugriffe auf Ressourcen des Endsystems werden zusätzlich durch die Zugriffskontrollpolitik des Endsystems beschränkt.

6.3.1 Adressierung der einzelnen Entitäten von MASA

Um Zugriffe autorisieren zu können, müssen einzelne Entitäten innerhalb der Zugriffskontrollpolitiken eindeutig adressiert werden können. Um die einzelnen Entitäten von MASA adressieren zu können werden definierte Eigenschaften (vgl. Kapitel 4.3) der einzelnen Entitäten herangezogen. Anhand dieser Eigenschaften ist dann die eindeutige Adressierung sichergestellt.

Adressierung der Entitäten innerhalb der Zugriffskontrollpolitik
In dem folgenden Abschnitt werden die einzelnen Entitäten von MASA entsprechend ihrer Funktion, die sie bei einem Zugriff einnehmen, zugeordnet. Durch die Zuordnung wird ersichtlich an welcher Stelle eine Entität innerhalb der Zugriffskontrollpolitik auftreten kann.

Adressierung der Subjekte
Personen, Agentensysteme und Agenteninstanzen können innerhalb von MASA die Funktion eines Subjekts einnehmen, d.h. sie können die aktive zugreifende Entität sein (vgl. Kapitel 2.1).

Subjekte werden innerhalb der Zugriffskontrollpolitik durch geeignete Attribute adressiert. Die Attribute beschreiben dabei Eigenschaften des jeweiligen Subjekts. Damit das Subjekt innerhalb der Zugriffskontrollpolitik eindeutig adressiert werden kann, müssen die zur Adressierung herangezogenen Eigenschaften das Subjekt eindeutig identifizieren können. Um dies zu gewährleisten werden die folgenden Eigenschaften der einzelnen Subjekte zur Adressierung der Entitäten innerhalb der Zugriffskontrollpolitik herangezogen:

Die Adressierung von Personen erfolgt durch den Personennamen.

```
[Personenname]
```

Die Adressierung des Agentensystems (AS) erfolgt anhand des Eintrags "AS" und der folgenden Eigenschaften des Agentensystems:

```
[AS: Eigentuemer, Anwender, Implementierer]
```

Die Adressierung der Agenteninstanzen (MA) erfolgt anhand dem Eintrag "MA" und den folgenden Eigenschaften der Agenteninstanz:

```
[MA: Eigentuemer, Anwender, Implementierer, Gattung, History]
```

Betrachtet man die History, so sind Adressierungen unterschiedlicher Formen möglich. Einerseits können hier Agentensysteme benannt werden, die nicht besucht wurden durften (`notvisit`), andererseits können auch Einträge vorgenommen werden, die besagen, welche Agentensysteme besucht werden durften (`visit`).

(Die History einer Agenteninstanz taucht dabei in der Reihe der Eigenschaften der Agenteninstanz auf, da sie eine für eine Agenteninstanz eindeutige Eigenschaft über den gesamten Zeitraum, über den sich die Agenteninstanz auf dem Agentensystem befindet, darstellt.)

Anhand der oben aufgeführten Eigenschaften können die Subjekte eindeutig autorisiert werden. Durch die Markierung "AS" und "MA" können Zugriffsrechte explizit einem Agenten bzw. einem Agentensystem zugeteilt werden.

Der Anwender einer Agenteninstanz kann eine Agenteninstanz, ein Agentensystem oder eine Person sein. Ist der Anwender eine Agenteninstanz oder ein Agentensystem, ist eine in sich verschachtelte Adressierung innerhalb der Zugriffskontrollpolitik möglich. Die Adressierung eines Agentensystems beispielsweise könnte, falls der Anwender eine Agenteninstanz ist, wie folgt aussehen:

```
[AS: Eigentuemer, [MA: Eigentuemer, Anwender, Implementierer, History], Implementierer]
```

Adressierung der Objekte und der Aktionen
Die von dem Endsystem, dem Agentensystem und den Agenteninstanzen zur Verfügung gestellten Operationen können die Funktion eines Objekts einnehmen, d.h. sie können der passive Teil eines Zugriffs sein (vgl. Kapitel 2.1).
Da in diesem Rechtekonzept objektspezifische Zugriffsrechte (vgl. Kapitel 2.6.2) verwendet werden, wird innerhalb des Zugriffsrechts das jeweilige Objekt adressiert. Die Adressierung der jeweiligen Aktion, erfolgt ebenfalls innerhalb des Zugriffsrechts.

Zugriffsrechte
Um anwendungsspezifische Zugriffsrechte vergeben zu können, d.h. Zugriffsrechte, die auf die spezifische Aufgabe des Subjekts zugeschnitten sind und somit dem Least-Privilege Prinzip entsprechen, werden objektspezifische Zugriffsrechte verwendet (vgl. Kapitel 2.6.2). Objektspezifische Zugriffsrechte enthalten neben der Aktion auch das jeweilige Objekt des Zugriffs. Dadurch ist die Vergabe eines Zugriffsrechts, welches nur eine bestimmte Funktionalität auf dem Objekt erlaubt, möglich.
Zugriffsrechte haben die Form:

```
[Aktion "z" auf "xy" ausfuehren]
```

Dieses Recht erhält dann ein Subjekt.

Beispielsweise können folgende Berechtigungen erteilt werden:
Beispiel 1: `Historyliste der Agenteninstanz XY auslesen`
Beispiel 2: `Agenteninstanz der Gattung XY starten`

Aus Sicht der Autorisationsinstanz muss die zu autorisierende Aktion, die innerhalb des Zugriffsrechts vergeben werden kann, bekannt sein. Handelt es sich um Aktionen auf ein Objekt des Agentensystems bzw. Endsystems, so kann nur der Administrator des Agentensystems Zugriffe innerhalb der AS-Policy autorisieren. Dabei ist vorauszusetzen, dass der Administrator die innerhalb der Zugriffsrechte zu vergebenden Aktionen kennt.

Handelt es sich um Operationen einer Agenteninstanz, so sind allgemeine Operationen und deren Aktionen, die jede Agenteninstanz zur Verfügung stellen muss, als bekannt vorausgesetzt.
Spezielle Operationen mit speziellen Aktionen, die durch die Agenteninstanz implementiert werden, sind dem Agentensystem in der Regel unbekannt und müssen von der Agenteninstanz entsprechend autorisiert werden können.
Spezifische Operationen der Agenteninstanzen können daher nur von dem Administrator der jeweiligen Agenteninstanz autorisiert werden. Die Autorisation erfolgt dabei innerhalb der Wunsch-MA-Policy. Die Agenteninstanzen müssen sich für die Durchsetzung der Zugriffsrechte auf das Agentensystem verlassen.
Die Namenvergabe der Zugriffsrechte erfolgt anhand eines festgelegten Schemas, welches im Kapitel 6.3.3 vorgestellt wird. So ist sichergestellt, dass die Zugriffsrechte, die von den Administratoren vergeben werden können, nicht falsch interpretiert werden.

6.3.2 Die Politik-Sprache der Zugriffskontrollpolitiken

Um Zugriffsbeschränkungen innerhalb einer Zugriffskontrollpolitik formulieren zu können, ist eine geeignete, den Anforderungen entsprechende, Politik-Sprache notwendig. Die Subjekte, Objekte und Aktionen sind innerhalb der Politik-Sprache feingranular adressierbar, so dass in MASA eine anwendungsspezifische Zugriffsbeschränkung realisiert ist.

Ein Zugriffsrecht wird durch einen `grant`-Eintrag ausgedrückt. Dabei bestimmt der Eintrag, welches Subjekt auf welches Objekt in welcher Art und Weise zugreifen darf. Innerhalb der Filter-Politiken werden Zugriffsrechte, die nicht durch eine Wunsch-MA-Policy integriert und durchgesetzt werden dürfen, mit dem `forbid`-Eintrag ausgedrückt. Die Terminalsymbole `ma`, `as` und `person` geben dabei den jeweiligen Typ des Eintrags an (Agenteninstanz, Agentensystem und Person). Nach dem Terminalsymbol `permission` werden die spezifischen Zugriffsrechte aufgeführt.

Folgend wird die kontextfreie Grammatik der Politik-Sprache eingeführt, die für die Wunsch-MA-Policy, die AS-Policy und der Filter-Policy verwendet wird. Die Notation erfolgt in der erweiterten Backus-Naur-Form (EBNF) [SCH 95].

Die Klammer "[ ]" bedeutet, dass der Eintrag eingefügt werden kann aber nicht muss. Die Klammer "{ }" bedeutet, dass der Eintrag beliebig oft (auch nullmal) wiederholt werden kann. Bei mehreren Regeln, die dieselbe linke Seite haben, wird eine einzige "Metaregel" (unter Verwendung des "Metasymbols" |) angegeben. Zur besseren Lesbarkeit werden Terminalzeichen unterstrichen und fett gedruckt. (Bei der Klammer **{}** handelt es sich demnach um ein Terminalzeichen und nicht um das {}-Symbol der EBNF).

policy_file -> *as_policy_file* | *as_filter_file* | *ma_policy_file* | *ma_filter_file*
as_policy_file -> *as_policy_entry* [**;** *as_policy_file*]
as_policy_entry -> **grant** *subject_entry* **{** *es_permission_entry* **}** **;** |
grant *subject_entry* **{** *as_permission_entry* **}** **;** |
grant *subject_entry* **{** *ma_permission_entry* **}** **;**

as_filter_file -> *as_filter_entry* [**;** *as_filter_file*]
as_filter_entry -> **forbid** *subject_entry* **{** *es_permission_entry* **}** **;** |
forbid *subject_entry* **{** *as_permission_entry* **}** **;** |
forbid *subject_entry* **{** *ma_permission_entry* **}** **;**

ma_policy_file -> [*time*] *ma_policy_entry* [**;** *ma_policy_file*]
ma_policy_entry -> [**<location** *as_specification* [**;** *as_specification*] **≥**]
grant *subject_entry* **{** *es_permission_entry* **}** **;** |
[**<location** *as_specification* [**;** *as_specification*] **≥**]
grant *subject_entry* **{** *as_permission_entry* **}** **;** |
[**<location** *as_specification* [**;** *as_specification*] **≥**]
grant *subject_entry* **{** *ma_permission_entry* **}** **;**
time -> **[***hh* **:** *mm* **-** *dd* **.** *mm* **.** *yyyy* **]**
as_specification -> **as.***as_entry* **,** *secure_flag*
secure_flag -> *encrypted* | *not_entcrypted*
encrypted -> **1**
not_encrypted -> **0**

ma_filter_file -> *ma_filter_entry* [**;** *ma_filter_file*]
ma_filter_entry -> **forbid** *subject_entry* **{** *ma_permission_entry* **}** **;** [*ma_filter_entry*]

```
subject_entry          -> ma.ma_entry | as.as_entry | person.person_entry
ma_entry               -> authority_entry . owner_entry .
                          coder_entry . kind_entry . [ [history_entry] ]
as_entry               -> authority_entry . owner_entry . coder_entry
person_entry           -> person_name

es_permission_entry    -> es_one_permission [es_permission_entry]
es_one_permission      -> permission es_permission_class_name
                          [ " target_name " ] [ , " action_list " ] [ , " signer_entry " ] ;

as_permission_entry    -> as_one_permission [as_permission_entry]
as_one_permission      -> permission as_permission_class_name
                          [ " target_name " ] [ , " action_list " ] [ , " signer_entry " ] ;

ma_permission_entry    -> ma_one_permission [ma_permission_entry]
ma_one_permission      -> permission ma_permission_class_name
                          [ " target_name " ] [ , " action_list " ] [ , " signer_entry " ] ;

history_entry          -> notvisit as_entry [; history_entry] | visit as_entry [ ; history_entry]

authority_entry        -> person_name
coder_entry            -> person_name
owner_entry            -> person_name
kind_entry             -> kind_name
```

Beschreibung der Variablen:

- *as_policy_file*
  Innerhalb der AS-Policy können Subjekte durch einen spezifischen Eintrag autorisiert werden, auf ein Objekt in einer bestimmten Art und Weise zuzugreifen.

- *as_policy_entry*
  Der as_policy_entry-Eintrag berechtigt ein Subjekt, auf ein Objekt in einer bestimmten Art und Weise zuzugreifen. Der Eintrag kann ein Subjekt und das dazugehörige Permission Objekt enthalten, welches auf das Endsystem, Agentensystem oder einer Agenteninstanz zugreifen kann.

- *as_filter_file*
  Der Eintrag bezeichnet die Filter-Policy des Agentensystems. Durch den Filter des Agentensystems können Zugriffsrechte, die durch die Wunsch-MA-Policy einer Agenteninstanz vergeben werden können, als unzulässig spezifiziert werden.

- *as_filter_entry*
  Der Eintrag enthält unzulässige Zugriffsrechte, die eine Agenteninstanz mittels der Wunsch-MA-Policy nicht durch das Agentensystem durchsetzen lassen kann.

- *ma_policy_file*
  Innerhalb der Wunsch-MA-Policy können Subjekte durch einen spezifischen Eintrag autorisiert werden, auf ein Objekt in einer bestimmten Art und Weise zuzugreifen.

- *ma_policy_entry*
  Der *ma_policy_entry*-Eintrag ist fast identisch mit dem as_policy_entry-Eintrag. Hier sind jedoch zusätzliche Einträge möglich, die das Agentensystem spezifizieren (*as_specification*), auf dem die Zugriffsrechte durchgesetzt werden sollen und die die Gültigkeitsdauer der Zugriffsrechte festlegen.

- *time*
  Die Angabe des Datums gibt das Gültigkeitsdatum des Wunsch-Zugriffsrechts an.

- *as_specification*
  Innerhalb der Wunsch-MA-Policy kann ein Agentensystem benannt werden, für das das Zugriffsrecht gelten soll. Neben der Bezeichnung des Agentensystems besteht weiterhin die Möglichkeit, das Zugriffsrecht mit dem Public Key des Agentensystems, für das das Zugriffsrecht gelten soll, zu verschlüsseln.

- *secure_flag*
  Das *secure_flag* kann den Eintrag "0" oder "1" haben. Wird das Zugriffsrecht mit dem Public Key des jeweiligen Agentensystems verschlüsselt, so muss das Flag auf "1" gesetzt werden.

- *ma_filter_file*
  Der Eintrag bezeichnet die Filter-Policy einer Agenteninstanz. Durch den Filter einer Agenteninstanz kann die Vergabe spezifischer Zugriffsrechte bezüglich der Objekte der Agenteninstanz als unzulässig spezifiziert werden. Diese Zugriffsrechte werden nicht durch das Agentensystem durchgesetzt.

- *ma_filter_entry*
  Der Eintrag enthält die unzulässigen Zugriffsrechte bezüglich der Objekte der Agenteninstanz, die eine Agenteninstanz mittels der Wunsch-MA-Policy nicht durch das Agentensystem durchsetzen lassen kann.

- *subject_entry*
  Ein Subjekt, das auf eine Ressource des Agentensystems zugreifen kann, ist ein Agentensystem, eine Agenteninstanz oder eine Person.

- *ma_entry*
  Der Eintrag beschreibt die Eigenschaften einer Agenteninstanz.

- *as_entry*
  Hier werden die Eigenschaften des Agentensystems beschrieben.

- *person_entry*
  Der Eintrag beschreibt den Namen einer Person.

- *es_one_permission*
  Der Eintrag bezeichnet Zugriffsrechte der Zugriffe auf Ressourcen des Endsystems.

- *as_one_permission*
  Der Eintrag bezeichnet Zugriffsrechte der Zugriffe auf Ressourcen eines Agentensystems.

- *ma_one_permission*
  Der Eintrag bezeichnet Zugriffsrechte der Zugriffe auf Ressourcen einer Agenteninstanz.

- *history_entry*
  Der Eintrag beschreibt die Eigenschaften der besuchten Agentensysteme, wobei der Eintrag `visit` angibt, welche Agentensysteme besucht werden durften und `notvisit`, welche Agentensysteme zuvor nicht besucht werden durften.

- *authority_entry*
  Der *authority_entry*-Eintrag bezeichnet den Anwender.

- *coder_entry*
  Hierbei handelt es sich um den Implementierer der Agentengattung.

- *owner_entry*
  Der Eintrag bezeichnet den Eigentümer.

- *kind_entry*
  Der Eintrag beschreibt den Namen der Agentengattung.

6.3.3 Syntax der Zugriffsrechte

Ein Zugriffsrecht besteht aus einer Menge von Subjekten, einer Menge von Objekten und einer Menge von Zugriffsarten. Ein Zugriffsrecht ermöglicht die Berechtigung eines oder mehrerer Subjekte, auf eine oder mehrere bestimmte Arten auf ein oder mehrere Objekte zuzugreifen.
Das Zugriffsrecht entspricht dabei dem folgenden Eintrag:

```
grant  <subject> {
  permission ObjektPermission {"<parameter>", "<action>"}
}
```

Folgend werden einige Beispiele für Zugriffsrechte gezeigt, die innerhalb von MASA eingesetzt werden können.

Zugriffsrechte auf Ressourcen des Endsystems
Beispiel 1.1: Zugriffsrecht für einen Zugriff einer Agenteninstanz auf eine Ressource des Endsystems [MA - ES]

```
grant ma.provider.provider.*.*[visit as.haendler.haendler.mnm,
                              [visit as.hersteller.hersteller.mnm,
                              [visit provider.provider.mnm]{

 permission java.security.FilePermission ("/results/-", "read")
}
```

Dieses Zugriffsrecht erlaubt den Agenteninstanzen, die die in dem Zugriffsrecht enthaltenen Eigenschaften (Eigentümer=[provider], Anwender=[provider], Implementierer= alle, Gattung=alle und höchstens die folgenden Agentensysteme in der History=[as.haendler.haendler.mnm, as.hersteller.hersteller.mnm, as.provider.provider.mnm]) erfüllen, auf die ES-Ressource (das Verzeichnis:"/results/") lesend zuzugreifen. Dabei wird von der Überprüfung der History der Agenteninstanz Gebrauch gemacht.

Die History der Agenteninstanz gibt an, welche Agentensysteme mit den genannten Eigenschaften besucht werden durften. Sind die Eigenschaften erfüllt, kann das Zugriffsrecht erteilt werden. Die Reihenfolge der History-Einträge spielt dabei keine Rolle.

Beispiel 1.2: Zugriffsrecht für einen Zugriff einer Agenteninstanz auf eine Ressource des Endsystems [MA - ES]

```
grant ma.provider.*.*.as-hersteller.[notvisit as.hersteller.hersteller.mnm]{
  permission java.security.FilePermission ("/results/-", "read")
}
```

Hierbei handelt es sich um ein Zugriffsrecht für Agenteninstanzen mit Beschränkung der History, wobei angegeben wird, welches Agentensystem zuvor nicht besucht werden durfte.

Die zugreifende Agenteninstanz darf nur auf das Verzeichnis zugreifen, wenn die History der Agenteninstanz nicht das angegebene Agentensystem aufweist.

Beispiel 1.3: Zugriffsrecht für einen Zugriff einer Agenteninstanz auf eine Ressource des Endsystems [MA - ES]

```
grant ma.provider.*.*.as-hersteller[visit as.*.provider.*.*]{
   permission java.security.FilePermission ("/results/-", "read")
}
```

Das Zugriffsrecht wird an die entsprechende Agenteninstanz vergeben, falls nur Agentensysteme besucht worden sind, dessen Eigentümer jeweils der [provider] war.

Beispiel 1.4: Zugriffsrecht für einen Zugriff einer Agenteninstanz auf eine Ressource des Endsystems [MA - ES]

```
grant ma.provider.*.*.as-hersteller[visit as.*.[as.*.provider.*.*].*.*]{
  permission java.security.FilePermission ("/results/-", "read")
}
```

Das Zugriffsrecht besitzt weitere Attribute innerhalb der History. Dabei dürfen von Agenteninstanzen nur Agentensysteme besucht worden sein, deren Eigentümer (authority) ein Agentensystem ist, dessen Eigentümer wiederum der Provider ist.

Es können also Eigenschaften von Eigenschaften eines Subjekts adressiert werden. Dies ist dann möglich, wenn eine Eigenschaft ein Principal darstellt. Die Attribute des Principals sind dann die weiteren Eigenschaften, die zur Adressierung herangezogen werden können.

Beispiel 1.5: Zugriffsrecht für einen Zugriff einer Agenteninstanz auf eine Ressource des Endsystems [MA - ES]

```
grant ma.provider.*.*.mnm.messe.[*] , ma.haendler.*.*.mnm.messe.[*]{
  permission java.security.FilePermission ("/temp", "read, write")
}
```

Ein Zugriffsrecht ist nicht nur an ein Subjekt oder Objekt gebunden. Es können mehrere Subjekte, mehrere Objekte und mehrere Aktionen, durch Hinzunahme des Wildcards * bestimmt werden. So impliziert das wie folgt modellierte Zugriffsrecht alle Zugriffe, deren Subjekte eine Agenteninstanz mit den folgenden Eigenschaften sind:
Authority = [provider] oder [haendler], Implementierer = [mnm] und der Gattung [messe]

Zugriffsrechte auf die Objekte des Agentensystems
Zugriffsrechte auf Objekte des Agentensystems beginnen mit der Kennzeichnung `masa.As`. Handelt es sich um ein Zugriffsrecht auf eine Agenteninstanz beginnt der Name des Zugriffsrechts mit `masa.AsAgent`.

Beispiel 2.1: Zugriffsrecht für einen Zugriff eines Agentensystems auf eine Ressource eines Agentensystems [AS - AS]

```
grant as.*.*.mnm {
  permission masa.AsPermission ("getOwner")
}
```

Ein Agentensystem mit dem Implementierer [mnm] darf den Eigentümer des Agentensystems auslesen.

Beispiel 2.2: Zugriffsrecht für einen Zugriff einer Agenteninstanz auf eine Ressource des Agentensystems [MA - AS]

```
grant ma.mnm.mnm.*.*.* {
  permission masa.AsAgentPermission ("ma.*.*.*.*.*", "createAgent")
}
```

Eine Agenteninstanz mit dem Eigentümer [mnm] und dem Anwender [mnm] darf auf dem Agentensystem eine Agenteninstanz jeglicher Art erzeugen.

Beispiel 2.3: Zugriffsrecht für einen Zugriff eines Agentensystems auf eine Ressource eines Agentensystems [AS - AS]

```
grant as.*.*.mnm {
  permission masa.AsAgentPermission ("ma.*.*.*.*.*", "createAgent")
}
```

Zugriffsrechte auf Objekte einer Agenteninstanz:
Zugriffsrechte auf Objekte einer Agenteninstanz beginnen mit Namen masa.MA. **Beispiel 3.1:** Zugriffsrecht für einen Zugriff einer Agenteninstanz auf eine Ressource einer Agenteninstanz [MA - MA]

```
grant ma.*.*.mnm.*.[*] {
  permission masa.MaPermission ("ma.mnm.mnm.mnm.messe.[*]", "getName")
}
```

Dieses Zugriffsrecht erteilt eine Berechtigung für den Zugriff auf eine Agenteninstanz. Agenteninstanzen, welche von [mnm] implementiert worden sind, dürfen die Aktion [getName] auf der Agenteninstanz [ma.mnm.mnm.mnm.messe.[*]] ausführen. Das Zugriffsrecht wird im Normalfall in der Wunsch-MA-Policy einer Agenteninstanz vergeben und muss daher vom Agentensystem akzeptiert werden, um auch durchgesetzt zu werden. Die Vergabe von Zugriffsrechten durch Agenteninstanzen können von dem Agentensystem beschränkt werden. Dadurch können z.B. von einem Agentensystem nur Zugriffsrechte einer Agenteninstanz durchgesetzt werden, die Objekte der Agenteninstanz selbst betreffen.

Beispiel 5: Zugriffsrecht für einen Zugriff eines Agentensystems auf eine Ressource einer Agenteninstanz [AS - MA]

```
grant as.*.*.mnm {
  permission masa.MaAgentPermission ("ma.mnm.mnm.mnm.messe.[*]", "getName")
}
```

Der folgende Eintrag berechtigt Agentensysteme, welche von [mnm] implementiert worden sind, auf der Agenteninstanz mit den genannten Eigenschaften, die Aktion [getName] auszuführen. Da es sich wiederum um ein Zugriffsrecht auf eine Agenteninstanzmethode handelt, wird es ebenfalls innerhalb der Wunsch-MA-Policy vergeben. Es liegt dann wiederum am Agentensystem, dieses Zugriffsrecht durchzusetzen.

6.3.4 Syntax der Zugriffskontrollpolitik

Die Zugriffskontrollpolitiken können aus mehreren Zugriffsrechten, also mehreren grant-Einträgen bestehen. Die Zugriffskontrollpolitik impliziert alle Rechte, die jedes einzelne Zugriffsrecht enthält.

Beispiel Agentensystem Zugriffskontrollpolitik (AS-Policy)

```
\\
\\   Agentensystem Policy
\\
grant ma.provider.provider.*.*.[visit as.haendler.haendler.mnm,
                                visit as.hersteller.hersteller.mnm,
                                visit as.provider.provider.mnm] {
   permission java.security.FilePermission("/results/", "read")
}
```

```
grant ma.provider.provider.mnm.messe.[*], ma.haendler.haendler.mnm.messe.[*]{
   permission java.security.FilePermission("/results/", "read, write")

}
```

Beispiel Agenteninstanz Zugriffskontrollpolitik (Wunsch-MA-Policy)

```
\\
\\   Agenteninstanz Policy
\\
grant ma.provider.provider.*.*.[visit as.haendler.haendler.mnm,
                               visit as.provider.provider.mnm] {
   permission java.security.FilePermission("/results/", "read")
}

[18:00 - 26.04.2001]
grant ma.provider.provider.mnm.messe.[*], ma.haendler.haendler.mnm.messe.[*]{
   permission java.security.FilePermission("/results/", "read, write")
}

<location as.haendler.haendler.mnm, 0>
grant ma.provider.provider.*.*.[*]{
   permission masa.MaAgentPermission ("readResult")
}
```

6.4 Modellierung der Zugriffe

Die hier eingeführte Modellierungssprache dient der Modellierung der Zugriffe. Durch die Modellierung eines Szenarios kann so leicht die notwendige Vergabe der Zugriffsrechte innerhalb der jeweils zuständigen Zugriffskontrollpolitik ermittelt werden. Innerhalb der Modellierung werden Subjekte, die auf Objekte zugreifen, modelliert. Dabei spielt vor allem die Aktion, die das Subjekt auf dem Objekt ausführt, eine wichtige Rolle.

Bei der Modellierung eines Zugriffs ist auch der Ort des Subjekts und des Objekts enthalten, um Zugriffe über Systemgrenzen hinaus modellieren zu können. Die modellierten Beispielzugriffe stellen die Zugriffe in Anlehnung an das Beispielszenario aus Kapitel 1 dar.

Modellierung eines Zugriffs: Ein Zugriff wird wie folgt definiert:

```
{[as: <as>](subject, <subject>)->[as: <as>](object,<object>) | (action, <action>)}
```

Beispiel 1: MA greift auf eine ES-Ressource des AS zu, auf dem er sich befindet

```
{
[as: as1.provider.provider.mnm]
(subject, ma.provider.provider.mnm.messe.(as.provider.provider.mnm)) ->
[as: as1.provider.provider.mnm]
(object, /results/erg-haendler-071200)|
(action, write)
}
```

Dies ist eine Modellierung des Zugriffs einer Agenteninstanz auf ein Objekt des Agentensystems, auf dem sich die Agenteninstanz befindet.

Modelliert wird der Zugriff der Agenteninstanz [ma-p-messe] mit den Eigenschaften: Eigentümer = [provider], Anwender = [provider], Implementierer = [mnm], Gattung = [messe] und History = [Agentensystem: Provider].
Die Agenteninstanz befindet sich auf dem Agentensystem [Provider], welches die folgenden Eigenschaften aufweist: Eigentümer = [provider], Anwender = [provider] und Implementierer = [mnm].
Bei der Ressource, auf die zugegriffen wird, handelt es sich in diesem Fall um eine Ressource des Endsystems, auf welchem das Agentensystem ausgeführt wird: ES-Ressource = [/results/erg-haendler-071200].
Auf diese Ressource wird folgende Aktion [write] ausgeführt.

Beispiel 2: MA mit Vergangenheit greift auf eine ES-Ressource

```
{
[as: as1.provider.provider.mnm]
(subject, ma.haendler.mnm.messe.(as.provider.provider.mnm
                                 as.haendler.haendler.mnm,
                                 as.hersteller.hersteller.mnm)) ->
[as: as1.provider.provider.mnm]
(object, /results/erg-hersteller-071200)|
(action, write)
}
```

Bei Beispiel 2 handelt es sich um eine Modellierung des Zugriffs einer Agenteninstanz, die zuvor andere Agentensysteme besucht hat, auf ein Objekt des Agentensystems, auf dem sich die Agenteninstanz befindet.

Modelliert wird der Zugriff der Agenteninstanz [ma-h-messe] mit den folgenden Eigenschaften: Eigentümer = [haendler], Anwender = [haendler], Implementierer = [mnm], Gattung = [messe] und der History = [as-haendler, as-hersteller], welche sich auf dem Agentensystem [provider] befindet, welches die folgenden Eigenschaften hat: Eigentümer = [provider], Anwender = [provider] und Implementierer = [mnm].
Zugegriffen wird auf die ES-Ressource: [/results/erg-hersteller-071200]. Auf der ES-Ressource wird die Aktion [write] ausgeführt. Die History gibt an, dass diese Agenteninstanz zuvor die Agentensysteme des Händlers und des Herstellers besucht hat.

6.5 Einsatz der Zugriffskontrollpolitiken

Ein Subjekt wird für einen Zugriff auf ein Objekt autorisiert, indem das Zugriffsrecht, welches das Subjekt, das Objekt und die Zugriffsart enthält, in die zuständige Zugriffskontrollpolitik eingetragen wird. Aufgrund der feingranularen Adressierung der Entitäten können Zugriffsrechte nach dem Least-Privilege Prinzip vergeben werden. Die Syntax der Einträge ist durch die in Kapitel 6.3.2 definierte Politik-Sprache festgelegt.

Agentensysteme autorisieren Zugriffe durch die AS-Policy. Eine Agenteninstanz kann innerhalb der Wunsch-MA-Policy Zugriffe autorisieren, die jedoch erst von dem Agentensystem, auf dem sich die Agenteninstanz befindet, akzeptiert werden müssen, um auch durchgesetzt zu werden. Das Agentensystem entscheidet anhand der für das jeweilige Zugriffsrecht zuständigen Filter-Politik, ob das Zugriffsrecht durchgesetzt wird.

Einerseits besitzt ein Agentensystem eine Filter-Politik (AS-Filter-Policy), wodurch das Agentensystem die Zugriffsrechte spezifizieren kann, die nicht durch eine Wunsch-MA-Policy einer Agenteninstanz in die ZKI-Policy integriert und durchgesetzt werden. Andererseits besitzt auch eine Agenteninstanz eine Filter-Politik (MA-Filter-Policy), in der beispielsweise eine Agenteninstanz "A" definieren kann, welche Zugriffsrechte das Agentensystem nicht durch eine Wunsch-MA-Policy einer weiteren Agenteninstanz "B", bezüglich Objekten der Agenteninstanz "A" durchsetzen soll.

Vergibt eine Agenteninstanz innerhalb ihrer Wunsch-MA-Policy Zugriffsrechte auf Objekte einer anderen Agenteninstanz, kontrolliert die ZKI die MA-Filter-Policy der entsprechenden Agenteninstanz und entscheidet daraufhin, ob die Wunsch-Zugriffsrechte in die ZKI-Policy integriert und somit von der ZKI durchgesetzt werden können.

6.5.1 AS-Policy

Der Eigentümer (bzw. Administrator) des Agentensystems ist berechtigt, Zugriffsrechte in die AS-Policy einzutragen. Er ist jedoch auch in der Lage, einem weiteren Subjekt die Berechtigung zu geben, Änderungen innerhalb der AS-Policy vorzunehmen. Hierfür muss er dem jeweiligen Subjekt ein spezielles Zugriffsrecht erteilen.

Findet ein Zugriff auf ein Objekt des Agentensystems statt, so kontrolliert die zentrale Kontrollinstanz (ZKI) den Zugriff, indem übeprüft wird, ob in der AS-Policy, die Bestandteil der ZKI-Policy ist, das notwendige Zugriffsrecht vorhanden ist (vgl. Abb. 34).

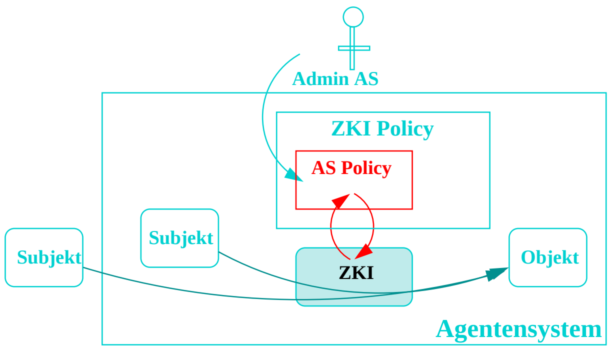


Abbildung 34: Autorisation der Zugriffe durch die AS-Policy

Innerhalb der Zugriffskontrollpolitik eines Agentensystems können folgende Zugriffe durch die Vergabe von Zugriffsrechten autorisiert werden:

- Zugriffe auf Objekte einer Agenteninstanz
- Zugriffe auf Objekte des Agentensystems
- Zugriffe auf Objekte des Endsystems

Der Administrator sollte jedoch beachten, dass Zugriffe auf Objekte von Agenteninstanzen im Normalfall nur von den Agenteninstanzen mittels der Wunsch-MA-Policy autorisiert werden.
Um die Informationssicherheit der Agenteninstanzen zu schützen, sollte diese Regel zwingend eingehalten werden. Die Möglichkeit trotzdem Zugriffe auf Agenteninstanzen autorisieren zu können, spiegelt die Einbettungsbeziehung einer Agenteninstanz gegenüber des Agentensystems wider. Das Agentensystem muss daher dem Agentensystem vertrauen können.

6.5.2 Wunsch-MA-Policy

Zugriffsrechte innerhalb der Wunsch-MA-Policy werden von dem Eigentümer der Agenteninstanz bei der Erzeugung der Agenteninstanz vergeben. Eine Agenteninstanz führt die Wunsch-MA-Policy mit sich. Um die Wunsch-MA-Policy vor böswilligen bzw. unrechtmäßigen Veränderungen zu schützen, muss sie von dem Eigentümer signiert werden und ist durch die Signatur eindeutig einer Person zuordenbar, wobei zusätzlich die Integrität der Wunsch-MA-Policy sichergestellt ist. Soll der Inhalt vor unberechtigtem Lesen geschützt werden, besteht die Möglichkeit, die Wunsch-MA-Policy mit dem Public Key des Agentensystems, welches die Wunsch-MA-Policy durchsetzen soll, zu verschlüsseln.

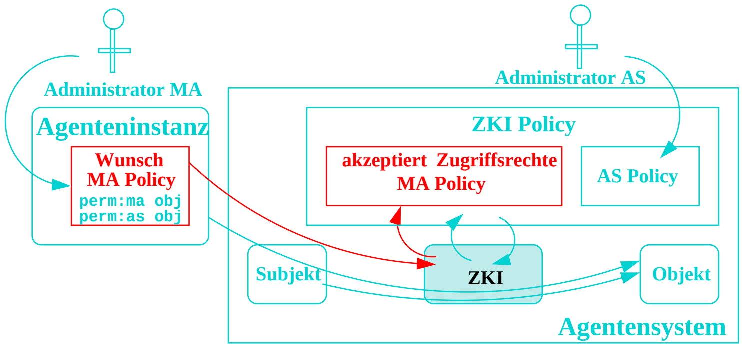


Abbildung 35: Autorisation der Zugriffe durch die Wunsch-MA-Policy

Innerhalb der Wunsch-Zugriffskontrollpolitik einer Agenteninstanz können Zugriffsrechte vergeben werden bezüglich:

- Zugriffe auf Objekte einer Agenteninstanz (a)
- Zugriffe auf Objekte des Agentensystems (b)
- Zugriffe auf Objekte des Endsystems (c)

Dabei sollten in der Wunsch-MA-Policy Zugriffsrechte vergeben werden, die bei ihrer Durchsetzung durch das Agentensystem der Agenteninstanz ermöglichen, ihre Aufgaben zu erledigen (a,b und c). Es ist jedoch darauf zu achten, dass einzelne Zugriffsrechte der Wunsch-MA-Policy unter Umständen nicht von einem Agentensystem akzeptiert und damit nicht durchgesetzt werden. Dies betrifft vor allem Zugriffsrechte, die Zugriffe auf Objekte des Agentensystems oder des Endsystems autorisieren (b und c). Diese Zugriffsrechte werden nur dann durchgesetzt, wenn die Zugriffsrechte nicht innerhalb der AS-Filter-Policy des Agentensystems bzw. der MA-Filter-Policy einer Agenteninstanz als unzulässig (`forbid`) deklariert worden sind.

Die Spezifizierung der unzulässigen Zugriffsrechte innerhalb der Filter-Politiken kann nur der Eigentümer des Objekts der Agenteninstanz oder des Agentensystems vornehmen. Bevor Zugriffsrechte der Wunsch-MA-Policies durchgesetzt werden, überprüft die zentrale Kontrollinstanz (ZKI), ob das Zugriffsrecht von der zuständigen Filter-Politik akzeptiert wird.

Zugriffsrechte der Wunsch-MA-Policy, welche Zugriffe auf die Schnittstelle der eigenen Agenteninstanz erlauben (a), werden in der Regel von der Kontrollinstanz (ZKI) des Agentensystems ausnahmslos akzeptiert und durchgesetzt.

Autorisation innerhalb der Wunsch-MA-Policy für spezielle Agentensysteme
Um innerhalb der Wunsch-MA-Policy die Möglichkeit zu bieten, Zugriffsrechte nach dem Least-Privilege Prinzip zu vergeben, kann die Wunsch-MA-Policy in Abschnitte unterteilt werden, die jeweils einem oder mehreren Agentensystemen zugeteilt werden. Zugriffsrechte, die nur von bestimmten Agentensystemen durchgesetzt werden sollen, besitzen innerhalb der Wunsch-MA-Policy einen Eintrag, der das Agentensystem angibt, bei welchem das Zugriffsrecht durchgesetzt werden soll.

```
<location as.as_entry, secure_flag>
```

Damit auch hier ein eventueller Missbrauch ausgeschlossen werden kann, kann das Zugriffsrecht mit dem Public Key des Agentensystems (bzw. einer Agentensystem Gruppierung) verschlüsselt werden. Das betreffende Agentensystem berücksichtigt bei der Durchsetzung der Zugriffsrechte die dem Agentensystem zugeordneten Abschnitte.

Wunsch-MA-Policy bei der Erzeugung einer neuen Agenteninstanzen durch eine Agenteninstanz
Wird durch eine Agenteninstanz auf einem fremden Agentensystem eine weitere Agenteninstanz erzeugt, so bekommt diese weitere Agenteninstanz die Wunsch-MA-Policy der Agenteninstanz in Form einer Kopie übermittelt. Zusätzlich wird die Kopie der Wunsch-MA-Policy von dem Agentensystem unterzeichnet, auf dem die "neue" Agenteninstanz erzeugt wurde. Die Entscheidung der Durchsetzung der in der Kopie der Wunsch-MA-Policy enthaltenen Zugriffsrechte basiert auf der jeweiligen Vertrauensbeziehung zu dem Agentensystem. Die Wunsch-MA-Policy ist hier nicht mehr von einem Eigentümer sondern von einem Agentensystem unterzeichnet.

Durch die Delegation der Zugriffsrechte besitzen die neu erzeugten Agenteninstanzen ebenfalls die für die Bearbeitung der Aufgabe notwendigen Zugriffsrechte.

6.5.3 Dynamische Zugriffsrechte (Agenteninstanz)

Die durch die Wunsch-MA-Policy vergebenen Zugriffsrechte sind in zweierlei Hinsicht statisch. Zuerst kann ein Zugriffsrecht nicht an bestimmte Bedingungen geknüpft werden. Zweitens sind die Zugriffsrechte nicht modifizierbar, d.h. sie können nur beim Start der Agenteninstanz vergeben werden.

Es gibt jedoch auch Szenarien, in denen eine dynamische Vergabe von Zugriffsrechten für Agenteninstanzen benötigt werden. Die Vergabe von Zugriffsrechten kann so z.B. von bestimmten Zustandsbedingungen abhängen.

Beispielsweise ist es sinnvoll, den Zugriff auf eine Methode einer Agenteninstanz, welche das Ergebnis einer Messung liefert, erst nach der erfolgreichen Messung zu erlauben. Hierfür müsste nach der erfolgreichen Messung die Agenteninstanz die Möglichkeit haben, das Zugriffsrecht auf diese Methode zu setzen bzw. auch wieder zurückzunehmen. Vorstellbar ist auch die Situation, dass eine Agenteninstanz sensitive Daten des Agentensystems bearbeitet und in diesem Zeitraum keine Zugriffe auf diese Agenteninstanz zugelassen werden sollen.

Daher sollte eine Agenteninstanz die Möglichkeit haben, Zugriffe dynamisch geknüpft an Bedingungen, beschränken zu können. Um dies zu ermöglichen, wird die Aktions-Policy eingeführt.

Damit Zugriffsrechte auch während des Lebenszyklus der Agenteninstanz dem Umfeld dynamisch angepasst werden können besteht die Möglichkeit, die Wunsch-MA-Policy einer Agenteninstanz durch eine "neue" Wunsch-MA-Policy auszutauschen.

Erneuerung der Wunsch-MA-Policy
Damit die Zugriffsrechte während der Lebenszeit einer Agenteninstanz modifiziert werden können, besteht die Möglichkeit, dass der Eigentümer der Agenteninstanz eine weitere Wunsch-MA-Policy erzeugen kann und diese dann die "alte" Wunsch-MA-Policy ersetzt. Die neue Wunsch-MA-Policy muss dabei von ein und demselben Eigentümer der Agenteninstanz unterzeichnet sein.

Damit der Missbrauch ausgeschlossen werden kann, ist nur der Eigentümer in der Lage, die Wunsch-MA-Policy zu erneuern. Das Agentensystem kontrolliert auch hier die Berechtigung. Bei einem Austausch wird dabei der Eigentümer der Agenteninstanz mit dem Unterzeichner der Wunsch-MA-Policy verglichen. Desweiteren kann auf jedem Agentensystem die Signatur der Wunsch-MA-Policy mit dem Eigentümer der Agenteninstanz verglichen werden.

Ein Austausch der Wunsch-MA-Policy kann aufgrund der notwendigen Signatur des Eigentümers in der Regel nur auf dem Home-Agentensystem der Agenteninstanz erfolgen.

Aktions Politik
Die Aktions-Policies sind Politiken, die Zugriffsrechte geknüpft an Bedingungen vergeben können. Die Aktions-Policy wird bei einem Eintreffen einer bestimmten Bedingung an das Agentensystem übergeben, welches die in der Aktions-Politik enthaltenen Zugriffsrechte durchsetzen kann. Die innerhalb der Aktions-Politik vergebenen Zugriffsrechte werden ebenfalls vor ihrer Durchsetzung auf Basis der Filter kontrolliert.
Hat eine Agenteninstanz beispielsweise die Aufgabe, den Netzdurchsatz zu messen, so kann die Agenteninstanz den Zugriff erst zulassen, nachdem das Ergebnis ermittelt wurde. Um dies zu erreichen wird die Wunsch-MA-Policy, die das notwendige Zugriffsrecht enthält, erst nachdem das Ergebnis vorliegt, an das Agentensystem überreicht.

Die Aktions-Policy ist mit der Wunsch-MA-Policy vergleichbar, jedoch können innerhalb der Aktions-Policy nur Zugriffsrechte für Objekte der Agenteninstanz vergeben werden.

Die Agenteninstanz kann unabhängig von dem Migrationszeitpunkt jederzeit dem Agentensystem die Aktions-Policy zur Durchsetzung vorlegen. Die Kontrollinstanz (ZKI) des Agentensystems überprüft, ob die Zugriffsrechte durchgesetzt werden können. Falls ja, werden die in der Aktions-Policy enthaltenen Zugriffsrechte von der zentralen Kontrollinstanz (ZKI) durchgesetzt.

Die Zugriffsrechte der Aktions-Policy können jederzeit von der Agenteninstanz wieder zurückgenommen werden. Wird der Agent vorher beendet oder migriert die Agenteninstanz auf ein anderes Agentensystem, werden die Zugriffsrechte automatisch von dem Agentensystem aus der ZKI-Policy entfernt.

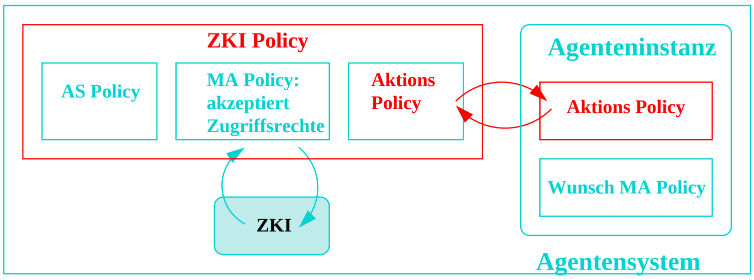


Abbildung 36: Dynamische Zugriffsrechte durch die Aktions-Policy

6.6 Die ZKI-Policy (AS-Policy und Wunsch-MA-Policy)

Die auf einem Agentensystem vorhandene AS-Policy ist Bestandteil der ZKI-Policy und wird daher von der zentralen Kontrollinstanz (ZKI) durchgesetzt. Migriert eine Agenteninstanz auf das Agentensystem oder wird eine Agenteninstanz auf dem Agentensystem neu gestartet, werden die von dem Agentensystem akzeptierten Zugriffsrechte der Wunsch-MA-Policy ebenfalls von der zentralen Kontrollinstanz (ZKI) durchgesetzt. Welche Zugriffsrechte einer Agenteninstanz durchgesetzt werden, entscheidet das Agentensystem unter Berücksichtigung der Filter-Politiken. Die von dem Agentensystem akzeptierten Zugriffsrechte einer Agenteninstanz werden zusammen mit den Zugriffsrechten der AS-Policy durchgesetzt, wobei die Zugriffsrechte der Politiken verknüpft werden. Die aus der Verknüpfung resultierenden Zugriffsrechte bilden die vom Agentensystem durchzusetzenden Zugriffsrechte.

Das Ergebnis der Verknüpfung ist die Vereinigungsmenge der Zugriffsrechte. Zugriffe werden erlaubt, falls mindestens ein Zugriffsrecht der AS-Policy oder der zugelassenen Zugriffsrechte der

Wunsch-MA-Policy den Zugriff erlauben.

Bei der Verknüpfung der Zugriffskontrollpolitiken werden die Autonomie-Anforderungen des Agentensystems und der betreffenden Agenteninstanz erfüllt. D.h. die Zugriffe, die nach der Zugriffskontrollpolitik AS-Policy erlaubt sind, sind nach der Verknüpfung weiterhin erlaubt. Zugriffe die explizit verboten sind, sind nach der Verknüpfung weiterhin verboten.
Die Autonomie-Anforderungen der Wunsch-MA-Policy sind nur erfüllt, falls sich die Zugriffsrechte auf Objekte der Agenteninstanz beziehen.

Betrachtet man Zugriffsrechte innerhalb der Wunsch-MA-Policy, die Objekte des Agentensystems bzw. anderer Agenteninstanzen betreffen, so werden die vergebenen Zugriffsrechte nur durchgesetzt, falls sie nicht innerhalb der jeweiligen Filter-Politik als unzulässig deklariert worden sind.

Das Agentensystem entscheidet bei der Verknüpfung der Wunsch-MA-Policy mit der AS-Policy anhand der authentifizierten Agenteninstanz und der Signatur der Wunsch-MA-Policy, ob die Agenteninstanz die Zugriffsrechte vergeben darf.

Das Agentensystem überprüft vor der Integration der Wunsch-MA-Policy:

- Ist die Agenteninstanz vertrauenswürdig?
- Ist die History der Agenteninstanz in Ordnung?
- Ist die Wunsch-MA-Policy von einer vertrauenswürdigen Person unterzeichnet?
- Ist die Wunsch-MA-Policy modifiziert worden (Integrität)?
- Werden nur erlaubte Zugriffsrechte vergeben (Filter)?

Die Verknüpfungsmöglichkeit der Wunsch-MA-Policy gestattet eine dynamische Erweiterung der durchzusetzenden Zugriffsrechte des Agentensystems (ZKI-Policy). Durch mobile Agenteninstanzen ist daher eine dynamische und flexible Autorisation von Zugriffen möglich.

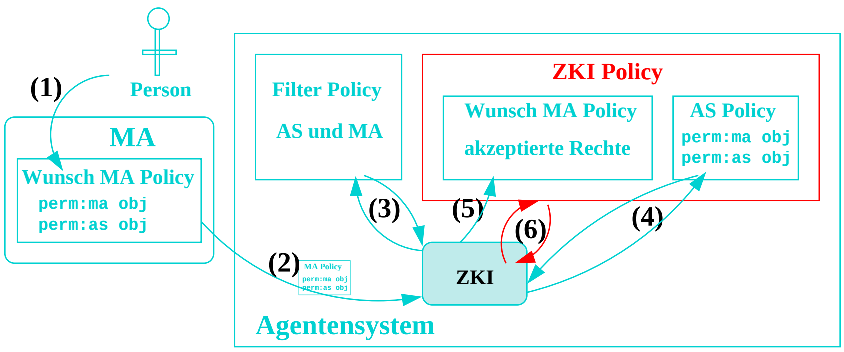


Abbildung 37: Verknüpfung der Wunsch-MA-Policy

(1) In der Wunsch-MA-Policy können Zugriffe auf Objekte des Agentensystems und der Agenteninstanzen autorisiert werden.
(2) Migriert nun die Agenteninstanz auf ein Agentensystem, wird dem Agentensystem die Wunsch-MA-Policy übergeben, um die darin enthaltenen Zugriffsrechte durchzusetzen.
(3,4) Zuerst kontrolliert die zentrale Kontrollinstanz (ZKI) die Integrität und die Signatur der Wunsch-MA-Policy. Danach überprüft die ZKI auf Basis der AS-Policy und der Filter-Politiken, welche Zugriffsrechte der Wunsch-MA-Policy durch die ZKI durchgesetzt werden können.

(5) Die vom Agentensystem akzeptierten Zugriffsrechte der Wunsch-MA-Policy bilden zusammen mit den Zugriffsrechten der AS-Policy, die vom Agentensystem durchzusetzende ZKI-Policy.
(6) Von der zentralen Kontrollinstanz (ZKI) werden die Zugriffsrechte der AS-Policy und die akzeptierten Zugriffsrechte der Wunsch-MA-Policy durchgesetzt.

Wunsch-MA-Policy enthält: Zugriffsrechte für Objekte des Agentensystems (AS-Obj)
Werden durch die Wunsch-MA-Policy Zugriffsrechte auf ein Objekt des Agentensystems *(AS-Obj)* vergeben, überprüft die ZKI die Zugriffsrechte, so dass nur solche Zugriffsrechte durchgesetzt werden, die den Sicherheitsbelangen des Agentensystems nicht widersprechen. Es werden nur Zugriffsrechte von der ZKI durchgesetzt, die nicht innerhalb der AS-Filter-Policy als unzulässig deklariert worden sind.

Wunsch-MA-Policy enthält: Zugriffsrechte für Objekte einer Agenteninstanz (MA-Obj)
Handelt es sich um Zugriffsrechte auf Objekte einer anderen Agenteninstanz *(MA-Obj)*, wird die MA-Filter-Policy der Agenteninstanz berücksichtigt. Die innerhalb der MA-Filter-Policy als unzulässig deklarierten Zugriffsrechte werden von der ZKI nicht durchgesetzt.

6.6.1 Einsatz des Filters

Durch den Filter Mechanismus können spezifische Zugriffsrechte, welche eine Agenteninstanz mittels der Wunsch-MA-Policy durchsetzen will, ausgeschlossen werden.

Filter Politik des Agentensystems
Durch den Filter besitzt das Agentensystem einen Mechanismus, der die Vergabe der Zugriffsrechte auf Objekte des Agentensystems durch die Integration der Wunsch-MA-Policy einer Agenteninstanzen beschränkt.
Einträge innerhalb der Filter-Policy werden in der Regel ausschließlich von berechtigten Personen vorgenommen. Dies ist in erster Linie der Administrator des Agentensystems auf dem der Filter eingesetzt wird. Der Administrator hat jedoch auch die Möglichkeit Zugriffsrechte auf die Filter-Policy zu vergeben, die einem weiteren Subjekt erlauben, Änderungen vorzunehmen. Beispielsweise könnte so einer spezifischen Agenteninstanz, die als Administrator tätig ist, ein solches Zugriffsrecht zugeteilt werden.

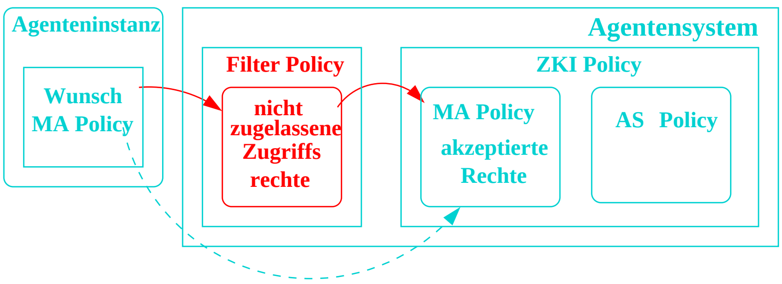


Abbildung 38: Filter des Agentensystems

Filter Politik der Agenteninstanz
Da eine Agenteninstanz ebenfalls Zugriffsrechte innerhalb ihrer Wunsch-MA-Policy vergeben kann, die die Objekte einer weiteren Agenteninstanz betreffen, muss die zentrale Kontrollinstanz, um die Informationssicherheit der weiteren Agenteninstanzen zu bewahren, die Filter-Policy der jeweiligen Agenteninstanz, auf die die Zugriffsrechte zum Zugriff berechtigen, berücksichtigen.

Ein Zugriffsrecht einer Wunsch-MA-Policy auf ein Objekt einer "fremden" Agenteninstanz wird nur dann durchgesetzt, wenn die Agenteninstanz des Objekts innerhalb der Filter-Policy die Durchsetzung des Zugriffsrechts zulässt.

6.6.2 Kollisionen der Zugriffsrechte MA und AS

Die in [ROE 99] Kapitel 5.8.2 aufgezeigte Problematik der Kollisionen gestaltet sich durch dieses Rechtekonzept etwas anders.

Eine Kollision kann auftreten

- Falls in der Wunsch-MA-Policy Zugriffsrechte enthalten sind, die das Agentensystem nicht zulassen will
- Falls in der Wunsch-MA-Policy einer Agenteninstanz Zugriffsrechte bezüglich Objekten einer Agenteninstanz “B” enthalten sind, die die Agenteninstanz “B” nicht zulassen will.

Durch den Einsatz der AS-Filter-Policy ist ausgeschlossen, dass die zentrale Kontrollinstanz (ZKI) unzulässige Zugriffsrechte einer Wunsch-MA-Policy durchsetzt. Die Autonomieanforderungen des Agentensystems werden somit erfüllt.

Durch die MA-Filter-Policy einer Agenteninstanz ist ebenso ausgeschlossen, dass Zugriffsrechte bezüglich Objekten einer Agenteninstanz “A” durchgesetzt werden, die von der Agenteninstanz “A” als unzulässig deklariert worden sind.

Die Überprüfung, ob ein Zugriffsrecht von der zentralen Kontrollinstanz (ZKI) durchgesetzt wird, übernimmt ebenfalls die ZKI.

6.6.3 Beschreibung der Autorisationsabläufe

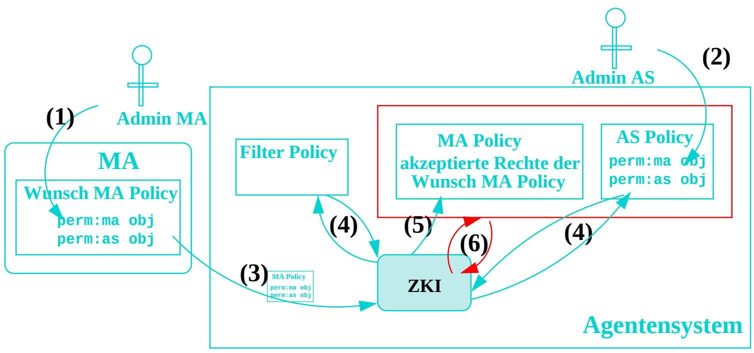


Abbildung 39: Autorisationszusammenhänge

Die in der Abb. 39 dargestellte Grafik beschreibt die Autorisationszusammenhänge.
(1) Das Agentensystem autorisiert durch seine AS-Policy Subjekte auf Objekte des Agentensystems oder einer Agenteninstanz auf eine bestimmter Art und Weise zuzugreifen. Einträge in die AS-Policy können über autorisierte Personen erfolgen.
(2) Eine Agenteninstanz autorisiert durch seine Wunsch-MA-Policy ebenso Subjekte auf Objekte des Agentensystem bzw. einer Agenteninstanz zuzugreifen. Die Wunsch-MA-Policy wird dabei von dem Eigentümer der Agenteninstanz erstellt.
(3) Migriert die Agenteninstanz auf ein Agentensystem, übergibt sie dem Agentensystem die Wunsch-MA-Policy. Die zentrale Kontrollinstanz (ZKI) des Agentensystems überprüft die Integrität der Wunsch-MA-Policy und die Signatur.

(4) Danach überprüft die ZKI, ob die Agenteninstanz die in der Wunsch-MA-Policy enthaltenen Zugriffsrechte vergeben darf. Die Überprüfung basiert auf Grundlage der AS-Policy und der Filter-Politiken.
(5) Die zugelassenen Zugriffsrechte werden letztendlich von der zentralen Kontrollinstanz (ZKI) bei der Zugriffskontrolle mitberücksichtigt und durchgesetzt.
(6) Greift nun ein Subjekt auf ein Objekt des Agentensystems zu, überprüft die ZKI, ob das benötigte Zugriffsrecht vorhanden ist. Dabei werden die Zugriffsrechte der AS-Policy und die zugelassenen Zugriffsrechte der Wunsch-MA-Policy berücksichtigt.

Die akzeptierten Zugriffsrechte der Wunsch-MA-Policy werden bis zur Beendigung bzw. einer Migration der Agenteninstanz auf ein anderes Agentensystem von der zentralen Kontrollinstanz (ZKI) des Agentensystems berücksichtigt und durchgesetzt.

6.7 Gültigkeitsdauer der Zugriffsrechte

AS-Policy
Zugriffsrechte, welche das Agentensystem vergibt, sind solange gültig, bis sie aus der AS-Policy entfernt werden.

Wunsch-MA-Policy
Vom Agentensystem zugelassene Zugriffsrechte, welche eine Agenteninstanz mittels der Wunsch-MA-Policy vergibt, sind über den gesamten Zeitraum, während sich die Agenteninstanz auf dem Agentensystem befindet, gültig. Terminiert die Agenteninstanz bzw. migriert die Agenteninstanz auf ein anderes Agentensystem, werden die akzeptierten Zugriffsrechte aus der Policy entfernt.

Enthält ein Wunsch-Zugriffsrecht ein Verfallsdatum (`time`), so ist das entsprechende Zugriffsrecht nur bis zum Eintreffen des Datums gültig.

6.8 Rücknahme der Zugriffsrechte

Die Problematik der Rücknahme von Rechten erfordert die differenzierte Betrachtung der Rücknahme von Zugriffsrechten der ZKI-Policy (AS-Policy und akzeptierte Zugriffsrechte der Wunsch-MA-Policies) und der Zugriffsrechte der Wunsch-MA-Policy einer Agenteninstanz.

1) Rücknahme der Zugriffsrechte der AS-Policy:

a) Zugriffsrechte für Objekte des Agentensystems (Vergabe vom AS)
Die Rücknahme der Zugriffsrechte, die die Objekte des Agentensystems betreffen, ist denkbar einfach. Soll ein Zugriffsrecht zurückgenommen werden, wird es einfach aus der AS-Policy entfernt. Damit ist dieses Zugriffsrecht nicht mehr innerhalb der ZKI-Policy enthalten und wird somit auch nicht mehr durchgesetzt.

Zu beachten ist, falls das Agentensystem z.B. Zugriffe auf ein bestimmtes Verzeichnis nicht mehr gestatten will, es nicht reicht, das betreffende Zugriffsrecht, welches Zugriffe auf dieses Verzeichnis zulässt, aus der AS-Policy zu entfernen. Es können weiter Zugriffsrechte in der AS-Policy enthalten sein, die explizit das Zugriffsrecht auf ein Unterverzeichnis dieses Verzeichnisses beinhalten und damit weiter Zugriffe auf dieses Verzeichnis ermöglichen. Zudem besteht die Möglichkeit, dass das Zugriffsrecht erneut durch die Integration von Zugriffsrechten einer Wunsch-MA-Policy in die ZKI-Policy aufgenommen und durchgesetzt wird. In einem solchen Fall muss darauf geachtet werden, dass die Filter-Policy die erneute Integration des Zugriffsrechts unterbindet.

Die spezifischen Zugriffsrechte müssen innerhalb der AS-Policy gesucht und entfernt werden. Um eine sichere Rücknahme bestimmter Zugriffsrechte zu ermöglichen, sollte die AS-Policy klar und unmissverständlich interpretierbar sein. Dies setzt eine geeignete Strukturierung voraus. Ein entsprechendes Tool könnte die Einträge geeignet repräsentieren, das z.B. alle Zugriffsrechte, welche den Zugriff auf das Dateisystem erlauben, ausgeben kann.

Wurden alle Zugriffsrechte entfernt, kann durch den Filter-Mechanismus ein Zugriffsrecht als unakzeptabel deklariert werden wodurch das Zugriffsrecht nicht mehr durch eine Wunsch-MA-Policy integriert werden kann.

b) Zugriffsrechte auf Objekte einer Agenteninstanz (Vergabe AS)
Das Agentensystem vergibt in der Regel keine Zugriffsrechte für Objekte einer Agenteninstanz. Zugriffsrechte auf Agenteninstanz-Methoden werden normalerweise von der Agenteninstanz durch die Integration der Wunsch-MA-Policy in die ZKI-Policy vorgenommen. Hat das Agentensystem dennoch ein solches Zugriffsrecht vergeben, gestaltet sich die Rücknahme ebenso durch das Entfernen des Zugriffsrechts aus der AS-Policy.

c) Zugriffsrechte für Objekte des Agentensystems (Vergabe vom MA)
Zugriffsrechte auf Objekte des Agentensystems, die durch die Integration der Wunsch-MA-Policy in die ZKI-Policy aufgenommen wurden, werden bei der Migration der Agenteninstanz auf ein anderes Agentensystem bzw. bei der Beendigung der Agenteninstanz wieder aus der ZKI-Policy entfernt.

d) Zugriffsrechte für Objekte eines Agenten (Vergabe MA)
Zugriffsrechte auf Objekte von Agenteninstanzen, welche durch die Integration der Wunsch-MA-Policy vergeben wurden, werden nur solange durchgesetzt, wie sich die Agenteninstanz auf dem Agentensystem befindet.

2) Rücknahme der Zugriffsrechte der Wunsch-MA-Policy:
Die in der Wunsch-MA-Policy enthaltenen Zugriffsrechte können einer Agenteninstanz nicht entzogen bzw. auch nicht verändert werden. Sie bleiben solange erhalten, bis die Agenteninstanz beendet wird. Die Gültigkeitsdauer eines Zugriffsrechts innerhalb der Wunsch-MA-Policy kann durch ein Datum beschränkt werden. Das Datum gibt dabei an, bis wann das Zugriffsrecht gültig ist. Aussagen, wer wann Zugriff auf welches Objekt hat, sind somit über dieses Zeitfenster möglich. Die Rücknahme der Zugriffsrechte einer Wunsch-MA-Policy muss somit im Voraus bestimmt werden. Um innerhalb der Wunsch-MA-Policy Änderungen bezüglich der Zugriffsrechte vorzunehmen, muss die Agenteninstanz auf ihr Home-Agentensystem migrieren. Nach der Modifikation muss die Wunsch-MA-Policy wiederum durch den Eigentümer der Agenteninstanz signiert werden.

Einschränkungen bezüglich der Rücknahme der Berechtigungen
Wird eine Berechtigung zurückgenommen während ein Zugriff stattfindet, so hat die Rücknahme für diesen Zugriff keine Auswirkungen.

6.9 Delegation von Zugriffsrechten der Agenteninstanzen

Die Delegation von Zugriffsrechten ist für bestimmte Szenarien wünschenswert. Durch die Delegation soll es Subjekten ermöglicht werden, Zugriffsrechte auf Objekte zu erlangen, die sie vorher nicht hatten.

Die Delegation muss jedoch ebenfalls kontrollierbar sein, so dass die Informationssicherheit nicht gefährdet ist. Die Delegation der Zugriffsrechte ist auf drei Arten möglich:

1. Delegation durch explizite Berechtigung innerhalb der Wunsch-MA-Policy
2. Delegation durch Vertrauen des Agentensystems
3. Delegation durch explizite Weitergabe der Berechtigung durch ein Agentensystem

1. Explizite Berechtigung
Die Delegation der Rechte in MASA, also die Weitergabe von Zugriffsrechten von Agenteninstanzen an Agenteninstanzen, kann durch die Weitergabe der Wunsch-MA-Policy von einer Agenteninstanz an eine weitere Agenteninstanz realisiert werden.
Die Agenteninstanz, welche die Wunsch-MA-Policy erhalten hat, kann sie dem Agentensystem zur Integration vorlegen und so die innerhalb der Wunsch-MA-Policy vergebenen Zugriffsrechte durchsetzen, wenn sie von dem Agentensystem akzeptiert werden.
Ein Eintrag innerhalb der Wunsch-MA-Policy kann Zugriffsrechte enthalten, die die Agenteninstanz, die die Wunsch-MA-Policy empfängt, explizit adressiert. Ebenso können Zugriffsrechte enthalten sein, die spezifische Eigenschaften der Agenteninstanz adressiert (z.B. die Agentengattung), so dass die Agenteninstanz durch die Integration der Wunsch-MA-Policy die ihr in der Wunsch-MA-Policy zugeteilten Zugriffsrechte erlangt.

Ist beispielsweise die Agenteninstanz [ma-1] nicht autorisiert den Netzverkehr auf dem Agentensystem des Hersteller zu messen, kann eine weitere Agenteninstanz [ma-2] durch ihre Wunsch-MA-Policy ein Zugriffsrecht vergeben, welches die Agenteninstanz für den Zugriff berechtigt. Somit kann der Agenteninstanz das Messen des Netzverkehrs ermöglicht werden.

Die zentrale Agenteninstanz kontrolliert in diesem Fall ob das Zugriffsrecht durchgesetzt werden kann. Dabei werden Sicherheitsvorkehrungen der Agenteninstanz und des Agentensystems berücksichtigt.

In der Wunsch-MA-Policy können die Zugriffsrechte der Agenteninstanz und der für die Erledigung der Aufgabe involvierten weiteren Agenteninstanzen vergeben werden.

2. Delegation durch Vertrauen des Agentensystems
Migriert eine Agenteninstanz [ma-1] auf ein Agentensystem, ist der Anwender der Agenteninstanz bekannt. Dies kann z.B. eine weitere Agenteninstanz [ma-2] sein. Greift nun die Agenteninstanz [ma-2] auf eine Ressource zu, wozu jedoch nur die Agenteninstanz [ma-1] berechtigt wäre, kann das Agentensystem den Zugriff gewähren lassen. Die Gewährung hängt von der Zugriffskontrollpolitik des Agentensystems und dem Vertrauen zu der Agenteninstanz ab.

Um die Vergabe der Zugriffsrechte der Agenteninstanzen nach dem Least-Privilege Prinzip zu vergeben, kann die Wunsch-MA-Policy in Abschnitte der Agentensysteme unterteilt werden. Jeder Abschnitt wird dabei nach einem Agentensystem benannt, welches nur die in diesem Abschnitt vorhandenen Zugriffsrechte bei der Integration berücksichtigt.

3. Delegation der Zugriffsrechte durch explizite Weitergabe durch das Agentensystem
Die letzte Form der Delegationsmöglichkeit ist die Autorisation einer Agenteninstanz durch das Agentensystem. Beispielsweise kann es vorkommen, dass ein Agentensystem selbst nicht in der Lage ist, die von der Agenteninstanz geforderte Aufgabe zu bearbeiten, jedoch ein weiteres Agentensystem konsultieren kann, welches dazu in der Lage ist. Das Agentensystem kann daher die Migration der Agenteninstanz auf das entsprechende Agentensystem veranlassen. Damit die Agenteninstanz auf diesem Agentensystem ebenfalls über die notwendigen Berechtigungen verfügt, kann nun das Agentensystem eine zusätzliche Wunsch-MA-Policy für die Agenteninstanz erzeugen. Diese zusätzliche Wunsch-MA-Policy enthält dabei die Berechtigung, die die Agenteninstanz benötigt, um auf dem weiteren Agentensystem die Aufgabe erledigen zu können. Damit das weitere Agentensystem erkennt, dass es sich quasi um eine Delegation der Aufgabenerledigung handelt, wird die zusätzliche Wunsch-MA-Policy von dem Agentensystem, welches die Aufgabe delegiert, mit dem Private Key des Agentensystems unterzeichnet.

Die zusätzliche Wunsch-MA-Policy enthält neben den Zugriffsrechten den Namen der Agenteninstanz.

6.10 Verwaltungsstruktur der Zugriffskontrollpolitiken

Die Zugriffskontrollpolitiken müssen noch einer geeigneten Verwaltungsstruktur untergeordnet werden. Die Verwaltungsstruktur legt fest, wer Änderungen an den jeweiligen Zugriffskontrollpolitiken vornehmen darf. Die Änderungen betreffen dabei die Vergabe, Rücknahme und das Löschen von Zugriffsrechten.

Zu unterscheiden sind dabei Änderungen durch das Agentensystem und Änderungen, die durch die Verknüpfung einer Wunsch-MA-Policy erfolgen. Ebenfalls unterschieden werden muss, ob es sich bei den Zugriffsrechten um Berechtigung für den Zugriff auf Objekte des Agentensystems oder um Objekte einer Agenteninstanz handelt.

1) Änderungen durch das Agentensystem

a) betreffend der AS-Objekte
Das Agentensystem bestimmt innerhalb der AS-Policy, wer berechtigt ist, Änderungen bezüglich der Zugriffsrechte für Objekte des Agentensystems innerhalb der AS-Policy vorzunehmen. Enthält die AS-Policy kein Zugriffsrecht, welches einem Subjekt erlaubt, Änderungen an der AS-Policy vorzunehmen, so darf nur der Eigentümer des Agentensystems Änderungen an der AS-Policy vornehmen.

b) betreffend der MA-Objekte
Das Agentensystem vergibt keine Zugriffsrechte auf Objekte von Agenteninstanzen. Die Agenteninstanz alleine vergibt Zugriffsrechte auf ihre Objekte. Die Zugriffsrechte sind in der Wunsch-MA-Policy enthalten.

2) Änderung durch die Integration der Wunsch-MA-Policy

a) betreffend der AS-Objekte
Zugriffsrechte, welche die Objekte des Agentensystems betreffen, können nur von einer durch ihre Signatur berechtigten Wunsch-MA-Policy vorgenommen werden. Zudem durchlaufen diese Zugriffsrechte den Filter des Agentensystems, wobei dadurch die Restriktionen des Agentensystems beachtet werden.

b) betreffend der MA-Objekte
Sollen Zugriffsrechte durchgesetzt werden, welche sich auf Objekte einer Agenteninstanz beziehen, so dürfen in erster Linie nur Zugriffsrechte auf Objekte der eigenen Agenteninstanz vergeben werden. Zugriffsrechte bezüglich Objekten anderer Agenteninstanzen werden erst durchgesetzt, nachdem die jeweilige MA-Filter-Policy berücksichtigt wurde.

Explizite Zugriffsrechte für Zugriffskontrollpolitiken
Um Änderungen der Zugriffskontrollpolitiken kontrollieren zu können, werden Zugriffsrechte, die das Modifizieren der AS-Policy, der Wunsch-MA-Policy und der Filter-Policy erlauben, eingeführt. Eine Änderung der Zugriffskontrollpolitiken ist jeweils nur dann gestattet, wenn das notwendige Zugriffsrecht vorhanden ist.
Die Zugriffsrechte sind ebenfalls feingranular adressierbar, damit beispielsweise einer Agenteninstanz erlaubt werden kann, nur Zugriffsrechte zur Änderung der Zugriffskontrollpolitik bezüglich ihrer Objekte zu vergeben.

Zugriffsrecht: Änderung der AS-Policy (betreffend AS-Objekte) durch eine Person
Das Zugriffsrecht zur Änderung der Zugriffsrechte in der AS-Policy, die Objekte des Agentensystems enthalten, hat die Form:

```
permission AsAccessPolicyAsPermission ("target", "action")
```

Folgende Aktionen sind möglich:
appendPermission, deletePermission, changePermission

Das Target bezeichnet dabei das Objekt, für das die Zugriffsrechte vergeben werden dürfen.

Zugriffsrecht: Änderung der AS-Policy (betreffend AS-Objekte) durch MA
Änderungen können nur vorgenommen werden, wenn die AS-Filter-Policy dies zulässt.

Zugriffsrechte: Änderung der AS-Policy (betreffend MA-Objekte) durch AS
Das Agentensystem vergibt keine Zugriffsrechte, welche Zugriffe auf Objekte von Agenten autorisieren.

Zugriffsrechte: Änderung der AS-Policy (betreffend MA-Objekte) durch MA
Änderungen können nur vorgenommen werden, wenn die zuständige MA-Filter-Policy dies zulässt.

6.11 Rechtedurchsetzung

Für die Durchsetzung der in der Zugriffskontrollpolitik des Endsystems vergebenen Zugriffsrechte ist das jeweilige Betriebssystem zuständig. Die Zugriffskontrollpolitiken der Agenteninstanzen und des Agentensystems werden von der zentralen Kontrollinstanz (ZKI) des Agentensystems durchgesetzt, auf dem sich das Objekt, auf das zugegriffen wird, befindet.

Die zentrale Kontrollinstanz (ZKI) überprüft vor dem eigentlichen Zugriff, ob das zugreifende Subjekt die notwendige Berechtigung für den Zugriff besitzt. Hierfür werden die ZKI-Policy nach dem spezifischen Zugriffsrecht durchsucht. Wird kein entsprechendes Zugriffsrecht gefunden, wird der Zugriff nicht erlaubt.

Die ZKI-Policy wird dabei wie folgt durchsucht:

1. Suche nach dem authentisierten Subjekt.
2. Wurde das passende Subjekt gefunden, wird nach dem Objekt gesucht, auf das zugegriffen wird.
3. Ist das passende Objekt gefunden worden, wird nach der passenden Aktion und ihrer Parameter gesucht.

Angaben innerhalb der Wunsch-MA-Policy, welches Agentensystem die Zugriffsrechte durchsetzen soll, werden dabei berücksichtigt.

6.12 Zugriffskontrolle anhand von Beispielszenarios

6.12.1 Beispielszenario aus Kapitel 1

Die Funktionsweise des vorgestellten Rechtekonzepts soll nun anhand des Beispielszenarios aus Kapitel 1 erläutert werden.

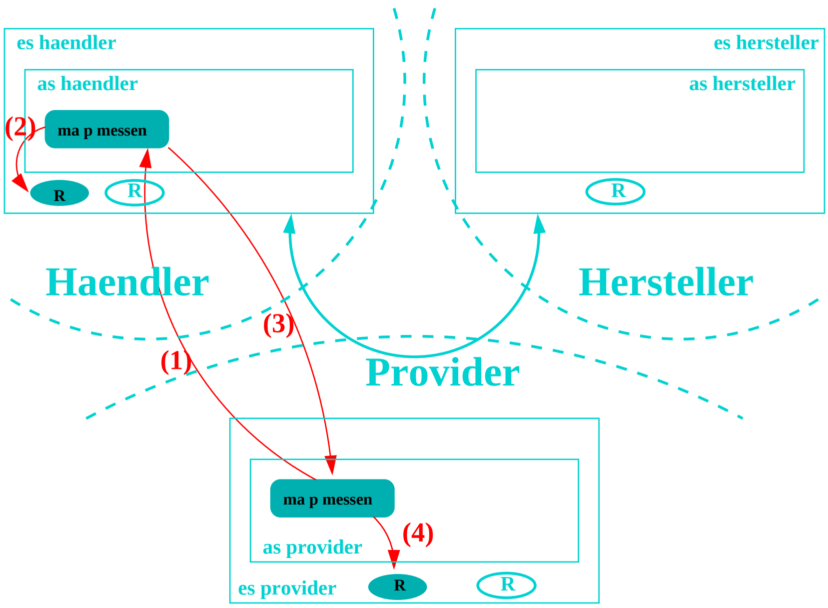


Abbildung 40: Beispielszenario aus Kapitel 1

Der mobile Agent des Providers *ma-p-messen* migriert auf das Agentensystem des Händlers *as-haendler* (1) und nimmt dort Messungen des Netzdurchsatzes vor (2). Der mobile Agent *ma-p-messen* kehrt auf das Agentensystem *as-provider* zurück (3), um dort die Resultate der Messungen in das Verzeichnis `/results/` abzuspeichern (4).

Die in den Beispielszenarios getätigten Zugriffe werden mithilfe der in Kapitel 6.4 definierten Modellierungssprache modelliert.

Folgende Zugriffe finden statt :

```
(Zugriff: 1)
[as: as.provider.provider.mnm](
(subject, as.haendler.haendler.mnm) ->
(object, ma.provider.provider.mnm.messe.(as.provider.provider.mnm)) |
(action, migrateTo(as.haendler.haendler.mnm)))
(Zugriff: 2)
[as: as.haendler.haendler.mnm](
(subject, ma.provider.provider.mnm.messe.(as.provider.provider.mnm,
                                          as.haendler.haendler.mnm)) ->
(object, socket-haendler-hersteller:4223) |
(action, durchsatzmessen))
(Zugriff: 3)
[as: as.haendler.haendler.mnm](
(subject, as.haendler.haendler.mnm) ->
(object, ma.provider.provider.mnm.messe.(as.provider.provider.mnm,
```

```
                                       as.haendler.haendler.mnm)) |
(action, migrateTo(as.provider.provider.mnm)))
(Zugriff: 4)
[as: as.provider.provider.mnm](
(subject, ma.provider.provider.mnm.messe.(as.provider.provider.mnm,
                                        as.haendler.haendler.mnm)) ->
(object, /results/erg-haendler) |
(action, write))
```

Benötigte Zugriffsrechte:
Betrachtet man die Modellierung der Zugriffe, so sind folgende Zugriffsrechte notwendig:

Zugriffsrechte die durch die ZKI des Agentensystems des Providers durchgesetzt werden müssen
Zugriffsrechte der Zugriffskontrollpolitik des Agentensystems [as.provider.provider.mnm]:

```
grant ma.provider.provider.mnm.[visit as.provider.provider.mnm] {
  permission masa.AsPermission ("as.haenlder.haendler.mnm", "migrateTo")
}                                                                     (1a)

grant ma.provider.provider.mnm.messe.[visit as.provider.provider.mnm,
                                     visit as.haendler.haendler.mnm] {
  permission masa.AsPermission ("as.provider.provider.mnm", "migrateFrom")
}                                                                     (3b)

grant ma.provider.provider.mnm.messe.[visit as.provider.provider.mnm
                                     visit as.haendler.haendler.mnm] {
  permission java.security.FilePermission ("/results/*", "write")
}                                                                     (4)
```

Zugriffsrechte die durch die ZKI des Agentensystems des Händler durchgesetzt werden müssen
Zugriffsrechte der Zugriffskontrollpolitik des Agentensystems [as.haendler.haendler.mnm]:

```
grant ma.provider.provider.mnm.messe.[visit as.provider.provider.mnm] {
  permission masa.AsPermission ("as.provider.provider.mnm", "migrateFrom")
}                                                                     (1b)

grant ma.provider.provider.mnm.messe.[visit as.provider.provider.mnm] {
  permission masa.AsPermission ("socket-h-hersteller:4223", "durchsatzMessen")
}                                                                     (2)

grant ma.provider.provider.mnm.messe.[visit as.provider.provider.mnm,
                                     visit as.haendler.haendler.mnm] {
  permission masa.AsPermission ("as.provider.provider.mnm", "migrateTo")
}                                                                     (3a)
```

Das Zugriffsrecht (1a) des Agentensystems des Providers gestattet der Agenteninstanz *ma-provider* auf das Agentensystem des Händlers zu migrieren. Im Gegensatz dazu gestattet das Zugriffsrecht (1b) des Agentensystems des Händlers, die Agenteninstanz *ma-provider* auszuführen. Das Zugriffsrecht (2) erlaubt zudem den Zugriff auf den Socket, auf welchem die Agenteninstanz die Aktion `durchsatzmessen` zum Messen des Durchsatzes durchführen darf. Die Zugriffsrechte (3a) und (3b) ermöglichen wieder die Rückkehr der Agenteninstanz zum Agentensystem des Providers, wobei das Zugriffsrecht abhängig von der History der Agenteninstanz ist.

Werden die oben genannten Zugriffsrechte durch die zentrale Kontrollinstanz (ZKI) des jeweiligen Agentensystems durchgesetzt, können alle Aktionen ausgeführt werden. Zudem ist durch die Betrachtung der History der Agenteninstanz sichergestellt, dass Zugriffsrechte nur an Agenteninstanzen vergeben werden, denen aufgrund ihrer Vergangenheit noch getraut werden kann. Damit kann ausgeschlossen werden, dass nicht vertrauenswürdige Agentensysteme die Agenteninstanz modifiziert haben.

Möglichkeiten der Autorisation durch die AS-Policy und der Wunsch-MA-Policy
Die notwendigen Zugriffsrechte können innerhalb der AS-Policy vergeben werden oder aber auch als Zugriffsrechte innerhalb der Wunsch-MA-Policy. Fehlen notwendige Zugriffsrechte innerhalb der AS-Policy, so können, abhängig von der AS-Filter-Policy des Agentensystems, die Zugriffsrechte der Wunsch-MA-Policy durch das Agentensystem durchgesetzt werden und somit die Bearbeitung der Aufgabe durch die Agenteninstanz ermöglichen.

Fehlen innerhalb der AS-Policy benötigte Zugriffsrechte und sind diese innerhalb der Wunsch-MA-Policy vergeben, ist die erfolgreiche Bearbeitung der Aufgabe der Agenteninstanz abhängig von den einzelnen Filter-Politiken. Nur die vom Agentensystem akzeptierten Zugriffsrechte werden von der ZKI durchgesetzt.

Sind die benötigten Zugriffsrechte innerhalb der AS-Policy vorhanden, ist die Bearbeitung der Aufgabe durch die Agenteninstanz sichergestellt. Die spezifische Vergabe der Zugriffsrechte innerhalb der AS-Policy bedeutet jedoch einen höheren Aufwand, da der Eintrag des benötigten Zugriffsrechts bei jedem Agentensystem, auf dem die Agenteninstanz die Aufgabe bearbeiten muss, zu erfolgen hat.

Durch die unterschiedlichen Autorisationsmöglichkeiten kann ein Kompromiss zwischen Informationssicherheit, Verwaltungsaufwand und Funktionalität getroffen werden.

Ein Agentensystem kann beispielsweise die jeweiligen Zugriffsrechte der Wunsch-MA-Policies überhaupt nicht betrachten und Zugriffe nur auf Basis der AS-Policy überprüfen. Hier wird also keiner Agenteninstanz getraut, wodurch die Funktionalität der Agenteninstanzen stark eingeschränkt sein kann.
Andererseits kann ein Agentensystem auch allen Agenteninstanzen trauen und jedes Zugriffsrecht der Wunsch-MA-Policy akzeptieren. Damit ist die Funktionalität der Aufgabenbearbeitung einer jeden Agenteninstanz sichergestellt. Jedoch besteht hier die Möglichkeit der Durchsetzung eines für das Agentensystem kritischen Zugriffsrechts.

Das Agentensystem kann die Vergabe spezifischer Zugriffsrechte durch den Einsatz des Filters mittels der AS-Filter-Policy unterbinden.

Das Zugriffsrecht (2) auf dem Agentensystem des Händlers stellt beispielsweise ein für ein Agentensystem kritisches Zugriffsrecht dar, während die Zugriffsrechte (1b) und (3a) in der Regel weniger kritisch für das Agentensystem sind.
Der Zugriff auf ein Socket sollte aus Sicht des Agentensystems nicht jede Agenteninstanz erlangen können. Daher ist es wahrscheinlich, dass diese Zugriffsrechte beispielsweise durch die AS-Filter-Policy unterbunden werden und so nur explizit durch das Agentensystem mittels der AS-Policy vergeben werden können.

Doch gerade bei expliziten Zugriffsrechten, die beispielsweise nur ein Subjekt adressieren, ist der Verwaltungsaufwand relativ hoch und bei zunehmender Anzahl der Subjekte schnell unpraktikabel. Hier empfiehlt sich der Einsatz der Wunsch-MA-Policy zur Vergabe der notwendigen Zugriffsrechte.

Aus Sicht des Agentensystems können spezifische Wunsch-MA-Policies entsprechend autorisiert und somit auch durchgesetzt werden.

6.12.2 Erweitertes Beispielszenario aus Kapitel 1

Das Beispielszenario wird zusätzlich erweitert, um die Möglichkeit der Rechtedelegation zu erläutern.

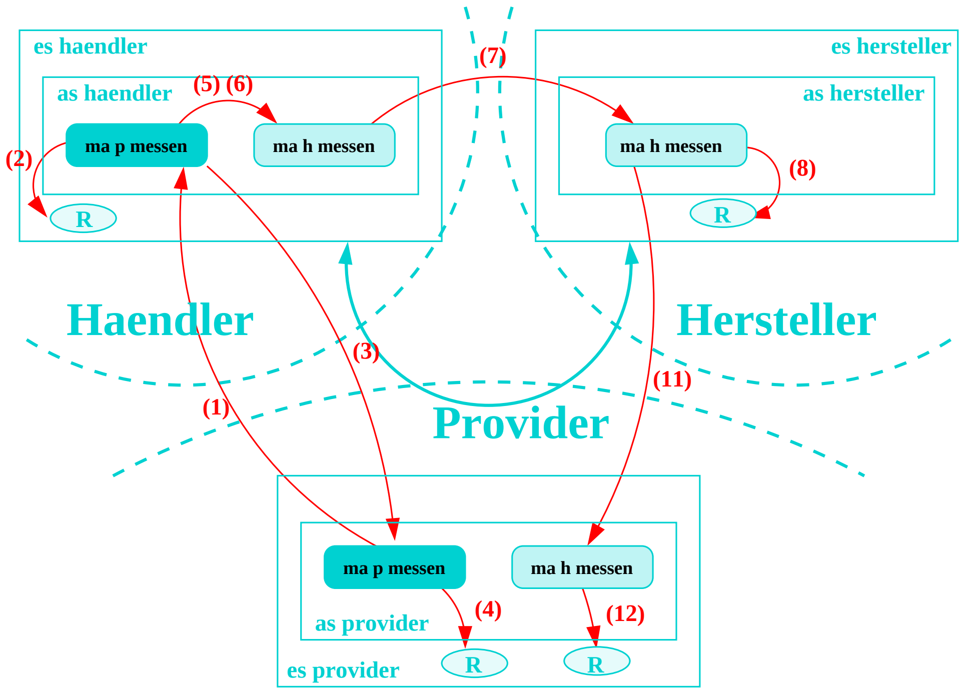


Abbildung 41: Beispielszenario

Die Agenteninstanz des Providers *ma-p-messen* migriert auf das Agentensystem des Händlers *as-haendler* (1) und nimmt dort Messungen des Netzdurchsatzes vor (2). Die Agenteninstanz *ma-p-messen* erteilt dem Agentensystem *as-haendler* den Auftrag eine Agenteninstanz der Gattung *messen* zu erzeugen (5), die den Netzdurchsatz auf dem Agentensystem *as-hersteller* messen soll (6). Hierfür muss eine Agenteninstanz des Händlers *ma-h-messen* auf das Agentensystem des Herstellers *as-hersteller* migrieren (7) und dort ebenfalls Messungen vornehmen (8). Die Agenteninstanz *ma-p-messen* kehrt auf das Agentensystem *as-provider* zurück (3), um dort die Resultate der Messungen in das Verzeichnis `/results/` abzuspeichern (4). Die Agenteninstanz des Händlers *ma-h-messen* migriert dann ebenfalls nach der Beendigung der Messung auf das Agentensystem des Providers *as-provider* (9) und speichert die Ergebnisse ebenfalls in das Verzeichnis `/results/` (10).

Folgende Zugriffe finden statt:

```
(1)
[as: as.provider.provider.mnm](
(subject, as.haendler.haendler.mnm) ->
(object, ma.provider.provider.mnm.messe.(as.provider.provider.mnm)) |
(action, migrateTo(as.haendler.haendler.mnm)))
(2)
[as: as.haendler.haendler.mnm](
(subject, ma.provider.provider.mnm.messe.(as.provider.provider.mnm,
```

```
                                          as.haendler.haendler.mnm)) ->
(object, socket-haendler-hersteller:4223) |
(action, durchsatzmessen))
(3)
[as: as.haendler.haendler.mnm](
(subject, as.haendler.haendler.mnm) ->
(object, ma.provider.provider.mnm.messe.(as.provider.provider.mnm,
                                         as.haendler.haendler.mnm)) |
(action, migrateTo(as.provider.provider.mnm)))
(4)
[as: as.provider.provider.mnm](
(subject, ma.provider.provider.mnm.messe.(as.provider.provider.mnm,
                                          as.haendler.haendler.mnm)) ->
(object, /results/erg-haendler) |
(action, write))
(5)
[as: as.haendler.haendler.mnm](
(subject, ma.provider.provider.mnm.messe.(as.provider.provider.mnm,
                                          as.haendler.haendler.mnm)) ->
(object, as.haendler.haendler.mnm) |
(action, createAgent(ma.haendler.haendler.mnm.messe)))
(6)
[as: as.haendler.haendler.mnm](
(subject, ma.provider.provider.mnm.messe.(as.provider.provider.mnm,
                                          as.haendler.haendler.mnm)) ->
(object, ma.haendler.haendler.mnm.messe.(as.haendler.haendler.mnm)) |
(action, auftrag(messung(as.hersteller.hersteller.mnm))))
(7)
[as: as.haendler.haendler.mnm](
(subject, as.haendler.haendler.mnm) ->
(object, ma.haendler.haendler.mnm.messe.(as.haendler.haendler.mnm)) |
(action, migrateTo(as.hersteller.hersteller.mnm))
(8)
[as: as.hersteller.hersteller.mnm](
(subject, ma.haendler.haendler.mnm.messe.(as.haendler.haendler.mnm,
                                          as.hersteller.hersteller.mnm)) ->
(object, socket-hersteller-haendler:4223) |
(action, durchsatzmessen))
(9)
[as: as.hersteller.hersteller.mnm](
(subject, as.hersteller.hersteller.mnm) ->
(object, ma.haendler.haendler.mnm.messe.(as.haendler.haendler.mnm,
                                         as.hersteller.hersteller.mnm)) |
(action, migrateTo(as.provider.provider.mnm))
(10)
[as: as.provider.provider.mnm](
(subject, ma.haendler.haendler.mnm.messe.(as.haendler.haendler.mnm,
                                          as.hersteller.hersteller.mnm)) ->
(object, /results/erg-hersteller) |
(action, write))
```

Benötigte Zugriffsrechte die durch die ZKI der Agentensysteme durchgesetzt werden müssen

Um die oben aufgeführten Zugriffe durchführen zu dürfen, müssen die folgenden Zugriffsrechte durch die ZKI der einzelnen Agentensysteme durchgesetzt werden. Es werden die minimal benötigen Zugriffsrechte der Agentensysteme aufgeführt (Least-Privilege Prinzip). Die History der Agenteninstanzen wird jedoch nicht betrachtet. Die Zugriffsrechte enthalten deshalb als Eintrag (`visit *`), was zum Ausdruck bringt, dass der Agent vorher beliebige Agentensysteme besucht haben darf.

ZKI-Policy des Agentensystems: as.provider.provider.mnm

```
grant ma.provider.provider.mnm.[visit *] {
  permission masa.AsPermission ("as.haendler.haendler.mnm", "migrateTo")
}                                                                    (1a)

grant ma.provider.provider.mnm.messe.[visit *] {
  permission java.secuity.FilePermission ("/results/*", "write")
}                                                                    (4)

grant ma.provider.provider.mnm.messe.[visit *] {
  permission masa.AsPermission ("as.haendler.haendler.mnm", "migrateFrom")
}                                                                    (3b)

grant ma.haendler.haendler.mnm.messe.[visit *] {
  permission masa.AsPermission ("as.hersteller.hersteller.mnm", "migrateFrom")
}                                                                    (9b)

grant ma.haendler.haendler.mnm.messe.[visit *] {
  permission java.security.FilePermission ("/results/*", "write")
}                                                                    (10)
```

ZKI-Policy des Agentensystems: as.hersteller.hersteller.mnm

```
grant ma.haendler.haendler.mnm.[visit *] {
  permission masa.AsPermission ("as.haendler.haendler.mnm", "migrateFrom")
}                                                                    (7b)

grant ma.haendler.haendler.mnm.messe.[visit *] {
  permission masa.AsPermission ("socket-h-haendler:4223", "durchsatzMessen")
}                                                                    (8)

grant ma.haendler.haendler.mnm.[visit *] {
  permission masa.AsPermission ("as.provider.provider.mnm", "migrateTo")
}                                                                    (9a)
```

ZKI-Policy des Agentensystems: as.haendler.haendler.mnm

```
grant ma.provider.provider.mnm.messe.[visit *] {
  permission masa.AsPermission ("as.provider.provider.mnm", "migrateFrom")
}                                                                    (1b)

grant ma.provider.provider.mnm.messe.[visit *] {
  permission masa.AsPermission ("socket-h-hersteller:4223", "durchsatzMessen")
}                                                                    (2)

grant ma.provider.provider.mnm.messe.[visit *] {
```

```
  permission masa.AsPermission ("ma.haendler.haendler.mnm.messe.
                              [as.haendler.haendler.mnm]", "createAgent")
}                                                                     (5)

grant ma.provider.provider.mnm.messe.[visit *] {
  permission masa.agentMesse.MaPermission ("as.hersteller.hersteller.mnm",
                                         "durchsatzMessen")
}                                                                     (6)

grant ma.haendler.haendler.mnm.messe.[visit *] {
  permission masa.AsPermission ("as.hersteller.hersteller.mnm", "migrateTo")
}                                                                     (7a)

grant ma.provider.provider.mnm.messe.[visit *] {
  permission masa.AsPermission ("as.provider.provider.mnm", "migrateTo")
}                                                                     (3a)
```

Um die Möglichkeit der Weitergabe von Zugriffsrechten zu verdeutlichen, soll das Beispielszenario 1 aus Kapitel 1 betrachtet werden, wobei beim Agentensystem *as-hersteller* folgendes, zum Messen des Durchsatzes notwendiges Zugriffsrecht *(perm-ma-h-messen)* fehlt:

```
grant ma.haendler.haendler.mnm.messe.[visit *] {
  permission masa.AsPermission ("socket:4223", "durchsatzMessen")
}                                                                     (8)
```

Migriert die Agenteninstanz `ma.haendler.haendler.mnm.messe(as.haendler.haendler.mnm)` auf das Agentensystem `as.hersteller.hersteller.mnm` und versucht dort auf das Socket `socket:4223` zuzugreifen, um den Durchsatz zu messen, wird aufgrund des fehlenden Zugriffsrechts der Zugriff vom Agentensystem des Herstellers nicht zugelassen.

Nun können folgende Delegationsmöglichkeiten genutzt werden (vgl. Kapitel 6.9):

1. Delegation durch explizite Berechtigung innerhalb der Wunsch-MA-Policy
   Die Wunsch-MA-Policy der Agenteninstanz des Providers oder einer anderen Agenteninstanz enthält das benötigte Zugriffsrecht und migriert auf das entsprechende Agentensystem, um das Zugriffsrecht zu setzen. Ist der Signierer der Wunsch-MA-Policy berechtigt das Zugriffsrecht auf dem Agentensystem zu setzen, so kann es in die ZKI-Policy des Agentensystems integriert werden. Die Filter-Policy des Agentensystems muss hier die Durchsetzung des Zugriffsrechts zulassen.

2. Delegation durch Vertrauen des Agentensystems
   Das Agentensystem entscheidet anhand des Anwenders der Agenteninstanz, ob der Zugriff genehmigt wird. Hierfür muss innerhalb der AS-Policy die notwendige Berechtigung enthalten sein.

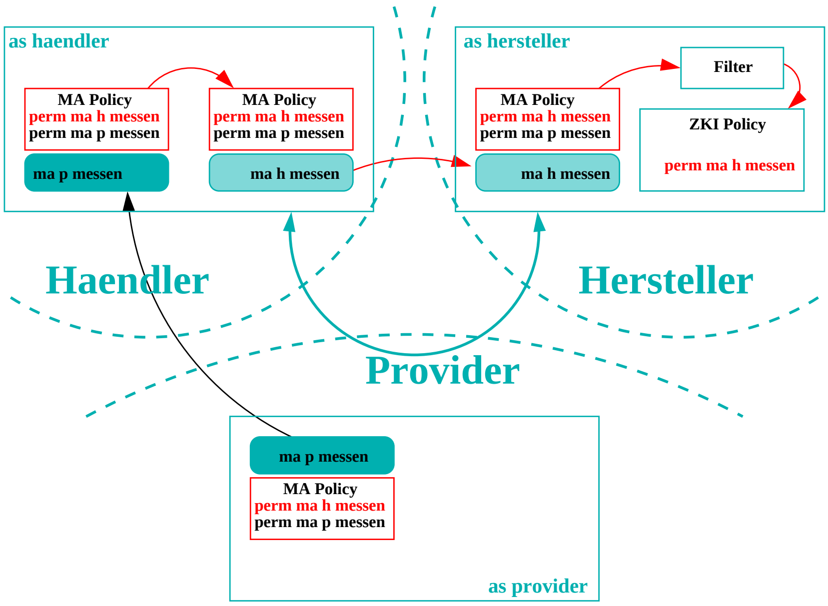


Abbildung 42: Delegation des Zugriffsrechts: perm-ma-h-messen

Beispiel eines Filter-Policy Eintrags, der die Integration nicht erlaubt:
In diesem Beispiel wird jeder Agenteninstanz verboten, das entsprechende Zugriffsrecht zu setzen.

```
\\
\\ Filter-Policy
\\
forbid ma.*.*.*.*.*.* {
 [
  grant signedBy "Netztester" {
   permission masa.AsPermission ("socket:4223", "durchsatzMessen")
  }
 ]
}
```

Beispiel eines AS-Policy Eintrags, der den Zugriff erlaubt:

```
\\
\\ AS-Policy
\\
grant ma.haendler.provider.*.*.[visit as.haendler.haendler.mnm] {
  permission masa.AsPermission ("socket:4223", "durchsatzMessen")
}
```

6.13 Zusammenfassung

Mit dem vorgestellten Rechtekonzept werden die folgenden Forderungen aus der Anforderungsanalyse erfüllt:

- Authentifikation:
  - Die Authentifikation der Entitäten ist sichergestellt.
- Zugriffsrechte:
  - Es werden feingranulare, objektspezifische Zugriffsrechte realisiert.
  - Durch die Namenstruktur der Zugriffsrechte ist sichergestellt, dass grobe Verwechslungen ausgeschlossen sind.
  - Die Zugriffsrechte sind systemweit gültig.
- Rechtevergabe:
  - Es können alle relevanten Subjekte und Objekte erfasst werden.
  - Die Vergabe der Zugriffsrechte kann anwendungsspezifisch erfolgen.
  - Die Zugriffskontrollpolitik des Agentensystems wird durchgesetzt.
  - Agenteninstanzen besitzen die Möglichkeit Wunsch-Zugriffskontrollpolitiken vom Agentensystem durchsetzen zu lassen.
  - Dabei werden die Autonomieanforderungen erfüllt.
  - Die Zugriffskontrollpolitik des Agentensystems erfüllt die dynamischen Anforderungen.
  - Dynamische Aspekte der Agenteninstanz-Zugriffskontrollpolitik werden beachtet.
  - Die Adressierung der Entitäten kann mit einer großen Anzahl sich dynamisch ändernder Subjekte und Objekte umgehen.
  - Autorisationen können an Bedingungen geknüpft werden.
  - Zugriffsrechte können zurückgenommen werden.
  - Die Eigentümer bestimmen über die Zugriffsrechte auf ihre Objekte.
  - Es besteht die Möglichkeit der Rechtedelegation.
- Rechtedurchsetzung:
  - Die Rechtedurchsetzung kann nicht umgangen werden.
  - Es werden alle Zugriffskontrollpolitiken durchgesetzt.
  - Informationen, die für die Entscheidungsfindung relevant sind, werden geschützt.
  - Jeder Zugriff, der nicht explizit erlaubt ist, wird unterbunden.

7 Implementierungskonzept

In diesem Kapitel werden die konkreten Techniken vorgestellt, die eine Umsetzung des entwickelten Modells ermöglichen.

7.1 Für MASA nutzbare Java Sicherheitsmechanismen

MASA ist vollständig in der Programmiersprache Java implementiert. Das Agentensystem wird auf einer JVM ausgeführt, wobei die Java Sicherheitsmechanismen für MASA eingesetzt werden können. Folgend werden Java Mechanismen vorgestellt, wie sie in der Standard Java 2 Security Architecture zum Einsatz kommen und in modifizierter Form für die Implementierung des Rechtekonzepts verwendet werden.

7.1.1 AccessController

Der AccessController wird herangezogen, um Entscheidungen zu treffen, ob ein Zugriff auf eine kritische Systemressource erlaubt oder unterbunden werden soll. Die Entscheidung basiert dabei auf Basis der System Security Policy (vgl. Kapitel 2.9.1).

Die `checkPermission()`-Methode des AccessControllers entscheidet, ob bei einem Zugriff die notwendige Berechtigung vorliegt. Der folgende Aufruf entscheidet beispielsweise, ob der lesende Zugriff auf die Temp-Datei erlaubt wird oder nicht:

```
FilePermission perm = new FilePermission("/temp/Datei", "read");
AccessController.checkPermission(perm);
```

Durch die System Security Policy können Zugriffsrechte auf spezifische Objekte vergeben werden. Zugriffsrechte können zur Laufzeit geändert werden. Damit die Änderung wirksam wird, muss das Policy-Objekt neuinitialisiert werden. Hierfür steht die `refresh()`-Methode des Policy-Objekts zur Verfügung.

Für die Bestimmung der Zugriffsrechte eines Programms, werden bei dem Einsatz der Default Policy Klasse PolicyFile die Einträge der System Security Policy durchsucht und mit der CodeSource des aufrufenden Programms verglichen. Die in den Einträgen, die mit der CodeSource übereinstimmen, vorhandenen Permission-Objekte, bilden das Permissions-Objekt der ProtectionDomain [OAK 00, GON 98, IBM 99b, ACC02].

CodeSource
Die CodeSource ist in der Standard Java Implementierung ein einfaches Objekt, welches lediglich die Herkunftsadresse der Klasse in Form einer URL und der Signatur der Klasse, falls vorhanden, widerspiegelt. Die SecureClassLoader-Klasse ist verantwortlich für die Erstellung und Manipulation des CodeSource-Objekts [COS02].

Während bei der Benutzung des URLClassLoaders jede URL ein unterschiedliches CodeSource-Objekt bekommt, besteht auch die Möglichkeit, dass jede Klasse ein eigenes CodeSource-Objekt besitzt, bzw. dass verschiedene Gruppen von Klassen derselben Herkunft (URL) verschiedene CodeSource-Objekte haben.

Permissions
Der Access Controller operiert auf Permission-Objekten. Ein Permission-Objekt ist eine Instanz der `Permission`-Klasse. Ein Permission-Objekt assoziiert mit einer Klasse repräsentiert die Berechtigungen der Klasse.

Der AccessController überprüft, ob das notwendige Permission-Objekt in der betreffenden ProtectionDomain vorhanden ist. Zur Überprüfung wird die `implies()`-Methode, die von dem betreffenden Permission-Objekt implementiert wird, herangezogen. Die `implies()`-Methode überprüft, ob ein Permission-Objekt das notwendige Permission-Objekt impliziert.

Bei dem folgenden Check-Aufruf überprüft der AccessController mittels der `implies()`-Methode der SocketPermission, ob der String "`host:6000, connect`" innerhalb der Policy vorhanden ist. Die Methode legt außerdem fest, welche weiteren String-Formen diese Berechtigung implizieren.

```
SocketPermission sp = new SocketPermission(host:6000, connect);
AccessController.checkPermission(sp);
```

Die Permissions enthalten für jeden Typ einer Permission (z.B. FilePermission) eine PermissionCollection. Die PermissionCollection enthält dabei jede Permission, die demselben Typ entsprechen. Dadurch wird die Abfrage, ob ein Subjekt eine entsprechende Permission besitzt, erleichtert.

System Security Policies
In der Default Konfiguration der Java 2 Security Architecture wird die Policy-Klasse durch die PolicyFile-Klasse (sun.security.provider.PolicyFile) repräsentiert, wobei Zugriffe in der default Einstellung durch Einträge in eine ASCII-Datei, der System Security Policy, autorisiert werden können. Es kann nur eine Instanz der Policy-Klasse geben, welche jedoch im Gegensatz zum SecurityManager ausgetauscht werden kann.

Die System Security Policy bildet die CodeSource auf Permissions ab. Geht man davon aus, dass durch den CodeSource das Subjekt adressiert wird und durch die Permission das Objekt und die betreffende Aktion, so wird auf Basis der Security Policy dem Subjekt die Berechtigung erteilt, auf ein Objekt mittels der spezifischen Aktion zuzugreifen.

Die System Security Policy wird durch eine oder mehrere persistente Policy Konfigurationen repräsentiert. Jede Policy Konfiguration kann dabei als eine ASCII-Datei, als eine serialisierte binäre Datei der Policy-Klasse oder in einer Datenbank gespeichert sein. Die Policy-Klasse konstruiert die Permissions anhand der Policy Konfiguration. Die Zugriffsrechte werden durch Einträge spezifiziert. Ein Eintrag enthält dabei die Informationen, die zur Erzeugung einer ProtectionDomain (CodeSource) und den zu dieser ProtectionDomain zugehörigen Zugriffsrechten herangezogen werden [POL 00].
Innerhalb der Security Policy sind Permissions, und innerhalb der Permissions sind Operationen eingekapselt.

ProtectionDomain
Eine ProtectionDomain entspricht einem CodeSource und die Gruppe von Permissions, welche dem CodeSource ereilt worden ist. Eine ProtectionDomain enthält demnach alle Zugriffsrechte für ein bestimmtes Subjekt auf Objekte in einer speziellen Art und Weise zuzugreifen.
Jede Klasse bekommt eine einzige ProtectionDomain zugewiesen, welche von dem ClassLoader gesetzt wird. Der ClassLoader ruft entsprechend die `getPermission(CodeSource)`-Methode auf, welche die für diese CodeSource in der Policy-Datei definierten Permissions zurückliefert. Mit der zugewiesenen ProtectionDomain stehen somit auch die zugeteilten Permissions zur Verfügung.

7.1.2 Das Policy-Objekt

Die Java Laufzeitumgebung erzeugt ein globales Policy-Objekt. Das Policy-Objekt trifft keine Policy Entscheidungen sondern repräsentiert die Zugriffsrechte der persistenten Security Policy Konfiguration. Das Policy-Objekt wird bei der Initialisierung der Zugriffsrechte einer ProtectionDomain in Anspruch genommen, und wird typischerweise durch Objekte wie den SecureClassLoader verwendet, sobald der Lader die Permissions einer ProtectionDomain benötigt.

Als Rückgabewert erhält der Lader vom Policy-Objekt das PermissionCollection-Objekt der jeweiligen Entität. Dieses PermissionCollection-Objekt wird zusammen mit der CodeSource der Entität zur Erzeugung der spezifischen ProtectionDomain verwendet.

```
policy = Policy.getPolicy();
PermissionCollection perms = policy.getPermissions(CodeSource)
```

Bei dem Aufruf wird die CodeSource der ProtectionDomain (CodeBase (URL) und Schlüssel Attribute) mitgeteilt. Das Policy-Objekt durchsucht (parst) daraufhin die globale Policy und liefert auf Basis der Policy Konfiguration das Permissions-Objekt zurück.

Das Policy-Objekt kann durch den `setPolicy()`-Aufruf ausgetauscht werden. Hierfür wird jedoch eine spezifische Berechtigung benötigt. Das aktuelle Policy-Objekt bekommt man mittels der `getPolicy()`-Methode. Die `refresh()`-Methode initialisiert die Konfigurationen des Policy-Objekts erneut. Werden die Policy Konfigurationen beispielsweise in Dateien gespeichert, so werden diese Dateien bei dem `refresh()`-Methodenaufruf erneut gelesen. Abhängig von gegebenen Cache-Strategien kann der `refresh()`-Methodenaufruf keine Auswirkungen auf schon geladene Klassen haben [POL 00].

7.1.3 DomainCombiner

Der DomainCombiner ermöglicht eine dynamische Änderung einer ProtectionDomain des aktuellen AccessControlContext [DOM02]. Der `DomainCombiner.combine` wird durch die Aufrufe `AccessController.getContext` oder `AccessController.checkPermission` involviert.

Die `combine()`-Methode enthält zwei Argumente. Das erste Argument enthält die ProtectionDomains des aktuellen execution-Thread und das zweite Argument enthält die ProtectionDomains des Parent-Thread.

Die `combine()`-Methode liefert als Ergebnis eine modifizierte ProtectionDomain. Durch den Einsatz des DomainCombiner können ProtectionDomains der auf dem Stack befindlichen Klassen modifiziert werden. Dem Execution-Thread können durch die `combine`-Methode ProtectionDomains hinzugefügt werden oder es können spezifische ProtectionDomains entfernt werden. Weiterhin besteht die Möglichkeit, nur einzelne individuelle ProtectionDomains zu modifizieren. Einer ProtectionDomain können so beispielsweise neue Permissions zugewiesen werden.

7.2 Realisation der zentralen Kontrollinstanz

Die zentrale Kontrollinstanz (ZKI) wird von dem jeweiligen Agentensystem realisiert und überwacht alle Schnittstellen. Das Agentensystem alleine realisiert daher die Überwachung der Schnittstellen.

Bei den einzelnen Schnittstellen muss einerseits die Überwachung der eigenen Schnittstellen und die Überwachung fremder Schnittstellen, das sind Schnittstellen, die von einer anderen als der überwachenden Entität implementiert werden, unterschieden werden.

- eigene Schnittstellen
  - CORBA Schnittstellen des Agentensystems werden vom Agentensystem überwacht
- fremde Schnittstellen
  - CORBA Schnittstellen einer Agenteninstanz, die durch das Agentensystem überwacht werden.
  - Schnittstellen des Endsystems werden durch das Agentensystem überwacht

Bei jedem Zugriff entscheidet die ZKI, ob die Aktion ausgeführt werden darf oder nicht. Um eine Entscheidung treffen zu können, besitzt die ZKI eine Schnittstelle, mit der abgefragt werden kann, ob der Zugriff eines bestimmten Subjekts erlaubt ist.

Betrachtet man die zentralen Aufgaben der ZKI, nimmt sie die Stellung der globalen Policy der JVM ein. Die ZKI implementiert dabei, um die feingranularen Zugriffskontrollentscheidungen treffen zu können, die eingeführte Policy-Sprache.

Eigene CORBA Schnittstellen
Die Methoden, die die einzelnen Funktionen der Schnittstelle implementieren, werden überwacht, indem vor der eigentlichen Ausführung der Methode die Berechtigung des zugreifenden Subjekts überprüft wird.
Dies geschieht, indem die ZKI befragt wird. Die Entscheidung der ZKI basiert auf den innerhalb der ZKI-Policy enthaltenen Einträgen.

Fremde CORBA Schnittstellen
Wird eine Funktion der CORBA-Schnittstelle einer Agenteninstanz aufgerufen, führt die ORB-Instanz die entsprechende Methode der Agenteninstanz aus. Der ORB muss daher explizit die ZKI nach der Entscheidung, ob der Zugriff zugelassen ist oder nicht, befragen.
Der ORB müsste, um dies bewerkstelligen zu können, die Access Decision-Objekte des Security Functionality Level 2 unterstützen [MAF 00, Kapitel 2.4]. Der in MASA eingesetzte ORB unterstützt diesen Level jedoch nicht, wodurch die Zugriffskontrolle in dieser Art und Weise noch nicht implementiert werden kann.

Schnittstellen des Endsystems
Die Schnittstellen des Endsystems werden durch das Java-API repräsentiert. Um Entscheidungen bezüglich der Zugriffsrechte zu treffen, werden Sicherheitsmechanismen von der Java 2 Security Architecture in modifizierter Form verwendet.

Migriert eine Agenteninstanz auf das Agentensystem oder wird auf dem Agentensystem eine Agenteninstanz gestartet, ermittelt die ZKI anhand der definierten Eigenschaften der Agenteninstanz (vgl. Kapitel 4.3), die der Agenteninstanz durch die ZKI-Policy zugeteilten Zugriffsrechte. Beim Ladevorgang der Agenteninstanz wird der Agenteninstanz eine ProtectionDomain mit den ermittelten Zugriffsrechten zugewiesen. Die CodeSource der ProtectionDomain ist die eindeutige Kennzeichnung der Agenteninstanz auf dem Agentensystem. Hierfür wird eine eigene CodeSource-Klasse für MASA implementiert. Die Zugriffskontrolle von Zugriffe von Agenteninstanzen, die auf dem Agentensystem ausgeführt werden, kann daher durch den Einsatz des AccessControllers erfolgen.

Greift die Agenteninstanz auf eine Ressource zu, wird anhand der Zugriffsrechte der Protection-Domain dieser Agenteninstanz entschieden, ob die notwendige Berechtigung vorhanden ist. Da die Zugriffsrechte durch den Einsatz der MASA Policy-Klasse anhand der definierten Eigenschaften einer Agenteninstanz ermittelt werden, erfolgt eine feingranulare Zugriffskontrolle.

7.3 Autorisation (Realisation der ZKI-Policy)

Agentensysteme realisieren die ZKI, indem die ZKI-Policy auf jedem Agentensystem als globale System Security Policy durchgesetzt wird. Bei einem Zugriff werden nur die innerhalb der ZKI-Policy autorisierten Zugriffe zugelassen. Dem Rechtekonzept entsprechend wird die ZKI-Policy aus der AS-Policy und der jeweiligen MA-Policy einer Agenteninstanz gebildet.

Damit die ZKI-Policy von der JVM, die die Ablaufumgebung des Agentensystems darstellt, als systemglobale Policy durchgesetzt wird, müssen folgende spezifischen Einträge innerhalb der Java

Security Property-Datei ("javahome"/lib/security/java.security) vorgenommen werden. Die Trennung der Politiken ist sinnvoll, um bei einem Zwischenfall die Durchsetzung der Agenteninstanz-Politiken unterbinden zu können.

```
# Agentensystem Policy
as-policy.url.1=file:{java.home}/lib/security/as.policy

# Policies der Agenteninstanzen
ma-policy.url.1=file:{java.home}/lib/security/ma.policies
```

Der erste Eintrag gibt den Ort der von der ZKI durchzusetzenden AS-Policy an. Der zweite Eintrag bestimmt den Ort der MA-Policies-Datei, in die die zulässigen Zugriffsrechte der einzelnen MA-Policy-Dateien integriert werden. Betrachtet man die Vereinigungsmenge der Zugriffsrechte, so ergibt sich die in dem Rechtekonzept dargestellte ZKI-Policy.

In dieser Implementierung werden die Konfigurationen der Politiken der Agentensysteme und Agenteninstanzen in einer ASCII-Datei vorgenommen. Die Einträge der ASCII-Datei entsprechen dabei der in Kapitel 6.3.2 eingeführten Policy-Sprache. Die Realisation mittels der ASCII-Dateien stellt dabei nur einen Lösungsweg dar. Die Implementierung kann ebenfalls mithilfe von Datenobjekten realisiert werden, um nicht explizit in eine Datei schreiben zu müssen.

Realisation der ZKI-Policy:

1. AS-Policy des Agentensystems (`as.policy`)
   Beim Start eines Agentensystems ist die `as.policy` die einzige von der ZKI durchzusetzende Policy. Die `ma.policies` Datei enthält zu Beginn keine Einträge.

2. MA-Policy einer Agenteninstanz (`ma.policies`)
   Migriert eine Agenteninstanz auf das Agentensystem oder wird eine Agenteninstanz neu gestartet, besitzt die Agenteninstanz eine MA-Policy. Die MA-Policy wird vom Agentensystem ausgewertet. Es werden nur die Zugriffsrechte einer MA-Policy in die `ma.policies`-Datei aufgenommen, die von den jeweiligen Filter-Politiken zugelassen werden. Die zulässigen Zugriffsrechte der MA-Policy bilden dann zusammen mit der AS-Policy die vom Agentensystem durchzusetzende ZKI-Policy des Rechtekonzepts.

Überprüfung der MA-Policy (`agentPolicyCheck()`)
Für die Überprüfung der Zugriffsrechte einer MA-Policy auf ihre Zulässigkeit hin, muss eine geeignete Methode (`agentPolicyCheck()`) bereitgestellt werden, welche die Zugriffsrechte der MA-Policies auf ihre Zulässigkeit hin überprüft. Die `agentPolicyCheck()`-Methode setzt dabei die Restriktionen der AS-Filter-Policy des Agentensystems und die betroffenen MA-Filter-Policies der Agenteninstanzen durch. Als Resultat liefert die Methode die akzeptierten, von dem Agentensystem durchzusetzenden Zugriffsrechte der MA-Policies.

Die `agentPolicyCheck`-Methode vergleicht jedes einzelne Zugriffsrecht der MA-Policy mit der zuständigen Filter-Politik. Ist das Zugriffsrecht innerhalb der Filter-Politik als unzulässig deklariert, wird es nicht in die `ma.policies`-Datei aufgenommen und daher nicht von der ZKI des Agentensystems durchgesetzt.
Handelt es sich um Zugriffsrechte bezüglich Zugriffen auf das Agentensystem bzw. Endsystem, wird der Eintrag mit jedem einzelnen Eintrag des Agentensystem-Filters (AS-Filter-Policy) verglichen. Autorisiert das Zugriffsrecht der MA-Policy einen Zugriff auf ein Objekt einer Agenteninstanz, so wird der Filter dieser Agenteninstanz (MA-Filter-Policy) zum Vergleich herangezogen.

Damit die in die `ma.policies`-Datei hinzugefügten Zugriffsrechte von der ZKI des Agentensystems durchgesetzt werden, muss das MASA Policy-Objekt neu initialisiert werden. Das MASA

Policy-Objekt stellt hierfür die `refresh()`-Methode bereit.

Integration einer neuen MA-Policy (`combineAgentPolicies()`)
Für die Integration der akzeptierten Zugriffsrechte in die `ma.policies`-Datei muss eine geeignete Methode bereitgestellt werden (`combineAgentPolicies()`). Die `combineAgentPolicy()`-Methode erweitert die `ma.policies`-Datei um ein Zugriffsrecht und setzt zusätzlich für jedes Zugriffsrecht eine Marke. Aus der Marke ist neben dem Zeitpunkt der Integration auch die Agenteninstanz, welche das Zugriffsrecht integriert, ersichtlich.
Die Marke wird als Kommentar (`# MARKE (NOT EDIT): Marke`)in die `ma.policies` Datei eingetragen. Die MASA Policy-Klasse wertet die Marke daher nicht aus.
Verlässt eine Agenteninstanz das Agentensystem bzw. wird eine Agenteninstanz beendet, können so die von der Agenteninstanz vorgenommenen Einträge wieder entfernt werden. Hierfür muss die Methode `deleteAgentPermissions(agentAttributs)` zur Verfügung gestellt werden. Die Methode sucht nach den jeweiligen Einträgen und entfernt diese aus der `ma.policies`-Datei. Durch einen erneuten `refresh()`-Aufruf werden die Änderungen von der ZKI übernommen.

7.3.1 Filter Politik

Die Filter Politiken werden ebenfalls innerhalb der Java Security Property-Datei spezifiziert.

```
# Filter Policy des Agentensystems
as-filter.url.1=file:{java.home}/lib/security/as.filter

# Filter Policy der Agenteninstanz
ma-filter.url.2=file:{java.home}/lib/security/ma.filters
```

Die Filter-Policy einer Agenteninstanz wird dabei zusammen mit der MA-Policy dem Agentensystem vor dem Laden der Agenteninstanz übergeben.

7.4 Rechtedurchsetzung (MASA Policy-Klasse)

Die Standard Policy-Klasse von Sun (`sun.security.provider.PolicyFile`) kann die Einträge in Form der neuen Policy-Sprache nicht auswerten und daher nicht durchsetzen. Um die Politiken auswerten zu können, ist daher eine eigene `masa.security.provider.MASAPolicyFile`-Policy-Klasse notwendig, die die in Kapitel 6.3.2 eingeführte Policy-Sprache implementiert.

Durch den folgenden Eintrag innerhalb der Java Security Property Datei wird die Masa Policy-Klasse von der JVM als systemglobale Policy verwendet:

```
policy.provider=masa.security.provider.MASAPolicyFile
```

Die Klasse MASAPolicyFile wird von `java.security.Policy` abgeleitet. Damit wird die MASA Policy-Klasse in die Java 2 Security Architecture integriert.
Innerhalb der MASA Policy-Klasse müssen neben den Funktionalitäten, die die Standard Policy-Klasse PolicyFile von Sun implementiert, bestehende Funktionen modifiziert bzw. neue implementiert werden.

Durch das MASA Policy-Objekt sind folgende Funktionen zu realisieren:

1. Initialisierung des Policy-Objekts und lesen der Konfigurationsdateien
   Bei einem Start eines Agentensystems erfolgt durch die Initialisierung des MASA Policy-Objekts die Verwendung der Zugriffskontrolle nach dem vorgestellten Rechtekonzept. Bei der Initialisierung werden die Eintragungen innerhalb der Java Security Property-Datei berücksichtigt. Die Bedeutung der Einträge innerhalb dieser Datei entsprechen zum größten Teil den Definitionen der Standard Implementierung.

Betrachtet man innerhalb der Java Security Property-Datei die Ortsangaben der durchzusetzenden Policy-Dateien, so sind hier die Änderungen bezüglich der `as.policy`- und der `ma.policies`-Dateien zu beachten. Zudem wird innerhalb dieser Datei die Lage der Filter-Politiken definiert.

2. Default Policy
   Für den Fall, dass keine AS-Policy Datei (as.policy) vorhanden ist, muss eine restriktive, an das jeweilige Agentensystem angepasste, Default-Policy durchgesetzt werden.

3. Parsen der Policy-Dateien
   Bei der Initialisierung des MASA Policy-Objekts werden anhand der in der Java Security Property-Datei spezifizierten Policy- und Filter- Dateien die entsprechenden Zugriffsrechte ermittelt. Hierfür muss das MASA Policy-Objekt die eingeführte Policy Sprache interpretieren können. Durch die Erstellung der jeweiligen Permissions-Objekte stehen die einer Instanz zugewiesenen Zugriffsrechte zur Verfügung.

4. Bilden des agentAttributs, des agentsystemAttributs und des personAttributs
   Aus den einzelnen Einträgen der Politiken muss das jeweilige Attributs-Objekt gebildet werden. Das agentAttributs besteht dabei aus den einzelnen Eigenschaften des Agenten. Das agentensystemAttributs-Objekt aus den Eigenschaften des Agentensystems und das personAttributs-Objekt aus den Eigenschaften der Person.

   Die Attributs-Objekte entsprechen der Funktionalität der CodeSource-Objekte der Standard Java 2 Security Architecture.

5. Erzeugung der Permission-Objekte
   Die in den Politiken vergebenen Permission-Objekte müssen erzeugt werden. Dabei bezeichnet der Name des Permission-Objekts den Typ des Zugriffsrechts. Desweiteren kann noch das spezifische Target-Objekt benannt sein und die Aktion, die auf dem Objekt ausgeführt werden kann (vgl. Standard Java Policy-Klasse von Sun).

6. Zertifikate der Aliase
   Es müssen alle Zertifikate der entsprechenden Aliase innerhalb der ZKI-Policy aufgenommen werden (vgl. Standard Java Policy-Klasse von Sun). (Aliase sind dabei die in der Security Policy verwendeten Namen, stellvertretend für die Signaturen).

7. Erzeugung des PermissionsCollection-Objekts
   Die Politiken müssen nach den spezifischen agentAttributs untersucht werden. Daraufhin muss das PermissionCollection-Objekt für die ProtectionDomain des Principals erzeugt werden.

   Die des agentAttributs-Objekts zugewiesenen Zugriffsrechte werden durch die Methode `getAgentPermissions(agentAttributs)` ermittelt.

   Auf Anfrage muss ebenso ein PermissionsCollection-Objekt auf Basis der AgentensystemAttributs- oder PersonAttributs-Objekte erzeugt werden.

   Folgende Methoden liefern das jeweilige Permissions-Objekt:
   `getAgentensystemPermissions(agentsystemAttributs)`
   `getPersonPermissions(personAttributs)`

7.4.1 Zugriffsrechte einer Agenteninstanz

Die in der ProtectionDomain der Agenteninstanz enthaltenen Zugriffsrechte, sind die Zugriffsrechte, die die Agenteninstanz auf dem jeweiligen Agentensystem besitzt.

Eine Agenteninstanz wird jeweils durch einen eigenen AgentenClassLoader geladen. Jede Agenteninstanz kann so unterschieden und eine eigene eindeutige ProtectionDomain zugewiesen werden. Der AgentenClassLoader wird von `SecureClassLoader` abgeleitet, wodurch der Sicherheitsmechanismus der ProtectionDomain für Agenteninstanzen eingesetzt werden kann.
Vor dem Laden einer Agenten-Klasse überprüft der AgentenClassLoader, ob der Agent die benötigte Berechtigung besitzt, um die Klasse zu laden und zu definieren. Realisiert wird dies durch die Verwendung der `System.getSecurityManager().checkPackageAccess()`-Methode innerhalb des AgentenClassLoaders. Desweiteren werden vor dem Laden der Agenteninstanz die der Agenteninstanz durch die ZKI-Policy zugewiesenen Zugriffsrechte ermittelt. Das MASA Policy-Objekt implementiert dafür die `getAgentPermissions()`-Methode, welche auf Basis der einzelnen Attribute der Agenteninstanz die in der ZKI-Policy zugeteilten Zugriffsrechte ermittelt.

```
agentPermissions =
  java.security.Policy.getPolicy().getAgentPermissions(agentAttributs);
```

Mit den ermittelten Zugriffsrechten und der MASA-CodeSource wird eine ProtectionDomain der Agenteninstanz erzeugt.

```
agentProtectionDomain =
  new java.security.ProtectionDomain(CodeSource, agentPermissions);
```

Durch die explizite Zuweisung ist es aus der Sicht der Zugriffskontrolle möglich, mehrere Agenteninstanzen der gleichen Gattung auf ein und demselben Agentensystem auszuführen und die Zugriffskontrolle durch den Einsatz des AccessControllers zu regeln, da die Eindeutigkeit der ProtectionDomain gegeben ist.

```
protected final Class defineClass(String name, byte date[],
                                  int offset, int length, agentProtectionDomain pd)
```

Durch den in der `define()`-Methode enthaltenen ProtectionDomain Parameter, werden der Agenteninstanz die expliziten Permission-Objekte, welche in der ProtectionDomain enthalten sind, zugewiesen.

Kritische Zugriffe auf Ressourcen des Endsystems werden durch den AccessController überprüft. Greift nun eine Agenteninstanz auf eine Ressource des Endsystems zu, so überprüft der AccessController den Zugriff. Hierfür werden die auf dem Stack befindlichen ProtectionDomains auf die in der ProtectionDomain enthaltenen Zugriffsrechte hin überprüft.

Da die ProtectionDomain der Agenteninstanz die mithilfe des MASA Policy-Objekts ermittelten Zugriffsrechte auf Basis der ZKI-Policy enthält, erfolgt eine feingranulare Zugriffskontrolle auf Basis der Eigenschaften der Agenteninstanz.

Die Abb. 43 stellt die involvierten Komponenten der Zugriffskontrolle dar.

7.5 Aktions-Policy

Der Mechanismus der Aktions-Policies kann durch den DomainCombiner (vgl. Kapitel 7.1.3) realisiert werden. Die ProtectionDomain einer Agenteninstanz kann mittels des DomainCombiners bei der Übergabe der Aktions-Policy und deren Überprüfung dynamisch um die in der Aktions-Policy enthaltenen und von dem Agentensystem akzeptierten Permission-Objekte erweitert werden.

Eine Agenteninstanz kann so ein dynamisches Zugriffsrecht, geknüpft an bestimmten Bedingungen, auf Ihre Objekte vergeben. Tritt eine bestimmte Bedingung ein, so überreicht sie die Aktions-Policy an das Agentensystem. Das Agentensystem kontrolliert die in der Aktions-Policy vorhande-

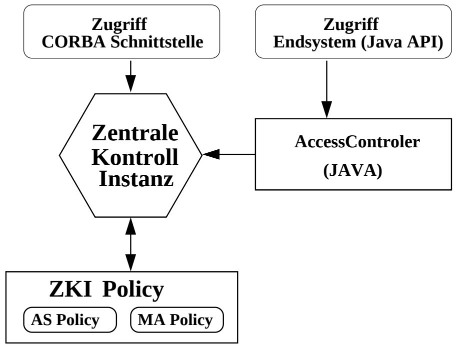


Abbildung 43: Zugriffskontrolle in MASA

nen Zugriffsrechte. Die ProtectionDomain der Agenteninstanz wird durch den DomainCombiner-Mechanismus um die von dem Agentensystem akzeptierten Zugriffsrechte der Aktions-Policy erweitert.

7.6 Zugriffsrechte

Die Zugriffsrechte können durch die `java.security.PermissionObjekte` der Java 2 Security Architecture dargestellt werden. Zugriffsrechte, die Objekte des Endsystems betreffend, werden durch das API definiert. Zugriffsrechte für Objekte der Agenteninstanzen oder des Agentensystems werden durch neue Permission-Objekte erzeugt die von `java.security.Permission` abgeleitet werden und somit vollwertige Zugriffsrechte darstellen.

7.7 Kontrolle der Zugriffe von Agenteninstanzen

Der AgentenClassLoader, welcher den SecureClassLoader erweitert, ruft bei der Erstellung einer ProtectionDomain die `getAgentPermission()`-Methode der MASA Policy-Klasse auf. Die `getAgentPermission()`-Methode liefert als Resultat das Permissions-Objekt, welches für die Erzeugung der ProtectionDomain der Agenteninstanz verwendet wird.

Bedingungen:

- Attribute der Agenteninstanz müssen der Policy-Klasse bei der Ausführung der `getAgentPermission()`-Methode bekannt sein.
- Das agentAttributs-Objekt (bestehend aus den Eigenschaften der Agenteninstanz) muss eindeutig sein.
- Die `getAgentPermission()`-Methode der MASA Policy-Klasse liefert das gewünschte Permissions-Objekt, welches alle Permissions enthält, die der Agenteninstanz innerhalb der ZKI-Policy zugewiesen worden sind.

Ablauf: Erzeugung der ProtectionDomain einer Agenteninstanz (vgl. Abb. 44):

1. Eine Agenteninstanz wird auf dem Agentensystem neu gestartet oder migriert auf dieses.
2. Die Agenteninstanz wird durch den AgentenClassLoader geladen, welcher den SecureClassLoader erweitert. Der SecureClassLoader ruft die `getAgentPermission()`-Methode der MASA Policy-Klasse auf, welche das Permissions-Objekt der Agenteninstanz liefert. Das Permissions-Objekt beinhaltet die der Agenteninstanz in der ZKI-Policy zugewiesenen Zugriffsrechte auf Basis der Eigenschaften der Agenteninstanz

3. Der AgentenClassLoader erzeugt die ProtectionDomain der Agenteninstanz. Die ProtectionDomain enthält dabei das Permissions-Objekt der Agenteninstanz.

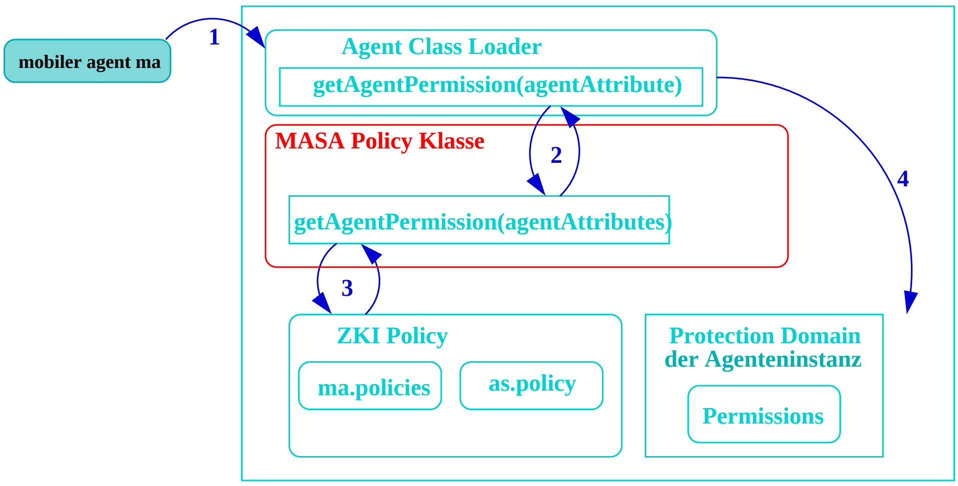


Abbildung 44: Erzeugung der ProtectionDomain der Agenteninstanz

Der Ablauf der Zugriffskontrolle ist in Abb. 45 dargestellt. Dabei entspricht der Ablauf der Sicherheitsarchitektur der Java 2 Security Architecture mit Ausnahme der eigenen Policy-Klasse.

1. Die Agenteninstanz des Agentensystems greift auf eine Ressource des Endsystems zu (z.B. Agenteninstanz liest die Temp-Datei (`/temp`)).

2. Durch den Einsatz des SecurityManagers wird bei einem sicherheitskritischen Zugriff der SecurityManager vor der eigentlichen Ausführung der kritischen Methode aufgerufen.

3. Der SecurityManager ruft defaultmäßig bei einer Standard Java API-Methode den AccessController auf.

4. Der AccessController überprüft daraufhin die Berechtigung der auf dem Stack befindlichen ProtectionDomains.

5. Haben alle sich auf dem Stack befindlichen ProtectionDomains die notwendige Berechtigung in Form des speziellen Permission-Objekts, so erfolgt keine Ausnahmebehandlung und die eigentliche Methode kann ausgeführt werden. Ist das notwendige Permissions-Objekt nicht vorhanden, hat dies eine Exception zur Folge. Die geschützte Methode kann somit nicht ausgeführt werden.

7.8 Beispielszenario Aktions-Policy

Erweiterung der ProtectionDomain durch Zugriffsrechte der Aktions-Policy
Mit dem Interface DomainCombiner aus dem Package `java.security` kann ein dynamisches Updaten der `ProtectionDomain`, die mit dem momentan aktuellen `AccessControlContext` verbunden ist, erfolgen. Durch die Modifizierungen mittels `combine` können neue ProtectionDomains in den Stack aufgenommen werden, ProtectionDomains aus dem Stack entfernt werden oder die existierenden ProtectionDomains upgedatet werden. Die `combine()`-Methode richtet sich bei einem Update nach Informationen, welche innerhalb des DomainCombiners eingekapselt sind.

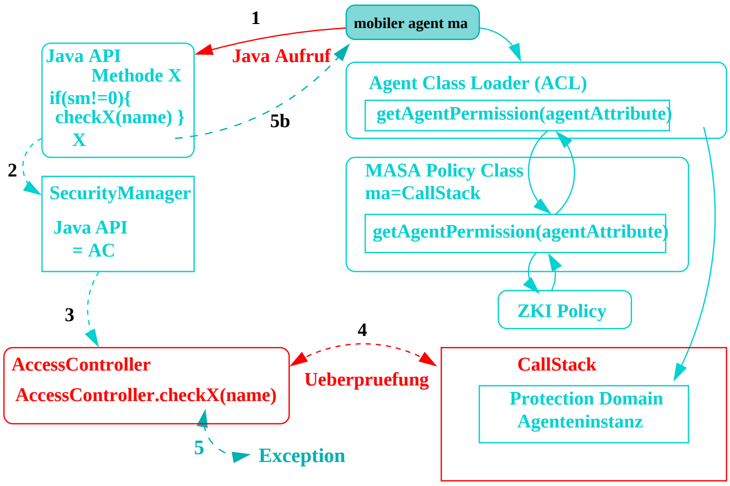


Abbildung 45: Ablauf der Zugriffskontrolle

Aufrufe der Methode:
`AccessController.getContext` und der Methode
`AccessController.checkPermission` involvieren die Methode `DomainCombiner.combine`.

Für die Integration der vom Agentensystem akzeptierten Zugriffsrechte der Aktions-Policy einer Agenteninstanz, muss die ProtectionDomain der Agenteninstanz um zusätzliche Zugriffsrechte erweitert werden. Beim Beenden bzw. bei Migration auf ein anderes Agentensystem müssen die nachträglich hinzugefügten Einträge wieder entfernt werden.

Ablauf: Integration der Aktions-Policy:

1. Eine Agenteninstanz wird auf dem Agentensystem neu gestartet oder migriert auf dieses.
2. Die Agenteninstanz wird durch den AgentenClassLoader geladen. Der Agenteninstanz wird auf Basis der ZKI-Policy eine ProtectionDomain mit den der Agenteninstanz zugewiesenen Zugriffsrechte zugewiesen.
3. Die Ermittlung der Zugriffsrechte der ProtectionDomain erfolgt anhand des agentAttributs-Objekts der Agenteninstanz, welche mit den Einträgen der ZKI-Policy (ma.policies und as.policy) verglichen werden.
4. Die ProtectionDomain der Agenteninstanz wird upgedatet. Damit sind alle Permission-Objekte der Agenteninstanz in der Protection Domain vorhanden. Das Updaten der ProtectionDomain kann mithilfe des DomainCombiner Interface der Java1.3 API erfolgen.

Damit besitzt die ProtectionDomain der Agenteninstanz die von dem Agentensystem akzeptierten und in den Aktions-Policy enthaltenen Zugriffsrechte.

8 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Rechtekonzept für die Mobile Agent System Architecture (MASA) entwickelt. Dabei wurden grundlegende Anforderungen einer Agentensystemarchitektur beachtet.

In dem vorgestellten Rechtekonzept werden, aufbauend auf den vorhandenen Authentifikationsmechanismen, die Rechtevergabe (Autorisation) und die Rechtedurchsetzung genauer betrachtet und realisiert. Dabei wird auf die strikte Trennung der Sicherheitspolitik und des Durchsetzungsmechanismus Wert gelegt.

Der Autorisation liegt eine sprachbasierte Spezifizierung von Zugriffsrechten mithilfe bestimmter Kriterien der Subjekte, Objekte und Zugriffsarten zugrunde. Durch die dafür eigens entwickelte Politik-Sprache wird eine anwendungsspezifische, feingranulare Zugriffskontrolle der Agentensysteme und der Agenteninstanzen ermöglicht. Zusätzlich kann die Entscheidungsfindung noch von der Vergangenheit einer Agenteninstanz abhängig gemacht werden, wodurch die Sicherheit entsprechend erhöht wird.

Das Agentensystem stellt die oberste Hierarchie des Rechtekonzepts dar. Ein Agentensystem kann jederzeit Zugriffsrechte zurücknehmen, die Filter-Policy konfigurieren und damit Regeln bezüglich der Integration von Zugriffsrechten einer Wunsch-Zugriffskontrollpolitik einer Agenteninstanz treffen.

Um die Informationssicherheit von Agenteninstanzen gewähren zu können, wurde neben der Zugriffskontrollpolitik eines Agentensystems, die Wunsch-Zugriffskontrollpolitik der Agenteninstanzen eingeführt. Eine Agenteninstanz ist in der Lage, dem Agentensystem eine Wunsch-Zugriffskontrollpolitik vorzulegen, um diese, unter Beachtung der Autonomieanforderungen, durchsetzen zu lassen. Durch die Integration der von einem Agentensystem akzeptierten Zugriffsrechte der Wunsch-Zugriffskontrollpolitik einer Agenteninstanz wird eine Autorisation von Subjekten über mehrere Agentensysteme und daher auch über Administrationsgrenzen hinaus ermöglicht. Die Autorisation bezieht sich dabei sowohl auf Objekte des Agentensystems als auch auf Objekte einer Agenteninstanz. Die Autorisation von Objekten einer Agenteninstanz können dabei der Eigentümer der Agenteninstanz aber auch weitere Subjekte, welche von dem Eigentümer autorisiert wurden, vornehmen.

Um eine dynamische, an Bedingungen geknüpfte, Autorisation zu ermöglichen, wurden Aktions-Policies eingeführt. Durch die Aktions-Policies wird eine dynamische Zugriffsrecht-Vergabe auf die Objekte der Agenteninstanz realisiert. Eine Agenteninstanz kann dadurch Zugriffe auf ihre Objekte in Abhängigkeit bestimmter Zustände gewähren oder unterbinden.

Auch die Delegation der Zugriffsrechte wurde betrachtet und realisiert. Agenteninstanzen können so autorisiert werden, dass sie die ihnen zugeteilte Aufgabe einer Agenteninstanz bearbeiten können.

Für das Rechtekonzept wurde ein Implementierungskonzept angegeben. Das Implementierungskonzept integriert das Rechtekonzept in die Java 2 Security Architecture, wodurch MASA von den Sicherheit-Features von Java profitieren kann.

Ausblick
Mit dem vorgestellten Rechtekonzept ist die Informationssicherheit der Agentensystemarchitektur sichergestellt. Für zukünftige Arbeiten können die folgend genannten Punkte angegangen werden:

- Durch das vorgestellte Rechtekonzept wird ein DAC-Kontrollmechanismus realisiert. Die Möglichkeit einer MAC-Zugriffskontrollimplementierung ist jedoch jederzeit vorstellbar. So könnten mobile Agenten in Sicherheitsklassen aufgeteilt werden, wobei es z.B. nur Agenten einer Sicherheitsklasse erlaubt wird, miteinander zu kommunizieren. Die Integration einer Informationsflusskontrolle würde die Sicherheit der Agentensystemarchitektur erhöhen.

- Mischformen der Autorisation eines sprachbasierten Ansatzes und der subjekt- bzw. objektbasierten Methoden vereinen Vorteile beider Seiten. So könnte ein zusätzliches Kriterium in Form eines Capabilities oder eines Eintrags in eine Zugriffskontrollliste (ACL) in die Sicherheitspolitik eingeführt werden.

- Es wird ein geeignetes Logging-Verfahren benötigt, welches sicherheitskritische Zugriffe aufzeichnet. Das Logging-Verfahren sollte über Möglichkeiten verfügen, die Granularität der Aufzeichnungen und die Inhalte der Logdatei bestimmen zu können.

- Die Erstellung einer graphischen Benutzeroberfläche (GUI) würde die Verwaltung und Konfiguration der Sicherheitspolitiken benutzerfreundlicher machen.

- Die Ressourcen Limitierung ist als Fernziel zu nennen.

A Abkürzungsverzeichnis

API	- Application Programming Interface
CA	- Certificate Authority
CMIP	- Common Management Information Protocol
CORBA	- Common Object Request Broker Architecture
CRL	- Certificate Revocation List
CSP	- Cryptographic Service Provider
DAC	- Discretionary Access Control
DCE	- Distributed Computing Environment
DES	- Data Encryption Standard
DFS	- Distributed File Service (DCE)
DN	- Distinguished Name
DNS	- Domain Name Service
DOD	- Department of Defense
DSA	- Data Encryption Standard
FIPA	- Foundation for Intelligent Physical Agents
GUI	- Graphic User Interface
ID	- Identifier
IETF	- Internet Engineering Task Force
IIOP	- Internet Inter-ORB Protocol
JAAS	- Java Authentication and Authorisation Service
JAR	- Java Archive
JCA	- Java Cryptography Architecture
JCE	- Java Cryptography Extension
JDK	- Java Developement Kit
JVM	- Java Virtual Machine
MAC	- Mandatory Access Control
MAC	- Message Authentication Code
MASIF	- Mobile Agent System Interoperability
NFS	- Network File System
NIS	- Network Information Service
NIST	- National Institute for Standards and Technology
NOS	- Network Operating Service/System
OMG	- Object Management Group
ORB	- Object Request Broker
PKCS	- Public Key Cryptography Standards
QoS	- Qualitiy of Service
RMI	- Remote Method Invocation
RPC	- Remote Procedure Call
SKIP	- Simple Key Management for Internet Protocols
SNMP	- Simple Network Management Protocol
SPI	- Service Provider Interface
TCB	- Trusted Computing Base
URL	- Uniform Resource Locator
VPN	- Virtual Privat Network

Literatur

[ACC02] SUN-MICROSYSTEM: *AccessController.java.* Java Quellcode V. 1.3, 2002.

[ANO 99] ANONYMOUS: *Maximum Linux Security.* SAMS, 1999.

[ASA 92] SANDHU R. S., AMMAN P. E.: *Implementing transaction control expressions by checking the absence of rights.* Eighth Annual Computer Security Applications Confernce, 1992.

[BAF 00] WALLACH D. S., BALFANZ D., DEAN D. UND FELTEN E. W.: *Extensible Security Architectures for Java.* Princeton University, 2000, http://i30www.ira.uka.de/conferences/SOSP/sosp16/PAPERS/WALLACH/NODE3.HTM.

[BIS 90] BIC L., SHAW A. C.: *Betriebssysteme - Eine moderene Einfuehrung.* Carl Hanser und Prentince-Hall International, 1990.

[BOR 94] BORENSTEIN, N. S.: *Email with a mind of its own: The Safe-Tcl language for enabled mail.* IFIP International Working Conference on Upper Layer Protocols, Architectures and Applications, 1994.

[BRK 98] BREDIN J., RUS D., KOTZ D.: *Market-based Resource Control for Mobile Agents.* in Proceeding of Autonomous Agents, 1998, ftp://ftp.cs.dartmouth.edu/pub/kotz/papers/bredin:market.ps.Z.

[BRO 92] BROBEIL, H.: *Software Angriffe auf PCs und Netzwerke.* 1992.

[CDC 83] CONTROL-DATA-CORPORATION: *NOS Version 2.* 1983.

[CHE 00] CHEUNG K., SAU KOON NK: *Intention spreading: An extensible theme to protect mobile agents from read attack hosted by malicious hosts.* 2000.

[CLS 94] CAELLI W., LONGLEY D., SHAIN M.: *Information Security Handbook.* 1994.

[COS02] SUN-MICROSYSTEM: *CodeSource.java.* Java Quellcode V. 1.3, 2002.

[CSD 00] COMPUTER-SCIENCES-DEPARTMENT: *Bond.* Purdue Universität, 2000, http://bond.cs.purdue.edu.

[DAM 01] DARTMOUTH-COLLEGE: *D'Agents.* WWW, 2001, http://agent.cs.dartmouth.edu.

[DEA 98] DEAN, D.: *Formal Aspects of Mobile Code Security.* Doktor Dissertation, Princeton Univ., Dept. of Computer Science, 1998.

[DFW 84] DEAN D., FELTEN E. W., WALLACH D. S.: *Java security: From HotJava to Netscape and beyond.* In Proceseding of the 1996 IEEE Symposium on Security and Privacy (Oakland), 1984.

[DOM02] SUN-MICROSYSTEM: *DomainCombiner.java.* Java Quellcode V. 1.3, 2002.

[ECK 98] ECKERT, C.: *Sichere verteilte Systeme - Konzepte, Modelle und Systemarchitekturen.* Habilitationsschrift zur Erlangung des akademischen Grades eines Dr. rer. nat. habil., 1998.

[ECO 00] ELECTRONIC-COMMUNITIES: *Using the EC Trust Manager to Secure Java.* WWW, 2000, http://www.communities.com.

[FAL 00] FALK, R.: *Eine Methode fuer die Verwaltung von Zugriffsrechten in IT-Systemen.* WWW, 2000, http://elib.uni-stuttgart.de/opus/volltexte/1999/50.

[GAL 87] GALLAGHER, P.: *A Guide to Understanding Discretionary Access Control in Trusted Systems.* National Computer Security Center (NCSC-TG-003 Version-1), 1987, `http://nestroy.wi-inf.uni-essen.de/Lv/ibis2/discretionary_acces_control.html`.

[GEN 01] GENMAGIC: *Odyssey.* WWW, 2001, `http://www.genmagic.com/agents`.

[GHW 86] GLIGOR V., HUSKAMP J., WELKE S. LINN C. UND MAYFIELD W.: *Traditional Capability-Based Systems: An Analysis of Their Ability to Meet the Trusted Computer Security Evaluation Criteria.* IDA Paper P1 935, 1986.

[GLO 97] GUENTER K., LANGE D. B., OSHIMA M.: *A Security Model for Aglets.* IEEE Internet Computin, Vol. 1, No. 4, 1997, `http://computer.org/internet/ic1997/w4toc.htm`.

[GON 00] GONG, L.: *The Java Security Architecture (JDK 1.2).* WWW, 2000, `http://java.sun.com`.

[GON 89] GONG, L.: *A secure identity-based capability system.* In Proceedings of the 1989 IEEE Symposium on Security and Privacy (Oakland), 1989.

[GON 98] GONG, L.: *Secure Java Class Loading.* EEE Internet Computing, Vol. 2, No. 6, 1998, `http://www.computer.org/internet/ic1998/w6056abs.htm`.

[GRA 00] IKV++: *Grasshopper Programmers Guide Release 2.1.* WWW, 2000, `http://www.ikv.de/products/grasshopper/downexamples.html`.

[GRE 00] GREEN, D.: *The Reflection API.* WWW, 2000, `http://www.zdv.uni-mainz.de/~langc/java/tutorial/reflect/index.html`.

[HAE 78] HAERDER, T.: *Implementierung von Datenbanksystemen.* Carl Hansen Verlag, 1978.

[HAN 99] HEGERING H. G., ABECK S., NEUMAIR B.: *Integriertes Management vernetzter Systeme.* dpunkt Verlag, 2000.

[HEK 84] KARGER P. A., HERBERT A. J.: *Lattice Security and Traceability of Access.* Symposium on Security and Privacy IEEE, 1984.

[HER 99] HEROLD, H.: *Linux Unix Systemprogrammierung.* Addison Wesley ISBN 3-8273-1512-3, 1999, `http://www.pinguin.at/linux_systemprog.html`.

[HGR 99] HEILBRONNER S., GRUSCHKE B. UND REISER H.: *Mobile Agent System Architecture - Eine Plattform für flexibles IT-Management.* Technischer Bericht 9902, Ludwig-Maximilians-Universität München, Institut für Informatik, 1999.

[HOH 97] HOHL, F.: *Protecting mobile agents with blackbox security.* In Proceeding of the 1997 Workshop on Mobile Agents and Security, University of Maryland, 1997.

[HOH 98] HOHL, F.: *Time Limited Blackbox Security: Protection Mobile Agents from Malicious Hosts.* in G. Vigna (Ed) Mobile Agents and Security, 1998.

[IBM 83] IBM: *Access Method Services.* 1983.

[IBM 99a] IBM: *Securing Valuable Corporate Data.* WWW, 1999, `http://www.s390.ibm.com/products/racf/racfhp.html`.

[IBM 99b] PISTOIA M., RELLER D., GUPTA D. NAGNUR M. UND RAMANI A.: *Java 2 Network Security.* Reedbook IBM, 1999, `http://www.reedbooks.ibm.com`.

[JAW 01] JAVAWORLD: *An introduction to agents (first steps toward building your own simple agent architecture in Java).* WWW, 2001, `http://www.javaworld.com/javaworld/jw-06-1998/jw-`.

[JHM 99] JUN K., HAO R., MARINSECU D. C.: *Bond System Security and Access Control Model.* Computer Sciences Department, Purdue University, USA, 1998-1999.

[KAR 95] GUENTER, K.: *Integrated Access Control Management.* IBM Research Division, Zurich Research Laboratroy, Switzerland, 1995.

[KAT 00] TRIPATHI A. R., KARNIK N. M.: *Delegation of Privileges to Mobile Agents in Ajanta.* Computer Science Department, University of Minnesota, 2000, `http://www.cs.nmn.edu/Ajanta/publications.html`.

[KAT 98] TRIPATHI A. R., KARNIK N. M.: *Protected Resource Access for Mobile Agent-based Distributed Computing.* Department of Computer Science, University of Minnesota, 1998, `http://www.cs.umn.edu/Ajanta/publications.html`.

[KEM 98] KEMPTNER, B.: *Entwurf eines Java/CROBA-basierten Mobilen Agenten.* Diplomarbeit, Technische Universität München, 1998.

[KOO 98] KARJOTH G., OSHIMA M., ONO K.: *Aglets Specification 1.1 Draft.* WWW, 1998, `http://www.trl.ibm.co.jp/aglets`.

[KUF 00] FERRAIOLO D., KUHN R.: *Role Based Access Control.* WWW, 2000, `http://hissa.ncsl.nist.gov/rbac`.

[KUH 96] KUHN, W.: *Ressourcenkontrolle in einem Mobilen-Agenten-System.* 1996.

[LEV 84] LEVY, H. M.: *Capability-Based Computer Systems.* Digital Press, 1984.

[LIE 90] LIEB: *Sicherheitsaspekte des Betriebssystems UNIX.* Springer Verlag, Informatik Spektrum, Bd. 13, Heft 4, 1990.

[LWO 97] LEVY J. Y., WELCH B. B., OUSTERHOUT J. K.: *The Safe-Tcl security model.* Technical Report, Sun Microsystems Laboratories 1997, 1997, `http://www.sun.com/research/techrep/1997/abstract-60.html`.

[MAF 00] MAFS: *Mobile Agent Facility Specification.* WWW, 2000, `http://java.sun.com`.

[MOM 99] MIKAND D., MOSHAMER W.: *Java und Security.* www, 1999, `http://www.infosys.tuwien.ac.at/Teaching/Courses/AK2/vor99/t1`.

[NCS 85] NATIONAL-COMPUTER-SECURITY-CENTER: *Department of Defense Trusted Computer System Evaluation Criteria (The Orange Book).* NCSC, 1985.

[NET 00] NETSCAPE: *Object Signing Resources.* WWW, 2000, `http://developer.netscape.com/docs/manuals/signedobj`.

[NEU 93] NEUMANN, B.: *Proxy-based authorization and accounting for distributed systems.* In Proceedings of the Thirteenth International Conference on Distributed Computing Systems, 1993.

[OAK 00] OAKS, S.: *Java Security (Java 2).* O'Reilly, 2000, `http:www.oreilly.de/catalog/javasec`.

[OBJ 01] OBJECTSPACE: *Voyager.* WWW, 2001, `http://www.objectspace.com/products/vgrORBpro.asp`.

[OPP 97] OPPLIGER, R.: *IT Sicherheit.* Vieweg, 1997.

[ORD 98] ORDILLE, J.: *When agents roam, who can you trust?* Bell Labs, 2000, `http://cm.bell-labs.com/cm/cs/doc/96/5-09.ps.gz`.

[PES 97] PEINE H., STOLPMANN T.: *The architecture of the Ara platform for mobile agents.* In Proceeding of the First International Workshop on Mobile Agents (MA 97), Volume 1219 of Lectures Notes in Computer Science, Springer Verlag, 1997.

[POL 00] SUN-MICROSYSTEM, SCHEMERS R.: *Policy.java.* Java Quellcode V. 1.3, 2000.

[POW 93] WECK G., POHL H.: *Handbuch 1, Einführung in die Informationssicherheit.* 1993.

[PTP 96] PIKE R., THOMPSON K., PRESSOTTO D. UND TRICKEY H.: *Plan 9 from Bell Labs.* Tech. Rep. A.I. Memo No. 1564 MIT, 1996.

[RED 74] REDELL, D.: *Naming and Protection in Extensible Operating Systems.* M.I.T AD-A001 721, 1974.

[REI 97] REISER, H.: *Sichere TCP/IP-basierte Kommunikation bei der BMW AG.* Diplomarbeit, Technische Universität München, August 1997.

[REV 00] REISER H., VOGT G.: *Security Requirements for Management Systems using Mobile Agents.* Munich Network Management Team, 2000.

[RMS 96] RENESSE R., MINSKY Y., SCHNEIDER R. B. UND STOLLER S. D.: *Cryptographic support for fault-tolerant distributed computing.* In Proceeding of the seventh ACM SIGOPS European Workshop, 1996.

[ROE 99] ROELLE, H.: *Authentisierung und Autorisierung fuer das Java/CORBA-Agentensystem MASA.* Diplomarbeit, Technische Universität München, 1999.

[ROS 00] ROSKIND, J.: *Setscopepermission envolving the security model for Java from Navigator 2.X to Navigator 3.X Java Security Architecture.* WWW, 2000, `http://developer.netscape.com/docs/technote/index.html?content=security/sectn1.html`.

[SAL 74] SALTZER, J. H.: *Information protection and the control of sharing in the Multics system.* Communications of the ACM, 1974.

[SAN 92] SANDHU, R. S.: *The typed access matrix model.* IEEE Computer Society Symposium on Research in Security and Privacy, 1992.

[SAN 96] SANDER, T.: *On cryptographic protection of mobile agents.* In Proceeding of the 1997 Workshop on Mobile Agents and Security, University of Maryland, 1996.

[SAS 75] SALTZER J. H., SCHROEDER M. D.: *The protection of information in computer systems.* Proceedings of the IEEE 63, 1278-1308, 1975.

[SAT 98a] SANDER T., TSCHUDIN C. F.: *Protecting Mobile Agents Against Malicious Hosts.* in G. Vigna (Ed) Mobile Agents and Security, 1998.

[SAT 98b] SANDER T., TSCHUDIN C. F.: *Toward Mobile Cryptography.* In Proceeding of Security and Privacy, Oakland California, 1998.

[SCH 95] SCHOENING, U.: *Theoretische Informatik - kurzgefaßt.* Spektrum (Akademischer Verlag), 1995.

[SHA 86] SHAPIRO, M.: *Structure and Encapsulation in Distributed Systems - The Proxy Principle.* In Proceedings of the 6th International Conference on Distributed Computing Systems IEEE, 1986.

[SOM 00] SOMMER, M.: *Eine Falle in der Programmiersprache Java.* www, 2000, `http://www.zalf.de/fachinfo/fachdoku/pressemitteilung/idw/idw-99/idw-4-99/idw1204991.htm`.

[STE 95] STERN, H.: *Verwaltung von UNIX-Netzwerken mit NFS und NIS.* O'Reilly, 1995.

[SUM 97] SUMMERS, R. C.: *Secure Computing (Threats and Safeguards).* 1997.

[SUN 00a] SUN: *Java Development Kit (JDKTM) 1.1.x - Signed Applet Example.* Sun Microsystems, Inc., 2000, `http://java.sun.com/security/signExample/index.html`.

[SUN 00b] SUN: *JavaTM Authentication and Authorization Service (JAAS) 1.0.* Sun Microsystems, Inc., 2000, `http://java.sun.com/security/jaas`.

[SUN 00c] SUN: *JavaTM Authentication and Authorization Service (JAAS) 1.0.* Sun Micorsystems, Inc., 2000, `http://java.sun.com/products/jaas`.

[TNK 99] TRIPATHI A. R., NEERAN K. M., VORA M. TANVIR A. RAM D. S.: *Mobile Agent Programming in Ajanta.* In Proceedings of the 19th International Conference on Distributed Computing Systems, 1999, `http://www.cs.nmn.edu/Ajanta/publications.html`.

[TUX 01] UNIVERSITÄT-TROMSO: *Tacoma UniX (TuX).* WWW, 2001, `http://www.tacoma.cs.uit.no`.

[UMN 01] MINNESOTA, UNIVERSITY OF: *Ajanta.* WWW, 2001, `http://www.cs.umn.edu/Ajanta`.

[VIG 98] VIGNA, G.: *Cryptographic Traces for Mobile Agents.* in G. Vigna (Ed) Mobile Agents and Security, 1998.

[VOS 95] VOSSBEIN, R.: *Organisation sicherer Informationsverarbeitungsysteme.* 1995.

[WAE 93] WAEHNER, G.: *Datensicherheit und Datenschutz.* VDI Verlag, 1993.

[WAK 00] JANSEN W., KARYGIANNIS T.: *NIST Special Publication 800-19 Mobile Agent Security.* WWW, 2000, `http://hissa.ncsl.nist.gov`.

[WOL 92] WOO T. Y. C., LAM S. S.: *Authorization in distributed systems: A formal approach.* IEEE Computer Society Symposium on Research in Security and Privacy, 1992.

[YEL 96a] YELLIN, F.: *Low Level Security in Java.* Sun Microsystems Java Products Group, 1996, `http://www.javasoft.com/sfaq/verifier.html`.

[YEL 96b] YELLIN F., LINDHOLM T.: *The Java Virtual Machine Specification.* Sun Microsystems, Inc., 1996, `http://www.javasoft.com/docs/books/vmspec/html/VMSpecTOC.doc.html`.