Johann Wolfgang Goethe-Universität
Frankfurt am Main

Fraunhofer Institut für
Graphische Datenverarbeitung

Diplomarbeit

Sichere Kommunikation zwischen Mobilen Agenten

von

STIVENS MILIĆ
Matrikelnummer 1233135

Johann Wolfgang Goethe-Universität Frankfurt
Fachbereich Biologie und Informatik
Institut für Informatik
Robert-Mayer-Straße 11-15
60325 Frankfurt am Main
und
Fraunhofer Institut für Graphische Datenverarbeitung
Fachgebiet Graphisch-Interaktive Systeme
Fraunhoferstraße 5
64283 Darmstadt

Prüfer: PROF. DR.-ING. DETLEF KRÖMKER
Betreuer: JAN PETERS
ULRICH PINSDORF

Aufgabenstellung für die Diplomarbeit des
Herrn cand.-Inform. Stivens Milić
Matrikel-Nr. 1233135

Thema: "Sichere Kommunikation zwischen Mobilen Agenten"

Sicherheit ist ein fundamentales Kriterium für die Verbreitung von mobilem Code. Am Fraunhofer IGD wird im Projekt SeMoA (Secure Mobile Agents) an einer sicherheitsorientierten Plattform für mobile Agenten geforscht. Ein wichtiger, jedoch oft unbeachteter Bereich in diesem Forschungszweig ist die Bereitstellung sicherer Kommunikationsmechanismen für mobile Agenten.

Im Rahmen der Diplomarbeit soll eine Infrastruktur entworfen und implementiert werden, die mobilen Agenten erlaubt, auf sichere Weise plattformübergreifend zu kommunizieren. Der Aspekt der Kommunikationssicherheit ist der Schwerpunkt der Arbeit. Daher sind folgende Eigenschaften zu gewährleisten:

- Informationsvertraulichkeit
- Datenintegrität
- Authentizität
- Schutz vor Wiedereinspielung

Im theoretischen Teil der Arbeit sollen folgende Aspekte berücksichtigt werden:

Literaturrecherche Es soll der State-of-the-Art in den Bereichen Sicherheitsmechnismen für Kommunikationssysteme, sowie Agentenkommunikationssprachen zusammengetragen werden.

Generelle Probleme Es sind die generellen Schwierigkeiten aufzuzeigen, die sich durch herkömmliche Verfahren (z.B. ortsfeste Mailboxen und Forward References) für die Kommunikation zwischen mobilem Code ergeben.

Sicherheitsimplikationen Eine Ausarbeitung über die Bedrohungen und impliziten Vertrauensbeziehungen, die sich durch Verwendung von mobilem Code im Bezug auf die Kommunikation ergeben.

Evaluierung Es ist zu prüfen, ob und in wie fern existierende Lösungen zur Kommunikationssicherheit wie SSL, IPsec oder S/MIME den Anforderungen an mobile Agentenkommunikation genügen.

Related Work Sicherheitslösungen für den Nachrichtentransport in anderen Plattformen für mobile Agenten.

Die zu entwerfende Architektur soll eine flexible Adaption und Konfiguration der Sicherungsmechanismen erlauben. Vorbild für die Architektur soll ein Modell sein, dass sich in SeMoA bereits für die Transportmechanismus bewährt hat. Nachrichten werden nach dem Absenden und vor der Zustellung auf dem jeweiligen Host von Filtern in einer Pipeline bearbeitet. Die einzelnen Filterstufen können unabhängig voneinander der Nachricht Sicherungsinformationen hinzufügen und prüfen. Das dynamische Hinzufügen und Entfernen der Filtern sorgt für eine flexible Adaption auch zur Laufzeit.

Die Sicherungsmechanismen sollen für den Agenten transparent erfolgen. Auf Basis der Annahme, dass ein Agent nicht auf Daten operieren kann, ohne dass diese dem Agentenhost potentiell bekannt sind, sollten die notwendigen Operation vom Host durchgeführt werden. Dies führt dazu, dass die Glaubwürdigkeit einer Nachricht direkt ableitbar ist aus der Glaubwürdigkeit des sendenden Hosts. Weiterhin ergibt sich durch die transparenten Sicherungsmechanismen ein einfaches Programmiermodell für das Entwickeln von Agenten.

Das Lokalisieren von Kommunikationspartnern stützt sich auf den ATLAS-Mechanismus und erfolgt für den sendenden Agenten ebenfalls transparent.

Frankfurt am Main/Darmstadt, den 25. Juni 2002

Prüfer:
Prof. Dr.-Ing. Detlef Krömker
J. W. Goethe-Universität Frankfurt a. M.
Betreuung vor Ort:
Dipl.-Ing.(FH) Ulrich Pinsdorf
Dipl.-Inform. Jan Peters
Fraunhofer IGD, Darmstadt

Frankfurt am Main/Darmstadt, den 27. Dezember 2002

Hiermit erkläre ich an Eides statt, dass ich diese Arbeit eigenhändig und nur mit den angegebenen Hilfsmitteln angefertigt habe.

Stivens Milić

Danksagung

An erster Stelle möchte ich meinen Eltern Branka und Stevo Milic, sowie meiner Schwester Tajana Milic danken, die mich in meinem Tun unterstützt und mir das Studium der Informatik ermöglicht haben.

Weiterhin gilt der Dank meinen Betreuern seitens der Fraunhofer IGD, Jan Peters und Ulrich Pinsdorf, die mir dieses interessante Thema angeboten und mich während dieser Zeit tatkräftig unterstützt haben. Vor allem gilt der Dank Jan Peters der mir stets mit Rat und Tat zu helfen wusste.

Für die Betreuung der Arbeit seitens der Universität Frankfurt, sowie für wertvolle Tips, möchte ich mich bei Prof. Dr.-Ing. Detlef Krömker bedanken.

Selbstverständlich danke ich all denen, die mich während der Entstehung dieser Arbeit unterstützt, mir geholfen und für die nötige Ablenkung und Ruhe gesorgt haben, besonders meiner Freundin Christina Allekotte, meinem Cousin Deniz Raic, meinen Mitbewohnern und all meinen Freunden, Verwandten und Bekannten.

Stivens Milić Frankfurt am Main/Darmstadt, Dezember 2002

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Teil I

Grundlagen

Kapitel 1

Einleitung

1.1 Zielsetzung

Das Konzept des Mobilen Agenten verdankt seine Beliebtheit in der Forschungsgemeinde weltweit, seiner Einfachheit und breitem Spektrum der Anwendungsmöglichkeiten. Mobile Agenten sind Softwareeinheiten die Aufgaben im Auftrag ihres Benutzers autonom ausführen. Sei es die Aufgaben als Hilfe bei der alltäglichen Arbeit oder zum Beispiel bei der Informationssuche im Internet sollen die Agenten ihren Besitzern jedenfalls helfen und zeitintensive und lästige Arbeit abnehmen. Sie sollten ihre Aufgaben möglichst eigenständig lösen können. Falls das bei komplexeren Problemstellungen nicht möglich ist, sollten sie die Möglichkeit haben, andere Agenten zu kontaktieren, um das Problem gemeinsam zu lösen. Sie müssen also in der Lage sein ihr Wissen und Absichten mit anderen Agenten und dem eigenen Besitzer teilen zu können. Weiterhin müssen sie die Möglichkeit haben mit Systemen, Diensten und Anwendungen zu interagieren. Deshalb gehört Kommunikation zu den wichtigsten Diensten der SeMoA-Plattform. SeMoA (Secure Mobile Agents) ist ein Projekt, das an dem Fraunhofer IGD entstanden ist. Genauso wie andere Agentensysteme ist SeMoA für die Anwendung in unsicheren Umgebungen, wie zum Beispiel dem Internet, vorgesehen. Dabei sind die Daten, die Agenten austauschen müssen, gewissen Gefahren ausgesetzt. Sie können ausgespäht, verändert oder vernichtet werden. Da SeMoA ein Agentensystem darstellt, in dessen Mittelpunkt die Sicherheit steht, wird demzufolge die Absicherung der Kommunikation, als eine nötige Voraussetzung für die Weiterentwicklung des Systemes angenommen.

Ziel dieser Arbeit ist es, sichere Kommunikation im SeMoA zu ermöglichen. Die Realisierung der sicheren Kommunikation soll durch die Erweiterung des bestehenden Grundmechanismus für Kommunikation erfolgen. Um der potentiellen Gefahr entgegenzuwirken, sind die wichtigsten Aspekte der sicheren Kommunikation zu berücksichtigen - Authentizität der Kommunikationspartner, sowie Verschlüsselung und Prüfung der Integrität der übertragenden Daten. Aus diesem Grund werden einige aktuelle Sicherheitslösungen im Bereich der Kommunikation, auf ihre Tauglichkeit in SeMoA geprüft und die Wahl einer für die Realisierung geeigneten Sicherheitslösung getroffen.

In Abhängigkeit von dieser Auswahl und mit Berücksichtigung, der in Aufgabenstellung ge-

stellten Anforderungen wird ein Konzept für sichere Kommunikation definiert. Das Konzept wird als Teil der Arbeit daraufhin entsprechend umgesetzt und in SeMoA integriert werden. Großer Wert wird dabei auf eine saubere Implementierung, Erweiterbarkeit und Integration in bestehende Umgebung gelegt, da die sichere Kommunikation ein Bestandteil jeder SeMoA-Plattform werden soll.

Um die Implementierung empirisch zu prüfen, wird weiterhin eine Testumgebung aufgebaut, die sich an einem bestimmten Kommunikationszenario orientiert. Dieses Szenario soll möglichst realistischen Bedingungen bei der Agentenkommunikation entsprechen. In dieser Testumgebung wird die Implementierung ausführlich auf ihre Anwendbarkeit getestet.

1.2 Aufbau der Arbeit

In dem ersten Teil der Arbeit bzw. in den Kapiteln 2, 3 und 4 werden die Grundlagen behandelt und für die Arbeit benötigte Grundbegriffe und Konzepte eingeführt. Jedes dieser Kapitel beschäftigt sich mit einem der Gebiete, die dieser Arbeit zu Grunde liegen. Im Kapitel 2 wird der Begriff des Mobilen Agenten eingeführt bis hin zu dem Begriff des Agentensystems. In diesem Teil der Arbeit wird auch das SeMoA-Projekt mit seinen Grundkonzepten und der Basisarchitektur dargestellt. Besonderes Augenmerk wird dabei auf die Konzepte der Kommunikation und der Sicherheitsarchitektur von SeMoA gerichtet.

Kapitel 3 behandelt die Kommunikation, insbesondere die in Agentensystemen. Durch Darlegung der verschiedenen Kommunikationsformen, der Konzepte der Kommunikationssprachen für Agentensysteme sowie einige realisierte Kommunkationsmodelle wird Einsicht in die aktuelle Welt der Kommunikation zwischen Agenten und Dienste gegeben.

Der Sicherheit als eines der Grundkonzepte von SeMoA wird im Kapitel 4 besonderes viel Aufmerksamkeit geschenkt. Die Sicherheit wird hier durch ein Modell, das die Abhängigkeit zwischen den Sicherheitsbedrohungen, -anforderungen und -maßnahmen aufzeigt, erklärt. Dabei wird bei dieser Darlegung auf die Sicherheit aus dem Aspekt der Kommunikation eingegangen. Zum Abschluss des Kapitels werden existierende Konzepte für die Absicherung der Kommunikation, konkret der SSL, IPSec und S/MIME, unternommen, um die für Realisierung der sicheren Kommunikation in SeMoA in Frage kommende Mechanismen darzustellen.

Der zweite Teil der Arbeit der aus den Kapiteln 5, 6 und 7 besteht, stellt den Hauptteil dieser Arbeit dar. Hier werden in Kapitel 5 zuerst die Anforderungen an sichere Kommunikation in SeMoA, wie auch ein Konzept für ihre Realisierung, dargestellt. Eine Darstellung des vorhandenen Kommunikationsmechnismus in SeMoA, wie auch eine ausführliche Diskussion über den zu verwendenden Sicherheitsmechanismus zu seiner Absicherung, gehören auch zu diesem Kapitel. Abschließend werden hier noch vergleichbare Konzepte für die Absicherung der Agentenkommunikation kurz beschrieben.

Danach folgt in Kapitel 6 die Darstellung der Implementierung mit ihren technischen Einzelheiten und internen Abläufen, für jeden der realisierten Dienste und Klassen. Als nächstes

werden hier noch die Einbettung in die SeMoA-Umgebung beschrieben, sowie die Funktionsweise der realisierten Kommunikationsschnittstelle.

Durch Anwendung eines realitätsnahen Kommunikationsszenario wird die Schnittstelle für sichere Kommunikation noch ausgiebig auf ihre Tauglichkeit in SeMoA getestet. Dazu werden das Testen und die Testergebnisse, die durch das Variation verschiedener Parameter gewonnen wurden, ausführlich in Kapitel 7 dargestellt und diskutiert.

Der letzte Teil dieser Diplomarbeit ist für die Schlussbetrachtung reserviert. Hier werden noch einmal die Ergebnisse der Arbeit zusammengefasst und eine Aufstellung der zukünftigen Entwicklungsmöglichkeiten gegeben.

Kapitel 2

Mobile Agenten

2.1 Einführung

In der letzten Zeit, besonderes in den 90er Jahren hat sich der Bedarf gezeigt, die Technologie der Verteilten Systeme anderes zu gestalten. Die wachsenden Anforderungen an die Technologie der verteilten Systeme resultieren aus rasanten Veränderungen in dem Bereich der neuen Technologien Multimedia, mobile Kommunikation, elektronischer Handel. Zudem haben die wachsenden Bedürfnisse seitens der Benutzer die Entwicklung der Technologie der mobilen Agenten weiter vorangetrieben. Die Technologie der mobilen Agenten ermöglicht eine flexiblere Gestaltung der bestehenden Technologie der verteilten Systeme, die vorwiegend auf dem Client/Server Modell aufbaut.

Das Client/Server Modell ist kaum mehr in der Lage die große Flut an Informationen zu bewältigen und den Anforderungen der Flexibilität, Portabilität und Mobilität, die von dem Systemen heutzutage verlangt werden, gerecht zu werden. Will man über das Internet beispielsweise eine Reise buchen oder einfach ein günstiges Fahrrad kaufen, müssen normalerweise größere Datenmengen an Informationen über die Produkte übertragen und zeitaufwendige Recherche betrieben werden. Diese dienen allein der Entscheidungsfindung beim Kauf eines bestimmten Produkts. Solche Aufgaben können viel effizienter von mobilen Agenten durchgeführt. So könnten hier mobile Agenten selbst verschiedene Online-Shops auf der Suche nach einem passenden Angebot (Reise/Fahrrad) durchsuchen und dem Nutzer am Ende ihrer Reise die Ergebnisse in Form des besten Angebots präsentieren.

Durch die Affirmation des E-Business besonders des E-Commerce hat die Technologie der mobilen Agenten einen neuen Aufschwung bekommen. Außer dem Bereich des E-Commerce sind aber auch viele andere Anwendungsgebiete denkbar oder schon Realität. Die interessantesten Anwendungen sind:

- *Verteilte Informationssuche*: Statt große Datenmengen durch das Netz zu bewegen, können Agenten für den Benutzer benötigte Informationen direkt vor Ort ausfindig machen, bearbeiten und die Ergebnisse zurückliefern.
- *Überwachung und Steuern*: Agenten können zu entfernten Rechnersystemen geschickt

werden um ihren Zustand zu überwachen, sie zu steuern oder Software und Programmupdates zu installieren.

- *Paralleles Rechnen*: Hier werden die Agenten benutzt, um auf verschiedenen Rechnern, parallel jeweils einen Teil einer zeitintensiven Aufgaben zu lösen.

- *Interaktive Hilfe*: Dem Benutzer stehen die Agenten in der Form eines Assistenten für alltägliche Aufgaben zur Seite. Die Hilfe bei Interaktion mit dem System, bei der Benutzung der elektronischer Post, Terminverwaltung usw. sind denkbar.

Weitere Anwendungsbeispiele der Agententechnologie kann mit in [**?**] finden. Durch die Einfachheit des Konzepten und eine breite Palette der Anwendungsmöglichkeiten schreibt man der Technologie der mobilen Agenten heutzutage ein großes Potential zu.

2.1.1 Was sind mobile Agenten? (Modell + Definition)

In der Literatur findet man eine Reihe von Definitionen für Agenten. (siehe z.B. [**?**]). Mit dem Begriff Agent beschäftigen sich viele verschiedene Disziplinen, die mit ihren Beiträgen die Eigenschaften des Agenten prägen. Deshalb ist es schwer eine allgemeine, umfassende Definition geben, die allen Disziplinen genügen würde. Eine der gängisten Definitionen ist allerdings die Definition von *Jennings* und *Woolridge* [**?**], nach der der Begriff Agent ein softwarebasiertes System bezeichnet, welches folgende Eigenschaften erfüllt:

- *Autonomie*: Agenten arbeiten ohne direkte Einflussnahme durch Menschen oder andere Systeme, haben Kontrolle über ihren eigenen Zustand und können ihre Aktionen selbst planen.

- *Sozialfähigkeit*: Agenten kommunizieren mit anderen Agenten oder Menschen über eine Agentensprache (Agent Communication Language, ACL)

- *Reaktionsfähigkeit*: Agenten nehmen ihre Umwelt (die reale Welt, einen Benutzer, andere Agenten, das Internet) wahr und reagieren auf Veränderungen.

- *Selbstständigkeit*: Agenten reagieren nicht einfach nur auf Reize, die sie aus ihrer Umwelt empfangen, sie können auch selbstständig aktiv werden, um ihre Ziele zu erreichen.

Diese Definition beschreibt eher allgemeine Eigenschaften, die Agenten aufweisen sollten, unabhängig davon, ob es sich um einen Agenten in digitaler oder physikalischer Umgebung handelt. Mit den Möglichkeiten selbst zu handeln, zu lernen, zu agieren und auf Reize zu reagieren, ist dieses Konzept nahe dem Bereich der Künstlichen Intelligenz (KI). Deshalb nennt man solche Agenten auch intelligente Agenten. Obwohl das Konzept des intelligenten Agenten eine Erweiterung eher statischer Konzepte der verteilten Verarbeitung wie z.B. *RPC* (Remote Procedure Call) oder Objekttechnologie ist, bewegen sich die Bemühungen der Forschungsgemeinschaft in Richtung mehr Intelligenz und intelligenter Agentensysteme.

In dieser Diplomarbeit beschäftigt ich mich mit den auf Software-Systemen basierten Agenten, deshalb wird der Begriff des Agenten wie folgt definiert und benutzt:

> *Ein Agent ist eine autonome Softwareeinheit die bestimmte Aufgaben im Auftrag der Person oder Organisation durchführen kann.*[**?**]

Unter einem Softwareagent kann demnach alles verstanden werden, was den Benutzer bei der Arbeit unterstützt, ihm aufwendige Aufgaben abnimmt, in seinem Sinne und ohne seine Mitwirkung handelt. Sehr wichtig für die Softwareagenten ist aber die Eigenschaft, die Lokalitäten oder Agentenplattformen ändern zu können, bzw. das mobile Verhalten. Man spricht hierbei von Migration, wobei ein Agent der die Eigenschaft der Migration besitzt, als mobiler Agent genannt wird. Mit der Eigenschaft der Migration zusammen mit den Definition von *Jennings* und *Wooldridge* kann man den mobilen Agenten auch als Softwareeinheit charakterisieren die folgende Eigenschaften besitzt:

- *Autonomie*
- *Intelligenz*
- *Kooperation*
- *Kommunikation*
- *Migration*[**?**]

Ein mobiler Agent kann als ein Stück Software betrachtet werden, daß in der Lage ist, durch das Netzwerk zu wandern und verschiedene Aufgaben im Namen des Benutzers durchzuführen. So könnte er z.B. nach Informationen in verschiedenen Datenbanken suchen, billigste Angebote bei verschiedenen Online-Händler ausfindig machen oder den Zustand bestimmter Rechnersysteme kontrollieren. Der mobile Agent bekommt in diesem Fall ein festgelegtes Ziel vorgegeben. Um seine Aufgabe zu erfüllen, kann er dabei vom Rechnen zu Rechner wandern und seine Daten mitzunehmen. Der Benutzer bekommt am Ende eventuell die Ergebnisse, wie beispielsweise in dem oben eingeführten Online-Händler Beispiel, in der Form des billigsten Angebots eines bestimmten Produkts präsentiert, oder der Agent erledigt den Kauf sogar selbst.

Außer den mobilen Agenten gibt es auch sog. stationäre Agenten. Die stationäre Agenten sin im Gegensatz zu den mobilen nicht in der Lage zwischen verschiedenen Plattformen zu wechseln, sondern verbringen ihre gesamte Laufzeit, auch als Lebenszyklus (lifecycle) genannt, auf einem System. Ein Beispiel für stationäre Agenten sind z.B. Agenten die Benutzer bei ihrer täglicher Arbeit mit dem Betriebssystem unterstützen. Für diese Agenten gebe es kein Bedarf an Migration, da ihre Aufgabe und Bedeutung nur auf einem System Definition findet. Stationäre Agenten gehören nicht zum Gegenstand dieser Arbeit. Wenn im Folgenden über Agenten gesprochen wird, sind immer mobile Agenten gemeint.

Der Vorteil der mobilen Agenten gegenüber anderen Konzepten aus verteilter Welt ist der, daß der Agent in der Lage ist, autonom zu handeln, zu entscheiden, seine Recherche und Berechnungen lokal durchzuführen und ihre Ergebnisse mitzunehmen. Es muß keine permanente Netzverbindung gehalten werden, der Agent arbeitet sozusagen "off-line". Auf diese Weise

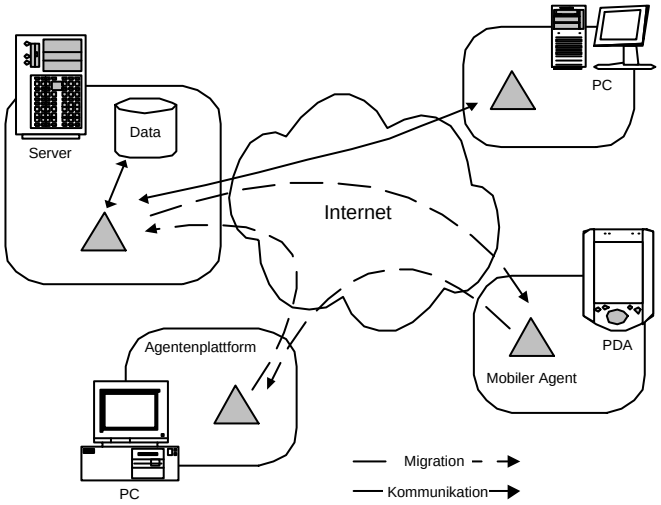

Abbildung 2.1: Agenten

werden Software- und Hardwarekomponenten durch verringerten Netzverkehr und Rechenzeit, Lastverteilung entlasten, wie auch der Benutzer, dem die Aufgabe dadurch abgenommen wird. Der weitere entscheidende Vorteil der Agenten ist die Möglichkeit ihre Ausführung auf einem neuen Rechner idealerweise an der Stelle fortzusetzen, wo sie auf dem vorigen aufgehört hat. Mobiles Code im Allgemeinem hingegen, hat diese Eigenschaft nicht und ist somit zustandslos.

2.1.2 Mobile-Agenten-Systeme

Durch die Eigenschaft der Migration sind die Agenten in der Lage vom Rechner zu Rechner zu wandern um ihre Aufgaben zu erfüllen. Mobile Agenten sind dadurch nicht an die Lokalität gebunden, wo sie entstanden sind, sondern bewegen sich selbstständig durch das Netz und benutzen die Dienste, die ihnen auf den bestimmten Wirtsystemen zur Verfügung stehen. Um die Migration zwischen Rechnern zu ermöglichen braucht man spezielle Netzwerk-Infrastruktur. Diese besteht aus dem Agentenserver der die Ausführungsumgebung bereitstellt und Diensten für Management und Kontrolle über Agenten, Kommunikation zwischen Agenten, Migration und weitere. Diese Infrastruktur wird auch Agentenplattform oder Agentensystem genannt.

Das Agentensystem nimmt den Agenten auf, ermöglicht ihm die Ausführung und weitere Migration. Will ein Agent migrieren, verpackt ihn die Agentenplattform samt seiner Daten (Serialisieren) und transferiert ihn zum gewünschtem Zielsystem. Dort wird der Agent wieder ausgepackt (Deserialisieren) und ausgeführt. Dabei bleiben sein Zustand und die Daten erhalten.

Agentensysteme sind ähnlich wie Agenten durch ihre Autorität (Person oder Organisation) charakterisiert. Die Agenten eines bestimmten Types laufen meistens nur auf dem Agentensystemen des gleichen Types. Agentenplattformen sind oft in der Lage die Existenz mehrerer

Agenten gleichzeitig zu ermöglichen und sie zu bedienen. In diesem Kontext spricht man von Multiagentensystemen. Systeme die nur aus einem einzigen Agenten bestehen, erweisen die Eigenschaften der Migration, Kooperation und Kommunikation nicht, deshalb sind die für diese Diplomarbeit eher uninteressant.

Bestehende Plattformen für Mobile Agenten unterscheiden sich in den Konzepten, in der Architektur und der Implementierung erheblich voneinander. Diese Unterschiede erschweren die Möglichkeit der Migration von Agenten zwischen der Agentensystemen verschiedener Hersteller. Um diesem zu entgegenwirken, spricht man von Bedürfnissen der Interoperabilität zwischen Systemen. Die Interoperabilität stellt die Grundlage für Akzeptanz und Verbreitung der Agententechnologie.

So entstanden auch verschiedene Konzepte, die als Referenz für Agentensysteme dienen und die Anforderungen an ihre Komponenten beschreiben. Die bedeutesten sind Bsp. MASIF und FIPA.

MASIF

Die Mobile Agent System Interoperability Facility (MASIF)[**?**] ist eine Entwicklung der *Object Management Group* (*OMG*). OMG wurde 1989 von mehreren großen Firmen gegründet, mit dem Ziel, die Anwendung der standardisierte Objektensoftware voranzutreiben. MASIF wurde 1997 von OMG als Standard veröffentlicht. Im Mittelpunkt des MASIF-Konzeptes steht die Unterstützung der Interoperabilität zwischen heterogenen Agentensystemen. Um die Interoperabilität zu sichern, war es aus Sicht der OMG nötig, folgende Bereiche zu standardisieren:

- *Agentenmanagement* zum Verwalten der Agenten
- *Agententransfer* zur Unterstützung der Mobilität
- *Namensgebung* für Agenten und Agenten Systeme, wegen eindeutiger Identifizierung
- *Lokalisierung* von Agenten und Agenten Systemen, um positionstransparente Kommunikation zu ermöglichen
- *Kommunikation* zwischen Agenten und Agentensysteme

Ein Agent im Sinne von MASIF ist ein autonome Softwareeinheit mit eindeutigem Namen, die entweder stationär oder mobil sein kann und im Auftrag seiner Authorität (Person oder Organisation) handelt. Das Agentensystem soll die Laufzeitumgebung für Agenten zur Verfügung stellen. Jeder Agent läuft in seiner eigenen Ausführungsumgebung, in der für ihn bestimmte Zugriffsrechte auf Ressourcen gelten. Um Kommunikation und Migration zwischen Agentensysteme zu ermöglichen, stützt sich MASIF auf die Mechanismen der Middleware CORBA (Common Object Request Broker). Für Lokalisierungs- und Namensdienste definiert MASIF den sog. `MAFFinder`, der als Schnittstelle von CORBA, seine Dienste anbietet.

FIPA

Die Foundation for Intelligent Physical Agents (FIPA)[**?**] wurde 1996 mit Sitz in Genf gegründet. Sie ist eine non-profit Organisation dessen primäre Aufgabe die Entwicklung und Spezifikationen von Konzepten im Bereich der Agententechnologie ist. FIPA's Beitrag zur Entwicklung von Agentensysteme basiert auf einer minimalen Architektur für Management der Agenten in einer offenen Umgebung. Mit ihrem FIPA-Referenzmodell ist eine normierte Beschreibung der wichtigsten Bestandteile einer agentenbasierende Softwarearchitektur gegeben. Zur Agentenkommunikation definiert FIPA das FIPA-ACL, eine an Knowledge Querying and Manipulation Language (KQML)[**?**] angelehnte Sprache, mit deren Hilfe Agenten die Informationen austauschen können. Die aktuelle FIPA98 Spezifikation schlägt folgende Architektur vor:

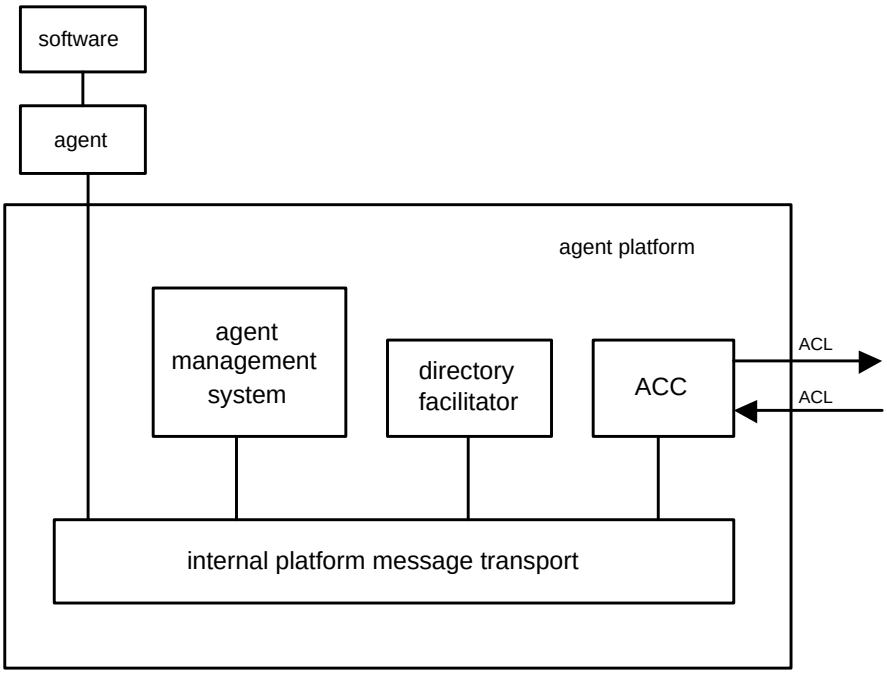

Abbildung 2.2: FIPA-Referenzmodell

FIPA baut auf intelligenten Agenten im Sinne der KI auf. Das agent management system (AMS) dient als Namensdienst. Es stellt einen Agenten dar, der ein Index mit Namen aller auf der Plattform registrierte Agenten hält und für den Agenten-Lebenszyklus zuständig ist. Der directory facilitator (DF) ist ein spezieller Agent, der als als Verzeichnisdienst auf der Platform agiert. Die Agenten melden eigene Dienste beim DF an und können Informationen über andere Agenten erfragen. Das DF definiert Domänen, die eine logische Organisation der Agenten mit der selben Aufgabe in Gruppen darstellt. Die Aufgabe des agent communication channel (ACC) ist die Kommunikation zwischen Agenten auf der selben Platform und zwischen den Agenten auf verschiedenen Agentenplattformen. Für die Kommunikation wird ACL eingesetzt.

Aus dem Dargestellten kann man erkennen. daß sich Bestrebungen bei MASIF auf die Interoperabilität, während sie sich bei FIPA eher auf die Kommunikation konzentrieren. In seinem Aufbau stützt sich MASIF auf bestehende Objekttechnologie. FIPA auf der anderen Seite setzt auf die Kommunikationsmechanismen zwischen den intelligenten Agenten.

2.1.3 Beispiele für mobile Agentensysteme

Es gibt verschiedene Agentensysteme die praktisch realisiert sind. Besonderes das Entstehen der Programmiertechnologien wie z.B. Telescript, Tcl (Tool Command Language) und Java, welche die Entwicklung solcher mobilen Konzepten unterstützen, hatten eine positive Auswirkung auf die Entwicklung der Agentensysteme. Einen guten Überblick über die bestehenden Agentensysteme bekommt man beispielsweise in The Mobile Agent List[**?**], in der momentan über 60 Agentensysteme verzeichnet sind. Einige davon sind Aglets[1] von *IBM*, JADE[2] und SeMoA von *Fraunhofer-IGD*.

Aglets

Das IBM-Aglet-Konzept ist ein auf MASIF aufbauendes Konzept. Es erweitert das Konzept des Java-Applets, in dem Sinne, daß Bytecode samt seinem Zustand durch das Netz übertragen wird. So kann der Agent, nach dem erreichen des Zielsystems seine Arbeit ausgehend vom letzten Zustand, aufnehmen. Der Bytecode wird über einen Class Loader auf der Plattform zur Ausführung gebracht. Die Verwaltung des Agentensystems erfolgt über eine zentrale Instanz. Die Kommunikation findet lokal oder global statt, und zwar über sog. AgletProxies. Als Kommunikationsmechanismen werden RPC, um die öffentliche Methoden der anderen Agenten aufrufen zu können und Message Passing, zum Transfer von Daten zwischen Prozessen, benutzt. Die Proxies sollen dabei den Agenten vor unbefugter Manipulation schützen. Es sind auch andere Schutzmechanismen implementiert. So erfolgt der Schutz der einzelnen Komponenten des Agentensystems z.B. durch der Definition von Prinzipalen und ihren Privilegien. Prinzipale können Aglets, Agleterzeuger und -besitzer sein, und sind durch ihre Privilegien, bzw. Rechte auf dem System ausgezeichnet. Einen erheblichen Teil der Sicherheitsprüfung vor der Ausführung der Agenten erledigen die Java-Komponenten Verifier, die Aglets auf korrektes Format untersucht und Security-Manager, der die Restriktionen bezüglich einzelner Operationen vergeben kann.

JADE

JADE ist eine Agenten-Plattform das von CSELT Telecom Italia Group[3] und dem Universität in Parma (Italien) entwickelt wurde. JADE ist ein Beispiel eines Agentensystems, das im Java entwickelt wurde und auf FIPA aufbaut. Nach dem Entwickler von JADE kann JADE als eine:

[1] IBM Aglets, http://www.trl.ibm.co.jp/aglets
[2] JADE, http://sharon.cselt.it/projects/jade
[3] CSELT, http://www.cselt.it

Agenten-Middleware betrachtet werden, die eine Agentenplattform und ein Entwicklungsgerüst darstellt.[**?**]

JADE wurde vollständig nach den Spezifikationen der FIPA entwickelt. So implementiert es alle von der FIPA vorgeschriebene Komponenten wie AMS, ACC und DF. Die Verwaltung und Kontrolle über Agenten ist über eine Java Graphical User Interface (GUI) möglich. Über die GUI kann man die Agenten starten, stoppen und konfigurieren. Die Kommunikation baut auf Message Passing auf und benutzt FIPA-ACL als Kommunikationssprache. Die Agentenplattform kann in mehrere Teile aufgeteilt werden, die auf dem selben oder verschiedenen Hosts laufen. Jeder Teil läuft in einer eigenen Java Virtual Machine (JVM) und kommuniziert über Java RMI (Remote Method Invocation) mit anderen JVM's. Jede JVM stellt ein sog. Container für meherere Agenten dar, und stellt die Runtime-umgebung für sie bereit. Die Agenten laufen dabei in eigenen Treads. JADE implementiert auch ein paar Tools bzw. spezielle Agenten mit bestimmten Aufgaben im System, von denen der sog. "sniffer agent" vielleicht der interessanteste ist. Der "sniffer agent" ist in der Lage den Nachrichtenaustausch auf der Plattform zu verfolgen. So können eingehende und ausgehende Nachrichten der Agenten auf Wunsch protokolliert und gezeigt werden.

2.2 SeMoA

Secure Mobile Agents (SeMoA)[4] ist ein Forschungsprojekt, daß vom Fraunhofer Institut für Graphische Datenverarbeitung ins Leben gerufen worden ist. SeMoA stellt ein Agentensystem dar, bei dessen Entwicklung die Sicherheit in den Vordergrund gestellt worden ist. SeMoA ist vollständig in Java programmiert. Die Besonderheit von SeMoA gegenüber den meisten anderen Agentensysteme ist die, daß SeMoA durch Anwendung der Sicherheitskonzepten und -mechanismen einen gewissen Maß an Sicherheit in vielerlei Hinsicht bietet. Böswillige Server könnten die Agenten angreifen, verändern oder gar so manipulieren, daß sie Schaden auf den anderen Servern anrichten, bzw. als Trojanisches Pferd benutzen. Bösartige Agenten könnten anderseits Server angreifen, sie als Platform für Angriffe auf andere Systeme benutzen, Viren und Würmer übertragen oder andere Agenten angreifen. Da Agentensysteme und Agenten viel Angriffsfläche bieten, sind viele Möglichkeiten hier denkbar. Deshalb laufen die Anstrengungen der Entwickler von SeMoA in Richtung der Absicherung gegenüber böswilligen Angriffen an Agenten, Plattformen und Diensten die SeMoA bereitstellt. Die folgende Beschreibung des SeMoA Agentensystemen ist der Entwicklerdokumentation [**?**], [**?**] und [**?**] entnommen. Für eine tiefere Analyse sei der Leser auf diese Referenzen verwiesen.

2.2.1 Grundkonzepte und Architektur

SeMoA besteht in ihrer Grundform aus einem stationärem Teil - SeMoA-Plattform und mobilen Teilen - SeMoA-Agenten. Die Architektur der SeMoA kann grob in folgende Konzepte aufgeteilt werden:

[4] SeMoA, www.semoa.org

- *SeMoA-Plattform*
- *Migration*
- *Lokalisierung*
- *Kommunikation*

SeMoA-Plattform kann als eine Ansammlung von Diensten sog. Services und einer SeMoA-Shell verstanden werden, die innerhalb einer JVM laufen. Um diese Dienste lokal zu managen und benutzen zu können, benutzt SeMoA das Environment.

Beim Starten von SeMoA werden die Objekte durch die JVM geladen und durch einen key, daß im Environment abgelegt wird, eindeitig gekennzeichnet. Der Key kann als Stringrepräsentation eines Pfades vergleichbar mit den Pfaden im UNIX-Dateisystem, verstanden werden. Der Unterschied zum UNIX-Dateisystem ist der, daß bei SeMoA kein Unterscheidungen zwischen Dateien und Verzeichnisse gemacht werden. Der Namenraum im Environment ist dabei hierarchisch organisiert. So wird z.B. der Dienst, der für das Deseriallisieren von Agenten zuständig ist, unter dem Pfad "/transport/ingate"publiziert. Die folgende Operationen auf Objekten aus Environment sind möglich: `lookup`, `publish` und `retract`. Zugriff auf die Elemente des Environments unterliegt der Zugriffskontrolle, die durch die Sicherheitskontrollmechanismen konfigurierbar ist. Die Objekte in der Laufzeitumgebung sind von einer Sicherheitsschicht, einem sog. Proxy, umhüllt. Der Proxy stellt die Schnittstelle der Objekte zur Außenwelt dar. Er schützt dabei das Objekt von ungewollten Zugriffen außerhalb.

Ein weiteres wichtiges Konzept in SeMoA ist die Resource. Eine Resource stellt eine Abstraktion der Datenspeicherung dar. Sie besteht aus einer Menge an Dateien mit verschiedenen Namen. Sie wird im Sinne von *streams* in Java behandelt, wobei es für Benutzer der Resource transparent ist, ob sich die Datei lokal auf der Festplatte, im Speicher, in eine Datenbank oder auf einem entfernten System befindet.

Die Kontrolle über die Plattform und darauf befindlicher Services bzw. Agenten ist durch konfigurierbare Einstellungen und durch der SeMoA-Shell möglich. Die SeMoA-Shell kann man als eine Art Betriebssystem verstehen. Sie ist in Form einer Eingabeaufforderung präsent, welche die direkte Eingabe von Kommandos ermöglicht. Unter SeMoA-Shell kann man die Dienste starten, stoppen, Agenten starten und den Zustand des Systems kontrollieren.

Der Agent im Sinne von SeMoA ist eine Java-Klasse, die Java-Schnittstellen `Runnable` und `Serializable` implementiert. Diese ermöglichen das Serialisieren des Agenten und seine Ausführung in der Multi-Thread Umgebung jeden Servers. Die Agenten werden in einer eigenen Laufzeitumgebungen ausgeführt und laufen in einer eigenen *thread group*. Agenten werden in SeMoA von dem sog. `AgentContext` gekapselt. `AgentContext` enthält wichtige Informationen, über den Agenten selbst oder die mobilität- und kommunikationbezogenen Daten, auf die der Agenten zugreifen kann. Außredem sind Agenten weiterhin in der Lage, falls sie über entsprechende Rechte verfügen, Objekte (Daten und Dienste) in dem Environment zu publizieren und abzufragen. Jeder Agent besteht aus einem statischen und einem

dynamischen Teil. Der statische Teil bleibt während des Lebenszeit des Agenten unverändert, anderseits kann sich der dynamische Teil des Agenten, der normalerweise die serialisierten Klassen und andere Daten enthält, ändern. Jedem Agent ist außerdem ein eindeutiger Identifikator, bzw. Name zugewiesen. Dieser setzt sich aus der Anwendung einer Hashfunktion auf die digitale Unterschrift des statischen Teil des Agenten zusammen. Die Lebenszeit (lifecycle) des Agenten, die Aufgaben, die er erfüllen sollte, seine Migrations- und Kommunikationsziele sind Gegenstand der Agentenprogrammierung. Der Agentprogrammierer bestimmt die Parameter des Agenten während die SeMoA-Plattform die Laufzeitumgebung, Resourcen und die Dienste zur Verfügung stellt.

2.2.2 Migration

Die Migration ist die essentielle Eigenschaft in mobilen Agentensystemen. Deshalb wurde ihr bei der Entwicklung von SeMoA besondere Aufmerksamkeit geschenkt. Die Migration stellt einen einfachen Mechanismus vor, der aus dem Serialisieren, Deserialisieren und Transport von Agenten besteht.

Die Agenten signalisieren selbst den Wunsch zu migrieren. Die Plattform schickt den Agenten daraufhin durch eine Pipeline aus Sicherheitsfilter, die den Agenten samt seinen Daten in eine JAR-Datei verpackt, verschlüsselt und signiert (Serialisieren). So verpackt wird der Agent über einen Transportport auch `gateway` genannt auf die Ziel-Plattform verschickt. Es gibt zwei Arten von Gateways - solche für den Empfang (`ingate`) und solche für das Versenden von Agenten (outgate). Der Transportweg kann von SeMoA gesichert werden. Auf dem Zielsystem durchläuft der Agent, bevor er zur Ausführung kommt, wiederum die Reihe der Sicherheitsfilter, die den Agenten entpacken, seine Signatur prüfen und teilweise entschlüsseln (Deserialisieren).

Der Agent wird erst dann Migrieren können nachdem er terminiert ist, bzw. seine Ausführung geendet hat. Eine weitere Voraussetzung für die Migration ist das Setzen eines Tickets. Das Ticket ist eine Referenz, die in der Form einer URL, die Adresse des nächsten Zielsystems enthält.

2.2.3 Lokalisierung

Der Lokalisierungsdienst ist ein wichtiger Bestandteil des Agentenmanagements und der Kommunikation. Damit die Agenten effektiv verwaltet werden können und damit die Kommunikation zwischen Agenten überhaupt möglich ist, muss das Kenntnis über den Aufenthaltsort des Agenten vorhanden sein. Die Lokalisierung wie auch die Namensgebung wird in SeMoA durch einen Dienst mit Namen Agent Tracking und Location Service (ATLAS)[**?**] ermöglicht. Die Agenten werden vom ATLAS durch einen eindeutigen Identifikator (Hashcode des Agenten) identifiziert. Für die Zuordnung dieser Identifikatoren zu dem Agenten ist der Namensdienst zuständig. Die Lokalisierung des Agenten und deren Registrierung bei dem Lokalisierungsdienst wird von einem System, das auf einer Client-Server Architektur aufbaut, durchgeführt. Wenn ein Agent auf einem Agentenserver ausgeführt wird, verschickt der auf

diesem Server installierte Client die Registrierungsanfrage (`register`) mit dem Agentenidentifikator an den Lokalisierungsserver. Um die Position eines bestimmten Agenten herauszufinden, verschickt der Client eine Lokalisierungsanfrage (`lookup`) an den Server, der mit der aktuellen Position bzw. Kontaktadresse des Agenten antwortet. Dieses Konzept ist skalierbar, da es die Existenz mehrerer Server zur Agentenlokalisierung ermöglicht. Die Kommunikation zwischen den Clients und Server kann durch ein spezielles Protokoll geschützt werden.

2.2.4 Kommunikation

Der Kommunikationsdienst gehört zu den Punkten der SeMoA-Plattform die noch in der Entwicklungsphase stecken. Die Basisfunktionalität der Kommunikation ist schon vorhanden. Diese reicht zum einfachen Austausch der Nachrichten zwischen Agenten aus. Der Kommunikationsdienst wird in der Form eines Systems, das aus einem augehendem, eingehendem Puffer und Kommunikationskanal besteht, für den Nachrichtenaustausch den Agenten zur Verfügung gestellt. Die Platform kümmert sich um ordnungsgemäße Auslieferung der Nachrichten, der Agent bzw. Agentenprogrammierer um Adressierung, Initiierung der Kommunikation, Nachrichten- und Fehlerbehandlung.

Unabhängig davon, ob der Empfänger der Nachricht lokal auf dem selben Server oder einem entfernten Server befindet, wird die Nachricht, für den sendenden Agenten völlig transparent, verschickt. Dabei benutzt SeMoA das Konzept der lokalen Mailbox. Alle Nachrichten für Agenten landen in seiner Mailbox. Für lokale Auslieferung werden die Nachrichten direkt an die Mailbox verschickt, für entfernte Auslieferung bedient sich SeMoA an einem speziellem Gateway. Dieses besteht aus einem Client, der die Nachricht verschickt, einem Server auf dem Zielsystem, der auf eingehende Nachrichten wartet und einem Kommunikationskanal mit speziellem Protokoll, der für Kommunikation zwischen den beiden zuständig ist. Nachdem die Nachrichten an die Mailbox des empfangenden Agenten ausgeliefert sind, muß er sie dann selbst abholen. Will ein Agent migrieren, werden bei dem Serialisieren des Agenten auch seine, aus dem Mailbox nicht abgeholte Nachrichten mit verpackt. Auf dem Zielsystem werden sie dann ausgepackt und in seine Mailbox gestellt. Auf solch eine Weise gehen die Nachrichten nicht verloren. Falls Nachrichten nach einigen Versuchen doch nicht ausgeliefert werden können, wird der sendende Agent mit entsprechender Fehlermeldung benachrichtigt.

Der Kommunikationsdienst ist von besonderer Wichtigkeit. Ohne Kommunikation haben Agenten keine Möglichkeit mit ihrem Besitzer oder anderen Agenten ihr Wissen zu teilen und zu kooperieren.

2.2.5 Sicherheitsarchitektur

Die Sicherheitsstruktur der SeMoA gehört zum Hauptpunkt der Entwicklung. Sie definiert spezielle Schutzmechanismen gegen folgende Szenarien:

- Angriffe des Agentenservers auf Agenten
- Angriffe der Agenten auf Agentenserver

- Angriffe zwischen Agenten
- Verändern und Ausspähen von Agenten während des Transports
- Verändern und Ausspähen anderer Daten

Um diesen Szenarien vorzubeugen, baut SeMoA auf eine Sicherheitsstruktur, die symbolisch mit einer Zwiebel (vgl. Abbildung **??**) verglichen werden kann. In der Mitte der Sicherheitsstruktur befindet sich das Laufzeitsystem unter dem die Agenten laufen. Auf dem Weg zur Laufzeitumgebung müssen die Agenten eine Reihe an Sicherheitschichten passieren.

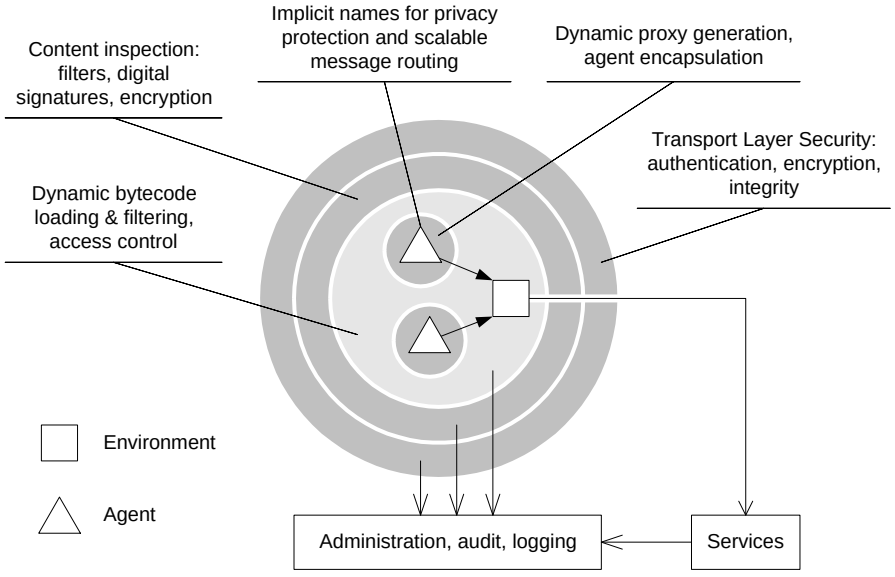

Abbildung 2.3: SeMoA-Sicherheitsarchitektur

Die äußerste Schicht der SeMoA-Sicherheitsstruktur ist ein transport layer, der momentan durch Kommunikationsprotokolle wie TLS oder SSL abgesichert wird. Diese Protokolle unterstützen gegenseitige Authentisierung, transparente Verschlüsselung und Integritätsschutz. Auf sie wird näher im Kapitel 4.4 eingegangen.

Die zweite Schicht besteht aus einer Pipeline aus Sicherheitsfilter. Diese Pipeline wird von jedem eingehenden und ausgehenden Agenten durchlaufen. Jeder Sicherheitsfilter hat dabei eine bestimmte Aufgabe. Momentan sind in der Pipeline vier verschiedene Filtertypen realisiert. Zwei gegensätzliche Paare der Filter sorgen dafür, daß die Agenten signiert, dessen Signatur geprüft wird, selektiv verschlüsselt und entschlüsselt werden. Sie können den Agenten akzeptieren oder verwerfen. Weitere Filter haben als Aufgabe die Vergabe der Rechte an Agenten auf Basis der Sicherheitseinstellungen und Informationen vorherige Filter. Am Ende gibt es einen eher informativen Filter, der den Lokalisierungsdienst in SeMoA über eingehende und ausgehende Agenten informiert.

Nach dem erfolgreichem Durchlaufen der Filter, wird für den Agenten eine sog. "Sandbox" bzw. Laufzeitumgebung erstellt. Dem Agenten wird eine eigene *thread group* und ein eigener *class loader* zugewiesen. Die Sandbox stellt die vierte Sicherungsschicht von SeMoA dar.

Jeder Agent wird durch einen eigenen class loader geladen. Neben der Agentenbasisklasse werden auch Klassen geladen, die mit dem Agenten mittransportiert werden. Diese Klassen müssen von dem Benutzer digital unterschrieben werden. Dies wird beim Laden auch geprüft um sicher zu gehen, daß alle Klassen dem eigentlichen Agenten gehören und nicht vielleicht trojanische Pferde sind. Bevor die Klasse aber durch den class loader in die JVM der Maschine geladen wird, muß sie durch eine Pipeline aus Filter für den Bytecode durchlaufen. Momentan wird der Bytecode durch einen Filter auf die Implementierung der `Finalize()` Methode geprüft. Mit `Finalize()` könnte ein böswilliger Agent den *garbage collector* der JVM blockieren und so die JVM zum Absturz bringen. Es sind aber auch andere Prüfungen und Änderungen des Bytecodes an dieser Stelle denkbar. Diese Filterung kann als dritte Sicherheitsschicht aufgefasst werden.

Anschließend gelangen die Agenten in die Sandbox. In dieser Umgebung sind sie aber von den anderen Agenten strikt getrennt. Um die Angriffe zwischen den Agenten vorzubeugen, ist es für die Agenten nicht möglich Objektreferenzen mit anderen Agenten auszutauschen. Das Environment stellt die einzige Möglichkeit, Informationen bzw. Objekte zu publizieren und auszutauschen. Dabei sind die Objekte im Environment, wie oben erwähnt, von einem Proxy umhüllt, so daß keine direkten Referenzen auf die eigentlichen Objekte möglich sind.

Auch außerhalb der SeMoA-Plattform sind Agenten geschützt. Die JAR-Dateien in die Agenten bei der Migration verpackt werden, werden durch das zusätzliche digitale Signieren und Verschlüsselung sicherer gegen Ausspähung und Veränderung ihrer Daten gemacht. Diese Aufgaben übernehmen die schon erwähnte Sicherheitsfilter. Weiterhin werden die Agenten, außer dem erwähntem Signieren des statischen Teiles, von jeder SeMoA-Platform komplett signiert. Dieses zweifache Signieren schützt den Agenten vor sog. "copy & paste"-Angriffen. Bei dieser Art von Angriffen werden die verschlüsselte Daten von einem in einen anderen Agenten kopiert. So könnte man an sie bei nächstem Deserialisieren herankommen. Außerdem ist es auch noch möglich, den kompletten Transportweg zwischen den Plattformen, durch den Aufbau einer sicherer Verbindung zu schützen.

Die in SeMoA verwendeten Sicherheitsmechanismen bauen auf den Sicherheitsarchitekturen der Java auf: Java Cryptography Architecture/Extension (JCA/JCE) und Java Secure Socket Environment (JSSE) auf. Diese Architektur ermöglicht einem einfachen Austausch der tatsächlich genutzten Kryptoverfahren, durch die Auswahl des aktuellen Krypto-Providers.

2.2.6 Offene Punkte

Trotz einem sehr gutem Konzept, gibt es bei SeMoA noch paar offene Fragen. Da SeMoA auf dem größtenteils unverändertem core package von Java gebaut ist, erbt es automatisch einige Nachteile der Java-Implementation.

So ist es bei SeMoA nicht möglich die Agenten vorzeitig zu terminieren bzw. zu stoppen. Das liegt daran, daß es bei Java ab Version 2 keine Möglichkeiten mehr gibt die Threads zu manipulieren. Die früheren Methoden, die das ermöglichten, wurden in der neuen Version deprecated[5], da sie früher zu Unstabilitäten im System führten. Außerdem ermöglichen die Methoden aus Java das Synchronisieren auf Objekten, was mit anderen Worten heißt, daß die Agenten, die Zugriff auf ein Objekt haben, den Zugriff der anderen Threads blockieren.

Weiterhin ist SeMoA auch anfällig gegen vielerlei (heute oft thematisierte) Denial-Of-Service-Attacken. Die Plattform kann beispielsweise mit einer großen Anzahl an Agenten überflutet werden, bis sie nicht mehr antwortet. Es ist auch denkbar, daß der Speicher des Rechnersystems durch übermäßige Nutzung der Ressourcen zur Ausschöpfung gebracht wird. Gegen solche Angriffsformen gibt es momentan in Java keinen wirksamen Schutz.

Außer diesen Punkten, ist es nötig einige weitere Erweiterungen und Verbesserungen in SeMoA durchzuführen. So müssen SeMoA-Dienste wie beispielsweise Kommunikationsdienst besser ausgebaut werden. Außerdem, eine einfachere und benutzerfreundlichere Verwaltung des Systems und Agenten durch z.B. GUI-basierte Elemente wäre ebenfalls denkbar.

[5] Die Klassen werden in Java mit `deprecated` bezeichnet, falls sie entweder in der neueren Java-Version durch neue Klassen ausgetauscht werden oder ihre Benutzung wegen der Problemen gemieden werden sollte.

Kapitel 3

Kommunikation in Agenten-Systemen

3.1 Kommunikationsformen

Nach Brockhaus ist Kommunikation:

> *allgemein ein Austausch von (objektiven oder subjektiven) Informationen, dabei steht zwischen Sender und Empfänger ein Übertragungsmediums (Kanal).*[**?**]

Mobile Agenten sind Softwareeinheiten die Aufgaben in Auftrag ihres Benutzers ausführen. Dabei müssen sie wissen, was sie tun, wohin sie gehen und welche Mittel sie dazu benutzen sollen. Sie müssen in der Lage sein mit Systemen, Diensten und Anwendungen interagieren zu können. Weiterhin müssen sie in der Lage sein ihr Wissen mit anderen Agenten und ihrem Besitzer teilen zu können. Sie sollten ihre Aufgaben möglichst eigenständig lösen können. Falls das beispielsweise bei komplexeren Problemstellungen nicht möglich wäre, sollten sie die Möglichkeit haben, andere Agenten zu kontaktieren, um das Problem gemeinsam zu lösen. Somit ist die Bedeutung der Kommunikation bei Agenten enorm.

Die wichtigste Forderung an die Kommunikation in einem mobilen Agentensystem ist nach *Westhoff*[**?**] an erster Stelle die Möglichkeit des Informationsaustausches. Als nächstes und nicht weniger wichtig, wird die Gewährleistung eines Höchstmaßes an Autonomie der einzelnen Agenten gefordert. Diese Forderung ist erfüllt, falls der Kommunikationsmechanismus die Eigenschaften der Ortstransparenz, Zeittransparenz, Unabhängigkeit gegenüber dem Kommunikationsprotokoll und Kompatibilität zwischen verschiedenen Agentensystemen erfüllt.

Der Begriff der Kommunikation wird öfters in Verbindung mit dem Begriff der Kooperation gebracht. Nach *Franklin*[**?**] sind Agenten mit der Eigenschaft der Kommunikation, der Gruppe der kooperierenden Agenten untergeordnet. Die andere Gruppe der kooperierenden Agenten machen sog. nicht-kommunikative Agenten aus. Nicht-Kommunikative Agenten sollten nach *Franklin* Agenten sein, die keine Kommunikationsfähigkeiten erweisen, sondern aufgrund von Beobachtungen und Reaktionen auf das Verhalten der anderen reagieren. Da das Kommunikationskonzept von SeMoA Agenten beinhaltet, die kommunikationsfähig sind, werden die

Begriffe kommunizierender und kooperativer Agent in dieser Arbeit gleich gestellt.

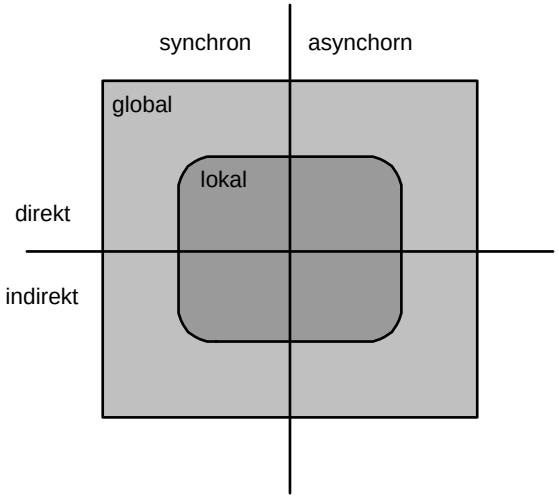

Abbildung 3.1: Kommunikation von mobilen Agenten

Eine einfache Kategorisierung des Begriffes Kommunikation bei Agentensystemen zu geben, ist nicht einfach. Im Allgemeinem unterscheidet man zwischen lokaler und globaler Kommunikation. Bei lokaler Kommunikation findet die Kommunikation auf demselben Agentensystem statt, bei der globaler hingegen, wird zwischen verschiedener Agentensystemen kommuniziert. In Abhängigkeit davon ob die Kommunikation direkt zwischen Kommunikationspartner oder über Vermittler verläuft, unterscheidet man weiterhin zwischen direkte und indirekte Kommunikation. Wenn die Kommunikationspartner beim Versenden einer Nachrichten aufeinander abgestimmt sind, spricht man von synchroner Kommunikation,andernfalls von asynchroner Kommunikation. Die Kommuniktionsteilnehmer können beim Austausch von Nachrichten den gemeinsamen Raum benutzen, auch Informationsraum (information space) gennant, oder den eigenen Nachrichtenpuffer bzw. Mailbox. Hier spricht man dann von Informationsraum- oder Mailboxbasierter Kommunikation.

3.1.1 Direkte und indirekte Kommunikation

Direkte Kommunikation findet auf direktem Weg zwischen Kommunikationspartner statt. Die Agenten, die direkt miteinander kommunizieren wollen, müssen über den Namen, die Fähigkeiten und den Aufenthaltsort ihres Kommunikationspartners informiert sein. Außerdem müssen sie darüber informiert sein, was für ein Kommunikationsprotokoll der Partner benutzt. Wenn keine Dienste vorhanden sind, die diese Art von Informationen bereitstellen, eignet sich diese Art von Kommunikation dadurch nur für eine Gruppe von Agenten dessen Eigenschaften untereinander bekannt sind. Die Agenten sollten in der Lage sein, lokal aber auch global zwischen verschiedenen Rechnern im Netz, zu kommunizieren. Falls dabei unerheblich ist, an welchem Ort sich die Kommunikationspartner befinden, dann nennt man diesen Kommunikationsmechanismus ortstransparent. Bei der direkten Kommunikation erreicht man die

globale Ortstransparenz, indem man Lokalisierungsmechanismen dazwischen schaltet.

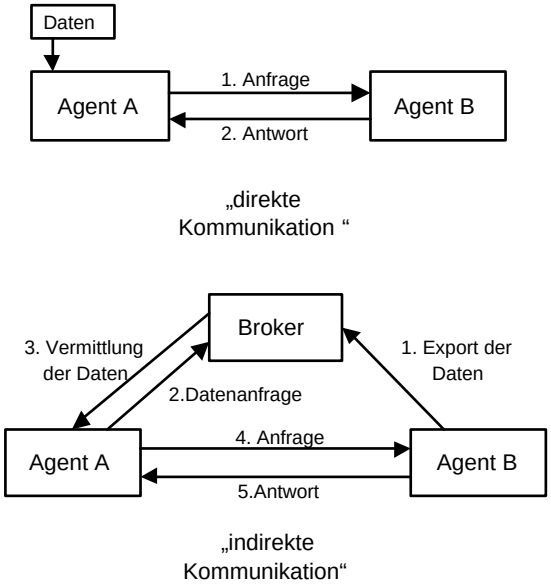

Abbildung 3.2: Direkte und indirekte Kommunikation

Die indirekte Kommunikation verwendet hingegen immer eine zwischengeschaltete Instanz zur Vermittlung der Kommunikation. Man kann dieses Konzept mit dem Konzept des Traders bzw. Brokers in der Objekttechnologie vergleichen. Der Broker ist ein Objekt bzw. ein Dienst mit dessen Hilfe Dienste vermittelt wird. Er tritt als Dienstvermittler zwischen Exporteur und Importeur auf. In einem Agentensystem stellt der Exporteur einen (Server-)Agenten dar, der einen Dienst anbietet, beim Importeur handelt es sich um einen (Client-)Agenten, der einen Dienst nutzt. Der Broker akzeptiert Dienstangebote von Exporteuren, in der Form von Informationen über Agenten, Dienste und Anwendungen und speichert sie lokal in einer Datenbank. Wenn ein Agent mit einem anderen kommunizieren will, stellt er eine Anfrage an den Broker. Dieser durchsucht seine Datenbank und gibt z.B. die Informationen über den Agent, seine Position und Kommunikationsprache an den Anfragenden zurück. Nachdem der Agent die benötigten Informationen über seinen Kommunikationspartner erhalten hat, kann er die Kommunikation mit dem anderen Agenten beginnen.

Dieses Konzept kann auch als eine Erweiterung des klassischen Client/Server-Paradigmas verstanden werden. Es wurde konzipiert, um eine bessere Vermittlung der Dienste in der Welt der verteilten Objekttechnologie zu sichern, und wie beispielsweise bei MASIF erfolgreich in die Theorie der Agentensysteme integriert. Der Vorteil eines solchen Konzeptes ist, daß der Agent bei der Kommunikation mit anderen Agenten außer der Kenntnis des Namens des Kommunikationpartners keine weitere Informationen benötigt. Der Nachteil ist, daß die Kommunikation erst nach dem Erfragen des Brokers stattfindet, womit sich die Komplexität und Reaktionszeiten bei der Kommunikation vergrößern.

3.1.2 Synchrone und Asynchrone Kommunikation

Wenn die Kommunikationspartner bei der Kommunikation aufeinander abgestimmt sind, bzw. wenn der Empfänger die Nachricht ohne nennenswerte zeitliche Verzögerung direkt nach ihrem Versenden erhält, spricht man von einer synchronen Kommunikation. Das Telefon ist ein Beispiel solcher Kommunikation. Die Informationen werden hier direkt ausgetauscht. Anderseits bei einem Briefkasten, bekommt der Empfänger seine Nachrichten erst, nachdem er sie aus dem für sie vorgesehenen Ort (Briefkasten) geholt hat. Dies geschieht mit etwas zeitlicher Verzögerung, deshalb spricht man hier von einer asynchroner Kommunikation. In dem Fall, daß es unerheblich ist, in welchen Zeitabständen die Kommunikationspartner aufeinander reagieren, erfüllt der Kommunikationsmechanismus die Eigenschaft der Zeittransparenz.

Synchrone Kommunikation bedarf einer permanenten Verbindung z.B. Socketverbindung zwischen den Plattformen. Der Agent der kommunizieren will benutzt beispielsweise den Lokalisierungsdienst um seinen Kommunikationspartner ausfindig zu machen, iniziert dann die Kommunikation und hält die Verbindung zu seinem Partner. Die Agenten dürfen dabei nicht migrieren, solange die Nachrichten nicht übermittelt sind. Dies entspricht dem Konzept einer Session aus der Welt der Kommunikationen. Diese Kommunikationsart ermöglicht deshalb die freie Kommunikation zwischen Agenten, mit der Einschränkung, daß die Agenten die Position ihres Kommunikationspartner in Erfahrung bringen müssen und ihre Aufgabe für die Dauer der Kommunikation blockiert ist.

Manchmal ist es nicht möglich, eine Verbindung zwischen mobilen Agenten herzustellen, da der Agent beispielsweise nicht kommunizieren will/kann, bzw. sich im Migrationsprozeß, oder auf einer "offline" Plattform befindet. In diesem Fall ist eine Verzögerung des Nachrichtempfangs erwünscht bzw. notwendig. Hier bietet sich deshalb das Konzept der asynchronen Kommunikation an. Die Nachrichten werden demnach über spezielle Kommunikationsmechanismen an eine sog. Mailbox (Briefkasten) des Empfängers verschickt. Die Mailbox ist ein Nachrichtenpuffer, der benutzt wird um eingehende Nachrichten zu speichern. Der Agent kann über den Erhalt der Nachricht benachrichtigt werden, dies ist aber nicht zwingend erforderlich. Die Nachrichten werden aus der Mailbox entweder direkt an den Agenten ausgeliefert (Push-Prinzip) oder erst nach der Aufforderung des Agenten (Pull-Prinzip). Hierbei werden verschiedene Ansätzen unterschieden: die Mailbox des Agenten kann sich auf einem bestimmten Server befinden (wie das bei der E-Mail Kommunikation der Fall ist) oder der Agent nimmt seine Mailbox mit. Es ist auch eine Kombination der beiden Ansätze denkbar, bei der die Mailbox auf einem Server bleibt und später unter bestimmten Bedingungen migriert (vergl. dazu [**?**]).

Ein anderes Beispiel für asynchrone Kommunikation und eine Alternative zur Mailbox-basierten Kommunikation stellt die Kommunikation über den Informationsraum (tuple space)[**?**] dar. Bei dem Informationsraum-Konzept findet die Kommunikation in einem, in abstraktem Sinne gemeinsamen Kommunikationsraum statt, der von jedem Agenten zugänglich ist, unabhängig davon, wo er sich momentan befindet. In physikalischen Sinne stellt der Informationsraum einen geteilten Speicherraum dar, auf den Agenten über den Agentenserver zugreifen können. Die Agenten können Nachrichten in Informationsraum hineinstellen, sie lesen und

aus dem Informationsraum herausnehmen. Der Informationsraum ist nicht nur auf bestimmte Teile des Agentensystems beschränkt, sondern kann z.B. durch Replikationsmechanismen[1] ins ganze Agentensystem abgebildet werden.

Der Vorteil der asynchronen Kommunikationsart liegt auf der Hand. Agenten müssen nicht immer für die Kommunikation zur Verfügung stehen und es muß keine permanente Verbindung zum Nachrichtenaustauch aufgebaut werden, somit werden der Sender und Empfänger währenddessen auch nicht blockiert. Weiterhin entscheidet der empfangende Agent selbstständig, wann er auf die erhaltene Nachricht reagiert. Der Nachteil ist, daß der Nachrichtenaustauch zeitverzögert stattfindet und der sendende Agent keine Sicherheit über dem Erhalt der Nachricht seitens seines Kommunikationpartners hat.

3.2 Agentenkommunikationssprachen

Kommunikation ist einer der Bereiche, mit denen sich die Forschungsgeimeinschaft um Agenten schon seit Anfängen der Agenten beschäftigt. Diese Bemühungen führten besonders im Bereich der Theorie zu bedeutenden Ergebnissen. So entstanden einige Standards für Agentenkommunikationssprachen wie KQML[**?**] und schon erwähntes FIPA-ACL[**?**]. Diese Standards ebneten den Weg in der Entwicklung der Kommunikationsmechanismen und dienten als Vorlage, auf denen verschiedene Agentensyteme ihre Kommunikation aufbauten. Das primäre Ziel bei der Definition dieser Standards war es, ein universelles Kommunikationsmittel zu schaffen. Kommunikationsmechanismen, die auf diesen Konzepten aufbauen, sind unabhängig gegenüber Kommunikationsprotokollen. Das Erfüllen dieser Eigenschaft ist ein Schritt weiter in Richtung Interoperabilität und Unterstützung der sog. offener Agentensysteme bzw. Agentensysteme, die in der Lage sind, mit verschiedenen Agententypen zu arbeiten.

Interoperabilität ist nach *Labrou, Finin* und *Peng*[**?**] von zentraler Wichtigkeit bei Agentensystemen. Agenten bewegen sich in einer heterogener Umgebung. Viele verschiedene Implementierungen von Agentenplattformen und besonders der Kommunikationsmechanismen erschweren die Möglichkeit der Zusammenarbeit. Agenten sind aber dafür gedacht, ihr Wissen mit anderen zu teilen, von anderen zu lernen, gemeinsam Aufgaben zu erledigen, kurz gesagt zu interagieren. Die Interaktion zwischen Agenten stellt dabei nicht nur den einfachen Austausch der Informationen dar, sondern ist mehr ein komplexer Prozeß zur Bewertung von Informationen nach Bedeutung und Wichtigkeit. Diese Art der Interoperabilität unterscheidet sich insofern von der, die MASIF in seinem Vorschlag anbietet als, daß MASIF eine Interoperabilität zwischen Systemen durch uniforme Umgebung und gleiche Komponenten in Agentensystemen erreichen will. Die Interoperabilität in der Kommunikation verlangt aber die gleiche Verständigungsbasis zwischen verschiedenen Agententypen.

Diese Verständigungsbasis sollte durch eine spezielle Kommunikationssprache für Agenten also Agent Communication Language (ACL) gegeben werden. ACL ermöglicht den Agen-

[1] Replikation stellt einen Prozeß dar, der für Erzeugen und Verwalten der Kopien der Datenbanken zuständig ist. Replikation kopiert nicht nur die Datenbank dabei, sondern synchronisiert die verschiedenen Kopien, so daß sich die Änderungen in einer Kopien auf die Anderen auswirken. Quelle: www.webopedia.com

ten den Austausch von Informationen und Wissen. Der Austausch des Wissens diente als Grundlage mit der Anfang der 90er alles anfing. Damals startete *DARPA (Defence Advance Research Projects Agency)* ein Projekt mit dem Namen Knowledge Sharing Effort (KSE). Das Ziel des Projektes war es, Techniken, Methoden, und Softwarewerkzeuge zu entwickeln, um Wissenstausch und Widerverwendung von Wissen zu ermöglichen. KSE kam zum Beschluss, daß der Austausch von Wissen Kommunikation voraussetzt, und das wiederum eine gemeinsame Sprache. Deshalb konzentrierten sich die Bemühungen von KSE auf die Definition einer universellen Kommunikationssprache.

Nach KSE sind Softwaresysteme (virtuelle) Wissensbasen, die ihre Bedürfnisse durch eine Sprache mit einer einfacher Begriffsmenge, ausdrücken können. KSE schlägt Schichten mit verschiedenen Aufgaben vor. Die erste sollte sich um die syntaktische Übersetzung zwischen Sprachen, die derselben Sprachgruppe angehören, kümmern. Eine andere Schicht sollte dafür sorgen, daß der semantische Inhalt der Tokens bzw. ihre Bedeutung überall gleich ist. Jedes Kommunikationssubjekt hat eine bestimmte Sicht seiner Umgebung, bzw. der Wissensbasis. Dieses, durch diese Sicht präsentes Hintergrundwissen, ist auch als Ontologie (ontology) bekannt. Ontologie bezeichnet eine bestimmte Konzeptualisierung einer Menge von Objekten, Konzepten und anderen Entitäten, die Wissen präsentieren und die Beziehungen, die zwischen ihnen gelten. Die Ontologie besteht aus Begriffen, deren Definitionen und Axiome die Beziehungen zwischen den Begriffen definieren. Die Begriffe sind in einer sog. Taxonomie (taxonomy) organisiert. Die letzte Schicht definiert die Kommunikation zwischen den Kommunikationssubjekten. Hierbei geht es nicht um die "low-level"-Kommunikation und Bitübertragung, sondern mehr um die Kommunikation auf einer komplexeren Ebene, bei der der Gegenstand des Austausches die Standpunkte über Informationen und Wissensinhalte der Kommunikationssubjekte sind. Obwohl beim KSE ursprünglich nicht von den Agenten die Rede war, sind die Konzepte die daraus hervorgingen universell und somit auch gut anwendbar.

KSE entwickelte zur Kommunikation Knowledge Query and Manipulation Language (KQML)[**?**], eine auf einfachen Sprachelementen basierende Sprache, aber auch eine Menge anderer Konzepte, Werkzeuge und Dienste zum Wissensaustausch. So entstanden auch eine logische Sprache, die speziell zur Beschreibung der Inhalte in einer computerbasierten Umgebung angewendet wird, das sog. Knowledge Information Format (KIF)[**?**] und Ontolingua, eine Sprache zur Beschreibung der Ontologien. Ontolingua war vorgesehen, um die Vorteile der Nutzung des World Wide Web (WWW) auszunutzen, um die Ontologien besser über das Web erzeugen, nachzufragen und ergänzen zu können.

3.2.1 KQML

Wie der Name Knowledge Query and Manipulation Language bereits aussagt, ist KQML eine Sprache, die dem Zweck dient, Wissen abzufragen und zu verändern. Grundlage für KQML ist die sog. Sprechakttheorie (Speech Act Theory), die noch in den 60er Jahren von Linguisten entwickelt wurde. Ein Sprechakt ist eine Nachricht, die mit einem ganz bestimmten Ziel geäußert wird.

Es sind zwei Aspekte der Sprechakte für KQML wichtig. Zum einen unterschiedet man verschiedene Typen von Sprechakten, und anderseits besteht jede Aussage aus zwei Teilen. Beispiele für Sprechakttypen sind:

- *Feststellungen* sind Aussagen, die einfach als Benachrichtigung oder als Information gedacht sind.
- *Weisungen* sind Versuche, den Hörer zu einer Aktion zu bewegen.
- *Verpflichtungen* stellen Verpflichtungen des Sprechers gegenüber seines Hörers dar, etwas zu tun.
- Mit *Erklärungen* werden die Beziehungen zwischen Sprachobjekten oder ihre Zustände erklärt.

Es gibt noch weitere Sprechakttypen, dessen Einteilungen aber durchaus umstritten sind, da die Forscher verschiedene Typisierungen vorgeschlagen haben.

Außer der Definition der verschiedenen Sprechakttypen, kümmert sich die Sprechaktentheorie auch um den eigentlichen Inhalt ihrer Nachrichten. So hat beispielsweise die Verpflichtung des Agenten "die Informationen über den Preis des Produktes A zurückzugeben" einen anderen Inhalt als die Verpflichtung, die Information A von Agenten C zu holen.

Auch in KQML besteht jede Nachricht zwischen zwei Agenten aus diesen beiden Teilen: Sie hat einen Sprechakttyp - in KQML heißt er Performative - und einen Inhalt. Bei KQML gibt es auch einen zusätzlichen Teil, man spricht hier von Schichten - die Kommunikationsschicht. Die Kommunikationsschicht (communication layer) ist für Parameter für die Kommunikation auf den niedrigen Schichten zuständig. Diese Parameter beschreiben den Sender, Empfänger der Nachricht und identifizieren sie eindeutig. Die Sprechakttypen bzw. Performative sind in der Nachrichtenschicht (message layer) angesiedelt. Die Nachrichtenschicht ist für das Entschlüsseln der Nachrichten zuständig und formt das Kernstück der KQML. Sie bestimmt die Art der Interaktion, die mit einem KQML-sprechenden Agenten möglich ist. In dieser Schicht findet die Verknüpfung zwischen der Performative und dem Inhalt der Nachricht statt. Es werden durch die Parameter des Performative Angaben über die verwendete Ontologie, die Sprache zur Beschreibung des Inhaltes und sogar über das Thema innerhalb der Ontologie gemacht. Der Inhalt der Nachricht ist Teil der Inhaltsschicht (content layer) und wird durch eine Sprache zur Inhaltsbeschreibung wie erwähntes KIF oder Extensible Markup Language (XML) angegeben.

Die Syntax der KQML basiert auf der sog. S-Ausdruck (s-expression), einer Art geklammerte Liste, vergleichbar mit der Syntax von Lisp. Jede solche Liste fängt mit einem Performative an, weiterhin folgen die Parameter des Performative sog. Tokens, die in der Form Schlüsselwort/Wert angegeben werden. Wie folgt sieht z.B. eine KQML-Nachricht aus, die eine Anfrage vom Sender *joe* an einen Börsenserver darstellt, in der *joe* den aktuellen Preis der *IBM-Aktie* erfragen möchte:

```
(ask-one
:sender joe
:content (PRICE IBM ?price)
:receiver stock-server
:replay-with ibm-stock
:language LPROLOG
:ontology NYSE-TICKS
)
```

In dieser Nachricht ist das KQML performative `ask-one`, was eine Anfrage darstellt. Der Inhalt ist `(PRICE IBM ?price)`, ein in LPROLOG geschriebener Ausdruck, der für eine Preisanfrage steht. Der Inhalt wird durch den Token `:language LPROLOG` und `:ontology NYSE-TICKS` beschrieben. Diese Tokens gehören zu der Nachrichtenschicht. Die Tokens `:sender`, `:receiver` und `:replay-with` sind ein Teil der Kommunikationsschicht. Die Inhaltsschicht ist mit dem Token `:content` gegeben.

KQML ist ein ISO-Standard für die Kommunikation, der auch die Performatives festlegt, die benutzt werden können. Dessen Anzahl ist relativ groß - ca. 35 verschiedene. KQML-Agenten haben aber die Möglichkeit, sich zur Kommunikation auf eine beliebig große Menge davon festzulegen und sie zu unterstützen.

Außer diesem auf den Performatives basierten Kommunikationsparadigma, sieht KQML auch eine spezielle Klasse der Agente vor, die sog. communication facilitator (Kommunikationsvermittler)[**?**]. Ein Kommunikationsvermittler ist ein Agent, der eine Reihe an nützlichen Kommunikationsdiensten erledigt. Er führt Daten über verfügbare kommunikative Agenten und Dienste, routet Nachrichten zu den Diensten und zurück. Er kann auch als Mediator und Übersetzer dienen.

3.2.2 FIPA-ACL

Obwohl KSE den größten Teil der Forschungsarbeit im Bereich der Interoperabilität zwischen Agentensystemen betrieben hat, verpasste es einen systematischen Vorgang zur weiteren Entwicklung des Kommunikationsstandards zu definieren. Diese Lücke wurde durch FIPA geschloßen. Die FIPA wurde mit dem Ziel gegründet, eine Spezifikation zu entwickeln, welche die Interoperabilität zwischen Agentensystemen weiter verbessern soll. Die Kommunikationssprache FIPA Agent Communication Language (FIPA-ACL)[**?**] ist entstanden als ein Standard, den die *Foundation for Intelligent Physical Agents* als ein Teil der *FIPA97-Spezifikation* veröffentlichte.

FIPA-ACL baut ähnlich wie KQML auf der Spechakttheorie auf. Nachricht ist nach FIPA eine Aktion, bzw. ein sog. kommunikativer Akt (communicative act, CA), der bestimmte Aktion durchführt. Der kommunikative Akt hat die selbe Funktion wie Performatives bei KQML. FIPA hat als Basis für ihre ACL nicht nur KQML, sondern auch eine Sprache mit dem Namen ARCOL herangezogen. ARCOL ist eine wohldefinierte, formale Sprache für Agentenkommunikation, die auf einer Menge von Grundbegriffen aufbaut.

FIPA-ACL kann wie auch KQML in Schichten aufgeteilt werden, allerdings besteht die FIPA-ACL aus 5 verschiedenen Schichten:

1. *Die Protokolschicht* definiert die Regel zur Strukturierung des Dialoges zwischen den Agenten,
2. *Der kommunikative Akt* definiert den Typ der Kommunikation,
3. *Die Nachrichtenschicht* enthält die Parameter über die Kommunikation, z.B. Informationen über Absender, Empfänger, und den Inhalt,
4. *Die Sprache zur Inhaltsbeschreibung* bezeichnet die für Beschreibung des Inhaltes verwendete Sprache,
5. *Die Ontologie* definiert wie bei KQML, den im Inhalt der Nachricht benutzten Wortschatz.

Die erste Schicht, Protokollschicht definiert die Struktur des Dialoges zwischen den Agenten und ist im Unterschied zu den Schichten 2 bis 5, nicht in jeder Nachricht enthalten. Die in FIPA-ACL präsente Protokollschicht unterscheidet FIPA-ACL von KQML. Das Protokoll welches für die Interaktion zuständig ist, definiert die Sequenz der Nachrichten, die den Dialog zwischen den Agenten aufbauen. Das ermöglicht den Agenten, auf die Antworte ihrer Kommunikationspartner, gemäß der vordefinierten Konversationsmustern zu reagieren, sofern diese innerhalb der Protokollschicht gegeben sind. Diese Konversationsmuster sind ein Teil der FIPA-Spezifikation. Sie sind allerdings auch erweiterbar.

Die FIPA-ACL-Spezifikation besteht aus einer Menge von Nachrichtentypen bzw. CAs und Deskriptiver als auch formaler Beschreibungen ihrer Bedeutungen. FIPA definiert zudem eine erweiterbare Basismenge an CAs die aus einer Reihe einfacher und komplexen CAs besteht. Die einfachen CA's stellen die Grundaktionen einer Konversation dar, wie `inform`, `confirm` und `request`, während die komplexen eine Kombination der ersten darstellen, wie `inform-if` oder z.B. `request-wenn`.

Nachrichten werden bei FIPA-ACL auch in Form von S-Ausdrücken aufgebaut. Folgendes Beispiel stellt eine Nachricht von `bt-agent` an `customer-agent` dar, in der `customer-agent` über die aktuelle Quote bei einer Auktion benachrichtigt wird:

```
(inform
:sender bt-agent
:receiver customer-agent
:content (Line_quote(bt_customer,123),300))
:in-replay-to round-4
:language prolog
:ontology bt-auction
:protocol fipa-contract-net
)
```

Der kommunikative Akt ist hier durch `inform` gegeben, und stellt somit die Absicht des Informierens dar. Die Bedeutung der Token `:sender`, `:receiver`, `:content`, `:in-reply-to`, `:language` und `:ontology` ist dieselbe wie bei KQML. Das einzig neue ist der Token `:protocol fipa-contact-net`. Dieser definiert das Protokoll dieser Kommunikation. Hier handelt es sich um ein Konversationsmuster, das den Prozeß des Abschlusses eines Vertrags beschreibt. Der Inhalt der Nachricht ist weiterhin in PROLOG verfasst. Zur Inhaltsbeschreibung hätte auch jede andere Beschreibungssprache verwendet werden können. Die FIPA definiert auch eine eigene Beschreibungssprache, die sog. FIPA-semantic language (FIPA-SL). Ihre Nutzung ist optional.

Vergleich zwischen KQML und FIPA-ACL

In ihrem Grundkonzept und Prinzipien mit denen sie sich beschäftigen, sind die beiden Sprachen gleich. Die Unterschiede entstehen in den Details ihres semantischen Grundgerüstes. Eine exakte Abbildung bzw. Transformation zwischen den *performatives* von KQML und *communicative acts* von FIPA ist nicht möglich. Abgesehen von diesem kleinen Unterschieden, sind die beiden Sprachen syntaktisch fast gleich. Diese Tatsache erleichtert die Aufgabe den poteziellen Programmierern eines ACL-Übersetzters.

KQML definiert weiterhin eine Fülle an verschiedenen Mechanismen bzw. Diensten für Registration und Vermittlung, wie z.B. Kommunikationsvermittler. Die Spezifikation der FIPA-ACL hingegen, beschäftigt sich nur mit den Sprachkonstrukten. Dienste sind Teil der gesamten *FIPA98-Referenz*.

Die Protokollschicht bei FIPA-ACL ermöglicht zusammen mit der Definition der Konversationsmuster einen strukturierten Ansatz zur Agenteninteraktion. Dies wurde bei KQML nicht berücksichtigt. Die Semantik bei KQML ist im Unterschied zu FIPA-ACL nicht formal definiert sondern nur sprachlich beschrieben und deshalb ist ihre Interpretation dem Entwickler überlassen.

Trotzt dieser Unterschiede schließen sich die Sprachen nicht gegenseitig aus. Die Auswahl der richtigen ACL ist dem Anwender überlassen. Sie hängt von seinen Bedürfnissen und Anforderungen ab. Nach *Finin*, *Labrou* und *Peng*[**?**] müssen die Kommunikationsmechanismen, die auf diesen Sprachen aufbauen wollen, folgende Bedingungen erfüllen:

1. einige APIs (application programming interface), die das Erstellen, Senden und Empfangen der ACL-Nachrichten unterstützen,

2. eine Infrastruktur aus Diensten, um die Namensgebung, Registrierung und andere Vermittlungsfunktionen bereitzustellen,

3. die Implementierung jeden reservierten Nachrichtentyps (performative oder kommunikativen Akt) gemäß seiner semantischen Definition.

Als Entwickler würde man sich normalerweise nur um die Umsetzung des dritten Punkten kümmern, da er von dem konkreten Anwendungskontext abhängig ist. Da die ersten zwei

Punkte sich kaum ändern, ist es zu erwarten, daß sie auch spezifizierte, wiederverwendbare Komponenten sind. Aus den ersten zwei Punkte machen die beiden Spezifikationen aber keinen Gegenstand der Definition. Solche Kommunikationsmechanismen sind normalerweise nicht in den selben Kommunikationsschichten wie die Sprachelemente selbst angesiedelt, sie gehören eher in die unteren Kommunikationsschichten. Deshalb wurden sie von den Entwicklern von KQML und FIPA-ACL auch kaum berücksichtigt.

Es bleibt auch noch zu erwähnen das KQML heute nicht mehr weiterentwickelt wird und deswegen FIPA-ACL eher die Zukunft gehört.

3.3 Existierende Kommunikationsmodelle

Die Unterstützung der Interoperabilität, Bestrebungen in Richtung von mehr Offenheit der Agentensysteme und vor allem die Definition der Kommunikationsstandards KQML und FIPA-ACL ließen die Kommunikation zu der wichtigsten Eigenschaften von Agenten werden. Als Folge solcher Verhältnisse entstanden einige Entwicklungsinfrastrukturen bzw. Softwarewerkzeuge, deren Schwerpunkt besonders auf die Kommunikation gelegt worden ist. Diese Infrastrukturen ermöglichen dem Entwickler einen einfachen Aufbau agentenbasierter Systeme, wodurch sich dieser mehr auf den eigentlichen Anwendungsbereich des Systems konzentrieren kann. Im Folgenden werden die JATLite von *Stanford University* und FIPA-OS als Entwicklung von *Nortel Network* vorgestellt.

3.3.1 JATLite

Java Agent Template Lite (JATLite)[**?**][2] ist ein Java-Archiv bzw. Java-Klassenarchiv, das eine Infrastruktur zur Erzeugung von Agenten stellt. Es ermöglicht dem Benutzer, durch eine Menge an Vorlagen ein schnelles Erzeugen von Agenten, die in der Lage sind, über das Internet zu kommunizieren. Neben dem Werkzeug zur Erzeugung der Agenten, bietet JATLite auch eine robuste Infrastruktur für die Verwaltung von Agenten und das Senden und Empfangen von Nachrichten und Daten. JATLite nutzt die Portabilität der Java-Sprache aus, um ein System für Agenten aufzubauen, das ein hohes Maß an Interoperabilität unterstützt.

JATLite ist ein Kommunikationsparadigma für die Kommunikation über das Internet, deshalb baut es auf Transfer Control Protocol/Internet Protocol (TCP/IP) auf, dem Basisprotokoll des Internets auf. Für die Kommunikation zwischen den Agenten wird KQML benutzt. Die Entwickler der JATLite lassen aber die Möglichkeit offen, auch andere ACL's wie z.B. FIPA-ACL zu verwenden.

Die Architektur von JATLite ist als eine Hierarchie aus speziellen Schichten organisiert. Die untersten Schichten sind die sog. abstract layer und base layer. Sie stellen die Basisklassen für die Unterstützung von TCP/IP und der Kommunikation über TCP/IP bereit. Darauf baut die KQML Schicht auf, welche als Basis die eigentliche Kommunikation über

[2] JATLite, URL: http://java.stanford.edu/

KQML ermöglicht. Sie enthält die von dem KQML-Standard vorgeschlagene Performatives, wie auch eine Menge administrativer Performatives, wie z.B. `connect`, `disconnect` und `registration`. Die letzten beiden Schichten sind die sog. Routerschicht und Protokollschicht. Die Routerschicht übernimmt die Aufgaben, der Namensregistrierung, des Routing und des Puffern der Nachrichten, während die Protokollschicht einige offenen Internetprotokolle wie FTP und SMTP unterstützt. Diese Protokolle werden zum Transport größerer Datenmengen bzw. zum eventuellen Versenden der KQML-Nachrichten via E-Mail verwendet.

Die Kommunikation von JATLite basiert auf der Theorie der sog. typed-message Agenten. Diese Theorie besagt, daß Agenten durch ihre Zugehörigkeit zu einer Gruppe definiert sind. Die Gruppe muß die Nachrichten austauschen können, um ihre gemeinsame Aufgabe zu lösen. Dabei müssen Agenten zur Kommunikation ein Nachrichtensemantik benutzen, die unabhängig vom Anwendungskontext ist und sich gut zur "peer-to-peer"-Kommunikation eignet. Da diese Bedingungen von KQML erfüllt werden, fiel die Auswahl auf sie.

Der Kommunikationsmechanismus ist weiterhin um einen zentralen Dienst, mit dem Namen Agent Message Router (AMR) aufgebaut. Die Aufgabe des AMR ist es, die Agenten und Nachrichten zu verwalten. Es ermöglicht den Agenten das Starten, Beenden und die Migration. Im Bezug auf die Nachrichtenverwaltung verfolgt das AMR das indirektes und asynchrones Prinzip. AMR funktioniert ähnlich wie ein Emailserver, da es empfangene Nachrichten puffert und weiterleitet[3]. Um die Nachrichten zu verschicken müssen Agenten eine Socket-Verbindung zum AMR aufbauen. Die Nachricht wird anschließend zum AMR übertragen, gespeichert und dann dem empfangenden Agenten weitergeleitet, wenn sich dieser mit dem AMR verbindet. Zu diesem Zweck müssen Agenten nur die IP-Adresse des AMR kennen, es gibt keinen Bedarf nach Lokalisierungsdiensten. Auf diese Weise ist auch kein Verlust von Nachrichten möglich, der durch temporäre Abwesenheit verursacht werden könnte.

JATLite stellt also eine einfache Infrastruktur, welche die Kommunikation zwischen Agenten ermöglicht. Durch Verwendung des zentralen Diensten zur Kommunikation bietet JATLite einen zuverlässigen und robusten Kommunikationmechanismus. Der modulare Aufbau von JATLite, die Wahl der Java zur Realisierung und der Bedarf keinerlei spezieller Software ermöglichen eine schnelle Anpassung, Erweiterung und die Verwendung in verschiedenen Anwendungen in dem Bereich der neuen Technologien.

3.3.2 FIPA-OS

FIPA Open Sorce (FIPA-OS)[**?**][4] stellt ähnlich wie JATLite eine Infrastruktur dar, die entwickelt worden ist, um die Konstruktion von heterogenen Agentenplattformen, Agenten und Diensten zu erleichtern. Diese Infrastruktur unterstützt, durch die Nutzung der Agentenkommunikationsprache FIPA-ACL, die Kommunikation zwischen mehreren Agenten. Die Plattform unterstützt weiterhin durch ihren modularen und erweiterbaren Aufbau die Verwendung bekannter Konzepte.

[3] sog. "store and forward"-Verfahren

[4] FIPA-OS, URL: http://www.nortelnetworks.com/fipa-os

FIPA-OS wurde in Java realisiert und seine Architektur baut vollständing auf dem FIPA-Refenzmodell auf. Es implementiert aus diesem Grund alle von FIPA vorgeschriebenen Komponenten und erweitert sie durch einige andere. FIPA-OS kann nicht als striktes mehrschichtiges Modell angesehen werden, sondern eher als ein modulares System mit leicht austauschbaren und erweiterbaren Komponenten.

Für die Erzeugung der Agenten in FIPA-OS kann man sich an verschiedenen sog. Agentenhüllen (agent shells) orientieren, die eine vordefinierte Vorlage mit allen benötigten Bindungen zu den Diensten der Platform darstellen. Einige Hüllen unterstützen beispielsweise Basisdienste für Nachrichtentransport und -verwaltung, andere implementieren zusätzlich die Unterstützung für FIPA-Nachrichten. Außer diesen Agentenhüllen erlaubt FIPA-OS aber auch die vollständige eigene Implementierung von Agenten.

Die "low lewel"-Kommunikation bei FIPA-OS baut ähnlich wie JATLite auf bekannten und zuverlässigen Kommunikationsprotokollen auf. FIPA-OS verwendet zum Zweck der Kommunikation zwischen Agenten aber auch für plattforminternen Nachrichtenaustausch das Kommunikationsprotokoll Internet Inter-ORB Protocol (IIOP)[5], das von CORBA stammt. CORBA-Elemente werden auch verwendet um andere Komponenten der Platform zu implementieren. So werden die AMS- und DF-Komponenten durch Benutzung des *CORBA-Namensdienstes* und die Unterstützung der Migration durch Verwendung des *CORBA Interoperability Object References (CORBA IOR)* realisiert.

Der Nachrichtenaustausch selbst basiert auf FIPA-ACL. Für die Konversation ist eine Kombination verschiedener von FIPA-ACL bekannter Kommunikationsmechanismen zuständig. Neben dem Nachrichtenmangement von FIPA-ACL hat FIPA-OS in dem Bereich noch eine Besonderheiten eingebaut. FIPA-ACL-Standard definiert die Tokens `replay-with` und `in-replay-to`. Es schreibt aber die Semantik der Benutzung dieser beiden Tokens nicht vor. Es ist nicht klar, was die zu Benutzen sind und in welchem Verhältnis sie zueinander stehen. Aus diesem Grunde definiert FIPA-OS ein eigenes Feld, bzw. Token, das sog. Conversation-ID. Conversation-ID stellt eine Nummer dar, mit der man den Sender und Empfänger zu einer bestimmten Konversation eindeutig zuordnen kann. Die Nachrichten werden durch einen ACL-Parser, der XML und FIPS-SL versteht, übersetzt.

FIPA-OS ist als ein Werkzeug zu verstehen, das den Entwickler zum Aufbau der für bestimmte Anwendungen vorgesehenen Agentensysteme dient. Mit seinem Konzept, das mit einem Baukastensystem verglichen werden kann, dessen Komponente leicht zusammengestellt und verändert werden können, stellt es eine praktische Hilfe für Entwickler dar, die ihre Kommunikation auf FIPA-Standard aufbauen wollen.

[5]Internet Inter-Orb Protocol ist ein von der Object Management Group entwickeltes Protokoll zur Implementierung der CORBA-Lösung über WWW. IIOP ermöglicht den Browsern und Servern den Austausch der Integers, Arrays und anderer komplexen Objekten. Quelle: Webopedia - http://www.webopedia.com

Kapitel 4

Sicherheit

Die immer größer werdende Bedeutung der Informations- und Kommunikationstechnologie, als Resultat des wachsenden Grades ihrer Integration in das alltägliche Leben, hat eine tiefere Auseinandersetzung mit den Vor- und Nachteilen dieser Technologien zur Folge. Die Nutzung solcher Technologien bringt gewisse Chancen aber auch Risiken mit sich. So wird durch die Benutzung dieser Technologien, die Arbeit erleichtert und verbessert - der Mensch wird flexibler, wissender und kompetenter. Sie birgt aber auch gewisse Risiken, wie zum Beispiel: Fehleranfälligkeit, Informationsüberflutung, erhöhte Verletztbarkeit gegen kriminelle Angriffe und Datenverlust. Um diesen Risiken vorzubeugen, werden verschiedene Maßnahmen ergriffen, die unter dem Begriff Sicherheit vereint sind. Sicherheit kann man dabei wi folgt definieren:

> *Sicherheit ist die Eigenschaft eines Systems, die dadurch gekennzeichnet ist, daß die als bedeutsam angesehenen Bedrohungen, die sich gegen die schützenswerten Güter richten, durch besondere Maßnahmen so weit ausgeschlossen sind, daß das verbleibende Risiko akzeptiert wird.*[**?**]

Die Sicherheit in der Informationstechnologie (IT-Sicherheit) beschäftigt sich vor allem mit zwei Bedrohungen:

- *unabsichtliche Fehler*, wie beispielsweise Übertragungsfehler bei der Kommunikation
- *Angriffe* stellen die absichtliche, unauthorisierte Manipulationen dar, wie zum Beispiel das Manipulieren der Nachrichten im Informationskanal zum Zwecke der Fehlinformation

Die IT-Sicherheit im engeren Sinne (security), richtet sich nur gegen die Angriffe auf Informationstechnologien. Sie geht davon aus, daß die Sicherheitsprobleme auf Interessenskonflikten zwischen Parteien beruhen. Das Ziel eines Angriffs ist der persönliche Vorteil des "Angreifers" auf Kosten des "Angegriffenen". Aus diesem Grund definiert Sicherheit eine Reihe von Maßnahmen, die bestimmte Anforderungen an die Informationstechnologie setzt, um das Risiko, das durch Angriffe entsteht, möglichst klein zu halten. Geschützt werden dabei verschiedene Werte und Güter von Daten bis zu ganzen IT-Systemen. Dieser Prozeß ist in Abbildung **??** ersichtlich.

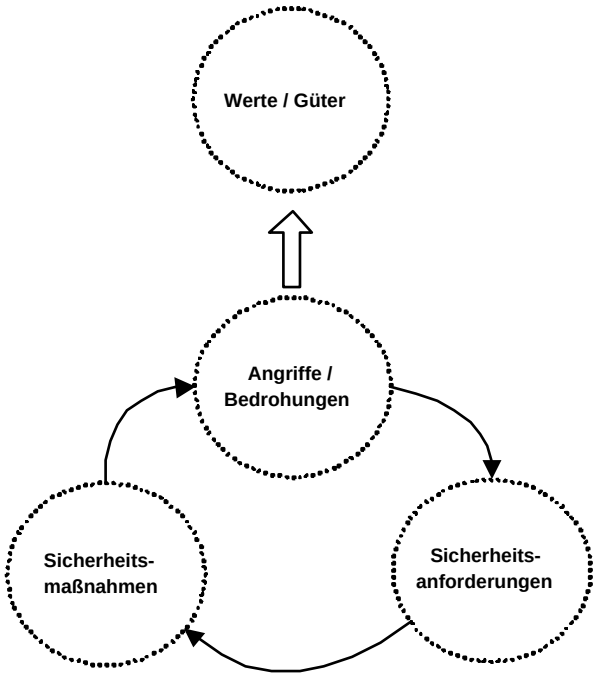

Abbildung 4.1: Grundsicht der IT-Sicherheit

4.1 Sicherheitsbedrohungen

Rechnersysteme und Daten sind ständig der potentiellen Gefahr ausgesetzt, gestohlen, ausgespäht, verfälscht, zerstört oder in ihrer Funktionalität eingeschränkt zu werden. Um die konkrete Gefahr für ein System zu bestimmen, muß man die Sicherheitsbedrohungen feststellen. Man unterschiedet dabei drei Grundbedrohungen:

- *unbefugter Informationsgewinn* (Verlust der Vertraulichkeit),
- *unbefugte Modifikation* von Informationen (Verlust der Integrität),
- *unbefugte Beeinträchtigung* der Funktionalität (Verlust der Verfügbarkeit).

Diese sind einerseits von der Umwelt abhängig, in der das System betrieben wird, andererseits von der Sensitivität, der im System vorhandenen Informationen. Auf diese Weise werden die Bedrohungen für unterschiedliche Systeme verschieden gewichtet.

Wie in dem vorigen Abschnitt bereits erwähnt, beschäftigt sich die IT-Sicherheit im engeren Sinn mit Bedrohungen in Form von Angriffen. Diese Angriffe stellen eine Bedrohung auf Systeme von "außen", der Angreifer befindet sich außerhalb des Systems, und von "innen", der Angreifer befindet sich direkt im System, dar. Die Angriffe können darüber hinaus generell in zwei weitere Gruppen aufgeteilt werden - *aktive* und *passive Angriffe*.

4.1.1 Passive Angriffe

Unter dem Begriff passive Angriffe versteht man, daß ein Angreifer versucht, unrechtmäßig in Besitz von Daten zu gelangen, ohne diese zu verändern. Hier sind einige passive Angriffsmöglichkeiten aufgelistet:

Abhören (Eavesdropping und Wiretapping) - der Angreifer hört den Nachrichtenaustausch von Nutzern ab, um auf die entsprechenden Nutzinformationen zu schließen.

unbefugter Zugang (Unauthorized Access) - der Angreifer verschafft sich unrechtmäßig Zugang, um gespeicherte Daten und Speichermedien auszulesen.

Abstreiten (Repudiation) - die Teilnahme von einem der Kommunikationspartner an einer Transaktion oder Kommunikation wird abgestritten. Dieses kann absichtlich oder unabsichtlich passieren. Eine Folge davon kann die Entstehung von ernsthaften Inkompatibilität zwischen den Kommunikationspartner sein, die ohne Gegenmaßnahmen nicht zu lösen sind.

Verkehrsflußanalyse (Traffic Analysis) - durch die Beobachtung anderer Nutzer bzw. ihres Netzverkehrs und der Kommunikation erlangt der Angreifer Kenntnisse über deren Aktivitäten.

4.1.2 Aktive Angriffe

Aktive Angriffe sind eine Bedrohung, bei denen der Angreifer zusätzlich versucht die Daten zu verändern, zu löschen oder die Verfügbarkeit von Ressourcen zu stören. Hierzu zählen:

Maskierung (Masquerade) - bei dieser Angriffsart versucht der Angreifer dem System, das er angreift, oder seinem Kommunikationspartner eine andere Identität vorzutäuschen. Es geht ihm darum, an die Rechte des Getäuschten zu kommen, um bestimmte Aufgaben durchführen zu können. Das klassische Beispiel ist hier das sog. IP-Spoofing, bei dem der Angreifer seine IP-Adresse verändert, um den Anschein zu erwecken die Daten kämmen von einem anderen Rechner. Ein weiteres Beispiel sind die Trojanischen Pferde. Hierbei handelt es sich um Programme, die neben ihrer eigentlichen Aufgaben auch noch versteckte schädliche Funktionen beinhalten, die zum Beispiel die Paßwörter des Benutzers ausspähen.

"Man-in-the-Middle"-Angriff - kann als eine spezielle Art der Maskierung angesehen werden. Bei dieser Art befindet sich der Angreifer zwischen zwei Kommunikationspartnern, und täuscht jedem der Kommunikationsteilnehmern jeweils die Identität des anderen vor. Auf diese Weise kommunizieren die eigentlichen Kommunikationspartner direkt über den Angreifer und er gelangt gleichzeitig an die Daten von beiden gleichzeitig, ohne daß einer von ihnen was merkt.

"Denial-of-Service"-Angriff (DoS) - ist eine Art des Angriffs bei der, der Angreifer versucht die Verfügbarkeit von Ressourcen bzw. Daten oder Diensten einzuschränken. Die Ressourcen werden dabei in einer Weise überbeansprucht, die dessen Funktionalität sehr

einschränkt oder total ausfallen läßt. So beispielsweise beim Überfluten (Flooding Attack), bei dem ein Zielrechner mit einer Flut von Anfragen bombardiert wird, bis dieser nicht mehr in der Lage ist, Anfragen zu beantworten.

Wiedereinspielung (Replay Attack) - hier werden schon verwendeten oder abgehörten Daten bzw. Nachrichten (evtl. verändert) zu einem späterem Zeitpunkt wiederbenutzt oder eingespielt um dem rechtmäßigen Empfänger eine neue Nachricht bzw. neue Verbindung vorzutäuschen. So kann der Angreifer sich bei bestimmten Diensten mit beschränktem Zugang mittels ausgespähten und wiedereingespielten Daten als berechtigter Nutzer ausgeben.

Täuschung der berechtigten Nutzern (Social Engineering) - dient dem Angreifer dazu, durch die Täuschung von berechtigten Nutzern in den Besitz von Zugangsdaten zu kommen. So kann der Angreifer den Nutzer zum Beispiel anrufen und sich als Systemadministrator ausgeben, um an die Paßwörter des Angegriffenen zu kommen.

Außer diesen Formen der aktiven Angriffe, gibt es auch noch weitere Formen, die sich auf bestimmte Teile von Sicherheitsmechanismen spezialisieren, wie zum Beispiel auf spezielle Kryptoalgorithmen und Kommunikationsprotokolle.

4.1.3 Angriffe in mobilen Agentensystemen

In einer stark vernetzten Welt, in der die Agentensysteme immer öfter eine praktische Anwendung finden - von Systemen für das elektronische Handeln bis zu solchen für Informationssuche - spielt Sicherheit eine immer wichtigere Rolle. Es wird immer mehr mit sensitiven Daten wie Kreditkartennummern, Authentifizierungsdaten und weiteren geheimen Informationen gearbeitet, dabei wird eine tadellose Abwicklung und hohe Verfügbarkeit solcher Systemen erwartet. Auch Mobilen-Agenten-Systeme und Agenten sind aus diesem Grund Sicherheitsbedrohungen ausgesetzt. Ihre Daten können ausgespäht, manipuliert, die Agenten können angegriffen, ihr Code verändert und die Agentenplattformen können in ihre Funktionalität eingeschränkt werden.

Generell drohen den Agenten nach *Jansen*[**?**] vier Arten der Sicherheitsdrohungen:

- *Offenlegung der Daten* (Disclosure of Data) - beispielsweise durch Ausspähung, Abfangen und unberechtigte Zugriffe
- *Denial-of-Service*
- *Verfälschung der Daten* (Corruption of Information) - beispielsweise durch Manipulation von Code und Nutzdaten der Agenten
- *Störung und Beeinträchtigung* (Interference or Nuisance) - z.B. durch Verweigerung, Beeinträchtigung der Ausführung, Manipulation der Interaktion mit anderen Agenten usw.

Nicht nur Agenten sondern auch die Agentenplattformen als Wirte der Agenten sind denselben Risiken ausgesetzt. Grob können die Angriffe, in Abhängigkeit davon, gegen wen sie gerichtet sind, in "Angriffe gegen Agenten" und "Angriffe gegen Agentenplattformen" eingeteilt werden.

Angriffe gegen Agenten

Bei dieser Art von Angriffen handelt es sich um Angriffe, die direkt auf Agenten ausgerichtet sind. Agenten enthalten Daten, sie kommunizieren miteinander, bei manchen Systemen führen sie bestimmte Aufgaben (Verzeichnisdienst, interne Kommunikation) auf den Agentenplattformen aus. Das sind alles mögliche Angriffspunkte, die böswillige Angreifer in Form von Agenten und Agentenplattformen, ausnutzen können. Nach *Hohl*[**?**] muß man bei Angriffen auf Agenten zwischen denen unterscheiden, die durch lokale Angreifer verursacht werden und solchen, die durch entfernte Angreifer zustande kommen.

Unter lokalen Angriffen versteht *Hohl* Angriffe, die von Agenten initiiert werden, die sich auf der selben Plattform wie der Opfer-Agent befinden, oder die Plattform selbst. Er unterscheidet weiterhin zwischen technischen Angriffen und anwendungsspezifischen Angriffen.

Bei den technischen Angriffen nutzen die Angreifer technische Mängel wie beispielsweise Sicherheitslöcher in der Implementierung der Plattform aus, um auf die Ressourcen der Agenten zuzugreifen. So kann der böswillige Agent oder die Plattform über ungeschützte Speicherbereiche unberechtigt an die Daten der Agenten kommen und sie dann kopieren oder manipulieren. Diese Problematik läßt sich durch einige bestehende Sicherheitskonzepte lösen. Die Programmiersprache *Java* enthält beispielsweise Elemente die den Schutz des mobilen Codes ermöglichen. Einige interessante Ansätze für den Schutz des Agenten gegenüber der böswilligen Plattform findet man beispielsweise in den Werken von *Sander* und *Tschudin*[**?**] und im Werk von *Hohl*[**?**].

Unter dem Begriff anwendungsspezifische Angriffe sind die Angriffe gemeint, die im Kontext einer für bestimmte Zwecke auf Agenten basierenden Anwendung möglich sind. So sind beispielsweise verschiedene Angriffe auf eine zum elektronischen Handeln vorgesehene Agentenplattform möglich. Es können beispielsweise Schwachstellen im Bezahlprotokoll genutzt werden um Zahlungen vorzutäuschen oder an die Kreditkartennummer zu kommen. Da diese Art der Angriffe nicht spezifisch für Agenten sondern eher allgemeiner Art sind, können hier Lösungen angewendet werden, die in allgemeinen Fällen ohne mobilen Agenten bekannt sind.

Entfernte Angriffe sind solche die von Angreifern ausgeführt werden, die sich außerhalb der Plattform befinden. Hierzu zählen Angriffe gegen die Migration und Kommunikation der Agenten und Denial-of-Service-Angriffe. Im Grunde genommen gehören Angriffe gegen die Migration und gegen die Kommunikation der Agenten zur selben Art der Angriffe, da die Migration, ähnlich wie die Kommunikation, aus Verpacken von Daten und der Übertragung derselben zwischen Agentenplattformen besteht. Deshalb können hier bestehende Mechanismen zur Verschlüsselung, Integritätssicherung und Authorisierung angewendet werden, um

die Agenten vor dem Abhören ihrer Kommunikation, Abstreitung, Maskierung und Wiedereinspielung zu schützen.

Denial-of-Service-Angriffe auf Agenten sind zum Beispiel durch das Überfluten mit Nachrichten möglich. Der Agent wird solange mit Nachrichten überschüttet bis seine weitere Ausführung eingeschränkt ist. Dieses Problem läßt sich durch eine effiziente Kommunikationsverwaltung auf die Problematik der Agentenplattform einschränken. So können solche Angriffe z.B. dadurch verhindert werden, daß dem Agenten nur eine bestimmte Anzahl der Nachrichten übermittelt werden, der Rest wird zum Problem der Plattform. Der Agent kann auf diese Weise weiterhin seine Aufgaben erfüllen.

Angriffe gegen Agentenplattformen

Die Agentenplattformen sind wie oben erwähnt auch gewissen Sicherheitsbedrohungen ausgesetzt. Böswillige Agenten und Plattformen, die anderen Interessensgruppen als die Plattform selbst angehören, könnten versuchen die Plattform in ihrer Funktionalität einzuschränken. Es können die Dienste der Plattform, ihre Ressourcen und sogar das zugrundeliegende Rechnersystem angegriffen werden. Die Gefahr kann wie bei Angriffen auf Agenten von lokalen oder entfernten Angreifern kommen. Nach ihrer Art können die Angriffe wieder in technische und anwendungsspezifische Angriffe eingeteilt werden. Die meisten Angreifer nutzen die Schwachstellen in der Implementierung der Plattform aus. Solche sind trotz sorgfältiger Programmierung oft vorzufinden. Dem böswilligen Agenten geht es dabei darum, Dienste und Ressourcen unberechtigt zu benutzen und/oder die Kontrolle über diese Dienste oder gar die ganze Plattform und das zugrundeliegende System zu übernehmen. Aus diesem Grunde basieren einige Agentensysteme auf dem Konzept der kontrollierten Ausführungsumgebungen, einer Art *Sandbox* für Agenten, in denen die Agenten mit eingeschränkten Rechten, kontrolliert ausgeführt werden.

Die gefährlichsten Angriffe sind jedoch die Denial-of-Service-Angriffe. Die Plattform kann durch Verschwendung der verfügbaren Ressourcen seitens eines böswilligen Agenten oder durch das Überfluten mit Nachrichten von anderer Plattform in ihrer Funktionalität stark eingeschränkt werden, bis sie nicht mehr in der Lage ist ihre Aufgabe zu erfüllen. Gegen eine solche Art von Angriffen ist ein Schutz nur teilweise möglich. Gegen die gefährlichste Art, die sog. verteilten Denial-of-Service-Angriffe (distributed DoS, DDoS), bei denen gleichzeitig von mehreren Rechnern gleichzeitig Anfragen ausgeführt werden, gibt es momentan keinen wirksamen Schutz.

4.2 Sicherheitsanforderungen

Die Definition der Sicherheitsanforderungen stellt neben der Identifikation der Sicherheitsbedrohungen auch den ersten Schritt in Richtung der Definition von wirksamen Sicherheitsmaßnahmen dar. Sie sind anwendungsabhängig, da IT-Systeme vielen verschiedenen Anwendungen dienen und die Sicherheit eines Systems sich immer auf die in der Anwendung bedrohten Werte bezieht. Es ist auch die Aufgabe einer Anwendungsumgebung, die sicherheitsrelevana-

te Anforderungen aufgrund ihrer spezifischen schutzwürdigen Güter zu erstellen. Die Einstufung einer Anforderung als sicherheitsrelevant für ein System hängt allerdings vom konkreten Anwendungsfall und der Einschätzung des Anwenders ab. Die Anwender sind durch ihre Interessen geprägt, die wiederum die Sicherheitsbedrohungen an den Werten erkennen und die benötigten Sicherheitsanforderungen identifizieren lassen. Zur Klassifikation der Sicherheitsanforderungen die als Basis dienen gibt es verschiedene Konzepte. Hierbei wurde bewusst das Konzept der Sicherheitsanforderungen aus dem Umfeld der Kommunikation übernommen, um die Thematik dieser Diplomarbeit zu unterstützen.

4.2.1 Authentizität

Die Authentizität liefert die Gewissheit über die Identität eines Individuums, eines Nutzers oder in der Kommunikation eines Kommunikationpartners und somit die Sicherheit, daß die empfangenen Daten auch vom sich ausgebendem Sender stammen. Sie kann auch als das Recht des Nutzers betrachtet werden, seine Identität zu benutzen. Ein Individuum kann gleichzeitig mehrere Identitäten haben. Der Prozeß der Bindung des Nutzers an seine Identität wird Authentisierung genannt. Es gibt verschiedene Techniken um die Nutzer zu authentisieren, wie z.B. Paßwörter, Zertifikate[1] oder biometrische Methoden. Gemeinsam ist allen Methoden, daß das Individuum anhand einer zweifelsfreien Eigenschaften (Paßwort, Schlüssel, Augeniris) eindeutig identifiziert werden kann. Das Individuum muß die Korrektheit der von ihm behauptete Identität nachweisen. Nur wenn die Identität eines Nutzers zweifelsfrei besteht, können ihm bestimmte Rechte eingeräumt werden.

4.2.2 Verfügbarkeit

Bei der Verfügbarkeit wird ein gewisses Maß an Zugänglichkeit bzw. Verfügbarkeit von Daten und die Benutzbarkeit von Betriebsmitteln bzw. im Kontext der Kommunikation, der Komponenten zur Datenübertragung verlangt. Um die unbefugte Vorenthaltung von Daten bzw. Systemkomponenten zu verhindern, können entsprechende Mechanismen, wie beispielsweise redundante Kommunikationskanäle eingesetzt werden.

4.2.3 Vertraulichkeit

Vertraulichkeit ist die Eigenschaft, welche die Geheimhaltung von Daten sichert. Daten sollen nur den dazu berechtigten Subjekten (Personen oder Softwareeinheiten im Auftrag von Personen) zur Verfügung stehen, für alle Anderen sollten die Daten geheim bleiben. Um dies zu gewährleisten, können beispielsweise Verschlüsselungsverfahren eingesetzt werden.

4.2.4 Integrität

Die Integrität der Daten ist gewährleistet, falls es möglich ist, eine unauthorisierte Änderung der Daten, die entweder durch unabsichtliche Fehler oder beabsichtigte Manipulationen

[1] Zertifikate binden die Identität eines Subjekts bzw. seinen Namen an seinen öffentlichen Schlüssel. Mehr dazu in Abschnitt 4.4.

entsteht, sofort zu erkennen und eventuell rückgängig zu machen. So wird bei der Kommunikation die Eigenschaft der Integrität erfüllt, wenn durch besondere Mechanismen erkennbar ist, ob eine Nachricht auf ihrem Weg vom Absender zum Empfänger manipuliert worden ist oder ein Übertragungsfehler aufgetreten ist. Zur Gewährung der Integrität werden spezielle Mechanismen wie Hashfunktionen, Message Authentication Codes (MAC) und digitale Unterschrift benutzt.

4.2.5 Verbindlichkeit

Unter Verbindlichkeit versteht man in Allgemeinem:

> *die durch gesellschaftliche Kontrolle garantierte Erfüllung einer in die Zukunft gerichteten Aussage, meist eines Versprechens oder einer Weisung.[**?**]*

Die Verbindlichkeit kann durch IT-Systeme nicht realisiert, sondern nur durch geeignete Maßnahmen unterstützt werden. Zum Zweck der Unterstützung der Verbindlichkeit werden die Mechanismen der Integritätssicherung benutzt. Die Technologie die sich zu diesem Zweck etabliert hat, heißt digitale Unterschrift. Die digitale Unterschrift stellt einen Mechanismus dar, mit dem die Authentizität eines Dokumentes bzw. einer Nachricht nachweisbar ist. Dadurch wird der Nutzer auf ein bestimmtes Versprechen (Nachricht) gebunden und das Abstreiten (repudiation) desselben ist nicht möglich.

4.3 Sicherheitsmaßnahmen

Die Sicherheitsmaßnahmen stellen die Maßnahmen dar, die den Bedrohungen gegenübergestellt werden, um Ihnen entgegenzuwirken. Das Ziel dabei ist es, die Durchsetzung der Sicherheitsanforderungen zu erreichen. Zu diesem Zweck werden einige grundlegende Mechanismen in Form von Protokollen, Verfahren und Algorithmen definiert, die zu komplexeren Maßnahmen zusammengesetzt werden können. Die Realisierung der Sicherheitsmechanismen ist aber den Entwicklern überlassen. Die Aufgabe von Sicherheitskriterien ist die Definition, welche Sicherheitsmechanismen miteinanderkombiniert werden, um bestimmte Sicherheitsmaßnahmen zu realisieren. Mit Sicherheitskriterien beschäftigen sich einige verschiedende Konzepte, wie zum Beispiel die europäischen Sicherheitskriterien Information Technology Security Evaluation Criteria (ITSEC) und das Basisreferenzmodell der *ISO*, Open System Infrastructure (OSI). Eine Sicherheitspolitik hingegen, beschreibt den gesamten Abbildungszusammenhang von Sicherheitszielen bzw. zu schützenden Daten auf Sicherheitsanforderungen und zugehörige Sicherheitsmaßnahmen.

Im folgenden Abschnitt werden einige grundlegende Sicherheitsmechanismen mit zugehörigen Beispielen dargestellt. Eine detailliertere Beschreibung dieser Verfahren kann man beispielsweise in dem Buch von *Stallings*[**?**] finden.

4.3.1 Authentisierung

Unter Authentisierung werden Mechanismen für die Angabe und Prüfung von Identitäten (Verifikation) zusammengefaßt. Sie bilden die Basis für die Verwendung weiterer Sicherheitsmechanismen, da ohne Identifikation keine Rechtevergabe (Authorisation) und ihre Nutzung möglich ist. Die Authentisierung kann, einseitig (nur ein Teilnehmer wird authentisiert) oder beidseitig (beide Teilnehmer authentisieren sich gegenseitig) durchgeführt werden. Die Subjekte (Benutzer, Agenten, Prozesse) haben grundsätzlich drei Möglichkeiten, sich zu authentisieren:

1. durch Wissen
   Die Benutzer können sich durch Wissen eines gemeinsames Geheimnisses, wie z.B. Passwort oder PIN, dem System bzw. Kommunikationspartner gegenüber identifizieren. Die Authentisierung geschieht durch Übermittlung dieses Geheimnisses vom Benutzer zum System. Das System prüft das Geheimnis und identifiziert dadurch den Benutzer.
   Paßwörter und PIN's sind die meist genutzte Methode zur Authentisierung, da sie eine schnelle und relativ sichere Methode darstellen. Abgesehen von Schwachstellen die sich durch mangelnde Sicherheitsanforderungen ergeben, wie z.B. Abspeicherung und Übermittlung der Paßwörter. Im Klartext bedeutet dies, daß das größte Sicherheitsrisiko bei diesem Verfahren die Benutzer darstellen, die durch Auswahl von einfachen Paßwörter, dem böswilligen Angreifer die Einbruchsversuche im System erleichtern.

2. durch Besitz
   Bei dieser Art der Authentisierungsverfahren dient der Besitz eines gemeinsamen Geheimnisses (Schlüssel, Token) als Beleg dafür, daß das zu verifizierende Subjekt eindeutig zu identifizieren ist. Das bekannteste Beispiel hier ist das Verfahren auf Basis von kryptographischen Schlüsseln. Dabei teilen sich die Verfahren in solcke auf, die auf symmetrischen Schlüsseln und solche die auf asymmetrischen Schlüsseln basieren.
   Bei den symmetrischen Authentisierungsverfahren besitzen beide Seiten denselben Schlüssel. Dieser wird benutzt, um eine bestimmte Information (Zufallszahl, Namen des Subjekten, Zeit), die den beiden bekannt ist, ein sog. Nonce zu verschlüsseln. Die verifizierende Seite entschlüsselt die Nachricht, und prüft die Korrektheit des Nonce. Das Needham-Schroeder-Protokoll das im Kerberos[2] im Einsatz ist, baut sein Authentisierungsmechanismus auf symmetrischen Schlüssel auf.
   Bei den asymmetrischen Verfahren besitzt das sich authentisierende Subjekt einen privaten Schlüssel, den nur er kennt, und das verifizierende Subjekt kann die Authentizität mittels eines zugehörigen öffentlichen Schlüssels prüfen. Das verifizierende Subjekt muß Kenntnis über den öffentlichen Schlüssel des zu authentisiereden Subjekten haben. Entweder liegt dieser vor oder er wird beispielsweise durch Zertifikate ausgetauscht. Hierbei wird das Nonce mit dem privaten Schüssel des verifizierten Subjekts verschlüsselt, das nach dem Empfang mit dem öffentlichen Schlüssel entschlüsselt und daraufhin die Gültigkeit des Nonce überprüft werden kann. Einige Authentisierungs-

[2]Kerberos ist ein Authentisierungsdienst der vom Massachusets Institute of Technologies entwickelt wurde. Das Ziel dabei war, ein sicheres System aufzubauen, das einen Server bei der Authorisierung der diensanfragenden Clients unterstützt. Quelle: [**?**].

protokolle wie zum Beispiel X.509 Authentisierungsprotokoll von *ITU-T*.[3] basieren auf diesem Prinzip.
Diese Arten der schlüsselbasierenden Authentisierungsverfahren sind als relativ sicher einzustufen. In Abhängigkeit von Implementierung der Protokolle stellen die Replay- und Man-in-the-middle-Angriffe die größte Bedrohung für diese Sicherheitsmechanismen dar.

3. durch Merkmale
   Ein gutes Beispiel für die Authentisierungsmechanismen durch Merkmale stellen die biometrische Verfahren, bei denen die Identität eines Menschen aufgrund seiner biometrischen Merkmalen (Fingerabdruck, Irismuster, Stimme) festgestellt wird.

Als "starker" Authentisierungsmechanismus wird ein Mechanismus angesehen, der die Kombination von mindestens zwei dieser Verfahren darstellt. Zum Beispiel Paßwort in der Kombination mit dem auf der Smart-Card gespeicherten Schlüssel stellt ein starkes Authentisierungsmechanismus dar.

4.3.2 Zugriffskontrolle

Die Zugriffskontrolle stellt ein Mechanismus dar, der Ressourcen vor unberechtigten Zugriffen schützt. Nach *Fries et al.*[**?**] können die Maßnahmen zur Durchsetzung der Zugriffskontrolle im Allgemeinem in drei Grundtypen aufgeteilt werden, abhängig davon ob die Zugriffsrechte direkt auf Objekte oder für Subjekte definiert werden.

- Zugriffskontrolle aus Objektsicht:
  Bei dieser Art der Sicherheitsmechanismen wird für jedes Objekt beschrieben, welche Subjekte mit welchen Rechten zugreifen dürfen. Ein bekanntes Beispiel sind die Zugriffskontrolllisten (Access Control Lists). Sie definieren für jedes Objekt des Systems (Ressorcen, Dienste) eine Liste von Zugriffsrechten für Subjekte bzw. Benutzer. Die Liste besteht aus einer Zuordnung der Subjekte zu den Operationen mit denen dieses Subjekt auf das Objekt zugreifen darf. Somit kann sofort bei dem Zugriff auf das Objekt, festgestellt werden ob der Benutzer berechtigt auf dieses Objekt zugreift.
  Diese Zugriffskontrollisten sind beispielsweise bei dem Betriebsystem UNIX im Einsatz. Bei UNIX werden für jedes Objekt (Dateien, Verzeichnise, Prozesse) die Zugriffsrechte der Subjekte (Besitzer, Gruppe und Andere) durch drei Grundoperationen (Lesen, Schreiben und Ausführen) definiert.

- Zugriffskontrolle aus Subjektsicht:
  Diese Art der Mechanismen beschreibt für jedes Subjekt, auf welche Objekte mit welchen Rechten zuzugreifen ist. Im Gegensatz zu Zugriffskontrolllisten, definieren die Capabilities (als ein Beispiel für solche Zugriffsmechanismen) die Rechte von Subjekten im System. Sie stellen eine Art Ticket bzw. eine Eintrittskarte für Zugriffe dar. Ist ein Subjekt im Besitz eines gültigen Tickets für ein Objekt, so darf er das Objekt in entsprechender Weise nutzen. Die Rechte der Subjekte im System sind dann die Summe aller

[3] X.509 ist ein Teil der Spezifikation für ein Verzeichnisdienst, der unter den Namen X.500 bekannt ist. Es definiert ein Gerüst für Authentisierungsdienst innerhalb von X.500. Quelle: [**?**]. Mehr dazu im Abschnitt 4.4.

seiner Capabilities. Capabilities sind in einer Kombination mit Zugriffskontrolllisten bei dem Betriebssystem Windows NT (res. 2000) im Einsatz. Bei diesen Betriebssystemen wird jeder Benutzer zu einer Gruppe zugeordnet (Bsp. Hauptbenutzer, Administartoren), dessen Zuggrifsrechte im System gut definiert sind.

- Zugriffskontrolle aus Subjekt- und Objektsicht:
  Zugriffskontrollmatrix stellt eine Kombination beider Arten der Zugriffsmechanismen. Die Zugriffsrechte werden in der Form einer (j,k)-Matrix dargestellt, wo jeder Eintrag eine Menge der Zugriffsrechte repräsentiert, die das Subjekt j bezüglich des Objekten k besitzt. Dieser Ansatz ist eher theoretisch und findet kaum praktische Anwendung.

4.3.3 Verschlüsselung

Die Aufgabe der Verschlüsselungsmechanismen ist es, Daten so zu präparieren, daß sie nur für Befugte lesbar sind. Bei der Verschlüsselung werden Daten im Klartext in eine scheinbar zufällig generierte, unlesbare Form konvertiert, um sie dann beispielsweise zu einem bestimmten Kommunikationspartner zu transferieren. Der Verschlüsselungsprozeß besteht aus einem Algorithmus und dem Schlüssel oder Schlüsselpaar. Der Schlüssel stellt einen Wert dar, der unabhängig vom verschlüssenden Klartext ist. Der Algorithmus produziert durch Anwendung des Schlüssels einen verschlüsselten Output, der wiederum vom dem Schlüssel abhängig ist. Nach dem Empfang der verschlüsselten Daten kann der Empfänger sie mit dem eigenen Schlüssel und dem entsprechenden Algorithmus wieder in die lesbare Form zurücktransformieren. Dieser Umkehrprozeß wird auch als Entschlüsselung bezeichnet. Bezüglich der Verteilung der Schlüssel zwischen Sender und Empfänger der verschlüsselten Daten, unterscheidet man hierbei zwischen symmetrischen und asymmetrischen Verschlüsselungsverfahren.

- symmetrische Verschlüsselung
  Die symmetrische Verschlüsselung basiert auf der Verwendung des gleichen geheimen Schlüssel zwischen dem Sender und dem Empfänger der Daten. Sie benutzen einen Schlüssel und einen Algorithmus zum generieren des Verschlüsselten Daten und ihrer Entschlüsselung. Einer der bekanntesten Verfahren, Data Encryption Standard (DES) benutzt eine Kombination der Algorithmen für Permutationen (Bitverschiebung) und Substitutionen (Bitvertauschung) der Ziffern-Blöcke um die Nachricht zu verschlüsseln. Da DES mit seiner Schlüssellänge von 56 Bit als eher unsicher einzustufen ist (siehe Tabelle **??**), wird in der Praxis das dreifache DES (triple DES, 3DES) benutzt. 3DES kombiniert zwei verschiedene Schlüssel und mehrmalige abwechselnde Anwendung der Verschlüsselungs-/Entschlüsselungsalgorithmen um die Nachricht zu verschlüsseln. Somit erhöht sich die Sicherheit der Verschlüsselung um ein vielfaches gegenüber dem einfachen DES. Als Nachfolger von DES wird heutzutage Advanced Encryption Standard (AES)[4] verwendet.

- asymmetrische Verschlüsselung
  Bei den asymmetrischen Verschlüsselungsverfahren hingegen basiert der Verschlüsselungs-

[4]Der AES ist ein 128-Bit symmetrisches Verschlüsselungsverfahren, der von den belgischen Kryptographen *Daemen* und *Rijmen* entwickelt wurde. Zur Verschlüsselung verwendet AES den sog. *Rijndael* Algorithmus.

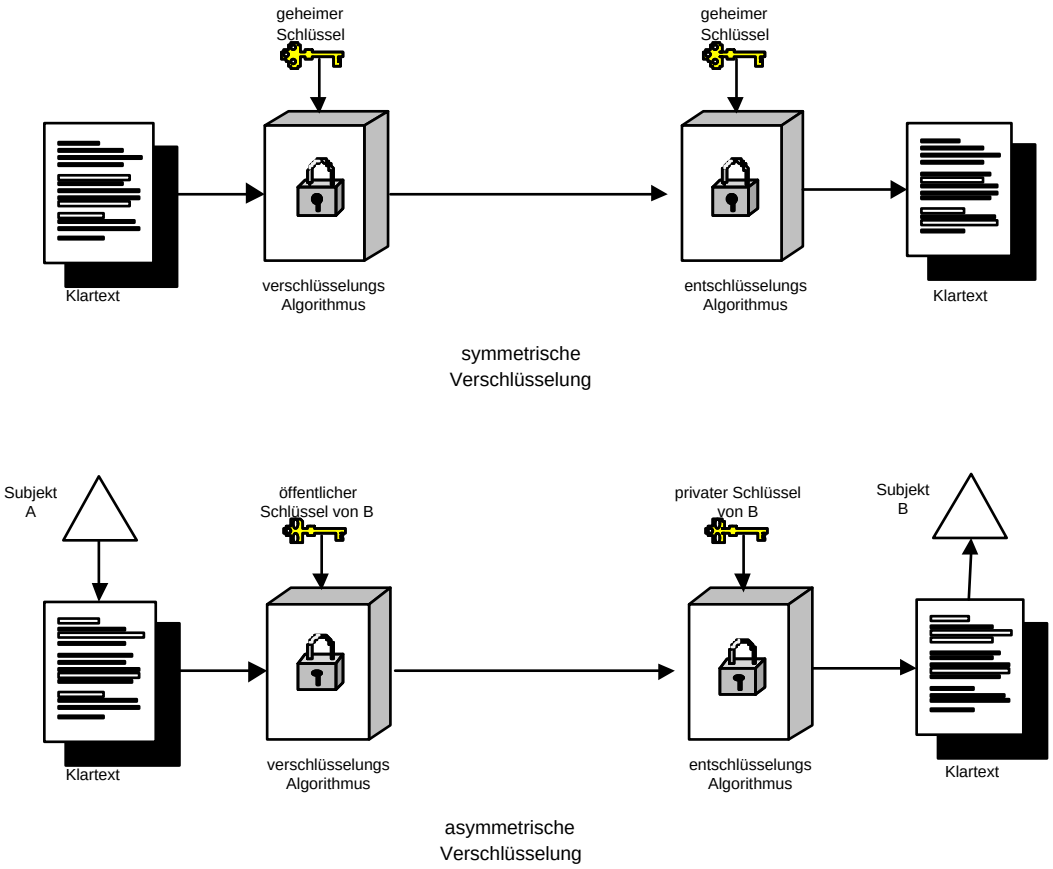

Abbildung 4.2: Symmetrische und asymmetrische Verschlüsselung

und Entschlüsselungsprozeß auf Schlüsselpaaren. Der Sender und Empfänger besitzen jeweils ein eigenes Schlüsselpaar, das aus einem privaten Schlüssel (private key) und einem öffentlichen Schlüssel (public key) besteht. Der Private Schlüssel dient der Entschlüsselung und bleibt ein Geheimnis des Besitzers, während mit dem öffentlichen Schlüssel verschlüsselt wird, dieser also nicht geheimgehalten werden muß. Die Schlüssel sind ungleich und bilden im krypthographischen Sinne ein Paar.
Die asymmetrischen Verschlüsselungsverfahren bauen auf einem einfachen Mechanismus auf. Zu der Verschlüsselung der Daten bedient sich der Sender an dem öffentlichen Schlüssel des Empfängers. Er benutzt ihn um die Nachricht zu verschlüsseln. Zu diesem Zweck müssen die öffentlichen Schlüssel vorher ausgetauscht werden oder von dafür vorgesehenen authoritären Instanzen sog. *Key Authorities* angefordert werden. Um die Nachricht wieder zu Entschlüsseln benutzt der Empfänger der Nachricht seinen privaten Schlüssel. Die Stärke dieses Verfahrens liegt darin, daß man die Nachricht mit einem Schlüssel (öffentlich) verschlüsselt, einfach mit seinem komplementären Schlüssel (privat) entschlüsseln kann, und man trotzdem aus dem Besitz des öffentlichen Schlüssels nicht auf den privaten Schlüssel schließen kann[5].

[5]Mathematisch ist die Unmöglichkeit dessen für jedes Verfahren nicht bewiesen.

Das bekannteste asymmetrische Verfahren ist mit Sicherheit RSA (benannt nach seinen Erfindern Rivest, Shamir und Adleman). RSA-Verfahren baut auf Schwierigkeit der Primfaktorisierung[6] der großen Zahlen auf. Um ein Schlüsselpaar zu generieren bedarf es eine Auswahl der genügend großen Primzahlen, dessen Produkt einen Teil beider Schlüssel ausmacht und die restlichen Teile der Schlüssel sich aus einem mathematischen Algorithmus ergeben. Die Tatsache das die Primfaktorisierung genügend großer Zahlen (die Primzahlen bei der Auswahl sollten mindestens 1024 Bit oder 100 Ziffern groß sein) in absehbarer Zeit durchführen fast unmöglich ist, macht dieses Verfahren sicher gegen Kryptanalysen[7]
Außer dieser asymmetrischen Verfahren gibt es auch noch die Verfahren, die auf sog. Elliptischen Kurven[8] basieren. Diese Verfahren kommen mit einer kleineren Schlüssellänge (vergl. Tabelle **??**) und demzufolge weniger Rechen- und Zeitaufwand aus, um denselben Grad an Sicherheit, wie z.B. RSA zu erreichen. Da sich diese Verfahren aber noch in der Standardisierungs- und Einführungsphase in die Praxis befinden, wird auf sie hier nicht näher eingegangen.

- hybride Verschlüsselungsverfahren
  Die hybriden Verschlüsselungen, wie der Name schon vermuten läßt, vereinen in sich beide Verschlüsselungskonzepte. Meist wird dabei das Konzept des Einwegschlüssels (session key) benutzt. Nach diesem Konzept werden die Nachrichten vertraulich übermittelt, indem man einen beliebigen, zufälligen, nur für dieses Mal gültigen symmetrischen Schlüssel generiert. Dieser Schlüssel wird daraufhin mit dem asymmetrischen Verfahren zu dem Kommunikationspartner übertragen und dann für die symmetrische Verschlüsselung der Daten benutzt, solange die beiden Partner kommunizieren.
  Nach dem selben Prinzip funktioniert auch das im Internet wohl bekannteste Protokoll zur Sicherung der Transportschicht, das sog. SSL/TSL-Protokoll (Secure Socket Layer/Transport Socket Layer), mit dem kleinem Unterschied, daß beim SSL/TSL nicht direkt die symmetrischen Schlüssel übertragen werden, sondern nur gemeinsame geheime Werte, aus denen jede Seite denselben Schlüssel generieren kann. Mehr zum Thema SSL/TSL im Abschnitt 4.5.

Die Sicherheit der aufgeführten Verschlüsselungsverfahren hängt an erster Stelle von Länge des Schlüssels ab. Tabelle **??** zeigt den Vergleich von *Silverman*[**?**] zwischen den Schlüssellängen der symmetrischen Verfahren, Verfahren basierend auf Elliptischen Kurven (EC) und RSA-Verfahren, und der Zeit die zum Brechen bzw. Kryptanalyse derselben benötigt wird. Je länger ein Schlüssel desto zeitaufwändiger die Verschlüsselung ist, aber gleichzeitig auch

[6] Unter der Primfaktorisierung versteht man die Zerlegung einer Zahl, bzw. Bestimmung ihrer Primfaktoren. Dieses Verfahren ist im Prinzip sehr einfach aber für große Zahlen noch sehr zeitaufwändig.

[7] Kryptanalyse befasst sich im wesentlichen mit dem Brechen von Verschlüsselungsverfahren.

[8] Elliptische Kurven sind Graphen, deren Punkte (x,y) Gleichungen folgender Art erfüllen:
$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$. Bei den kryptographischen Verfahren die auf elliptischen Kurven basieren (sog. ECC Verfahren) sind diese Gleichung, genauso wie ein Punkt der Ellipse bekannt. Dieser Punkt bildet durch gewisse Eigenschaften eine bestimmte Zahlengruppe, die auf Arithmetik Modulo n basiert. Aus dieser Gruppe werden die Elemente zur Bildung des geheimen Schlüssels gewählt. Die Schwierigkeit des Verfahrens besteht in der Komplexität der Berechnung von diskreten Algorithmen. Mit ECC Verfahren sind viele Verfahren der asymmetrischen Kryptologie realisierbar.

sicherer die Daten. Die Schlüssellänge wird in Bits angegeben. Als sicher gelten zur Zeit die Schlüssellängen von $\geq$ 128 bit für symmetrische und von $\geq$ 1024 Bit für asymmetrische Verfahren.

symm. Schlüssel	EC-Schlüssel	RSA-Schlüssel	Zeit zum Brechen
56	112	430	weniger als 5 Minuten
80	160	760	50 Jahre
96	192	1020	3 Millionen Jahre
128	256	1620	10^{16} Jahre

Tabelle 4.1: Vergleich zwischen Schlüssellängen und der benötigten Zeit für die Kryptanalyse.

Verschlüsselungsverfahren mit kürzeren Schlüssel sind durch die sog. Brute-Force-Angriffe extrem anfällig. Bei Brute-Force-Angriffen wird der gesamte Schlüsselraum[9] systematisch ausprobiert bis der richtige Schlüssel gefunden ist. Falls die Schlüssellänge klein ist, ist der Schlüsselraum auch dementsprechend klein und in absehbarer Zeit berechenbar. Eine weitere Gefahr für diese Verschlüsselungsarten stellen verschiedene andere kryptanalytische Angriffe dar, die direkt die Eigenschaften der Algorithmen aufnehmen, wie zum Beispiel differentielle Kryptanalyse[10] und mathematische Angriffe.

4.3.4 Datenintegrität

Bei den Mechanismen zur Sicherung der Integrität von Daten geht es hauptsächlich um zwei Aspekte - den Beweis für die Authentizität der Nachricht und des Autors zu erbringen, und Änderungen bzw. Manipulationen der Daten erkennbar zu machen. Aus diesem Grunde wird für jede Nachricht durch dafür vorgesehene Mechanismen eine zusätzliche spezielle und eindeutige Information erstellt, die dazu dient diese Nachricht zu prüfen. Hierzu gibt es verschiedene Verfahren: von einfachen, die aus einem Mechanismus bestehen (z.B. MAC) bis hin zu komplexeren die auf mehreren Mechanismen aufbauen (z.B. digitale Unterschrift).

- Message Authentication Codes (MAC)
  Der Message Authentication Code (Nachrichtenauthentifizierungscode) ist ein einfacher Mechanismus, der Nachrichten beliebiger Länge auf einen fälschungssicheren Wert der festen Länge (MAC) abbilden. Die Berechnung des MAC funktioniert dabei ähnlich wie ein symmetrisches Verschlüsselungsverfahren. Mit Hilfe eines symmetrischen Schlüssels wird die Nachricht verschlüsselt und als Zusatzinformation zur Nachricht dem Empfänger geschickt. Zur Prüfung der Nachricht verschlüsselt der Empfänger die empfangene Nachricht und prüft die Gleichheit beider MACs. Durch Prüfung des MAC seitens des Empfängers kann die Authentizität der Nachricht geprüft werden, da nur

[9] Schlüsselraum bezeichnet alle mögliche Schlüssel für eine bestimmte Länge. Bei 56 Bit Schlüssellänge ergeben sich somit 2^{56} mögliche Schlüssel.

[10] Differentielle Kryptanalyse ist eine Methode, mit der man versucht eine Verschlüsselung dadurch zu brechen, daß von beliebigen Klartext ausgehend mit Wahrscheinlichkeitsrechnung auf den Schlüssel geschlossen wird. Durch die statistische Betrachtung der Differenzen zwischen zwei Klartexte und entsprechender Krypttexte werden die Informationen über den Schlüssel gewonnen.

der Sender und Empfänger den Schlüssel besitzen, und eine eventuelle Veränderung aufgedeckt wird. Der Unterschied zur konventiellen symmetrischen Verschlüsselungsverfahren besteht darin, daß der Verschlüsselungsprozeß nicht umkehrbar sein muß, d.h. es wird nicht verlangt, daß die Nachricht mit dem selben Algorithmus entschlüsselbar ist, da der Empfänger die Nachricht zur Prüfung auch nur verschlüsselt. Aus diesem Grunde, durch eine geschickte Auswahl des Algorithmus, ist dieses Verfahren weniger gegen kryptanalytische Angriffe empfindlich als normale Verschlüsselungsverfahren.

- Hashalgorithmen
  Hashalgorithmen basieren ähnlich wie die MAC-Verfahren auf dem Prinzip der eindeutigen Abbildung einer Nachricht variabler Länge auf ein Prüfwert fester Länge (Hashwert). Dies ist bei Hashalgorithmen ein Einwegprozeß, d.h. die Umkehrung bzw. Berechnung der Nachricht aus ihrem Hashwert ist fast unmöglich. Anderes als das MAC-Verfahren benutzten die Hashalgorithmen keinen gemeinsame Schlüssel sondern basieren auf einem allgemeinem Verfahren. Deshalb sind die Verfahren mit Hashalgorithmen nur dafür geeignet, um die Unversehrtheit der Nachricht zu prüfen. Der Kern jedes Hashalgorithmus ist die sog. Hashfunktion (Message Digest). Sie besteht aus mehreren bitweise logischen Funktionen, die eine Nachricht beliebiger Länge auf ein Wert bestimmter Länge reduzieren. Dabei muß die Hashfunktion so gebaut sein, daß sie verschiedene Nachrichten auf möglichste verschiedene Hashwerte abbilden kann. Dieser Wert wird zusammen mit der Nachricht an den Empfänger weitergereicht, der wiederum den Hashwert der Nachricht erstellt und anschließend beide Hashwerte vergleicht. Der Unterschied zwischen den Hashwerten ist ein Hinweise auf Veränderung bzw. Manipulation der Nachricht.
  Die bekanntesten hashbasierte Verfahren sind der Message Digest 5 (MD5) und der Secure Hash Algorithm 1 (SHA-1). SHA-1 ist eine Entwicklung des *NIST (National Institute of Standards and Technology)* und stellt gemessen an der Länge des produzierenden Hashwertes ein sichereres Verfahren dar als MD5.

- Digitale Signatur
  Die digitale Signatur stellt ein Verfahren dar, das mehrere Sicherheitsmechanismen vereinigt, um die Integrität der Daten zu gewähren. Ziel der digitaler Signatur ist es, die Nachricht nicht nur vor Dritten zu schützen, wie daß bei MAC der Fall ist, sondern auch Schutz der Kommunikationspartner voreinander. Es sind Situationen denkbar in denen zum Beispiel:

  – ein Subjekt A den Schlüssel von B benutzt (den die beiden gemeinsam besitzen) um das MAC für eine Nachricht für C zu generieren und behauptet die Nachricht wurde von B stammen
  – der Subjekt B kann die an C verschickte Nachrichten abstreiten, da es für den A einfach ist, die MAC zu fälschen

  Aus diesem Grund kombinieren die Verfahren für digitale Signatur die Hashfunktionen mit den Verschlüsselungsverfahren um die Authentizität der Nachricht und eine eindeutige Zuordnung zu dem Autoren der Nachricht zu erreichen. Der Prozeß der digitalen Signatur und Verifizierung einer Nachricht kann man vereinfacht, wie auf dem Bild **??**

darstellen.

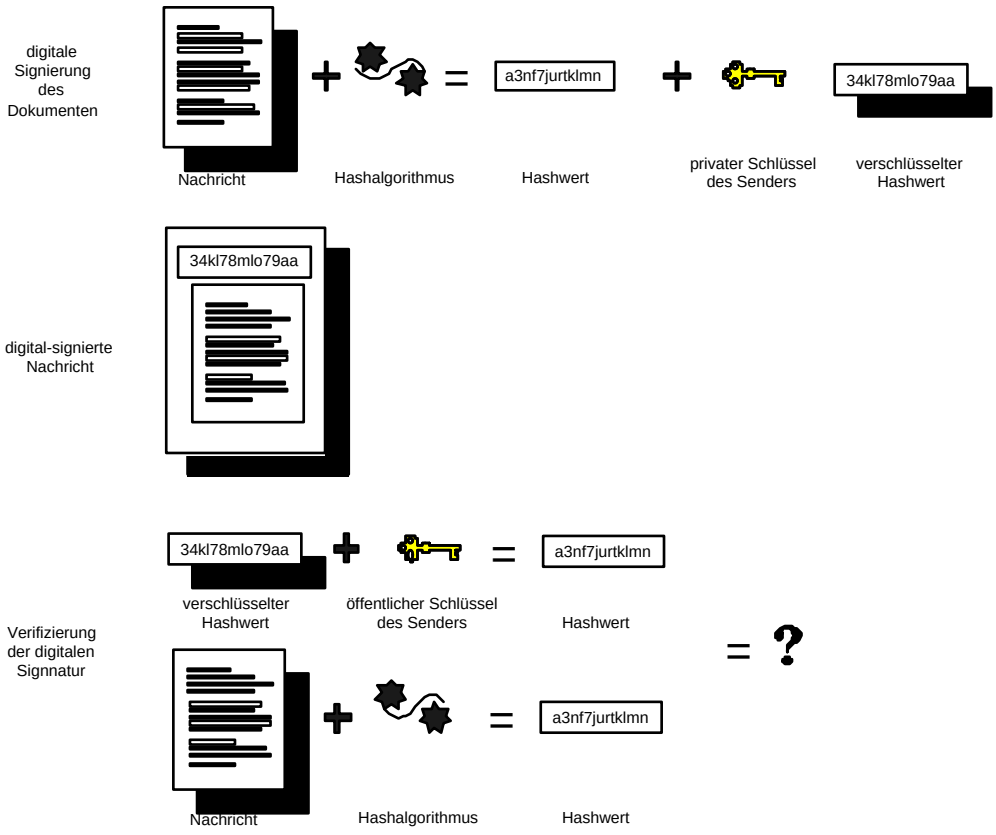

Abbildung 4.3: Digitales Signieren und Verifizieren einer Nachricht

Der Standard zur digitalen Unterschrift Digital Signature Standard (DSS) von *NIST* ist neben den Verfahren von RSA und ElGamal (benannt nach seinem Erfinder) mit Sicherheit der bekannteste Mechanismus für die digitale Signatur. DSS baut auf einem Algorithmus, dem sog. Digital Signature Algorithm (DSA) auf, der speziell und nur zum Zwecke der digitalen Signatur entwickelt worden ist. Dieser Algorithmus kombiniert ein symmetrisches und asymmetrisches Verschlüsselungsverfahren mit der Funktionalität von SHA-1 um die Signatur zu erzeugen und zu prüfen. Mit Hilfe von SHA-1 wird ein Hashwert produziert, der zusammen mit dem geheimen Schlüssels des Signierers, einer geheimen nur für diese Session gültige Zufallszahl und eine Menge an Parametern, die den globalen öffentlichen Schlüssel repräsentieren, in die Signierfunktion eingeht. Als Resultat der Signierfunktion bekommt man zwei Komponenten, welche die Signatur der Nachricht darstellen. Nach dem Empfang der Nachricht erstellt der Empfänger den Hashwert der Nachricht, der zusammen mit dem globalen öffentlichen Schlüssel und dem öffentlichem Schlüssel des Senders (der mit dem privaten Schlüssel des Senders ein krypthographisches Paar bildet) durch die Prüfungsfunktion muß. Die Signatur ist gültig, falls das Ergebnis der Prüffunktion einem der beiden Komponenten der Signatur gleicht. Die Signierfunktion hat die Eigenschaft, daß nur der Sender, der

im Besitz des entsprechenden privaten Schlüssels ist, die Signatur erstellt haben konnte. Dieser Algorithmus basiert auf mathematischen Berechnungen bzw. Restzahlen-Arithmetik und seine Stärke liegt in der Schwierigkeit der Berechnung der diskreten Logarithmen[11]. Dieses zusammen mit der kleinen Schlüssellänge und Komponenten des Algorithmus die vorrechenbar sind, geben diesem Verfahren gegenüber der konkurrierenden Signaturverfahren einen gewissen Vorteil im Puncto Effizienz.

Die dargestellten Verfahren zur Sicherung der Integrität der Daten, schützen die Daten nicht direkt vor den potenziellen Angreifern oder unabsichtlichen Fehlern, sondern sollten nur versichern, daß die Daten vom Sender stammen und unverfälscht sind. Der potenzielle Angreifer würde beispielsweise versuchen die Nachricht zu seinem Vorteil zu verfälschen und zwar so, daß sich ihr Hashwert nicht ändert. Auf diese Weise ändert sich ihre Signatur nicht und deshalb kann die Fälschung nicht erkannt werden. Der Hashwert der Nachricht stellt also den empfindlichsten Angriffspunkt dar. Um auf eine Nachricht mit dem selben Hashwert wie die Originalnachricht zu kommen kann der Angreifer die Brute-Force-Methode anwenden, die dank der mathematischen Eigenschaften der Hashfunktion[12] weniger Aufwand als Brute-Force bei Verschlüsselungsverfahren braucht. Außer Brute-Force könnte der Angreifer versuchen die Algorithmen direkt mit kryptanalytischen Methoden anzugreifen. DSA hingegen ist im Anbetracht der Verwendung der diskreten Logarithmen noch ziemlich sicher gegen kryptanalytische Angriffe.

4.4 Schlüssel und Zertifikatsmanagement

In den letzten Abschnitten haben wir die Sicherheitsverfahren kennengelernt, die auf Schlüsseln und Zertifikaten basieren. Diese Schlüssel und Zertifikate müssen erzeugt, sicher gespeichert, verteilt und eventuell widerrufen werden. Diese Aufgaben werden unter dem Begriff des Schüssel- und Zertifikatsmanagements (Key- and Certificate Management) zusammengefaßt.

4.4.1 Schlüsselmanagement

Schlüsselerzeugung

Die Schlüsselerzeugung ist primäre Aufgabe auf dem Weg zur Nutzung eines auf Schlüsseln basierenden Sicherheitsverfahrens. Zur Erzeugung ist, abhängig vom Sicherheitsverfahren, spezielle Software nötig. Das Ziel dieses Vorgangs ist die Erzeugung kryptographisch sicherer Schlüssel unter der Erfüllung der Bedingung der Perfect Forward Security. Diese verlangt eine unabhängige Erzeugung der Schlüssel, bzw. die Unabhängigkeit von den vorher erzeugten Schlüsseln.

[11] Diskrete Logarithmen sind Zahlen, die ein eindeutiges Exponent i in der folgenden Gleichung darstellen: $b = a^i mod p,\ wobei\ 0 \leq i \leq (p-1)$.

[12] Hier ist das Geburtstagsparadox gemeint. Geburtstagsparadox besagt, daß man bei 2^m möglichen Ergebnisse der Hashfunktion nur $2^{\frac{m}{2}}$ Werte prüfen muß, um mit einer Wahrscheinlichkeit von über 50% ein Duplikat zu finden.

Die Schlüssel, die für symmetrische Verfahren benutzt werden, werden meistens zufällig gewählt. Um dies zu bewerkstelligen, werden gute Zufallszahlengeneratoren[13] benötigt. In der Praxis erzeugen die meisten Zufallsgeneratoren nur pseudozufällige Zahlen. Bei der asymmetrischen Verschlüsselung hingegen müssen Schlüsselpaare erzeugt werden. Dazu werden zuerst genügend große Primzahlen ermittelt. Um die diese zu bekommen, werden zufällige große Zahlen mit speziellen probabilistischen Algorithmen auf ihre Primzahleigenschaft getestet.
Die Erzeugung der Schlüssel kann zentral oder dezentral erfolgen. Beim zentralen Ansatz erzeugt ein Server, sog. `Key-Server` die benötigte Schlüssel und verteilt sie an die Clients. Die Schlüssel müssen dabei über sichere Kanäle verteilt werden. Außerdem da der `Key-Server` den zentralen Angriffspunkt darstellt, muß er auch besonderes geschützt werden. Der dezentrale Ansatz erlaubt die Schlüsselerzeugung auf den jeweiligen Endsystemen. Allerdings stellt sich hier auch die Frage, nach der Sicherheit der Schlüssel auf den Endsystemen.

Speicherung und Verteilung von Schlüsseln

Die Schlüsselspeicherung verlangt, eine Speicherung der Schlüssel auf eine Weise, daß mitlesen, verändern und austauschen unmöglich ist.
Die sicherste und aufwendigste Methode ist, die Schlüssel auf einem externem Medium zu speichern, das geschützt wird, und auf das nur lesend zugegriffen werden kann. Eine andere Methode ist die Speicherung der Schlüssel im System und ihr Schutz durch geeignete Zugriffskontrolemechanismen. In Java zum Beispiel werden Schlüssel in speziellen Dateien gespeichert, sog. JKS/JCEKS (Java Key Store/Java Cryptography Extension Key Store), die mit einem Paßwort geschützt sind.
Auch bei der Speicherung können zentrale und dezentrale Ansätze verfolgt werden.

Damit die Kommunikationspartner kryptographische Verfahren anwenden können, müssen vorher benötigte symmetrische, asymmetrische Schlüssel und Werte zur Erzeugung derselben ausgetauscht werden.
Die sicherste Methode dabei ist der persönlicher Austausch oder zumindest der "Out-of-Band"-Austausch, d.h. außerhalb der Kommunikationsysteme, die zur späteren Datenübertragung verwendet werden. Dieser Prozeß wird problematisch bei einer größeren Anzahl von Kommunikationspartner und dementsprechend vielen Schlüsseln, die verteilt werden müssen.
Die Verteilung der symmetrischen Schlüssel ist im Allgemeinem problematischer als die der asymmetrischen, da die asymmetrische Schlüsselpaare lokal erstellt werden können und nur der öffentliche Schlüssel, der nicht geheim gehalten werden muß, ausgetauscht wird. Die symmetrische Schlüssel können genauso wie normale Daten mit asymmetrischen Verfahren übertragen werden.
Eine Möglichkeit sich auf einen gemeinsamen Schlüssel zu einigen, ohne ihn direkt zu übertragen, bietet das Diffie-Hellman Protokoll. Diffie-Hellman basiert auf mathematischen Algorithmen und verdankt seine Effizienz der Problematik, daß es kein eindeutiges, sicheres

[13] Zufallsgeneratoren sind im kryptographischem Sinne Algorithmen, die zufällige Zahlen erzeugen. Ein Zufallsgenerator wird als gut betrachtet, falls er Zahlen erzeugt, die gleichmäßig über ein Intervall gestreut sind und unabhängig voneinander sind.

mathematisches Verfahren zur Berechnung der diskreten Logarithmen gibt. Bei dem Diffie-Hellman-Verfahren werden zwischen den Kommunikationspartner statt Schlüssel, Werte ausgetauscht aus denen die beiden anschließend denselben Schlüssel erstellen.

Widerruf von Schlüsseln

Ein Subjekt sollte auch die Möglichkeit haben seinen Schlüssel zu widerrufen, falls er den Verdacht hat, daß dieser kompromittiert wurde, bzw. daß einer unberechtigter Dritte Kenntnis von dem Schlüssel erlangt hat. Der Widerruf des Schlüssels wird auch dann nötig, wenn das Subjekt sein Schlüssel verloren oder keinen Zugriff mehr auf ihn hat.
Bei symmetrischen Schlüsseln genügt es, wenn sich die Kommunikationspartner über dem Widerruf des Schlüssels einigen. Dieses kann wieder "Out-of-Band" erfolgen oder über im Voraus festgelegte Verfahren. Der Widerruf von asymmetrischen Schlüssel ist etwas aufwändiger und wird im nächsten Abschnitt näher beschreiben.
Um die Sicherheit der Kommunikation zu erhöhen, ist es generell sinnvoll, eine feste Gültigkeitsdauer für Schlüssel festzulegen. Denn je länger ein Schlüssel verwendet wird, desto größer ist die Gefahr, daß er kompromittiert wird. Die Gültigkeitsdauer des Schlüssels muß deshalb im Vorfeld festgelegt und den Bedürfnissen der Kommunikation angepasst werden. Grundsätzlich gilt es, je sensibler die zu übertragenden Daten, desto kürzer die Lebensdauer der dafür benötigten Schlüssel.

4.4.2 Zertifikatsmanagement

Der Begriff Zertifikat wurde in vorherigen Abschnitten eingeführt, um die Problematik des Schlüsselaustausch zu erläutern. Ein Zertifikat bindet die Identität eines kommunizierenden Subjekts sicher an seinen öffentlichen Schlüssel. Diese Bindung bestätigt und verantwortet eine vertrauenswürdige Instanz, die sog. Zertifizierungsstelle (Certification Authority, CA). Derjenige, der ein Zertifikat anfordert (subject) muß einer Zertifizierungsstelle seine Identität und Richtigkeit des Schlüssels beweisen. Die Zertifizierungsinstanz (issuer) überprüft die Angaben und im Falle, daß der Schlüssel vom angegebenen Subjekten stammt, signiert die CA den Schlüssel mit ihrem eigenen geheimen Schlüssel. Durch die digitale Signatur und den öffentlichen Schlüssel der Zertifizierungsstelle kann jeder Kommunikationspartner dann die Informationen im Zertifikat selbst überprüfen. Die digitale Signatur, Name des Subjektes und sein öffentlicher Schlüssel sind Informationen die jedes Zertifikat mindestens enthält. Einer der meistbenutzten Standards für Zertifikate ist der Standard der *ITU* (X.509). Die aktuelle Version des X.509-Zertifikates ist die Version 3 (X.509v3). In einem X.509 Zertifikat sind folgenden Informationen als <Variable>:<Wert> Paare enthalten:

- *Name der Zertifizierungstelle*
- *Daten des Zertifizierungssubjektes*
- *öffentliche Schlüssel des Subjektes*
- *Digitale Unterschrift der CA*
- *Seriennummer des Zertifikats* (wird von der CA vergeben)

- *Gültigkeitszeitraum des Zertifikats*
- *weitere Parameter*

Der X.509-Standard legt nicht fest welche Verschlüsselungsverfahren oder welche Schlüssellänge verwendet werden soll. Die Daten werden in der Form einer Datei gespeichert und weitergegeben. Weiterhin wird davon ausgegangen, daß der öffentliche Schlüssel der CA bekannt ist, da er zur Überprüfung der Zertifikate gebraucht wird.

Die Erzeugung, Verteilung und Verwaltung von Zertifikaten führen zum Bedarf an IT-Strukturen, die diese Aufgaben übernehmen und erfolgreich erfüllen könnten. Ein System aus digitalen Zertifikaten, Zertifizierungsstellen und Behörden, welche die Authentisierung und Verifikation jedes Subjektes übernehmen, das an elektronischen Transaktionen teilnimmt, ist unter dem Namen Public Key Infrastructure (PKI) bekannt. Die Regierungen hätten zum Beispiel Interesse an einer PKI um die Dienste Gesellschaften anzubieten. Größere Organisationen bräuchten eigene PKIs innerhalb der Organisation selbst oder zusammen mit anderen Organisationen, um die Arbeit besser zu organisieren. Einzelnen Personen hätten auch Interesse an solchen Systemen, um zum Beispiel am elektronischen Handel teilzunehmen oder die Sicherheit eigener Privatsphäre, zum Beispiel durch verschlüsselte elektronische Post, zu sichern. Im Grunde unterscheidet man zwischen zwei Ansätzen - dem zentralen bzw. hierarchischen Ansatz (baumähnlich) und dem dezentralen (graphenähnlich) Ansatz in Abhängigkeit davon wie die Verantwortungen über Zertifikate verteilt sind. Momentan gibt es weder eine globale universale PKI, noch einen allgemein gültigen Standard zum Aufbau dergleichen. Es existieren momentan mehrere verschiedene PKIs, die verschiedenen Anwendungsgebieten und Authoritäten zugeordnet sind. Allerdings, sind sich alle daran Beteiligten einig, daß bevor der E-Commerce globale Bedeutung im Handeln erlangt hat, eine zuverlässige PKI notwendig ist.

Zertifikatserzeugung

Die Erzeugung von Zertifikaten unterscheidet sich in Abhängigkeit davon, ob es sich um den zentralen oder um den dezentralen Ansatz handelt.
Der zentrale Ansatz hat eine hierarchisch geordneter Aufbau, die einer Baumstruktur ähnelt. An der Wurzel des Baumes befindet sich eine feste CA, die sog. Wurzel-CA (root CA) von der dann weitere Wege über untergeordneten CAs zum Endbenutzer führen. Um einen Schlüssel zu verifizieren, müssen die Zertifikate der untergeordnete CAs geprüft werden, solange bis eine untergeordnete CA erreicht wurde, deren Schlüssel bekannt ist oder die Wurzel-CA erreicht ist. Diese Kette der prüfenden Zertifikate wird als Zertifizierungspfad (Certification Path) bezeichnet. Das Zertifikat der Wurzel-CA, das sog. Wurzelzertifikat wird von der CA selbst zertifiziert, d.h. selbst unterschrieben. Das Bestehen der selbst unterschriebenen Zertifikate ist der Grund, warum das Zertifikatenmanagement in diesem Fall ein Problem darstellt. Manche Anwendungen (wie Internet-Browser, Programmiersprache Java) liefern deshalb öffentliche Schlüssel der bekanntesten CAs mit. Ein weiteres Problem stellt die Vertrauensproblematik dar, da die Tatsache, daß jemand ein Zertifikat besitzt, das von einer bekannten Zertifizierungsstelle herausgegeben worden ist, nur bezeugt, daß der private Schlüssel in seinem Besitz ist und nicht, daß ihm vertraut werden müsste.

Bei dem dezentralen Ansatz hingegen wirken die Kommunikationssubjekte selbst als Herausgeber der Zertifikate. Jeder Kommunikationspartner kann einer Gruppe der ihm bekannten und vertrauten Partner die öffentlichen Schlüsseln signieren. Diese haben weiterhin eigene Gruppen von Subjekten, denen sie den Schlüssel signieren. Auf diese Weise entsteht ein Web of Trust (Vertrauensnetz). Die Zertifizierungspfade sind in Vertrauensnetzen nicht beschränkt, wie das bei hierarchischen Ansatz der Fall ist. Allerdings, können die Kommunikationssubjekte selbst genau bestimmen, wem sie trauen und wem nicht. Den Vertrauensnetz-Ansatz findet man zum Beispiel bei Pretty Good Privacy (PGP)[14].

Speicherung und Verteilung von Zertifikaten

Da Zertifikate nur den öffentlichen Schlüssel des Zertifizierten enthalten und weiterhin durch den Schutz der digitalen Unterschrift der CA, vor Veränderungen geschützt sind, sind bei der Speicherung und Verteilung der Zertifikate keine zusätzliche Maßnahmen erforderlich.
Bei der Verteilung ist darauf zu achten, daß der Anfordernde das Zertifikat tatsächlich erhält. Da die Initiative vom Empfänger ausgeht, entweder durch eine Zertifikatsanforderung für sein eigenes Zertifikat oder im Rahmen eines Überprüfungsprozesses für ein Zertifikat eines Dritten, kann davon ausgegangen werden, daß der Empfänger solange nachfragen wird, bis er das Zertifikat erhalten hat. Als Mechanismus zur Verteilung können alle Verfahren verwendet werden, die auch bei der Übertragung normaler Daten Anwendung finden, z.B. E-Mail, WWW, ftp, usw.

Widerruf von Zertifikaten

Zertifikate werden widerrufen falls ihre Gültigkeitsdauer abgelaufen ist, oder falls der Verdacht besteht, daß der private Schlüssel des Subjektes kompromittiert wurde. Anders als bei symmetrischen Schlüssel, dessen Besitzer bekannt und in ihrer Anzahl beschränkt sind, ist es bei Zertifikaten nicht möglich sie bei jedem einzelnen Besitzer zu widerrufen. Zertifikate werden nur bei den Zertifizierungsstellen für ungültig erklärt bzw. widerrufen. Die CA vereinbart dazu mit dem Zertifizierten ein Paßwort, mit Hilfe dessen der Zertifizierte seinen öffentlichen Schlüssel zurücknehmen kann. Jede CA führt eine Certificate Revocation List (CLR), in der Seriennummer und Zeitpunkt des Rückrufs aller widerrufenen Zertifikate enthalten sind. Die CLR wird von der CA signiert um sie vor Veränderungen bzw. Manipulationen zu schützen. Sie wird in regelmäßigen Abständen von der CA veröffentlicht und jedes Kommunikationssubjekt kann die Gültigkeit der von ihm verwendeten und akzeptierten Zertifikate überprüfen. Andernfalls sind sie nicht in der Lage, eventuelle Angriffe über ungültige Zertifikate zu erkennen. Nach ihrem Widerruf werden Zertifikate nicht gelöscht sonder aus rechtlichen Gründen archiviert.

4.5 Sicherheit in der Kommunikation

Das World Wide Web (WWW) ist das größte Kommunikationsmedium heutzutage. Über die Bedürfnisse, die Sicherheit der Informationen zu sichern, wurde ausführlich in dem letzten

[14]PGP ist ein Verfahren für authentische und vertrauliche E-Mail Kommunikation.

Abschnitt geschrieben. Es gibt verschiedene Ansätze um die Sicherheit in einem öffentlichen Netz wie dem WWW zu unterstützen. Diese Ansätze unterscheiden sich aber in dem Bereich ihrer Anwendung. So wird IPSec beispielsweise in der Vermittlungsschicht des ISO/OSI Referenzmodells[15] eingesetzt. SSL/TSL bildet sozusagen eine eigenen Transport und Sicherheitsschicht, während S/MIME in der Anwendungsschicht angesiedelt ist.
Da diese drei Mechanismen in Betracht gezogen worden sind, um die Ansätze für die Realisierung der sicheren Kommunikation bei SeMoA zu analysieren, werden sie in diesem Abschnitt vorgestellt.

4.5.1 IPSec

IP Security (IPSec)[**?**][**?**] umfaßt eine Menge an Protokollen des *IETF (Internet Engineering Task Force)*[16], um sichere Übertragung von Paketen in der Vermittlungsschicht (OSI-Modell) zu unterstützen. IPSec kam Mitte der Neunziger als Antwort auf Sicherheitsbedürfnisse auf, die das IAB damals in einem Report mit dem Namen *Security in Internet Architecture* beschrieben hat. IPSec wurde mit dem Ziel konstruiert das neue Internetprotokoll Internet Protocol Version 6 (IPv6), um Sicherheitsfunktionalität zu erweitern. Die Konzepte sind aber auch auf das heute noch gültiges IPv4 anwendbar.

Die Aufgabe des IPSec ist die Sicherung der Netzwerkkommunikation auf Paketebene. Der Standard basiert auf verschiedenen Request for Comments (RFC)[17], die folgendes definieren:

- IPSec Protokolleigenschaften, definieren die Informationen, mit denen IP-Pakete erweitert werden, um Vertraulichkeit, Integrität und Authentisierung zu ermöglichen. Weiterhin definieren sie, wie die Daten verschlüsselt werden.
- Internet Key Exchange (IKE), das für die "Security Association" und den Schlüsselaustausch zwischen Kommunikationspartnern zuständig ist.

IPSec ermöglicht dem System, das benötigte Sicherheitsprotokoll auszuwählen, sich mit dem Kommunikationspartner über Sicherheitsmechanismen zu einigen und eventuell benötigte Schlüssel auszutauschen. Um die Sicherheitsdienste zu unterstützen definiert IPSec einige neue Paketenformate und zwar: Authentication Header (AH), um die Integrität der Daten und Encapsulating Security Payload (EPS), um die Vertraulichkeit und Integrität der Daten zu sichern. Sie können unabhängig voneinander oder zusammen benutzt werden, obwohl für die meisten Anwendungen eines von beiden ausreichend ist.

[15] ISO/OSI (International Standards Organisation/Open System Interconnection) ist ein Referenzmodell, das den Rahmen für die Definition von Standards für die Kommunikation offener Systeme beschreibt. Dieses Modell Besteht aus sieben Schichten, beginnend mit physikalischen und endend mit Anwendungsschicht.

[16] IETF ist die Organisation für die Standards im Internet. Sie ist eine große offene internationale Gemeinde von Netzwerkentwicklern, Betreibern, Herstellern und Forschern, die sich um die Entwicklung der Internetarchitektur und problemlosen Betrieb des Internets kümmert. Sie ist für jedes interessierte Individuum offen. Die IETF wird von dem IAB (Internet Architecture Board, technisches beratendes Gremium) beaufsichtigt. Quelle: www.webopedia.com

[17] RFC 2401-2411

Damit eine Verbindung Zustande kommen kann, muß erst einmal vereinbart werden, welche Sicherheitsdienste (Verschlüsselung, Integritätssicherung, Authentisierung) Verwendung finden sollen. Danach muß bestimmt werden welche Sicherheitsmechanismen bzw. Sicherheitsalgorithmen eingesetzt werden sollen. Der letzte Schritt ist die Bestimmung der Sitzungsschlüssel und dessen Austausch. Um dies effizient zu verwalten gibt es bei IPSec das Konzept, das unter dem Namen Security Association (SA) bekannt ist. Security Association ist eine Beziehung zwischen zwei oder mehreren Parteien bzw. Kommuniaktionssubjekten, die beschreibt, wie diese Parteien die Sicherheitsdienste benutzen sollten um sicher zu kommunizieren. SA wird zu diesem Zweck von einer Menge verschiedener Parameter um die Kommunikation herum und einer Datenbank, die sicherheitsbezogene Einträge über die Kommunikationspartner enthält, unterstützt.

In seiner Implementierung nimmt IPSec an, daß eine Realisierung des Security Association vorliegt. IPSec definiert keine Mechanismen um eine SA zu realisieren. Aus diesem Grund hat sich IETF entschieden den Prozeß der sicheren Kommunikation in zwei Bereiche einzuteilen. Die Aufgabe der IPSec ist dabei die Vermittlung der Pakete und das Internet Key Management Protocol (IKMP) kümmert sich um Aushandeln der für Verbindung benötigten SA. Als Standard für die Konfigurierung der SA wurde der Internet Key Exchange (IKE) ausgewählt. IKE verlangt eine gegenseitige Authentisierung der Kommunikationspartner und den Austausch des geheimen Schlüssels, der für den Prozeß des Aushandeln des SA benötigt wird. Der Authentisierungprozeß kann dabei mit Hilfe von Verfahren die auf symmetrischen Schlüsseln, asymmetrischen Schlüsseln oder digitaler Signatur basieren, durchgeführt werden. Der geheime Schlüssel wird mit Hilfe von Diffie-Hellman ausgetauscht. Nachdem diese zwei Schritte erfolgt sind, kommt es über eine sichere Verbindung zum eigentlichen Aushandeln des SA. Die beide Seiten einigen sich über die Algorithmen die benutzt werden und den gemeinsamen geheimen Schlüssel, der zur Verschlüsselung in IPSec benutzt wird.

In Bezug auf mögliche Verbindungsarten unterstützen beide Paketenformate AH und EPS, zwei verschiedene Operationsmodi - den Transport-Modus und den Tunnel-Modus:

Transport-Modus bietet Schutz primär für Protokolle der Schichten oberhalb der Vermittlungsschicht (TCP, UDP, ICMP). In dem Transport-Modus wird durch AH und ESP der Nutzdatenteil (Payload) eines IP-Pakets geschützt. Der Header bleibt ungeschützt.

Tunnel-Modus bietet hingegen den Schutz für das gesamte IP-Paket. Nach dem die AH und ESP-Header dem eigentlichen IP-Paket hinzugefügt wurden, wird das gesamte Paket geschützt in den Nutzdatenteil eines anderen Paketen verpackt. Auf diese Weise wird das neue Paket durch einen "Tunnel" von einem zu dem anderen Punkt des Netzwerks transportiert.

Die Voraussetzung für die beiden Modi sind die Systeme, die IPSec auf beiden Seiten, Senderwie auch Empfängerseite unterstützen. Welcher von den beiden Modi verwendet wird, hängt von den Bedürfnissen der Kommunikation ab. Der Tunnel-Modus wird allerdings eher verwendet, dank dem verbreitetem Konzept des Virtual Private Network (VPN)[18] (Virtual Pri-

[18]Ein Virtual Private Network stellt ein System bzw. ein Netzwerk dar, das öffentliche Kommunikationsmedien wie z.B. das Internet benutzt, um zwei Kommunikationspunkte zu verbinden. Solche Systeme benutzen

vate Network) und der Tatsache, daß nur Eingangs- und Ausgangspunkte eines Netzwerkes IPSec-konform sein müssen.

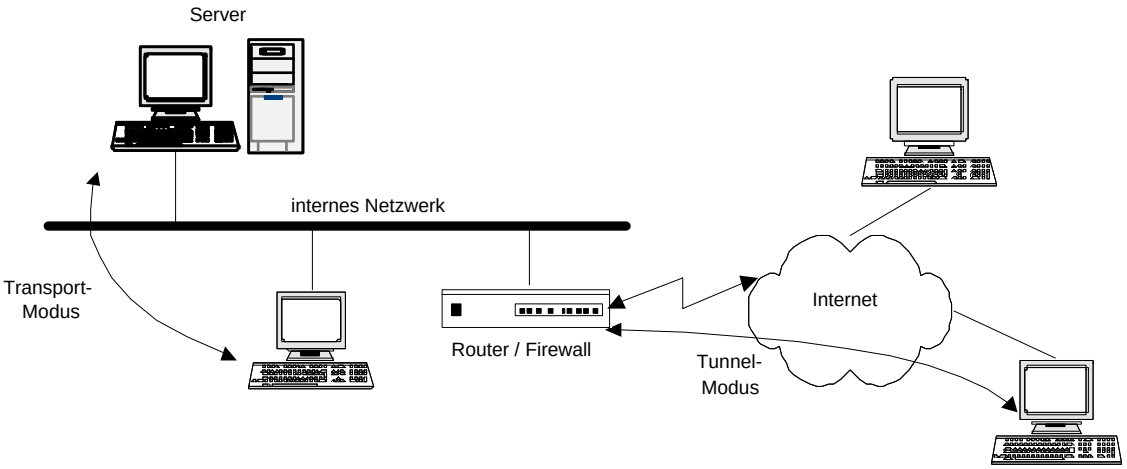

Abbildung 4.4: Ein Beispiel der IPSec-Anwendung

Authentication Header

Der Authentication Header unterstützt die Datenintegrität und Authentisierung der IP-Pakete. Durch Sicherung der Datenintegrität soll es möglich sein, jede Veränderung an dem Inhalt eines Pakets zu erkennen. Auf der anderen Seite, ermöglicht der Authentisierungsmechanismus, daß die Kommunikation nur zwischen den authorisierten Subjekten (Benutzer, Anwendungen) stattfindet.
Der Aufbau des AH ähnelt der Aufbau des Headers eines IP-Pakets. Der vordere Teil des AH besteht aus verschiedenen Feldern fester Länge, in denen sich verschiedene Parameter, wie Typ des nächsten AH, Länge des Nutzdatenteils, Sequenznummer, usw. befinden. Der hintere Teil ist für einen Nutzdatenteil variabler Länge reserviert, in dem sich der Prüfwert des Pakets befindet.
Die Authentisierung basiert auf der Verwendung von Prüfwerten in Form von MACs. Der MAC-Prüfwert kann über verschiedene Informationen berechnet werden - über dem IP-Header, Nutzdatenteil oder den AH selbst. Für die Erzeugung des Prüfwertes werden erst einmal die MD5 und SHA-1 Hash-Algorithmen verwendet, um den Hashwert zu berechnen und danach das MAC mit Hilfe des HMAC-Algorithmus erzeugt. Der MAC-Wert wird bei jedem Empfang eines Pakets geprüft. Bei Unstimmigkeiten wird das Paket verworfen. Die Authentisierung und dementsprechend die Anwendung des MACs unterscheidet sich in Abhängigkeit davon, welcher der Operationsmodi (Transport- oder Tunnel-Modus) und welcher Typ der Pakete (IPv4 oder IPv6) benutzt wird.
Um die Kommunikation gegen Widereinspielungsangriffe zu schützen, benutzt IPSec das im AH eingebaute Sequenznummer-Feld. Die Sequenznummer ist eine Art Kontrollnummer, die

Verschlüsselung und andere Sicherheitsmechanismen, um die Daten vor Angriffen zu schützen und zu sichern, daß nur authorisierte Personen den Zugang zum Netzwerk und Daten erlangen. Quelle: www.webopedia.com.

in jedem Paket enthalten ist. Sie startet beim Verbindungsaufbau bei 0 und wird bei jedem Versenden inkrementiert. Jede Seite führt eine eigene Liste der für diese Verbindung gültigen Sequenznummern, in der der Empfang jeder gültigen Sequenznummer entsprechend quittiert wird. Auf dieser Weise ist das Einspielen von Paketen mit schon empfangenen Sequenznummern ausgeschloßen.

Encapsulating Security Payload

Encapsulating Security Payload stellt ein Pakettyp dar, der für Vertraulichkeit der Daten zuständig ist. ESP verschlüsselt an erster Stelle den Paketeninhalt, optional kann er auch Pakete mit denselben Mechanismen wie AH authentisieren.
Der ESP-Header besteht aus mehreren Feldern fester Länge, die ähnliche Daten und Mechanismen wie der AH enthalten. Der Nutzdatenteil hat eine variable Länge die, falls die Verschlüsselungsmechanismen das verlangen, mit Füllbits zur benötigten Länge aufgefüllt wird. Der Teil mit Authentisierungsdaten ist optional und wird nur in dem Fall gebraucht, wenn die Authentisierung Verwendung findet.
Als Verschlüsselungsmechanismen können die meisten der gängigen Verschlüsselungsarten gewählt werden: DES, 3DES, IDEA, Blowfish, usw.. Die Wahl des Verfahrens hängt von der ausgehandelten Aushandeln des Security Association ab, der kleinste gemeinsame Nenner bei der Verschlüsselung ist jedoch DES.
Die Anwendung des ESP unterscheidet sich in Abhängigkeit davon, welcher Modus der Kommunikation gewählt wird. Bei dem Transport-Modus werden die Daten zuerst verschlüsselt in ESP verpackt und dann in einem IP-Paket verpackt zu ihrem Empfänger verschickt. Bei dem Tunnel-Modus hingegen, wird das komplette Paket in ESP verpackt und wiederum in ein neues IP-Paket, das zu einem vertrautem Host verschickt wird. Der Host entpackt das orginale Paket und routet es an den eigentlichen Empfänger weiter. Somit eignet sich Transport-Modus zum Schützen der Verbindungen zwischen Hosts und der Tunnel-Modus für Verbindungen zwischen vertrauten Netzwerken bzw. Firewalls oder anderen Sicherheitsgateways, die sie von der Außenwelt schützen.

4.5.2 SSL/TLS

Secure Socket Layer (SSL)[**?**][**?**][**?**][**?**] wurde von der *Netscape Communications Corporation* entwickelt, um sichere Kommunikation im Internet zu ermöglichen. SSL hat seit seinem ersten Erscheinen im Jahre 1994 mehrere Veränderungen erlebt, die zuletzt zur der SSL Version 3 (SSLv3) geführt haben. Die Weiterentwicklung des SSL wurde anschließend von *IETF* übernommen, was Ende der 90er die Erklärung des SSL zum Internet-Standard unter dem Namen Transport Layer Security (TLS)[**?**][19] zur Folge hatte. TLS Version 1.0 basiert auf SSLv3 und kann wegen den wenigen Unterschieden in seiner Definition im Prinzip als SSLv3.1 betrachtet werden. In dem nächsten Abschnitt wird wegen der breiteren Anwendung und Unterstützung in der Praxis auf die Beschreibung der SSLv3 eingegangen.

[19] RFC 2246

Aufbau von SSL

SSL ist ein Protokoll, daß die Authentisierung der Kommunikationspartner, sowie Vertraulichkeit und Integrität der Daten unterstützt. Es wurde entwickelt um die TCP/IP-Kommunikation zwischen einem Client und Server gegen Angriffe abzusichern. Aus diesem Grund, liegt das SSL-Protokoll unterhalb der Anwendungsschicht und oberhalb der TCP-Ebene in der Transportschicht (vgl. Abb. **??**).

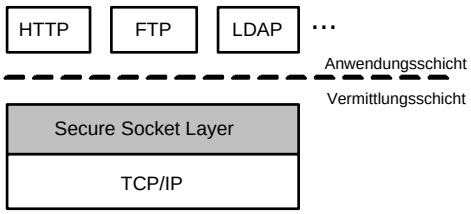

Abbildung 4.5: Position des SSL-Protokolls

SSL benutzt TCP/IP im Auftrag der Protokolle der höheren Ebenen (HTTP, FTP, LDAP, usw.), um ihnen Sicherheit zu bieten. Es ermöglicht dabei die Authentisierung des SSL-fähigen Server dem SSL-Client gegenüber, falls erwünscht auch des Client dem Server gegenüber und anschließend den Aufbau einer verschlüsselten Verbindung zwischen den beiden. Zum Beispiel wird eine SSL-Verbindung zwischen dem Rechner eines Online-Händlers (Server) und dem Browser (Client) eines Käufers hergestellt, um die Zahlung von den vom Käufer ausgesuchten Produkten zu tätigen. Da es sich an dieser Stelle um empfindliche Informationen (Name, Anschrift, Kreditkartennummer des Käufers) handelt, eignet sich SSL besonderes gut, um diese Daten vor Angriffen zu schützen. SSL verdankt heutzutage seinen Grad der Verbreitung, Popularität und Akzeptanz gerade solchen E-Business Anwendungen im Internet.

Bei SSL werden zwei wichtige Konzepte unterschieden: Sitzungen (Sessions) und Verbindungen (Connections). Eine Verbindung stellt eine "peer-to-peer"-Verbindung im Netzwerk dar. Jede Verbindung ist einer Sitzung zugeordnet. Eine Sitzung hingegen ist eine Zuordnung zwischen dem Client und dem Server, die bei dem Aufbau einer Verbindung erzeugt wird. Eine Sitzung definiert eine bestimmte Menge an kryptographischen- bzw. Sicherheitsparametern, die für verschiedene Verbindungen gleichzeitig verwendet werden können. Das Konzept der Sitzung wird benutzt, um das "teure" Aushandeln der Sicherheitsparameter für jede Verbindung zu vermeiden. Zwischen zwei Kommunikationspartnern sind gleichzeitig mehrere Verbindungen möglich, während in der Praxis meistens nur eine Sitzung zwischen den beiden Parteien aufgebaut wird.

SSL besteht aus zwei Teilprotokollen: SSL Handshake Protocol und SSL Record Protocol. Das Record Protocol definiert das Format, das benutzt wird um Daten auszutauschen. Auf der anderen Seite wird das Handshake Protocol benutzt, um eine Fülle von Nachrichten zwischen dem Server und Client auszutauschen, die bei dem Aufbau einer Verbindung benötigt werden.

SSL Handshake Protocol

Damit eine Verbindung zwischen zwei Kommunikationspartner zustandekommt, ist es an erster Stelle nötig, einige Parameter auszutauschen um die Verbindungseinstellungen auf beiden Seiten koordinieren zu können. Dies erfolgt durch den Austausch der Nachrichten am Anfang des Aufbaus einer SSL-Sitzung. Dieser Prozeß wird SSL-Handshake genannt. Der SSL-Handshake wird durch das SSL-Handshake Protocol eingeleitet und durchgeführt. Dieser Prozeß ermöglicht dem Client und dem Server, sich gegenseitig zu authentisieren, einige Parameter, die für die Generierung des Schlüssels für die symmetrische Verschlüsselung benötigt werden, auszutauschen und das Aushandeln der für Verbindung benötigten Sicherheitsmechanismen (Algorithmus zum Schlüsselaustausch, Verschlüsselungsverfahren und Verfahren zur Prüfung der Datenintegrität).

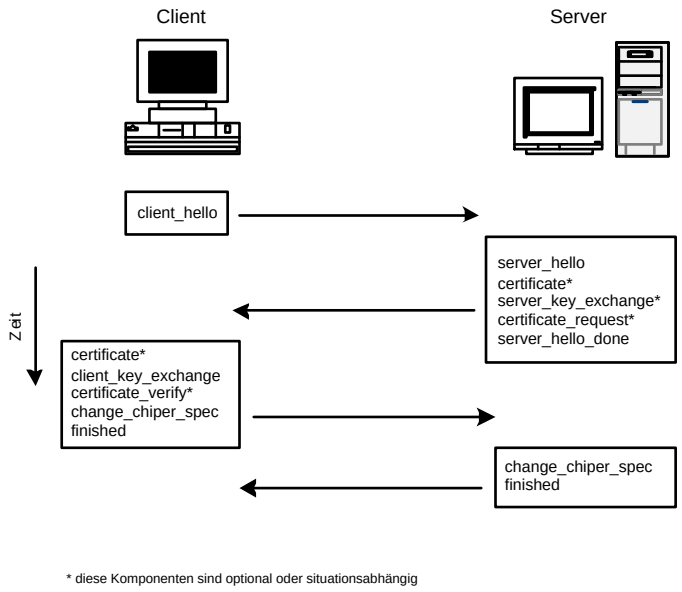

Abbildung 4.6: SSL Handshake Protokoll

Der SSL-Handshake erfolgt in Schritten wie auf der Abbildung **??** vereinfacht dargestellt und kann wie folgt beschrieben werden:

1. Aushandeln der Kommunikationsparameter
   Durch das Senden einer `client_hello` Nachricht initiiert der Client die Verbindung zu dem Server. Diese Nachricht muß der Server mit `server_hello` beantworten, ansonsten kommt keine Verbindung zustande. Die `hello` Nachrichten dienen dazu, die Versionsnummer des SSL und Kombination von möglichen Sicherheitsmechanismen, abzusprechen. Dazu schlägt der Client in seiner `hello` Nachricht eine Liste der Kombinationen der Sicherheitsmechanismen vor, die er unterstützt. Der Server wählt eine aus, und vermerkt es auch entsprechend in seinem `hello`. Außerdem werden generierte Zufallszahlen, die später zur Erzeugung des Kommunikationsschlüssel benötigt werden, und andere für die Verbindung benötigte Daten, ausgetauscht.

2. Authentisierung des Servers und Schlüsselaustausch
   In dem Fall, daß der Server authentisiert werden muß, sendet er sein Zertifikat in einer `certificate` Nachricht an den Client. Falls der Server keine Zertifikat besitzt oder das Zertifikat nur für Signierungszwecke benutzt wird (wie bei DSS) sendet der Server seinen öffentlichen Schlüssel mit der `server_key_exchange` Nachricht. Die `certificate_request` Nachricht wird von dem Server in dem Fall benutzt, wenn eine Authentisierung des Clients gefordert wird. Zum Abschluß signalisiert der Server durch eine `server_hello_done` Nachricht, das Ende der Hello-Phase.

3. Authentisierung des Client und Schlüsselaustausch
   Nach dem Empfang der `server_done` Nachricht prüft der Client das vom Server gesendete Zertifikat und die Gültigkeit anderer mitgegebenen Parameter. Falls der Server das Zertifikat des Client angefordert hat, muß der Client mit einer `certificate` Nachricht oder einer negativen Nachricht beantworten. Als nächstes wird eine sog. `client_key_exchange` Nachricht verschickt, deren Inhalt von dem ausgehandelten Algorithmus für Schlüsselaustausch abhängt. Der Client erzeugt in diesem Fall ein 48-Bit langes Datum, das sog. `pre_master_secret` und benutzt die Nachricht `client_key_exchange`, um das erzeugte `pre_master_secret` zum Server zu übertragen oder die Parameter zu ihrer Erzeugung auszutauschen. Im ersten Fall wird sie mit dem öffentlichem Schlüssel des Servers verschlüsselt und übertragen. Falls der Server die Authentisierung des Client angefordert hat und der Client im Besitz eines Zertifikats ist, das die digitale Signatur ermöglicht, schickt der Client eine sog. `certificate_verify` Nachricht, um die explizite Verifizierung seines Zertifikats zu ermöglichen. Zu diesem Zweck signiert der Client eine spezielle Nachricht und überträgt sie an den Server, um den Besitz des privaten Schlüssel zu beweisen. Nachdem der Client erfolgreich überprüft worden ist, benutzen die beiden Seiten das vorher erzeugte `pre_master_secrets` um das `master_secret` zu generieren. Mit Hilfe der in der Hello-Phase ausgetauschten Zufallszahlen, des `master_secret` und bekannter Hashalgorithmen erzeugt jede Seite den Sitzungsschlüssel bzw. einen symmetrischen Schlüssel in der benötigten Länge.

4. Endphase
   In der Endphase des Handshake, sobald der Sitzungsschlüssel vorhanden ist, verschickt der Client eine `change_chiper_spec` Nachricht, um seine Bereitschaft zur Benutzung der für die Verschlüsselung und Prüfung der Datenintegrität ausgehandelten Algorithmen zu signalisieren. Der Client quittiert sofort danach das Ende der Handshake Phase mit einer `finished` Nachricht. Als Antwort darauf schickt der Server seine `change_chiper_spec` und `finished` Nachrichten und beendet somit das Handshake. Die SSL-Sitzung ist jetzt gestartet und Client und Server können anfangen Daten austauschen.

Falls Client und Server eine vorausgegangene Sitzung wiederaufnehmen oder eine bestehende Sitzung duplizieren wollen, kann in diesem Fall ein verkürztes Handshake benutzt werden. Der Client sendet dabei eine `client_hello` Nachricht mit der Identifizierungsnummer (`session_ID`) der Sitzung, die wiederaufgenommen werden soll. Der Server überprüft daraufhin den eigenen Cache, ob die Parameter für diese Sitzung noch vorhanden sind. Falls er die

Parameter mit der zugehörigen `session_ID` findet, antwortet er mit dem `server_hello` und der `session_ID`. Anschließend werden die `change_chiper_spec` und `finished` Nachrichten ausgetauscht und die Datenübertragung kann beginnen.

SSL Record Protocol

Das SSL Record Protocol definiert das Datenformat, das benutzt wird um Informationen bei SSL zu übertragen, unabhängig davon, ob es sich um Daten des Handshake Protocols handelt, oder die der eigentlichen Nachricht. Der SSL Record besteht aus zwei Teilen - dem Header und dem Nutzdatenfeld (payload). Der Header enthält die Informationen über das zu übertragende Datenpaket - SSL-Versionsnummer, Typ des Dateninhaltes und die Länge des Pakets. In dem Nutzdatenfeld werden die Datenfragmente der Nachrichten, die zu den Protokollen der höheren Schicht gehören (z.B. http), übertragen. Neben diesen Daten werden hier auch die Daten des SSL Handshake Protocol und andere Kontrollinformationen übertragen. Das gemeinsame für alle zu übertragenden Informationen ist eine Fragmentation in Blöcke einer bestimmten Länge (16384 Bytes oder weniger). Falls erwünscht werden diese Blöcke auch durch bestimmte Algorithmen komprimiert. Als nächstes folgt die Berechnung des MAC über die Datenblöcke mit Hilfe von dem im Handshake gewonnenen symmetrischen Schlüssel und bekannten Hashalgorithmen. In die Berechnung der MAC geht auch die Sequenznummer des Datenblocks ein, die genauso wie beim IPSec benutzt wird, um die Datenpakete vor Widereinspielung zu schützen. MAC wird bei den SSL-Nachrichten benutzt, um die Integrität der Daten prüfen zu können. Der Datenblock zusammen mit dem MAC wird anschließend mit im Handshake ausgehandelten symmetrischen Verschlüsselungsverfahren verschlüsselt, zusammen mit dem Header zu einem SSL Record hinzugefügt und an den Kommunikationspartner übertragen.

Sicherheit im SSL

Wie aus der vorherigen Beschreibung ersichtlich stellt SSL ein hybrides Kryptoverfahren dar, bei dem das Public-Key-Verfahren benutzt wird, um die für die Erstellung der symmetrischen Schlüssel benötigten Informationen zu übertragen und die Daten anschließend mit dem erstellten Schlüssel und Symmetrischen Kryptoverfahren zu verschlüsseln und übertragen. SSL ermöglicht weiterhin die gegenseitige Authentisierung der Kommunikationspartner und die Prüfung der Integrität der zu übertragenden Daten. Um diese Aspekte der Sicherheit effektiv zu realisieren, unterstützt SSL eine Fülle an verschiedenen kryptographischen Algorithmen und Verfahren. Diese Algorithmen werden zusammengefaßt zu sog. Cipher Suite Kombinationen. Eine Cipher Suite besteht immer aus einem Algorithmus für den Schlüsselaustauch (Diffie-Hellman und RSA), dem Algorithmus für die symmetrische Verschlüsselung (DES, RC2, RC4 und 3DES) und einem Hashverfahren (SHA-1 und MD5). Diese Cipher Suites werden von der SSL-Spezifikation selbst vorgeschrieben. Ihre Unterstützung ist den Kommunikationspartnern überlassen. Bei dem Handshake wird (falls nicht anderes vorgegeben) versucht die stärkste Cipher Suite auszuhandeln.

Die Sicherheit der SSL-Kommunikation hängt in dem großen Maße davon ab, was für eine Kommunikationsart gewählt worden ist. SSL ermöglicht durch die Cipher Suitses die Kom-

munikation ohne Authentisierung oder ohne Verschlüsselung der Daten. In diesem Fall kann nicht nicht mehr von sicherer Kommunikation gesprochen werden, da sie potentiellen Angriffen ausgesetzt ist. Weiterhin hängt die Sicherheit der Kommunikation von den verwendenden Kryptoverfahren bzw. der dabei verwendeten Schlüssellängen ab. Die von SSL unterstützte Kryptoverfahren verwenden bei symmetrischen Verfahren Schlüssellängen ab 40 Bit und bei asymmetrischen Schlüssellängen ab 512 Bit. Die heute ziemlich populäre und verbreitete Nutzung der Schlüssellängen von 128 Bit für symmetrische Verschlüsselung kann als sicher[20] betrachtet werden.

Die relativ einfache Konfigurierbarkeit und Anwendungsfreundlichkeit des SSL und der hohe Grad an Sicherheit machen SSL besonderes im Internet zum defacto Standard für sichere Kommunikation. Dies garantiert eine breitgefächerte und gute Unterstützung, wie auch Weiterentwicklung des SSL-Mechanismus und sichern sein Fortbestand in der Zukunft.

4.5.3 S/MIME

Secure Multipurpose Internet Mail Extension (S/MIME)[**?**][**?**] ist eine Spezifikation für sichere elektronische Post. S/MIME wurde 1995 von einem privatem Konsortium von Herstellern mit dem Ziel entwickelt, die Email-Kommunikation vor Angriffen zu schützen. Die Anforderung an die Entwickler war es, eine Spezifikation zu entwerfen, die leicht in bestehende Email- und Kommunikationsprodukte zu integrieren ist. S/MIME baut aus diesem Grund auf der MIME-Spezifikation[21], PKI und der asymmetrischen Kryptotechnologie auf. Die letzte Version von S/MIME-Spezifikation, S/MIME v3 wurde von der *IETF S/MIME Working Group* Ende der 90er Jahre verabschiedet, was S/MIME zum defacto Standard für sichere elektronische Post erklärte.

S/MIME definiert die Möglichkeiten, Nachrichten vor Angriffen abzusichern. Folgende Möglichkeiten sind dabei realisierbar:

1. *Verschlüsselung der Daten* - entweder der Nachrichteninhalt oder die Schlüssel werden verschlüsselt

2. *digitale Signatur* - der Nachrichteninhalt wird unterschrieben, die Unterschrift dann mit dem privaten Schlüssel des Besitzers verschlüsselt, und die ganze Nachricht mittels Base64-Kodierung[22] kodiert. Sie kann nur von einem Empfänger mit S/MIME-Funktionalität gelesen und verifiziert werden.

[20] Nach Schätzungen von *Silverman*[**?**] von der RSA-Laboratories würde man für das Brechen eines symmetrischen Schlüssels von 128 Bit und einem entsprechend sicheren asymmetrischen Schlüssel von 1620 Bit Länge mit Hilfe der kryptanalytischen Verfahren ungefähr 10^{16} Jahren brauchen.

[21] Multipurpose Multimedia Internet Mail Extension (MIME) stellt eine Spezifikation zum Formatieren von nicht ASCII-Nachrichten dar. MIME ermöglicht Email-Clients das Versenden und Empfangen von Grafiken, Audio- und Video-Dateien und anderen Datentypen. Das Versenden von selbst definierten Datentypen ist auch möglich. Quelle: www.webopedia.com

[22] Die Base64-Kodierung stellt ein Kodierungsverfahren dar, das 6-bit-lange Textblöcke in 8-bit-lange konvertiert, die wiederum printbare ASCII-Zeichen darstellen.

3. *digital signierte und unverschlüsselte Daten* - die Nachricht wird wie in Punkt 2 behandelt, mit dem Unterschied, daß nur die digitale Signatur base64-kodiert wird. Die Nachricht ist auf dieser Weise auch für Empfänger lesbar (nicht aber verifizierbar), die keine S/MIME-Funktionalität besitzen.

4. *digital signierte und verschlüsselte Daten* - hier sind verschiedene Kombinationen möglich, da diese Möglichkeit die Verschachtelungen der Verfahren ermöglicht, so daß verschlüsselte Daten signiert werden können und signierte Daten wie unter den Punkten 2 und 3 verschlüsselt werden können.

Ähnlich wie SSL, verwendet S/MIME eine hybride Verschlüsselungstechnologie. Der Nachrichteninhalt wird mit einem Sitzungsschlüssel und symmetrischen Verschlüsselungsverfahren verschlüsselt, anschließend wird der Sitzungsschlüssel mit dem öffentlichem, asymmetrischen Schlüssel des Nachrichtenempfängers verschlüsselt. Wie oben erwähnt, kann die Nachricht verschlüsselt oder signiert werden oder gleichzeitig verschlüsselt und signiert werden. Die verwendbaren Algorithmen können der Tabelle **??** entnommen werden . Tabelle **??** beschreibt dabei die Mindestanforderungen und Empfehlungen seitens des Senders und Empfängers der Nachricht bei der Benutzung der Kryptoverfahren und -algorithmen. Die genaue Spezifikationen des Nachrichtenaufbaus, und der zu verwendenden Verschlüsselungsverfahren finden sich in RFC 2633[23].

Verfahren	Anforderungen
Hashwert für digitale Signatur	Es muß SHA-1 und MD5 unterstützt werden. Es sollte SHA-1 verwendet werden.
Verschlüsselung des Hashwertes für digitale Signatur	Sender und Empfänger müssen DSS unterstützen. Der Sender sollte RSA unterstützen. Der Empfänger sollte die Verifikation der RSA-Signaturen unterstützen.
Verschlüsselung des Sitzungsschlüssels	Sender und Empfänger müssen Diffie-Hellman unterstützen. Der Sender sollte RSA-Verschlüsselung unterstützen. Der Empfänger sollte RSA-Entschlüsselung unterstützen.
Verschlüsselung der Nachricht	Der Sender sollte Verschlüsselung mit 3DES und RC2/40 unterstützen. Der Empfänger muß die Entschlüsselung mit 3DES unterstützen und sollte die Entschlüsselung mit RC2/40 unterstützen.

Tabelle 4.2: Kryptographische Algorithmen in S/MIME

[23] S/MIME Version 3 Message Specification (RFC 2633), URL: ftp://ftp.ietf.org/rfc/rfc2633.txt

Die S/MIME-Spezifikation beschreibt auch die Prozedur, die benötigt wird, um die Entscheidung über die Verwendung der Kryptoalgorithmen zu treffen. Da die Email-Kommunikation eine indirekte Kommunikationsart darstellt, können im Unterschied zu IPSec und SSL/TSL die zu verwendenden Kommunikationsparameter bzw. Algorithmen und Verfahren nicht direkt ausgehandelt werden. Die Entscheidung darüber wird dem Sender der Nachricht überlassen. Im Prinzip muß der Sender zwei Entscheidungen treffen. Erstens muß der Sender feststellen ob der Empfänger in der Lage ist, mit Hilfe eines bestimmten Algorithmus, die Nachrichten zu entschlüsseln. Dabei liegt es im Vordergrund, das möglichst stärkere Verschlüsselungsverfahren zu benutzen. Zweitens, muß der Sender entscheiden, ob das Senden der Nachrichten erfolgt, falls der Empfänger nur in der Lage ist, eine schwache Verschlüsselungsverfahren zu benutzen. Um diesen Prozeß zu unterstützen, kann der Sender seine Verschlüsselungspräferenzen dem Empfänger mitteilen, die der Empfänger zur zukünftiger Benutzung speichern kann.

S/MIME benutzt zum Versenden der Nachrichten die MIME-Spezifikation. Zu diesem Zweck definiert S/MIME einige neue MIME-Typen. Die Namen dieser Typen fangen mit "pkcs" an und beziehen sich dabei auf Public-Key-Kryptospezifikationen (Public-Key Cryptography Standards, PKCS) die von *RSA Laboratories*[**?**] herausgegeben worden sind. Die PKCS#7 (Cryptographic Message Syntax, CMS) Spezifikation beschreibt dabei, wie die verschlüsselten und/oder signierten Daten innerhalb der Nachricht zu behandeln sind, während PKCS#10 für den Austausch der Zertifikate für benötigte öffentliche Schlüssel, zuständig ist. Unabhängig davon, ob die Nachricht nur verschlüsselt oder signiert wird, oder beide Sicherheitsverfahren angewendet werden, werden die Informationen nach der Behandlung (Verschlüsselung, Signieren) zusammen mit einigen sicherheitsrelevanten Informationen, wie dem Typ der angewendeten Algorithmen und Zertifikate, zu einem sog. PKCS-Objekt zusammengefügt. Dieses PKCS-Objekt wird dann wie regulärer Nachrichteninhalt behandelt und als ein MIME-Teil mit Base64-Kodierung in der Nachricht transportiert. Der Empfänger der Nachricht macht den umgekehrten Prozeß, um die Nachricht zu entschlüsseln und die Signatur zu prüfen.

Um den öffentlichen Schlüssel anderen für Verschlüsselung und Verifikation der Signatur verfügbar zu machen, werden verschiedene Verfahren eingeschlagen: zum einen kann dies durch Anfrage bei Zertifizierungstellen geschehen, zum anderen wird der öffentliche Schlüssel automatisch den signierten Nachrichten als Zertifikat angefügt. S/MIME benutzt X.509v3-Zertifikat, um die öffentliche Schlüssel auszutauschen. Das Zertifikatsmanagement stellt dabei eine Struktur dar, die zwischen der strikten hierarchischen und der "Web of Trust"-Struktur liegt. Jedes Kommunikationssubjekt ist selbst für das Warten einer eigenen Liste mit vertrauten Schlüsseln und widerrufenen Zertifikaten verantwortlich. Auf der anderen Seite, werden Zertifikate nur von Zertifizierungsstellen unterschrieben.

Teil II

Realisierung eines sicheren Kommunikationservices

Kapitel 5

Konzept

5.1 Grundüberlegungen zur Architektur

Bevor man sich mit der Konzipierung eines sicheren Kommunikationsdienstes befaßt, müssen die Grundzüge der Kommunikation in SeMoA definiert werden. Dies betrifft den Entwurf eines Dienstes im Bezug auf seine Implementierung und Anwendung. Aus diesem Grund müssen Antworten auf einige Fragen über die Eigenschaften dieses Dienstes gegeben werden:

- Wer soll und darf kommunizieren?
- Welche Kommunikationsformen werden realisiert?
- Wie wird der Kommunikationsmechanismus aufgebaut?
- Welche Sicherheitsmechanismen eignen sich für die Kommunikation?
- Sicherer Kanal zwischen Plattformen oder Sicherheitsfilter-Pipeline?
- Wie wird das Nachrichtenmanagement realisiert?
- Wie werden Fehler behandelt?
- Welche Kommunikationparameter werden benutzt?
- Eignet sich der realisierte Kommunikationsdienst zur Kommunikation zwischen Agenten?

Diese Fragen haben die Entwickler von SeMoA zum größten Teil durch bisherige Entwicklungsbemühungen bereits beantwortet, da ein Grundkonzept für die Kommunikation in SeMoA schon vorhanden und weitestgehend auch schon realisiert ist. Die Aufgabe dieser Diplomarbeit ist es, sichere Kommunikation zu ermöglichen. Dabei liegt der Schwerpunkt auf der Realisierung eines sicheren Kommunikationmechanismus, der auf dem vorhandenen Kommunikationsmechanismus aufbauen soll. Mögliche darüberliegende Mechanismen für die Übersetzung der Nachrichten in KQML oder FIPA-konforme Kommunikationskonstrukte, wie

auch dessen Interpretation, befindet sich außerhalb des Rahmens dieser Arbeit. Im Vordergrund der Arbeit steht die sichere Kommunikation zwischen verschiedenen Agentenplattformen, da hier im Gegensatz zu der lokalen Kommunikation innerhalb einer Plattform Sicherheitsaspekte auf un zwischen zwei verschiedenen Ausführungsumgebungen betrachtet werden müssen. Die Realisierung sollte auf eine Art erfolgen, welche Erweiterungen durch aufbauende Mechanismen, leicht und ohne Umstrukturierung des Grundmechnismus ermöglicht.

Es soll vertrauliche Kommunikation zwischen authentisierten Kommunikationspartner möglich sein, wobei auch Integrität der Daten, die übertragen werden, gesichert sein sollte. Dies ist in Anlehnung an bestehende und bewerte Sicherheitsmechanismen für die Kommunikation umzusetzen, welche eine flexible Adoption und Konfiguration ermöglichen (siehe Kapitel 4). Nach der Analyse möglicher Sicherheitsmechanismen wird die Implementierung, auf dem ausgewähltem Mechanismus basierend, beschrieben. Die implementierten Sicherheitsmechanismen sollten keinerlei Einschränkungen der Funktionalität von SeMoA zur Folge haben und ein gewisses Maß an Unabhängigkeit von den verwendeten Software- und Hardware-Ressourcen erweisen.

Die Kommunikationmechanismen sollen so realisiert werden, daß sie nicht nur Agenten zur Verfügung stehen, sondern einen universellen Charakter erweisen und den Nutzern, Agentenplatfformen und anderen Subjekten, genauso zur Verfügung stehen können. Dabei solle die Einfachheit und Transparenz bei der Nutzung im Vordergrund stehen. Außer dem Namen des Kommunikationpartners sollen die Kommunikationssubjekte keinerlei Kenntnisse über den Kommunikationsmechanismus selbst haben müssen.

Die Agenten sollen durch den Nachrichtenaustausch in ihren Aktivitäten nicht eingeschränkt werden. Weder in den Aufgaben die sie lokal durchführen noch in ihrer Mobilität. Weiterhin sollen Agenten die Möglichkeit haben, selbst über den Schutz der Nachrichten entscheiden zu können. D.h. die Entscheidung ob eine Nachricht sicher oder unsicher verschickt wird, befindet sich in Verantwortungsbereich des Agenten. Da die Absicherung der Nachrichten nur zwischen Agentenplattformen stattfindet und Nachrichten lokal unsicher behandelt werden, müssen die Agenten der Agentenplattform vertrauen.

Zuerst müssen aber die Grundrisse des sicheren Kommunikationsmechanismus in der Anlehnung an die in SeMoA bestehenden und Mechanismen dargelegt werden.

5.1.1 Bestehende Kommunikationsmechanismen in SeMoA

Auf die Kommunikationsmechanismen in SeMoA wurde teilweise bereits in dem Abschnitt **??** eingegangen. Für den Nachrichtenaustausch zwischen SeMoA-Plattformen wurde ein System entwickelt, das aus einem Nachrichtendienst für ausgehende Nachrichten (`Outbox`), einem Nachrichtendienst für eingehende Nachrichten (`Inbox`) und einem Kanal für den Nachrichtenaustausch (`MessageGateway`) besteht. Den Vorgang der Kommunikation kann der Abbildung **??** entnommen werden.

Damit die Agenten kommunizieren können muß der sog. `CommunicationContext` implementiert sein. Dieser ist ein Teil des `AgentContext` eines Agenten und definiert die Basismethoden, die der Agent zur Kommunikation benötigt. Er stellt die Schnittstelle für die Kommunikation zwischen dem Agenten (bzw. `AgentContext`) und der SeMoA-Plattform dar. Jeder `CommunicationContext` definiert außerdem einen Nachrichtenpuffer für eingehende Nachrichten (`Spooler`). Die Basismethoden der Kommunikation sind `send` und `next`. Mit `send` sendet der Agent eine Nachricht an einen bestimmten Agenten und mit `next` ruft er die nächste Nachricht aus dem `Spooler` ab. Die Nachrichten werden dem `Spooler` von der `InBox` als Pakete, dessen Syntax in ASN.1-Sprache[1] spezifiziert ist, übergeben und vom `Spooler` anschließend in die richtige Reihenfolge gebracht. In dem Fall, daß der Agent migriert und im `Spooler` noch Nachrichten vorhanden sind, wird der `Spooler` als ein Teil des `AgentContext` mitmigriert. Auf diese Weise bleiben die Nachrichten an Agenten erhalten.

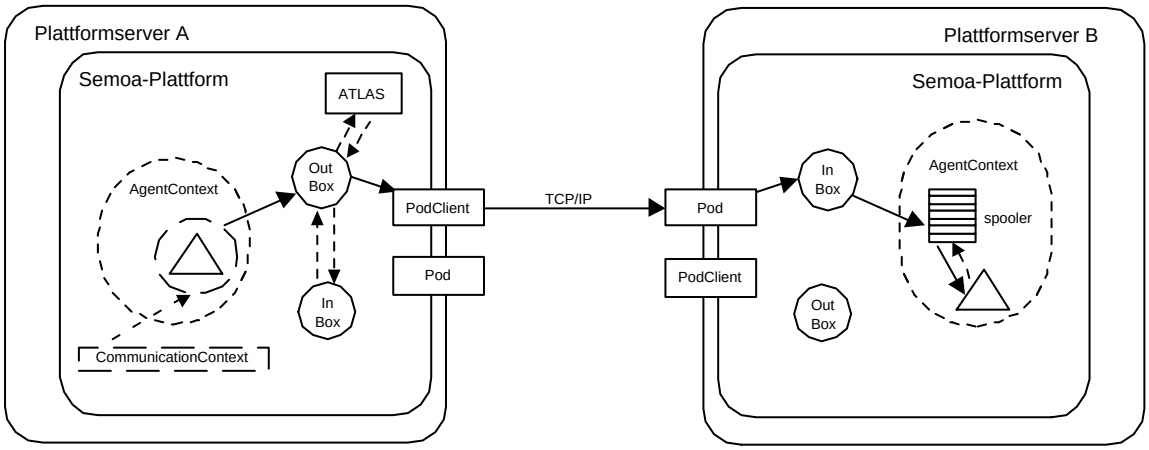

Abbildung 5.1: Beispiel der Kommunikation bei der SeMoA-Plattform

`InBox` und `OutBox` hingegen können als abstrakte Nachrichtenpuffer für die gesamte SeMoA-Plattform verstanden verden. Da sie selbst keine Nachrichten puffern sondern sie nur weiterleiten, stellen sie allerdings eher Nachrichtendienste der Plattform dar. Alle ausgehende Nachrichten landen erst einmal in der `OutBox`, alle eingehende Nachrichten werden zuerst an die `InBox` weitergeleitet, unabhängig davon, ob es sich um lokale oder globale Kommunikation handelt. Um die Nachricht an den richtigen Agenten zu verschicken ist die `OutBox` in der Lage zu prüfen, ob der empfangende Agent sich auf dem lokalen Host befindet und falls nicht, über ATLAS-Mechanismus seine Kontaktadresse ausfindig zu machen und die Nachricht an den richtigen Host weiterzuleiten. Auf der anderen Seite versucht die `InBox` nach dem Empfang einer Nachricht, den entsprechenden `CommunicationContext` des Agenten ausfindig zu machen und die Nachricht an seinen `Spooler` auszuliefern.

[1] ASN.1 (Abstract Notation One) ist ein Standard des ISO/ITU, der Art und Weise definiert, in der Daten zwischen unterschiedlichen Kommunikationsystemen übertragen werden. ASN.1 garantiert durch einheitliche Syntaxdefinition und Kodierung der Daten, daß diese unabhängig von der zu Grunde liegenden Plattform auf die gleiche Weise interpretiert werden können. Quelle: www.webopedia.com

Das MessageGateway besteht aus zwei Kommunikationskanälen, einem für den Empfang von Nachrichten (`MessageGateway.In`) und einem der für das Versenden der Nachrichten zuständig ist (`MessageGateway.Out`). Die Implementierung der beiden hängt von dem für Nachrichtentransport verwendeten Protokoll ab. Als Kommunikationsprotokoll ist Portal Daemon (Pod) realisiert. `Pod` stellt ein Protocol dar, daß nicht nur in der Lage ist die Nachrichten zu übertragen sondern auch Dienstanfragen und -antworten zwischen Agenten und Diensten erfolgreich zu übertragen. `Pod` baut auf einem System aus einem Server (`Pod`), der das `MessageGateway.In` implementiert, und einem Client (`PodClient`), der das `MessageGateway.Out` auf der Plattform realisiert, auf. Dabei enthält jede Plattform mit Kommunikationsmöglichkeit einen `Pod` (Server) und einen `PodClient`. Zwischen dem Server und Client wird für die Kommunikation eine Socket-Verbindung[2] aufgebaut, und die Daten in Form von Datenstreams (in Java `InputStream` und `OutputStream`) von einer auf die andere Plattform übertragen.

Um die Basisfunktionalität der Kommunikation zu ermöglichen, müssen `InBox`, `OutBox`, `Pod` und `PodClient` auf der SeMoA-Plattform bzw. in ihrem Environment vorhanden sein. Sie werden beim starten der SeMoA-Plattform geladen und mit einigen Parametern, wie z.B. maximale Anzahl und Größe der empfangenden Nachrichten und Nummer des Kommunikationsportes initialisiert. Die Kommunikationsdienste unterliegen genauso wie andere Dienste, der Sicherheitskontrolle der SeMoA-Plattform. Operationen, die einzelne Dienste ausführen dürfen, sind durch konfigurierbare Parameter in den Initialisierungsdateien genau definierbar.

Funktionsweise des Nachrichtenaustauschs

Der Nachrichtenaustausch funktioniert folgendermaßen: Der Agent holt sich mit der Methode `CommunicationContext.getContext()` die Referenz auf seinen `CommunicationContext`. Somit stehen die bereits erwähnten Methoden zur Kommunikation zur Verfügung. Durch den Aufruf der `send`-Methode unter Angabe der Kontaktadresse des empfangenden Agenten und optional der des sendenden Agenten sowie der eigentlichen Nachrichtendaten wird die Kommunikation initiiert. Dieser Abschnitt sieht zum Beispiel wie folgt aus:

```
CommunicationContext comm;
comm = CommunicationContext.getContext();
comm.send(to, from, message);
```

Wobei `to` die Kommunikationsadresse des empfangeneden und `from` die des sendenden Agenten darstellt. Das die Variable `message` enthält die zu übertragende Nachricht.
Beide Adressen `to` und `from` haben in allgemeinem Fall folgends Format:

```
protocol://(implicitName|nickName|spoolerName)@(host|automatic):port
```

[2] Die Socket-Verbindung stellt eine TCP/IP-Verbindung zwischen zwei Ports verschiedenen Rechnern dar.

Das Schlüsselwort `protocol` bezeichnet das für die Kommunikation verwendende Protokoll. Momentan ist nur `Pod` realisiert. Nach der Bezeichnung des Protokolls, folgt die Bezeichnung des Agenten. Hierzu kann man entweder den impliziten Namen des Agenten verwenden (`implicitName`), oder die für Menschen verständliche Bezeichnung - `nickName` oder `spoolerName`. Danach wird die SeMoA-Plattform, auf der sich der empfangende Agent befindet, durch `host` näher spezifiziert. Das Schlüsselwort `host` stellt die IP-Adresse des Servers dar. Falls diese Adresse nicht bekannt ist, kann man mit Schlüsselwort `automatic` eine Suche dieser Adresse, über dem ATLAS-Mechanismus erzwingen. Der Bezeichner `automatic` wird dann durch die gefundene Adresse ersetzt. Mit dem Schlüsselwort `port` wird der Kommunikationsport des empfangenden Server angegeben. So könnte zum Beispiel eine Kommunikationsadresse eines Agenten mit dem Namen (`nickName`) *Communication-Agent* wie folgt aussehen:

```
pod://CommunicationAgent@10.71.12.1:60001
```

Die nächste Station, nachdem die Nachricht durch die `send`-Methode abgeschickt wurde, ist die `OutBox`. Die `OutBox` ergänzt weiterhin als erstes die Adresse des Senders in dem Fall, daß diese nicht komplett angegeben wurde. Weiterhin versucht die `OutBox` die Nachricht lokal auszuliefern in dem Fall, daß sich der empfangende Agent auf der gleichen Plattform befindet. Falls der Agent nicht lokal vorliegt oder die Adresse des Empfängers das Schlüsselwort `automatic` enthält, ruft `OutBox` den ATLAS-Lokalisierungsdienst um die Kontaktadresse des Agenten zu bestimmen. Wenn die Adresse des empfangenden Agenten vorliegt, wird die Nachricht an den `PodClient` weitergeleitet. Der `PodClient` baut daraufhin eine Socketverbindung zur betreffenden SeMoA-Plattform auf, und liefert diese Nachricht an den `Pod`-Server auf der Gegenseite aus. Der `Pod`-Server leitet die Pakete dann weiter an `InBox`. Diese versucht dann den `Spooler` des empfangenden Agenten anhand seines impliziten Namen (`implicitName`), seines `nickName` oder den `spoolerName` zu bestimmen. Nachdem der richtige spooler lokalisiert ist, wird ihm das Nachrichtenpaket übergeben. Hier endet der Nachrichtenaustausch. Alles weitere wird dem empfangenden Agenten überlassen. Der Empfang der Nachricht wird dem empfangenden Agenten nicht direkt signalisiert. Um die Nachricht zu erhalten muß der Agent diese selbstständig bei seinem `spooler` anfordern. Dies kann der Agent mit der oben eingeführten Methode `next` bewerkstelligen, welche Nachrichten aus dem `Spooler` holt.

Falls an irgendeiner Stelle dieser Nacrichtenaustauschkette ein Fehler passiert, wenn z.B. der Agent nicht gefunden werden oder die Verbindung nicht aufgebaut werden kann, wird dies durch einen Fehler (`Exception`), der an den sendenden Agenten übermittelt wird, signalisiert. Hier sind SeMoA-eigene Exceptions (`CommunicationException`) oder Java-Exceptions möglich. Die Fehlerbehandlung befindet sich dadurch beim Agenten d.h. der Agent entscheidet, was in solchen Fällen unternommen wird.

5.1.2 IPSec, SSL oder S/MIME?

Um die geforderten Sicherheitsanforderungen (Vertraulichkeit, Authentizität, Schutz vor Wiedereinspielung und Datenintegrität) zu realisieren, müssen geeignete Sicherheitsmechanismen gewählt und implementiert werden. In Kapitel **??** wurden aus diesem Grund ausführlich IPSec, SSL und S/MIME vorgestellt. Diese Sicherheitslösungen ermöglichen eine Absicherung auf verschiedenen Ebenen der Kommunikation.

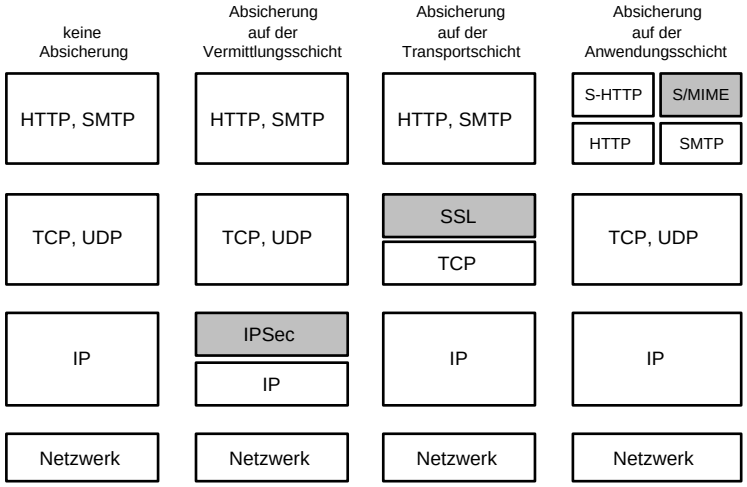

Abbildung 5.2: Absicherung der Kommunikation

Wie aus Abbildung **??** ersichtlich, sichert IPSec die Kommunikation auf der Vermittlungsschicht. Die darüberliegenden TCP oder UDP (in SeMoA nur TCP) Pakete werden verschlüsselt in die IP-Pakete eingebettet und übertragen. Die Kommunikation verläuft auf diese Weise völlig Transparent für die darüber liegende Transport- und Anwendungsschicht. In dem sog. Transport Modus sind Verbindungen zwischen den Rechnern problemlos möglich. Falls der Tunnel-Modus auch unterstützt werden sollte, entstehen allerdings Probleme bei der Kommunikation zwischen Netzwerken, die von Zugriffen von Außen geschützt sind. Man braucht in diesem Fall spezielle Server, die NAT[3] (Network Address Translation) durchführen. Abhilfe schaffen hier dann IPSec-Gateways, die sich um die richtige Zustellung der Pakete in lokalem Netz kümmern. Für die Authentisierung und das Aushandeln der Sitzungsschlüssels müssen IKMP-Mechanismen (Internet Key Management Protocol) realisiert sein. D.h. für die Unterstützung dieser Art der Kommunikation zwischen Agentenplattformen, würde man Mechanismen benötigen, die nicht mehr als Teil der Agentenplattform selbst realisiert werden könnten, sondern auf das ganze Netzwerk angewendet werden müßten.

Auf der anderen Seite benötigt SSL für dessen Realisierung keinerlei technischer Voraussetzungen, die noch nicht erfüllt sind, und die Rahmen der SeMoA-Plattform verlassen. SSL

[3]Network Address Translation (NAT) stellt ein Mechanismus dar, der die IP-Adressen eines lokalen Netzes in öffentliche Adressen umsetzt. Diese Umsetzung findet auf einem Grenzsystem des lokalen Netzes statt.

residiert in der Transportschicht und sichert somit die Protokolle der Anwendungsschicht (HTTP, SMTP, LDAP, usw.). Hier entscheidet die Applikation bzw. im Fall von SeMoA die Agentenplattform dynamisch, ob eine sichere Verbindung aufgebaut werden soll. SSL-Verbindungen werden auf Basis der für Kommunikation vorgesehenen Sockets auf den Rechnern aufgebaut und werden solange gehalten bis der Datenaustausch vollständig ist.

S/MIME sichert hingegen die Kommunikation in der Anwendungsschicht ab. Es stellt einen Sicherheitsmechanismus dar, das auf dem SMTP (Simple Mail Transfer Protocol) und MIME aufbaut, um den Austausch der elektronischen Post auf sicherem Wege zu ermöglichen. S/MIME definiert die Art und Weise, in der Nachrichteninhalte effizient verschlüsselt und entschlüsselt werden, während für den tatsächlichen Transport das darunterliegende Protokoll SMTP zuständig ist.

Da der Datenaustausch bei IPSec und SSL über Verbindungen erfolgt, die solange gehalten werden, bis die Daten vollständig übertragen sind, realisieren diese Mechanismen eine synchrone Art der Kommunikation. Im Gegensatz dazu verschlüsselt S/MIME die Nachrichteninhalte und gibt sie an Dienste weiter, die für ihre Auslieferung zuständig sind. Dadurch realisiert S/MIME eine asynchrone Art der Kommunikation. Weiterhin sicheren alle drei Sicherheitslösungen die Kommunikation auf der abstrakten Datenpaketebene. Während beim IPSec der Inhalts der IP-Pakete abgesichert wird, sichert SSL den Inhalt der TCP-Pakete und S/MIME den Inhalt der elektronischer Post. IPSec hat zusätzlich die Möglichkeit einen sicheren Kanal zwischen zwei Hosts aufzubauen (Tunnel-Modus).

In dem Punkto Sicherheit, leisten alle drei Sicherheitslösungen fast vollständige Deckung der geforderten Maßnahmen. Während IPSec und SSL eine ganze Palette verschiedener Verschlüsselungsalgorithmen unterstützen, unterstützt S/MIME nur zwei, RC2/40 und 3DES. Im Unterschied zu 3DES stellt RC2 mit 40 Bit Schlüssellänge ein schwaches Verschlüsselungsverfahren dar. Zur Absicherung der Kommunikation sollte deswegen 3DES Verwendung finden. Obwohl 3DES etwas zeitaufwendiger als andere sichere Verfahren ist, spielt die zeitliche Komponente bei einem auf S/MIME basierenden, asymmetrischen Kommunikationsystem nur eine unwesentliche Rolle. Alle Sicherheitsverfahren unterstützen in genügendem Maße Authentisierung und Schutz der Datenintegrität. Nur der Schutz vor Wiedereinspielung ist bei S/MIME durch fehlende Mechanismen in Vergleich zu den anderen beiden Sicherheitslösungen benachteiligt.

Als weitere Kriterien bei der Auswahl einer der Lösungen, werden der Aufwand der Implementierung und die Integrierbarkeit in die SeMoA-Umgebung betrachtet. Hier liegt das SSL-Verfahren deutlich im Vorteil, da durch in Java vorhandene Lösungen (JSSE, JCA/JCE), welche die Implementierung des SSL-Mechanismus in Java unterstützen, SSL leicht implementierbar und in SeMoA integrierbar ist. Anderseits gibt es für IPSec und S/MIME (außer der Unterstützung für PKCS#7 und PKCS#12 Standards) seitens Java keine Unterstützung. Die Implementierung dieser beiden Lösungen ist aus diesem Grund mit einem viel größerem Aufwand verbunden. Ein weiterer Vorteil von SSL ist die Möglichkeit des einfachen Aufbaus verschiedener Verbindungen mit verschiedenen Parametern von einem Host zur anderen

Kommunikationspartnern. Dies ist notwendig in einem Multi-Agenten-System, in dem verschiedene Agenten gleichzeitig kommunizieren sollen. Ähnliches ermöglicht S/MIME durch dessen asynchrone Kommunikationsart. Allerdings werden die Aufträge bei dem darunterliegenden Protokoll (SMTP) auch einer nach dem anderen abgearbeitet. IPSec hingegen erlaubt nur eine Verbindung gleichzeitig und eignet deswegen für die Multi-Agenten-Systeme nur bedingt. Der einzige Nachteil vom SSL ist die bedingte Wiederverwendbarkeit der Kommunikationsparametern. Eine Verbindung zwischen bekannten Kommunikationspartnern läßt sich bei SSL nur dann ohne zeitaufwändigen Handshake-Prozeß wiederverwenden, falls die Kommunikationsdaten noch in dem sog. *Session-Cache* der Sockets vorhanden sind. Andernfalls müssen die Kommunikationsparameter erneut ausgehandelt werden. IPSec und S/MIME ermöglichen im Gegensatz zu SSL, eine effizientere Verwaltung der Kommunikationsparametern. Bei IPSec wird dies durch Speicherung der Security Associations und bei S/MIME durch Speicherung des öffentlichen Schlüssels und Verschlüsselungspräferenzen der Kommunikationspartner realisiert.

Eigenschaft	**IPSec**	**SSL**	**S/MIME**
Ebene der Absicherung	Vermittlungsschicht	Transportschicht	Anwendungsschicht
technische Voraussetzung	NAT u. IKMP	-	SMTP u. MIME
Kommunikationsart	synchron	synchron	asynchron
Sicherungskonzept	Datenpaket/Tunnel	sicherer Kanal	Datenpaket
Vertraulichkeit (Algorithmen)	RC5, DES, 3DEC IDEA, 3IDEA CAST, Blowfish	RC4, RC5, DES und 3DES	RC2 u. 3DES
Authentisierung	symmetrisch und asymmetrisch	asymmetrisch (RSA und DSA)	asymmetrisch (Diffie-Hellman)
Datenintegrität	MAC (MD5 u. SHA-1)	MAC (MD5 u. SHA-1)	Digitale Signatur (DSS u. RSA)
Schutz vor Wiedereinspielung	Sequenznummer	Sequenznummer	-
Java-Lösung	-	JSSE + JCA/JCE	-
Verbindungen mit unterschiedlichen Sicherheitsparametern	-	JA	bedingt
Wiederverwendung der Parameter	JA	bedingt	JA

Tabelle 5.1: Vergleich zwischen IPSec, SSL und S/MIME

In Tabelle **??** wurden die Vergleichspunkte von IPSec, SSL und S/MIME noch einmal zusammengefasst. Diese Gegenüberstellung liefert eine gute Grundlage, um die Entscheidung über die Auswahl einer geeigneten Lösung zu treffen. Nach einer Analyse der Vergleichs-

punkte ist es offensichtlich, daß die Auswahl auf SSL fällt. SSL unterscheidet sich in den meisten Sicherheitsfragen nur unwesentlich von anderen zwei Verfahren und deckt dabei alle von der Aufgabenstellung geforderte Sicherheitsanforderungen und Voraussetzungen ab. Der wesentliche Vorteil von SSL liegt in der Einfachheit der Integration in SeMoA. Dies ist der wichtigste Punkt bei dem Prozeß der Realisierung. Da die Unterstützung für die Implementierung von SSL in Java vorhanden ist und die Erweiterung der vorhandenen Mechanismen leicht durchgeführt werden kann, erweist sich die Implementierung von SSL als einfache Aufgabe. Einen weiteren Vorteil stellt die Unabhängigkeit von anderen technischen Voraussetzungen dar. Die Realisierung von SSL hängt in diesem Zusammenhang nicht von anderen Mechanismen und Hardwarekomponenten ab, wie das bei IPSec und S/MIME der Fall ist. Weiterhin ist die Möglichkeit der Aufbau mehrerer Verbindungen mit verschiedenen Sicherheitsparametern gleichzeitig ein weiterer Pluspunkt für die SSL-Lösung.

5.2 Kommunikationsmodell

Der zu implementierende Sicherheitsmechanismus sollte wie bereits erwähnt auf dem vorhandenen Kommunikationsmechanismus in SeMoA aufbauen. Aus diesem Grund werden nur einige Veränderungen und Ergänzungen Basismechanismus selbst nötig sein. Der SSL-Mechanismus wird mit der Berücksichtigung der in SeMoA vorhandenen Sicherheitslösungen definiert und aufgebaut. Zur diesem Zweck müssen die Konfigurationsparameter der Kommunikation um SSL-betreffende Parameter erweitert werden. Außerdem wird die Realisierung einer Filter-Pipeline für ausgehende und eingehende Nachrichten in SeMoA diskutiert.

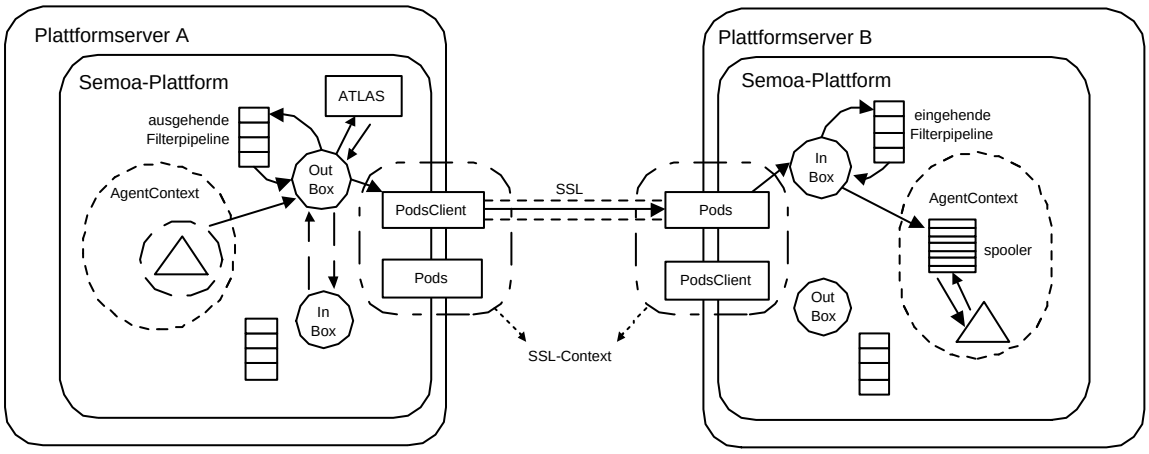

Abbildung 5.3: Anpassung der Kommunikation bei der SeMoA-Plattform

5.2.1 Kommunikationsparameter

Die Kommunikation wird seitens des Agenten eingeleitet. Der Agent entscheidet über Zeitpunkt und Art der Kommunikation. Die SeMoA-Plattform geht dem Wunsch des Agenten nach und unternimmt alle dafür notwendigen Schritte. Dabei kümmert sich der Agent um das Bilden der Kommunikationsadresse, und der Bindung dieser an die Nachricht, und auf der anderen Seite um den Empfang. Der Empfang der Nachricht wird dem Agenten seitens der Plattform nicht signalisiert, ein Signalisierungsmechanismus ist allerdings agenten- und anwendungsspezifisch leicht relisierbar. Nicht erfolgreiche Auslieferung wird dem Sender durch verschiedene Fehlermeldungen signalisiert.

Die Entscheidung darüber, ob sicher oder unsicher kommuniziert wird, liegt beim Agenten. Der Agent sollte explizit eine unsichere, sichere oder eine bedingt sichere[4] Kommunikation verlangen können. Aus diesem Grund werden einige Änderungen in der SeMoA-Klasse `CommunicationContext` nötig sein. Zusätzlich bestehenden Methoden wird eine neue Methode eingeführt `sendSecure()`, die diese Entscheidungsmöglichkeit ermöglicht.

Protokoll

Um dem Agenten die sichere Übertragung der Nachrichten zu ermöglichen und um diese Übertragung aber explizit von der unsicheren zu unterscheiden, wird ein neues Protokoll definiert. Das Protokoll hat den Namen Pods (**Po**rtal **D**aemon **S**ecure) und wird nach derselben Schemata wie das `Pod`-Protokoll verwendet, d.h. die Kommunikationsadresse des Agenten hat in diesem Fall folgendes Aussehen:

```
pods://(implicitName|nickName|spoolerName)@(host|automatic):PODS_PORT
```

Der einzigste Unterschied zum alten Adressaufbau liegt in der Tatsache, daß in der Kontaktadresse der Agenten der Kommunikationsport (bzw. die Socketnummer) für die sichere Übertragung des Kommunikationspartners (`PODS_PORT`) angegeben wird, da sich dieser von dem Kommunikationsport (`COMM_PORT`) für unsichere Kommunikation unterscheidet. [5] Die Aufgabe des `Pods`-Protokolls ist es, durch die Verwendung eines bestimmten Bezeichners (`pods`) dem Kommunikationsmechanismus zu signalisieren, eine sichere Verbindung für Kommunikationszwecke einzuleiten. Weiterhin, weist das Protokoll durch die Präsenz von Client und Server im Environment auf die Fähigkeit der Plattform hin, sicher zu kommunizieren. Das `Pods`-Protokoll läßt sich dann mit den Methoden des `CommunicationContext` genauso verwenden, wie die unsichere Variante.

[4] Bedingt sichere Kommunikation stellt hier eine Kommunikation dar, die in dem Fall, daß ein sichere Übertragung nicht möglich ist (z.B. wegen mangelnde Fähigkeit des Kommunikationspartner sicher zu kommunizieren) die Nachricht automatisch unsicher überträgt.

[5] Der `PODS_PORT` wird in der Form einer Zahl in der Kontaktadresse des Empfängers fest kodiert. Dieses wird in Zukunft durch geplante Einführung eines Informationsprotokolls mit Namen SHIP (Simple Host Information Protocol) in SeMoA, nicht mehr nötig sein. Dieses Protokoll wird einen Austausch aller für Migration und Kommunikation relevanten Daten zwischen Agentenplattformen ermöglichen. Auf dieser Weise wird der Kommunikationsport bekannt sein und durch geeignete Mechanismen automatisch in die Kontaktadresse eingesetzt.

Dienste

In Anlehnung an das Prinzip des Nachrichtenaustausch des in SeMoA vorhandenen Mechanismus, werden Dienste für sicheren Kommunikation definiert. Ähnlich wie bei der unsicheren Kommunikation baut der PodsClient-Dienst der SeMoA-Plattform, eine sichere Verbindung zu dem Pods-Dienst der SeMoA-Plattform, auf der sich der empfangende Agent befindet (vergl. die Abbildung **??**). Die Absicherung findet nur in dem Nachrichtenkanal zwischen den SeMoA-Plattformen bzw. zwischen beiden Diensten statt. Die Kommunikation die in diesem Kanal stattfindet und auf TCP/IP basiert, wird durch SSL-Mechanismen abgesichert. Die restliche Behandlung der Nachrichten funktioniert nach dem selben Prinzip wie bei dem bestehenden Mechanismus.

5.2.2 SSL-spezifische Parameter

Java Secure Socket Extension

SSL wird in SeMoA mit Hilfe der Java Secure Socket Extension (JSSE)[**?**][**?**] realisiert, die den Aufbau von SSL-Mechanismen unterstützt. JSSE stellt eine Ansammlung von Klassen rund um SSLv3.0 und TSLv1.0 dar, und ist in Java 2 ab der Version 1.4 standardmäßig vorhandenem. JSSE baut auf ähnlichen Mechanismen wie bei dem Java-Sicherheitsgerüst mit Namen Java Cryptography Architecture (JCA)[**?**] auf. Die Sicherheitmechanismen und -algorithmen werden bei JSSE von einem eigenen Provider implementiert, dem sog. SunJSSE-Provider. JSSE ist allerdings an die Verwendung der eigenen Implementation nicht gebunden. Es erlaubt z.B. die Verwendung der Verschlüsselungsalgorithmen aus dem Java-eigenen Kryptoprovider Java Cryptography Extension (JCE) oder der Algorithmen anderer Provider.

Um die Verbindung zwischen dem Client und Server-Rechner aufbauen zu können, unterstützt JSSE zwei Typen von Sockets, den `SSLSocket` (Client-Socket) und den `SSLServerSocket` (Server-Socket). Der Client baut eine SSL-Verbindung zum Server auf, indem er zu diesem Zweck bei dem SSLServerSocket eine Verbindung anfordert. Es folgt das Handshake zum Aushandeln des benötigten Sicherheitskontextes, falls dies zwischen beiden Sockets noch nicht erfolgt ist. Danach können die Daten übertragen werden. Die Sockets werden durch die sog. `SocketFactory`-Klassen generiert. Diese Klassen konfigurieren und initialisieren die Sockets mit den für die Verbindung wichtigen Parametern (z.B. Authentisierungsschlüssel, erlaubte Cipher-Suites usw.).

Wenn zwei Sockets verbunden sind, gehören sie automatisch zu einer `SSLSession`. Die SSLSession beinhaltet den Sicherheitskontext, der durch beide Seiten am Anfang der SSL-Verbindung durch den Handshake-Prozeß ausgehandelt worden ist. Nachdem eine SSLSession zwischen zwei Plattformen generiert worden ist, kann sie von verschiedenen Verbindungen zwischen den gleichen Rechnern benutzt werden. SSLSession enthält neben den Informationen über die angewendete Sicherheitsmechanismen bzw. Cipher-Suites auch das `master_secret`-Datum, das für Erzeugung des symmetrischen Schlüssels verwendet wird, sowie weitere SSL-Parameter.

SSLSessions werden werden vom `SSLContext` verwaltet und in einem Cache zum Zweck der Wiederverwendung gespeichert. SSLContext stellt den Hauptmechanismus zur Generierung des SSL-Mechanismus dar. Er definiert die Version des SSL-Protokolls (bzw. TLS-Protokolls), das in der Implementierung benutzt wird, und initialisiert die Mechanismen, die für Schlüssel- (`KeyManager`) und Zertifikatsverwaltung (`TrustManager`) zuständig sind. Die primäre Aufgabe des `KeyManager` ist es, alle für eine eventuell stattfindende Authentisierung benötigten Daten bereitzustellen. Der KeyManager hängt von dem für die Authentisierung verwendeten Algorithmus ab und benutzt dafür die im `KeyStore`[6] gespeicherten Schlüssel. Um den in JSSE vorhandenen (Standard-) `KeyManager` zu benutzen, muß dieser mit dem Typen, Namen und benötigtem Passwort zum Zugriff auf einen bestimmten `KeyStore` initialisiert werden. Dieser `KeyManager` wird von der sog. `KeyManagerFactory` erzeugt, welche im `SSLContext` initialisiert wird. Außer der Verwendung des Standard-`KeyManager` erlaubt JSSE auch die Implementierung eines eigenen `KeyManager`.

Im Gegensatz dazu, ist die Aufgabe des `TrustManager`, zu bestimmen, ob den Authentifikationsdaten des Kommunikationspartners vertraut wird. Falls diesen Daten nicht vertraut wird, wird die SSL-Verbindung unterbrochen. Ähnlich wie beim `KeyManager` hängt die Funktionsweise von dem verwendeten Authentifikationsmechanismus ab. Auch hier erlaubt JSSE die Nutzung des vorhandenen bzw. eines eigenen `TrustManager`. Der `TrustManager` wird ebenfalls durch die im `SSLContext` initialisierte `TrustManagerFactory` erzeugt. Die in JSSE standardmäßig vorhandene Key- und TrustManager arbeiten mit X.509 Zertifikaten. Außer der Generierung dieser `Factory`-Instanzen ist der `SSLContext` zusätzlich für die Erzeugung der von Sockets benötigten `SocketFactory`-Instanzen zuständig.

SSL in SeMoA

Die Implementierung von SSL in SeMoA beinhaltet die Realisierung aller wichtigen von JSSE benötigten Komponenten. Für die Realisierung der `SSLSocket` und `SSLServerSocket`-Komponenten sind die Klassen zuständig, die den `PodsClient` und `Pods` realisieren (vergl. Abb. **??**). Die Sockets werden auf diese Weise initialisiert, um die gegenseitige Authentisierung (Client und Server) und die Verwendung einer speziellen Cipher Suit zu unterstützen. Der `SSLContext` und somit auch der `KeyManager` und `TrustManager` werden von einem im Environment vorhandenen Objekt, dem sog. `SSLMaster` realisiert. Der `KeyManager` und `TrustManager` werden so initialisiert, daß diese auf den `KeyMaster` zugreifen, wenn es um Anforderung von Schlüsseln und Zertifikaten geht. Die Aufgabe des `KeyMaster` in SeMoA ist die Verwaltung aller Schlüssel und Zertifikate, die in `Keystores` der Agentenbesitzer gespeichert werden. Da die `Keystores` in SeMoA bereits alle benö-

[6] Ein Keystore ist eine Datei, die Schlüssel und Zertifikate enthält, die für Java-Sicherheitsanwendungen bereitgestellt werden müssen. Der Zugriff auf Keystores ist geschützt und nur mit einem Passwort möglich. Es gibt zwei Typen von Keystores - JKS (Java Keystore) und JCEKS (JCE Keystore). SeMoA benutzt JKS Keystores.

tigten Schlüssel bzw. X.509v3-Zertifikate enthalten[7], sind hier keine weitere Schritten nötig. Ein eigener `TrustManager` ermöglicht die Verifizierung der Zertifikate über den SeMoA-eigenen Verifizierer (`CertificateChainVerifier`) und `KeyMaster` somit eine dynamische und flexible Konfiguration der SSL-Umgebung.

5.2.3 Nachrichten-Filterpipeline

Die Nachrichten-Filterpipeline wird analog der Sicherheitspipeline aufgebaut, die von dem Migrationsmechanismus von SeMoA bekannt ist, aus zwei Reihen Filter, einmal für ausgehende und einmal für eingehende Agenten besteht und eine Reihe von sicherheitsbezogenen Operationen an Agenten ausführt (siehe dazu Abschnitt **??**). Im Unterschied zur Sicherheitspipeline soll die Nachrichten-Pipeline vorerst keine sicherheitsbezogen Aufgaben erfüllen, da die Absicherung der Nachrichten in dem Kommmunikationskanal zwischen den SeMoA-Plattformen stattfindet. Eine Realisierung von sicherheitsbezogenen Filtern, die den Sicherheitsmechanismus des SSL ergänzen (z.B. S/MIME) ist aber durchaus vorstellbar. Der eigentliche Sinn dieser Pipeline ist es, auf der Inhaltsebene der Nachrichten zu Arbeiten. So sind zum Beispiel Mechanismen vorstellbar, die den Agenten beim Parsen und der Interpretation der ACL-Nachrichten (z.B. FIPA-ACL) behilflich sind.

Da die Realisierung der Nachrichten-Pipeline nicht direkt zu der Aufgabe der Kommunikationsabsicherung gehört, wird nur die Pipeline-Struktur für ausgehende und eingehende Filter realisiert, welche ein flexibles Einfügen und Entfernen von zueinander unabhängigen Filterstufen ermöglicht. Es wird beispielhaft zumindest eine Filterstufe, d.h. ein eingehender und sein komplementerer ausgehender Filter realisiert.

Alle Nachrichten sollen im ausgehenden Fall vor dem Verlassen der `OutBox` und im eingehenden Fall, bevor sie `InBox` verlassen, gefiltert werden (siehe Abbildung **??**). Dazu sind einige Änderungen in den `InBox` und `OutBox`-Klassen nötig. Um die Implementierung der Filter zu unterstützen, wird eine Schnittstelle (Interface) mit Namen `MessageFilter` definiert. Alle Nachrichtenfilter müssen diese Schnittstelle implementieren.

5.2.4 Weitere Komponenten

Die Fehlerbehandlung bei der Kommunikation wird entsprechend der Neuerungen angepasst. Ziel ist es, eine möglichst genaue Beschreibung eventuell eintretender Exceptions zu geben, um den Agenten eine passende Reaktion auf Fehler zu ermöglichen.

Außer der Fehlerbehandlung wird eine Protokollierung von Vorgängen beim Nachrichtentransport implementiert. Diese soll relevante Informationen für den Fall der späteren Fehlersuche und Fehlerrekonstruktion speichern. Die Speicherung erfolgt in eine bestimmte Datei (Log-Datei) auf dem lokalen Dateisystem.

[7] In SeMoA befinden sich zur Zeit in den Keystores der Benutzer standardmäßig die DSA und RSA Schlüsselpaare, dessen öffentliche Teile von der lokalen CA "A8-SeMoA-CA"zertifiziert sind. Zum Zweck der Authentisierung enthält jeder Keystore auch das selbst-signierte Zertifikat der *A8-SeMoA-CA* Zertifizierungsstelle.

5.3 Vergleichbare Arbeiten

Es existieren bereits eine Reihe von Lösungen, die sich mit der Absicherung von Agentensystemen und speziell der Agentenkommunikation beschäftigen. Einige davon wie zum Beispiel die Arbeit von *Finnin*, *Mayfield* und *Thirunavukkarusu*[**?**] fangen bei dem Fundament der Kommunikation bzw. bei der ACL an. Sie schlagen ein Modell vor, welches KQML um geeignete Sprachkonstrukte ergänzt, um sichere Kommunikation bei den auf KQML basierenden Agentensystemen zu ermöglichen. Auf der anderen Seite gibt es Konzepte, die sich mit der Definition von geeigneten Mechanismen zum sicheren Nachtichtenaustausch beschäftigen, die ein Teil des Systems darstellen. Wegen der Ähnlichkeit mit dieser Arbeit werden hier kurz zwei solche Konzepte vorgestellt. Beim ersten wird die Absicherung der Kommunikation durch S/MIME vorgeschlagen, beim zweiten Konzept wird eine Sicherheitsschicht für Kommunikationszwecke in die Plattform eingeführt. Beide Konzepte bauen auf FIPA-basierten Systemen auf.

5.3.1 Absicherung der Kommunikation einer FIPA-Plattform

Dieses Konzept, das von *Tan*, *Titkov* und *Neophytou*[**?**] stammt, beschreibt eine Möglichkeit der Absicherung der Kommunikation in FIPA-basierten Systemen durch den S/MIME Mechanismus, wobei S/MIME für die Absicherung des Nachrichtenaustauschs zwischen den Agentenplattformen sorgt. Für S/MIME spricht bei FIPA die Tatsache, daß die Spezifikation für Nachrichten bei FIPA2000 auf dem MIME Standard für elektronische Post aufbaut. Das S/MIME Protokoll wird auf diese Nachrichtenstruktur einfach aufgesetzt. Das Konzept redefiniert den Nachrichtentransport bei FIPA so, daß die ACL-Nachrichten nach der Spezifikationen von S/MIME durch PKCS#7 (CMS) und Base64 kodiert werden (siehe Bild **??**). Die Aufgabe von PKCS#7 ist die Definition der Sicherheitskomponenten und die Verschlüsselung des Nachrichteninhaltes. Das digitale Signieren der Daten, daß S/MIME in seiner Definition vorsieht, wird hier nicht unterstützt und ist erst für zukünftige Entwicklungen vorgesehen.

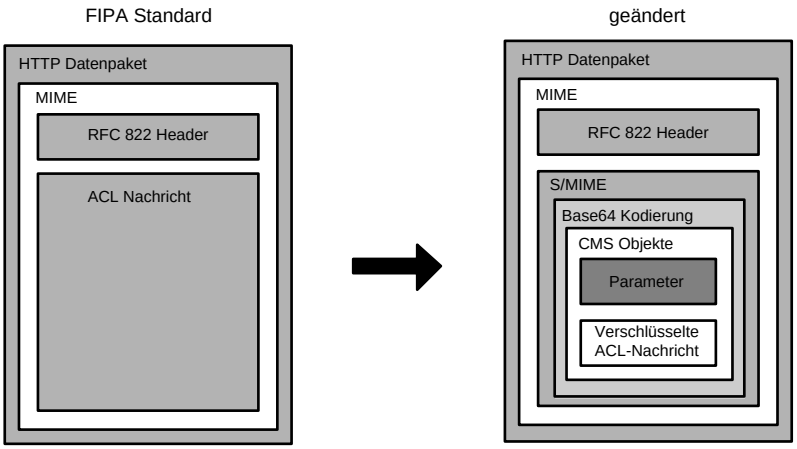

Abbildung 5.4: Änderungen im FIPA-Standard um das S/MIME zu unterstützen

Der Nachrichtenaustausch funktioniert nach folgendem Prinzip: Falls der Agent eine Nachricht verschicken will, sendet er sie vorerst an das ACC der Plattform. Der Nachrichtinhalt wird dann mit den zufällig generierten symmetrischen Schlüssel verschlüsselt. Dieser Schlüssel wird danach mit dem öffentlichen Schlüssel des Empfängers verschlüsselt und zusammen mit einigen sicherheitsrelevanten Parametern der Nachricht hinzugefügt. Der Schlüssel für Verschlüsselung des Nachrichteninhalts kann auch mit dem symmetrischen Schlüssel der letzten Session oder mit dem neu ausgehandelten Session-Schlüssel verschlüsselt werden. Base64-kodiert, wird die Nachricht anschließend zum ACC der Empfängerplattform übertragen. Das ACC unternimmt den Umkehrprozess um die Nachricht zu rekonstruieren und gibt sie daraufhin an das *Message Transport System* der Plattform weiter, um sie dann dem Empfänger auszuliefern.

Außer der Redefinition der Nachrichtenstruktur werden einige weitere Änderungen nötig. So sollten neue Parameter und ebenfalls eine neue Methode für sichere Kommunikation (z.B. `FIPAAgent.secureForward()`) definiert werden, um diese Art des Nachrichtentransportes von der unsicheren zu unterscheiden. Die Autoren schlagen weiterhin, die Definition von Optionen zur Auswahl des Sicherheitsgrades durch definierbare Cipher Suites vor.

5.3.2 Sicherheitsschicht in FIPA-OS

In der Arbeit von *Zhang*, *Karmouch* und *Impey*[**?**] wurde ein Konzept einer Sicherheitsschicht in der FIPA-OS gegeben. Dieses Konzept unterteilt die Agentenplattform in zwei Schichten, wie in der Abbildung **??** dargestellt. Die erste Schicht enthält die vorhandene FIPA-OS Implementierung und die zweite Schicht stellt die sicherheitsrelevanten Dienste bereit. Die Aufgabe der Sicherheitsschicht ist die Authentisierung, sichere Kommunikation und Überwachung der Ressourcen zu gewährleisten. Die Hauptkomponenten der Sicherheitsschicht sind die *Secure Agent Channel Communication* (SACC) und das *Credential Granting Center* (CGC). SACC stellt eine Komponente dar, die für die sichere Kommunikation über ACC zuständig ist und CGC ist eine Komponente, die für Authentisierung und Rechtevergabe zuständig ist.

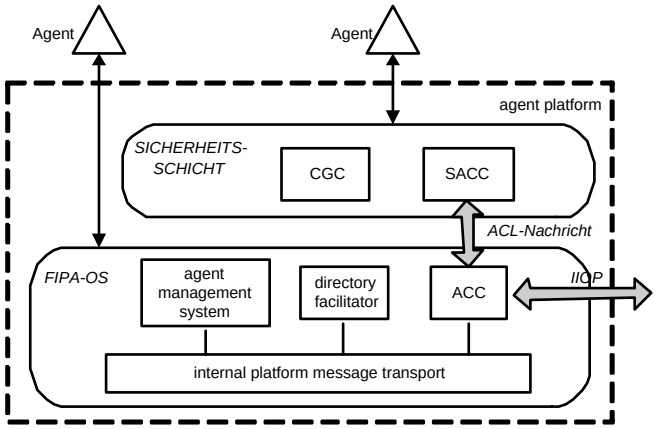

Abbildung 5.5: Sicherheitsschicht bei FIPA-OS

Das SACC schützt die Kommunikation durch Unterstützung von Authentisierung, Verschlüsselung und Datenintegrität. Aufbau eines sicheren Kanals besteht aus zwei Stufen - (1) Authentisierung der Plattform und (2) Aushandeln der Sicherheitsparameter.

Die Authentisierung basiert auf der gegenseitigen Prüfung der plattformeigenen Zertifikate und digitalen Signaturen. Jede Plattform besitzt ein Schlüsselpaar dessen öffentlicher Teil von einer CA zertifiziert wird. Den verifizierten Plattformen und ihren Agenten wird danach vertraut.

Falls der Authentisierungsprozeß erfolgreich war, werden in der nächsten Phase die Sicherheitsparameter ausgehandelt. Der Sinn des Aushandelns dieser Parameter ist die Definition der kryptographischen Mechanismen, die bei dem Nachrichtenaustausch benutzt werden. Einer der Parameter die hier ausgehandelt werden ist zum Beispiel der symmetrische Schlüssel für die Verschlüsselung der Nachrichten. Dieser wird vom Sender generiert und durch die Verschlüsselung mit dem öffentlichen Schlüssel der Empfängers ausgetauscht. Das Aushandeln der Sicherheitsparameter kann auf zwei Arten erfolgen - (1) der Empfänger entscheidet, ob er die Parameter des Senders akzeptiert oder nicht. (2) Bei der zweiten Art hat der Empfänger außerdem die Möglichkeit mit seinen eigenen Parametern zu antworten, im Fall daß er die Parameter des Sender nicht akzeptiert.

Nach dem erfolgreichen Abschluss beider Phasen, ist ein sicherer Kommunikationskanal etabliert, und ACL-Nachrichten können auf dieser Basis wie gewohnt ausgetauscht werden.

Kapitel 6

Implementierung

Die Implementierung der sicheren Kommunikationsmechanismen in SeMoA wurde in der Programmiersprache Java (JDK 1.4.1) durchgeführt. Dabei wurden die für SeMoA vorgeschriebenen Code Conventions[**?**] berücksichtigt. Obwohl die aktuelle Version von SeMoA vollständig kompatibel zum JDK 1.3 ist, wurde die Verwendung der Version 1.4 aus mehreren Gründen vorgezogen.

Das JSSE ist bei der Version 1.4 standardmäßig vorhanden, wohin gegen es bei früheren Version nur als Erweiterungspaket erhältlich war. Außerdem wurden beim Übergang in JSSE folgende Änderungen durchgeführt:

- `SunJSSE`-Provider kann JCE oder andere Kryptoprovider verwenden. Früher war nur die Nutzung der von `SunJSSE` implementierten Algorithmen möglich.
- Der Klassenpfad für die JSSE-Klassen wurde in `javax.net.ssl` geändert (früher `com.sun.net.ssl`).
- Es wurden einige neue Methoden und Klassen hinzugefügt. So wurden zum Beispiel die `setSessionTimeout()`- und `setSessionCacheSize()`-Methode zu der Klasse `SSLSessionContext` hinzugefügt, um die Gültigkeitsdauer und Anzahl der Sessions, die für eventuelle erneute Benutzung gespeichert werden, zu setzen.
- An manchen alten Klassen, die vor allem für den Authentisierungsprozeß zuständig sind, wurden einige Änderungen vorgenommen.

Eine Anpassung von SeMoA auf das neue JDK 1.4 ist geplant und solle im Kürze durchgeführt werden. Somit ist die Verwendung des hier realisierten sicheren Kommunikationsmechanismen nicht in Frage gestellt.

6.1 Klassen und Konzepte

In diesem Abschnitt wird auf die Beschreibung der implementierten Klassen und Implementationsänderungen in SeMoA eingegangen. Dabei werden zur Beschreibung nur die Methoden

herangezogen, die wesentlich für die Funktionalität der jeweiligen Klassen sind. Andere in der Regel durch Vererbung übernommene Methoden, werden in diesem Kontext nicht erwähnt.

6.1.1 Der Kommunikationskontext

Wie bei der Beschreibung der Konzepte (seihe **??**) bereits eingeführt, liegt die Entscheidung darüber, ob sicher oder unsicher kommuniziert werden soll, bei dem Agenten selbst. Um dieses zu unterstützen wird eine neue Methode mit Namen `sendSecure` in die SeMoA Klasse `CommunicationContext` eingefügt. Die `sendSecure`-Methode hat folgende Syntax:

```
public void sendSecure(URL to,URL from,byte[] data,int secflag)
```

Die Variablen `to`, `from` und `data` haben dieselbe Bedeutung, wie bei der `send`-Methode. Neu hier ist die ganzzahlige Variable `secflag` (von security flag). Sie erlaubt die genaue Definition des gewünschten Sicherheitsgrades der Kommunikation. Dabei kann `secflag` drei Werte annehmen - 0,1 oder 2. Falls `secflag` auf Wert 0 gesetzt ist, heißt es, daß der Agent explizit eine unsichere Verbindung wünscht. Bei dem Wert 1, möchte der Agent ausschließlich sicher kommunizieren. Wenn der Agent `secflag` auf Wert 2 setzt, heißt das, daß der Agent eine sichere Verbindung wünscht, falls das aber aus irgendeinem Grund nicht möglich ist, die Daten über eine unsichere Verbindung verschickt werden dürfen. In diesem Fall wird zuerst der Aufbau einer sicheren Verbindung versucht. Bei einem Fehler wird dann doch eine unsichere Verbindung initiiert.

Andernfalls, also bei Verwendung der regulären `send`-Methode, erfolgt die Unterscheidung zwischen sicherer und unsicherer Übertragung durch das Setzen des entsprechenden Protokollbezeichners in der Kontaktadresse des Empfängers. Für eine unsichere Verbindung wird als Protokollbezeichner `pod` in der Kontaktadresse gesetzt, für eine sichere der Bezeichner `pods`.

Die einzige Problematik besteht nun bei der Initialisierung von `secflag` mit dem Werk 2 darin, die Kontaktadresse des Empfängers mit dem richtigen Kommunikationsport zu initialisieren, sofern im Fehlerfall bei der sicheren Verbindung, eine unsichere Verbindung aufgebaut werden muß. Da der `CommunicationContext` die Portadresse der jeweiligen Pod(s)-Server nicht ermitteln kann, muß hier mit allgemein bekannten Ports gearbeitet werden.
In Zukunft wird dieses Problem durch die Anwendung von SHIP (siehe die Fußnote auf Seite 75) allerdings nicht mehr bestehen.

6.1.2 Server- und Clientklassen

Um den Mechanismus für den Aufbau einer sicheren Verbindung zwischen Agentenplattformen zu realisieren, werden die Klassen `PodsClient` und `Pods` implementiert. Sie implementieren die in Konzeptbeschreibung eingeführte Dienste zum Nachrichtenaustausch und sind für das Initialisieren der Sockets, den Aufbau einer Verbindung und das Übertragen der Daten zuständig. Die UML-Diagramme der in diesem Abschnitt beschriebenen Klassen finden sich in Abbildung **??**.

PodsClient

Die Klasse `PodsClient` realisiert dabei den größten Teil der Aufgabe beim Aufbau einer SSL-Verbindungen. Sie implementiert das `MessageGateway.Out` und initialisiert die Verbindung. Der Konstruktor des `PodsClient` ruft die Methode `initSocketFactory` auf, um die für Erzeugung des `SSLSocket` benötigte `SocketFactory` zu initialisieren.

Bevor die Verbindung benutzt werden kann, muß sie durch `SSLSocket` Generierung und Parameter initialisiert werden. Dies wird in der `createSocket`-Methode durchgeführt. Die Erzeugung des `SSLSocket` erfolgt unter Mitwirkung des `SSLSocketFactory` und geschieht durch den Aufruf ihrer `createSocket`-Methode unter Angabe der URL der Empfängerplattform bzw. ihres `ServerSocket`. Die Generierung des Sockets startet den internen SSL-Handshake Prozeß, baut die Verbindung auf und erzeugt eine für beide Rechner gültiges Session. Die Lebensdauer der Session wird dann in der `createSocket`-Methode des `PodsClient` explizit auf ein bestimmtes `TIMEOUT` (Zeitdauer in Sekunden) gesetzt.

Der Kommunikationsdienst wird typischerweise über die `send`-Methode des `CommunicationContext`, der wiederum an das `OutGate` weitergibt, angesprochen. Zum Versenden der Nachricht ruft die zur `OutBox` gehörende `send`-Methode, die `send`-Methode der `PodsClient`-Klasse auf. Ihre Aufgabe ist es, zu versuchen die Nachricht mit einer festgelegten Anzahl an Wiederholungen (die mit `MAX_RETRY` definiert ist), zu versenden. Bei einem erfolgreichem Versandt wird der Zähler für erfolgreiche Verbindungen (`good`-Methode) inkrementiert, andernfalls wird der Zähler für fehlerhafte Verbindungen (`bad`-Methode) hochgezählt und eine entsprechende Fehlermeldung zurückgegeben. Eine weitere Aufgabe dieser Methode ist es, die Daten die verschickt werden sollen, und die bis dahin als *Bytestream* vorliegen, in Pakete im ASN.1 Format zu verpacken.

Die Methode `sendAgain` leitet den Aufbau der Verbindung durch den Aufruf der Methode `createSocket` ein und versendet die Datenpakete über die aufgebaute Verbindung. Diese Methode bedient sich dabei dem `OutputStream`-Konzept in Java, das die einfache Byteübertragung unterstützt.

Die Methode `toString` gibt Status-Daten über stattgefundene Kommunikation zurück. So werden hier die oben erwähnte Zähler der `good`- und `bad`-Methoden über erfolgreiche und erfolglose Verbindungen zurückgegeben.

Pods

Die Klasse `Pods` implementiert das `MessageGateway.In`. Die primäre Aufgabe dieser Klasse ist einen `SSLServerSocket` aufzubauen. Dieser wird an einem bestimmten Port (`PODS_PORT`) gebunden, hört dort die Kommunikation ab und leistet Hilfe bei einem Verbindungs- und Sessionaufbau. Die Rolle des `SSLServerSocket` und somit des `Pods`-Dienstes ist eher passiv, da sie beim Verbindungsaufbau auf Verbindungsanfragen antwortet. `Pods` empfängt anschließend die Nachrichtendaten und gibt sie durch Aufruf der `Inbox`-Methode `deliver` an die `Inbox` weiter.

Die Erzeugung und Initialisierung des SSLServerSocket wird in createServerSocket-Methode durchgeführt. Die für Erzeugung des Sockets benötigte SSLServerSocketFactory wird in dem Konstruktor der Klasse und in der initServerSocketFactory-Methode initialisiert. Weiterhin werden nach der Erzeugung des Sockets, einige Verbindungsspezifische Parameter gesetzt. So wird durch einen setNeedClientAuth(true)-Aufruf die Authentisierung des Clients bei einer Verbindungsaufbau erzwungen. Außerdem wird die für die Verbindung benötigte Cipher Suite auf den in CIPHER_SUIT definierten Parameter gesetzt.

Die toString-Methode gibt einige technische Parameter der Sicherheitsmechanismen. Hier werden Informationen über die unterstützten Cipher Suites sowie die aktuell verwendete CIPHER_SUITE zurückgegeben.

6.1.3 SSLContext

Das JSSE verlangt bei Verwendung von SSL die Initialisierung des SSLContext. Die Initialisierung wird im Fall der Kommunikation von der SSLMasterImpl-Klasse veranlasst. SSLMasterImpl ist eine Implementierung der SSLMaster-Schnittstelle, die dafür gedacht ist, die Verwaltung des zentralen SSLContext in SeMoA zu gewährleisten.

SSLMasterImpl

Der SSLContext wird in Klasse SSLMasterImlp (siehe Abb. **??**) durch ihre Methode initSSLContext initialisiert. Diese Methode holt erst einmal eine Referenz aus dem Environment auf den benötigten KeyMaster. Darauf hin wird der SSLContext auf die SSL Version 3 gesetzt, und die benötigte KeyManagerFactory kreiert und auf JSSE-eigene Algorithmus zur Unterstützung der X.509-basierten Authentisierung (SunX509) gebunden. Umden Authentisierungsmechanismus zu realisieren verlangt SunX509 die Referenz auf einen Keystore mit den benötigten Zertifikaten und ein Passwort um auf den Keystore zuzugreifen. Um dieses zu ermöglichen wird als nächstes ein Keystore von Typ JKS mit benötigten Zertifikaten generiert und durch das in Variable PASSWD_ definiertes Passwort geschützt. Die benötigten Zertifikate und Schlüssel werden aus dem benutzereigenem Keystore ausgelesen und an den neugenerierten übergeben. Der nächste Schritt ist die Generierung einer Referenz auf den SeMoA-eigene X509TrustManager. Durch die Methode getKeyManagers des KeyManagerFactory wird eine Instanz des KeyManager zurückgegeben. Zuletzt werden die Referenzen auf die Objekte KeyManager und TrustManager zusammen mit der Referenz auf die SecureRandom-Klasse dem SSLContext übergeben.

X509TrustManager

Die Klasse X509TrustManager (siehe Abb. **??**) ist eine Implementierung des von JSSE verlangtem TrustManager. Wie oben erklärt, ist der KeyMaster in SeMoA auch für die Verwaltung der Zertifikate zuständig. Der KeyMaster verwaltet die benutzereigenen

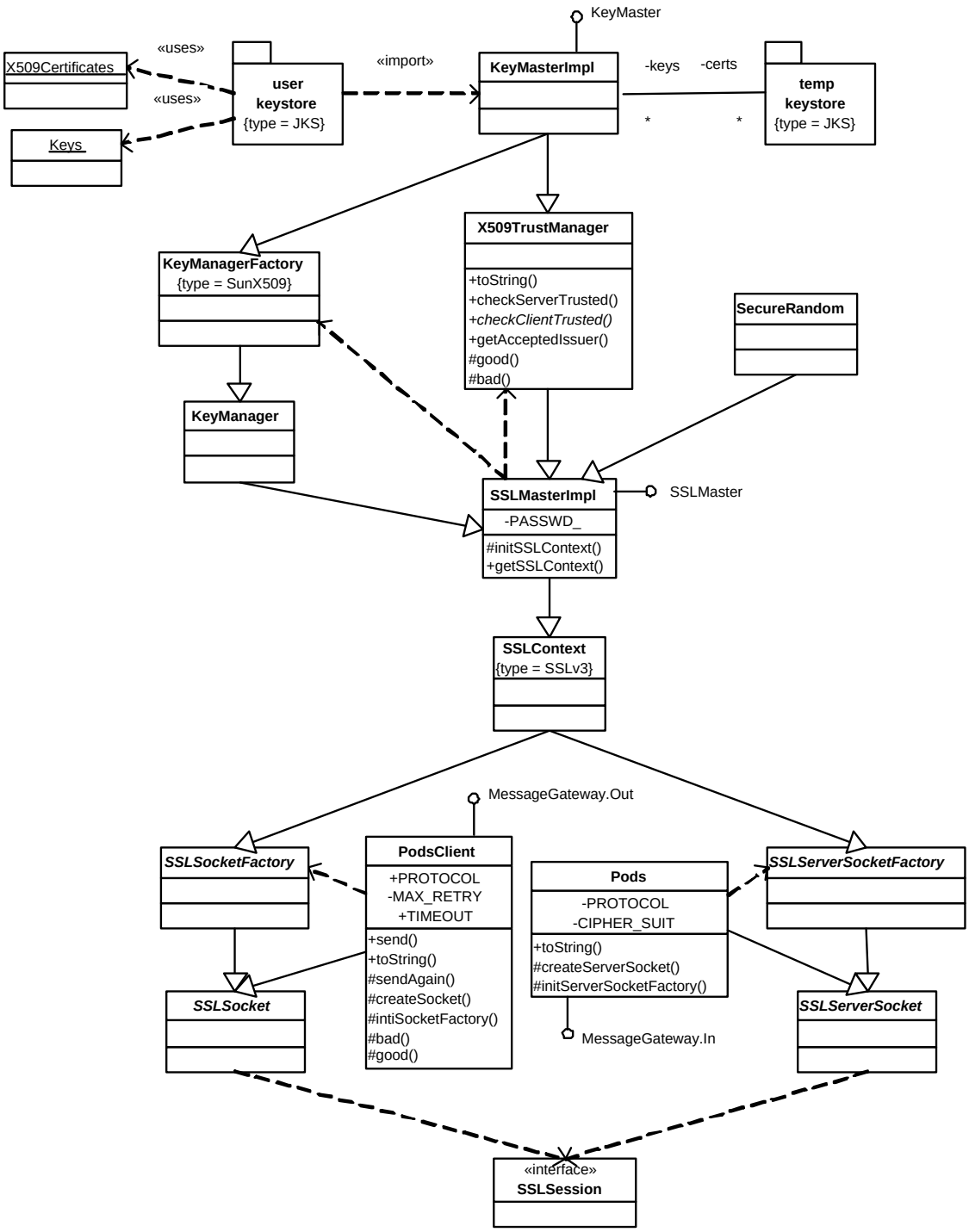

Abbildung 6.1: SSL-Mechanismus in SeMoA

Keystores und gibt auf Anfrage eine Zertifikatskette zurück, die daraufhin durch geeignete Methode von SeMoA geprüft wird. Aus diesem Grund muß der TrustManager so angepasst werden, daß die Zertifikatsprüfung über SeMoA-eigene Mechanismen läuft.

Die Neuimplementierung des `TrustManager` beinhaltet die Implementierung der Methoden `checkClientTrusted`, `checkServerTrusted` und `getAcceptedIssuer`. Die `checkServerTrusted`-Methode wird so umgeschrieben, daß die bei ihrem Aufruf übergebene Zertifikatskette vom `CertificateChainVerifier` geprüft wird. Der Verifier wird zuerst durch den `KeyMaster` initialisiert, daraufhin wird ihm die Kette übergeben. Dazu muß durch den Aufruf der Methode `getKeyMaster` eine Referenz auf den `KeyMaster` erzeugt werden. Nach einer erfolgreichen Prüfung durch den Verifier, wird die Anzahl der erfolgreich bzw. erfolglos geprüften Zertifikate durch den Aufruf der Methode `good` bzw. `bad` erhöht. Die Zertifikatsprüfung bei `checkClientTrusted` anderseits, wird durch den Aufruf von `checkServerTrusted` durch diese Methode durchgeführt.

Die Aufgabe der `getAcceptedIssuer`-Methode ist die Rückgabe, der über `KeyMaster` gewonnenen Liste der Zertifikate, denen vertraut werden kann. Falls diese Liste nicht ermittlet werden kann, wird eine leere Liste zurückgegeben.

Über die `toString` Methode können Daten über die vorhandenen vertrauenswürdigen Zertifikate, wie auch über erfolgreich und erfolglos geprüfte Zertifikate ausgegeben werden.

6.1.4 MessageFilter-Pipeline

Die Implementierung der `MessageFilter`-Pipeline verlangt die Definition einer allgemeinen Schnittstelle für Nachrichtenfilter `MessageFilter` und den Einbau des Filterungsmechanismus in die entsprechende Klassen. Außerdem zur Demonstration wird eine beispielhafte Filterstufe realisiert, die vor dem Senden in jede Nachricht einen Zeitstempel einfügt und ihn beim Empfang auf der Gegenseite wieder ausliest und protokolliert.

MessageFilter

`MessageFilter` (siehe Abb. **??**) ist eine öffentliche Schnittstelle, die es ermöglicht, zwei Typen von Filter zu definieren. Zum einen sind das Filter für eingehende Nachrichten, zum anderen Filter für ausgehende Nachrichten. Diese Filter werden unterschieden durch die Implementierung zweier interner Schnittstellen, und zwar `In` für eingehende und `Out` für ausgehende Filter. Außer diesen Definitionen, enthält die `MessageFilter` Schnittstelle zwei Methoden für Nachrichtenbehandlung.

Die `filter`-Methode wird verwendet, um die eigentliche Aufgabe des Filters aufzurufen. Als Ergebnis der Filterung wird ein Rückgabewert (`ErrorCode`) zurückgegeben, wobei dieser Rückgabewert folgende Werte annehmen kann:

`ErrorCode.OK` - alles in Ordnung,

`ErrorCode.MESSAGE_MODIFIED` - die Nachricht wurde durch die Filteroperation verändert,

`ErrorCode.REJECT` - die Nachricht wurde abgelehnt und ein Fehlerreport wird gesendet.

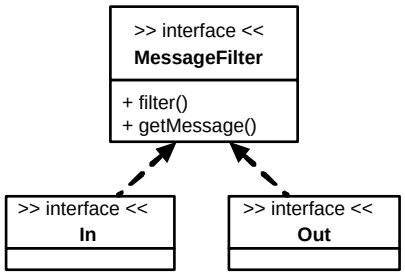

Abbildung 6.2: UML-Diagramm der `MessageFilter` Klasse

In dem Fall, des `ErrorCode.MESSAGE_MODIFIED`, kann man sich mit der Methode `getMessage`, die neue Nachricht von der Nachrichtenfilter-Pipeline zurückgeben lassen.

Um das Filtern der Nachrichten zu ermöglichen, müßten noch die entsprechenden Aufrufe, in die Klassen `InBox` und `OutBox` eingebaut werden. In der `OutBox` wird das Filtern nach dem Empfang der Nachricht, in der `InBox` vor der Auslieferung an den Agenten, eingebaut. Das Filtern in beiden Klassen wird durch die `filterMessage`-Methode durchgeführt. Diese Methode ruft allerdings zuerst die `getMessageFilters` Methode auf, die eine Liste der installierten Filter zurückgibt. Bei den ausgehenden Filtern werden alle Filter aus dem SeMoA-`Environment`, die unter dem `WhatIs`-Schlüssel `MESSAGE_FILTERS_OUT` publiziert sind, gesucht und zu einer Liste zusammengefügt. Bei den eingehenden Filtern alle mit dem `WhatIs`-Schlüssel `MESSAGE_FILTERS_IN`. Wenn die Liste der Filter vorhanden ist, kann mit dem Filtern angefangen werden. Dazu werden alle Filter, einer nach dem anderen, aufgerufen und ihr Rückgabewert (`ErrorCode`) beobachtet. Falls in irgendeinem Filter ein Fehler passiert, wird das Filtern abgebrochen und die eine Fehlermeldung ausgegeben. In dem Fall, daß eine Nachricht geändert wurde, wird sie durch die geänderte ausgetauscht und das Filtern wird fortgesetzt. Wenn alle Filter durchschritten wurden, wird die Nachricht durch die `filterMessage`-Methode an die `InBox` bzw. `OutBox` zurückgegeben.

AddTimeFilter und GetTimeFilter

Die Klassen `AddTimeFilter` und `GetTimeFilter` Klassen sind als beispielhafte Realisierung einer Filterstufe gedacht, die einer Nachricht beim Versenden mit einem Zeitstempel versehen und diesen beim Nachrichtempfang wieder auslesen.

Die Aufgabe des `AddTimeFilter` ist es dabei, den Zeitstempel einer Nachricht hinzufügen. Der Zeitstempel, der aus Datum und Zeitangabe der aktuellen Systemzeit besteht, wird als ein String fester Länge (26 Zeichen) vorne an die Nachricht angehängt. Eine Änderung der Nachricht in dieser Filter-Instanz wird der Filterpipeline durch Zurückgabe von `ErrorCode.MESSAGE_MODIFIED` signalisiert. Die Filterpipeline kann die neue Nachricht mittels der `getMessage`-Methode von dem Filter holen.

Die `GetTimeFilter`-Klasse extrahiert wiederum den an die Nachrichten angefügten Zeit-

stempel. Für den Fall, daß eine Nachricht empfangen wurde, die zuvor nicht den `AddTimeFilter` durchlaufen hat, wird ein String fester Länge entnommen und geprüft, ob dieser mit dem Schlüsselwort `Time:` beginnt. Falls dies der Fall sein sollte, dann wird der Zeitstempel aus der Nachricht herausgenommen und in eine spezielle Logdatei geschrieben. Die Signalisierung der Änderung und Weitergabe der neuen Nachricht findet statt, wie beim `AddTimeFilter`.

6.1.5 Weitere Anpassungen

Die Klassen `InBox`, `OutBox` und `GetTimeFilter` werden zusätzlich durch einen Mechanismus ergänzt, welcher die Speicherung von Protokoll-Nachrichten, ermöglicht. Dieser besteht den beiden Methoden `setLog` und `log`. Die `setLog`-Methode kreiert eine Datei auf dem lokalen System, in die die Protokoll-Nachrichten geschrieben werden sollen. Dieser Methode wird beim Publizieren der jeweiligen Klasse im SeMoA-Environment der Dateiname als Parameter übergeben. Die Aufgabe der `log`-Methode ist es, die Nachricht, die dieser Methode übergeben wird zusammen mit einem Zeitstempel in die von der `setLog`-definierten Logdatei zu schreiben.

Außer diesen Änderungen war noch eine Ergänzung der Klasse `ErrorCode` nötig. Um das Erkennen von Änderungen an gefilterten Nachrichten zu unterstützen, mußte der oben eingeführte Fehlercode bzw. Signalisierungsbezeichner `MESSAGE_MODIFIED` in die Klasse `ErrorCode` eingefügt werden. Neben diesem Fehlercode wird die Klasse noch um den Fehlercode `HOST_UNREACHABLE` ergänzt, um die Fehlerbenachrichtigung im Fall eines erfolglosen Verbindungsaufbaus in `PodsClient` zu unterstützen.

6.2 Sichere Kommunikation in SeMoA

Als nächster Schritt werden die neu implementierten Komponenten des Kommunikationsmechanismus in die SeMoA-Plattform integriert. In diesem Abschnitt wird beschrieben wie diese Komponenten in SeMoA integriert und wie sie initialisiert werden, um den Betrieb der sicheren Kommunikation zu ermöglichen.

6.2.1 Einbettung in SeMoA

Da die SeMoA-Plattform ein Produkt des Fraunhofer Institut für Graphische Datenverarbeitung ist, befinden sich alle Java-Klassen in *Java-Packages* mit dem Präfix `DE.FhG.IGD`. Dabei sind die Basisdienste von SeMoA unter dem Präfix `DE.FhG.IGD.semoa` zu finden. Die für die sichere Kommunikation entwickelten Klassen werden in *Packages* unterhalb dieses Klassenpfades eingebettet. Die Klassen, welche Kommunikationsdienste betreffen, liegen im `comm`-Package, die sicherheitsbetreffenden Klassen im `security`-Package, usw.. Die genaue Position der Klassen wie die Angaben darüber, welche Klassen und Schnittstellen sie implementieren, kann man der Tabelle **??** entnehmen.

Klassenpfad	Klasse/Objekt	implementiert
DE.FhG.IGD.semoa.comm	Pods PodsClient MessageFilter AddTimeFilter AetTimeFilter	MessageGateway.In MessageGateway.Out MessageFilter MessageFilter.Out MessageFilter.In
DE.FhG.IGD.semoa.net	SSLMasterImpl	SSLMaster
DE.FhG.IGD.semoa.security	X509TrustManager	javax.net.ssl. X509TrustManager

Tabelle 6.1: Einbettung der Klassen in SeMoA

Um den normalen Betrieb dieser Klassen zu ermöglichen, müssen noch einige Änderungen an den Initialisirungsdateien von SeMoA vorgenommen werden. Die SeMoA-Plattform wird beim starten durch eine Fülle verschiedener Initialisierungsdateien konfiguriert. Sie enthalten Angaben über Dienste die gestartet werden und die Parameter mit denen diese initialisiert werden. SeMoA kann auf diese Art vollständig flexibel konfiguriert werden.

Die durch die Implementierung des Mechanismen für sichere Kommunikation neu hinzugekommene Dienste müssen ebenfalls geeignet initialisiert und gestartet werden (siehe Tabelle **??** im Anhang). Dies beinhaltet die Definition eines geeigneten Ports für sichere Kommunikation. Dieser wird mit dem Bezeichner PODS_PORT in der benutzereigenen Konfigurationsdatei (z.B. Benutzername.conf) initialisiert. Dieser Port ist genauso wie die anderen Ports (z.B. Port für Migration oder unsichere Kommunikation) an einen bestimmten Benutzer gebunden.

Die Kommunikations- und SSL-Fähigkeit der Plattform wird durch Aktivierung der bestimmten Parameter ermöglicht. Dies geschieht durch das Setzen der AGENT_COMM und SSL Variablen in dem Skript rc.global auf den Wert enabled. Das Setzen dieser Variablen auf enabled ermöglicht, daß die Kommunikations- und SSL-Dienste gestartet und benutzt werden können. Für den Zweck der Protokollierung der Kommunikation, wird in der gleichen Datei noch die COMM_LOG_FILE-Variable mit dem Namen der Logdatei initialisiert. Mit der Variable TIME_LOG_FILE wird der Name der Logdatei, für die Zeitstempel, die der GetTimeFilter aus empfangenen Nachrichten entnimmt, initialisiert.

Die Schlüsselbezeichner, welche die Position und Bezeichnung der Dienste in Environment definieren, werden in der Datei whatis.conf definiert. So stehen die Bezeichner SSL_MASTER und SSL_TRUSTMASTER für Position dieser Dienste, die im Environment in dem Verzeichnis /security auffindbar sind. Pods und PodsClient sind hingegen unter derselben Bezeichnung PODS, aber verschiedenen Verzeichnissen in Environment zu finden (/comm/inbox bzw. /comm/outbox).

Das Starten und Initialisieren der Sicherheitskomponenten geschieht in dem Skript mit Namen rc.security. Hier wird zuerst der JSSE-Provider vom *SUN* (SunJSSE) aktiviert.

Weiterhin werden die von SSL benötigte Dienste `SSLMaster` und `TrustMaster` im `SeMoA-Environment` publiziert.

In dem Skript `rc.network` wird das Starten der Netzwerkdienste angestoßen. Diese Datei enthält Kommandos zum Publizieren der Dienste für die Migration, Kommunikation und Lokalisierung. Die Dienste für die sichere Kommunikation `Pods` und `PodsClient`, werden hier mit entsprechenden Optionen ebenfalls publiziert. Außer standardmäßig, benötigten globalen Optionen, müssen die Dienste mit einigen dynamischen Optionen initialisiert werden. So wird die `Pods`-Klasse durch das Publizieren auch mit dem Port für sichere Kommunikation (`setPort`), dem Wert für maximale Anzahl der gleichzeitigen Verbindungen (`setCapacity`), wie auch mit der maximalen Größe einer Nachricht (`setMax`, in Bytes) initialisiert, während die Klasse `GetTimeFilter` mit dem Namen der benötigten Logdatei initialisiert wird.

6.2.2 Betrieb

In Anlehnung an in den Abbildung **??** (Anhang) dargestellten Kommunikationsmechanismus kann der Vorgang der Kommunikation in zwölf wesentliche Schritte unterteilt werden. Dieser Vorgang entspricht größtenteils dem in Abschnitt **??** beschriebenem Kommunikationsprozeß. Neu hinzugekommen sind die Schritte (4) und (11) bzw. die Filterpipeline für Nachrichten, die Dienste für sicheren Nachrichtentransport `Pods` und `PodsClient`, sowie die Methode `sendSecure`.

Am Anfang des Kommunikationsprozesses (Schritt (2)) hat der Agent jetzt die Wahl zwischen der `send`- und `sendSecure`-Methode. Der Vorteil von `sendSecure` gegenüber der `send` ist der, daß zum einen die Kommunikationsart (unsicher, sicher oder kombiniert) jetzt durch das Setzen eines Parameters definierbar ist und somit kein explizites Setzen des Protokollsbezeichners in der Kontaktadresse nötig ist, und zum Anderen, daß falls sichere Kommunikation nicht erfolgreich war, die Nachricht trotzdem ohne weiteren Eingriff des Agenten unsicher verschickt werden kann.

Das Filtern der ausgehenden Nachrichten wird in der `OutBox` angestoßen und zwar nachdem die Suche des Agenten auf lokaler Plattform erfolglos war (Schritt (4)), d.h. wenn sicher ist, daß die Nachricht die Plattform verlassen wird. Die gefilterte Nachricht wird der `OutBox` von der Filterpipeline zurückgegeben, woraufhin die `OutBox` sie weiter an `Pods` verschickt. Das Filtern der Nachrichten in der `InBox` findet hingegen statt, bevor die Nachricht an den `Spooler` des Agenten (Schritt (11)) ausgeliefert wird.

Die Klassen `Pods` und `PodsClient` bauen jetzt anstatt einer normalen Socket-Verbindung, eine SSL-Verbindung auf. Wenn eine Nachricht vorliegt bzw. die `send`-Methode des `PodsClient` aufgerufen worden ist, beginnt der `PodsClient` durch die Erzeugung eines `SSLSocket` mit dem Aufbau einer SSL-Verbindung. Der durch die Klasse `Pods` auf der entfernten Plattform initiierte `SSLServerSocket` antwortet und der SSL-Handshake wird angestoßen. Die Sicherheitsparameter werden ausgehandelt und die Zertifikate der beiden Platt-

formen werden gegenseitig durch den Authentisierungsprozeß verifiziert. Nach der Erzeugung eines gemeinsamen Schlüssels für die Verschlüsselung wird die Nachricht ausgetauscht (Schritt (7)). Die erzeugte Verbindung wird danach für vordefinierte Zeit offen gehalten, für den Fall, daß mehrere Nachrichten zwischen den Plattformen übertragen werden müssen. Die Sicherheitsparameter bzw. Sessions werden vom SSL-Mechanismus für diesen Zweck zwischengespeichert. Falls eine Rückverbindung zwischen den Plattform aufgebaut werden sollte (z.B. wenn eine Antwort des empfangenden Agenten vorliegt), wird die noch bestehende Session zwischen den Plattformen für die Verbindung verwendet, ohne den zeitintensiven Handshake-Prozess noch einmal aufzubauen.

Kapitel 7

Testen und Bewertung

Der in den letzten Kapiteln konzipierte und realisierte Mechanismus für sichere Kommunikation wird in diesem Kapitel ausführlich auf seine Funktionsfähigkeit und Tauglichkeit getestet. Hier geht es darum die Tauglichkeit der Implementation unter normalen Bedingungen zu testen sowie Faktoren, die für zuverlässige Kommunikation nötig sind, zu identifizieren bzw. zu bestimmen.

In dem ersten der folgenden beiden Abschnitten wird die Testumgebung mit dem für den Testzweck aufgebauten Kommunikationszenario und zugehörigen Parametrisierung vorgestellt. Der zweite Abschnitt ist für die Darstellung der Ergebnisse und ihre Diskussion reserviert.

7.1 Testumgebung und Testszenario

Um den Kommunikationsmechanismus vollständig zu testen, sollten Tests durchgeführt werden, die möglichst reellen Bedingungen der Agentenwelt entsprechen, wie das zum Beispiel in einer heterogenen Umgebung, wie dem Internet der Fall ist. Da SeMoA ein Projekt ist, das sich noch in der Entwicklungsphase befindet, muß man sich hier mit Laborbedingungen zufrieden geben. Die Kommunikation wurde aus diesem Grund im Entwicklungsnetzwerk des Fraunhofer-IGD getestet. Dieses besteht aus mehreren Rechner, die verschiedene Parameter aufweisen.

Zum Testen wurde folgendes Szenario angewendet:

Zwei speziell für Kommunikationszwecke programmierte Agenten werden auf einem beliebigen Rechner gestartet. Sobald beide aktiv sind, tauschen sie ihre implizite Namen aus. Dies ist an dieser Stelle nötig, damit die Kommunikation durch richtige Adressierung der Kommunikationspartner zustande kommen kann. Daraufhin verlassen sie ihre initiale Plattform und migrieren zu ihren Zielen. Zu diesem Zweck wählt jeder Agent per Zufallsprinzip 10 verschiedene Ziele seiner Reise durch das Netz. Das Netz besteht dabei aus vier Rechner mit vergleichbaren Kenndaten (siehe Tabelle **??**).
Jedes Mal wenn die Agenten auf einem neuem ein Rechnersystem angekommen sind, ver-

CPU	333 MHz
RAM	256 MB
Model	Ultra 10
Betriebssystem	SUN Solaris
Java Version	1.4.1_01

Tabelle 7.1: Kenndaten der Rechner in der Testumgebung

suchen sie, mit ihrem Kommunikationspartner zu kommunizieren. Sie schicken jeweils eine Nachricht bestimmter Länge als Anfragenachricht an den anderen Agenten. Dabei wird ATLAS zur Bestimmung der Position des Partners benutzt. Nachdem sie die Nachricht verschickt haben, warten die Agenten solange, bis sie eine Antwortnachricht bzw. eine vom anderen Agenten abgeschickte Anfragenachricht erhalten haben. Gemessen wird dabei die Gesamtzeit die benötigt wird, um die Anfragenachricht zu versenden und die Antwortnachricht zu erhalten, also die Zykluszeit der Kommunikation. Dieser Testmodus mit Zykluszeiten wurde gewählt, da bei der Messung der einzelnen Zeiten für das Versenden und Empfangen der Nachrichten wegen unsynchonisierter Uhren auf den verschiedenen Rechnern sich andernfalls Abweichungen von tatsächlichen der Zeiten ergeben würden. Getestet wird dabei zum Vergleich die normale Kommunikation ohne der Absicherung durch SSL (Pod) und die Kommunikation mit Absicherung und verschieden starken Verschlüsselungsmechanismen bzw. Cipher Suites[1] (siehe dazu Tabelle **??**). Die Nachrichtenlängen wird von kurzen Nachrichten von 60 Bytes Länge über mittellangen der Länge 1 KB, bis zu den Langen mit der Länge von 10 KB variiert. Für jede Nachrichtenlänge werden 10 Testdurchläufe durchgeführt und somit theoretisch 200 Zeiten für jeden Test ermittelt.

Test	Cipher Suite	Schlüssellänge
Test 1	keine Absicherung	0
Test 2	SSL_DHE_DSS_WITH_DES_CBC_SHA	64
Test 3	SSL_RSA_WITH_RC4_128_SHA	128
Test 4	SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	192

Tabelle 7.2: Durchgeführte Tests

7.2 Ergebnisse und Bewertung

Die in Tabelle **??** dargestellte Zeiten ergaben sich als Mittelwerte der getesteten Kommunikationsarten, in Bezug auf die in Tabelle **??** aufgeführten Tests. Diese Ergebnisse werden noch

[1] Für das Test 3 war die Generierung eines RSA-Schlüsselpaares der Länge 2048 Bit und seine Integration in die SeMoA-Keystores notwendig. Dieses wird von JSSE für Authentisierungs- und Schlüsselaustauschzwecke bei Benutzung von RSA verlangt.

einmal durch die graphische Darstellung in der Abbildung **??** verdeutlicht.

Bei der Berechnung des Mittelwertes wurden die Kommunikationszeiten die am Anfang des jeweiligen Testes gemessen wurden, bzw. nach dem Starten der Agenten und ihrer ersten Migration, nicht berücksichtigt. Wegen unterschiedlichen Start- und Migrationszeiten der Agenten, die wiederum in die Zykluszeiten der ersten Kommunikation einfließen (die Agenten warten bei der Kommunikation auf einander), würde der Durchschnittswert der Kommunikationszeiten ansonsten verfälscht. Weiterhin ist es bei einer Testumgebung, die aus nur vier Rechner besteht durchaus möglich, daß sich die beiden Agenten mehrmals auf dem gleichen Rechner befinden und dadurch lokal kommunizieren. Da es die Absicht des Testens war, die Kommunikationszeiten zwischen verschiedenen Rechnern zu messen, wurden die Zeiten für die Kommunikation, in solchen Fällen, ebenfalls nicht berücksichtigt.

	60 Bytes	1 KB	10 KB	Δ Zeiten	Δ Fehler
Test 1	213 ms	229 ms	232 ms	225 ms	4,5%
Test 2	682 ms	671 ms	780 ms	711 ms	3,5%
Test 3	778 ms	749 ms	848 ms	792 ms	4,6%
Test 4	740 ms	746 ms	820 ms	769 ms	2,1%

Tabelle 7.3: Ergebnisse der getesteten Kommunikation

Aus den Ergebnissen der Tabelle **??** sind eindeutige Schlüsse zu ziehen. Die sichere Kommunikation braucht deutlich mehr Zeit (durchschnittlich Faktor 3,4) als die ungesicherte Kommunikation. Dies ist vor allem durch die zeitintensive Prozesse bei der SSL-Übertragung zu erklären. Dabei ist das Handshake mit Abstand der kostspieligste Teil einer SSL-Übertragung (Größenordnung der Kommunikationszykluses mit Handshake 2000 ms). Dank dem Caching-Mechanismus bei SSL muss das Handshake nicht bei jeder Verbindung durchgeführt werden, sondern es können die schon ausgehandelte Parameter aus vorausgegangenen Verbindungen wiederverwendet werden. Dies wirkt sich enorm auf die durchschnittliche Kommunikationszeit aus.

Aus Abbildung **??** ist ein interessanter Effekt ersichtlich: die Kommunikationsvorgänge bei Nutzung des komplexesten Cipher Suite mit 3DES (Test 4) kamen durchschnittlich schneller zustande, als das bei dem Cipher Suite mit einer einfacheren Verschlüsselung bzw. mit RC4 (Test 3) der Fall war. Dieses geht eindeutig auf das Konto des Handshake, das beim Cipher Suit in Test 2 etwas länger braucht, als das Handshake beim 3DES. Es läßt sich vielleicht dadurch erklären, daß RSA für Authentisierung und dem Schlüsselaustausch längere Schlüssel verwendet als die Diffie-Hellman/DSS Kombination (2048 gegenüber 1024 Bit).

Die Zahlen der Spalte "Δ Fehler" zeigen die Ausmaßen eines weiteren Effektes, der beim Testen beobachtet wurde. Sie zeigen den Anteil der Fälle, bei denen ein Kommunikationsvorgang nicht im erwarteten Anfrage/Antwort-Zyklus zustande kam, und deshalb keine Zeiten gemessen wurden. Solche Vorfälle ereigneten sich in Fällen, bei denen ein Agenten die Kommunikation initiiert, während sich der andere Agent gerade im Prozess der Migration be-

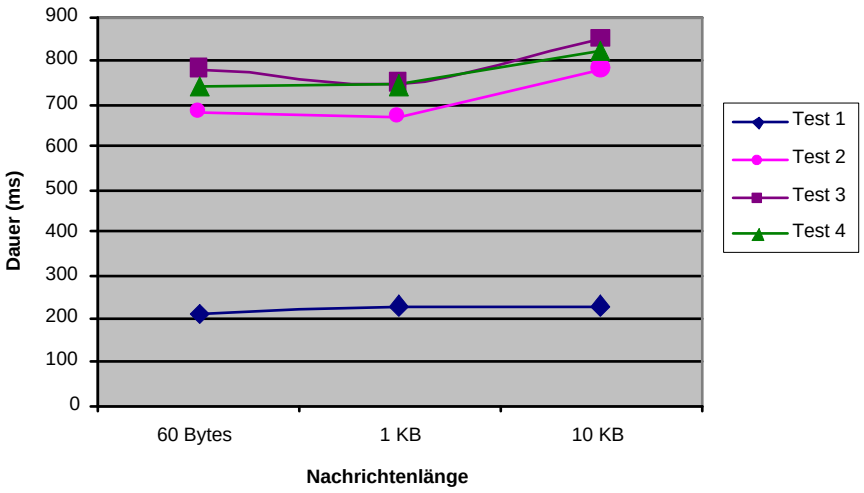

Abbildung 7.1: Graphische Darstellung der Testergebnisse

fand. Dieser Agent hat die Plattform noch nicht verlassen, und die Nachricht wird an seinen `Spooler` übermittelt. Da sich der zweite Agent gerade in Migration befindet, wird die Nachricht mit einer `CommunicationException` und der Fehlermeldung "bad time to deliever" nicht zugestellt. In diesem Fall, da der Fehlercode "RECIPIENT_UNREACHABLE" an den sendenden Agenten zurückgegeben wird, kommt keine Kommunikation zustande. Die Anzahl solcher Vorfälle nimmt aber mit wachsenden Zeiten und Komplexität der Kommunikation tendenziell ab, und liegt schon bei dem Test 4 im Bereich des Akzeptablen.

Bei dem angewendeten Testszenario wurde von dem "worst case" der Kommunikation ausgegangen, d.h. die Agenten haben versucht zu kommunizieren und sind anschließend gleich weiter zur nächsten Plattform migriert. Im Normalfall kann man davon ausgehen, daß die Verweilzeit eines Agenten bei seiner Aufgabenerfüllung auf einer Plattform etwas größer ist, als es im Test der Fall war. In diesem Fall wird auch die Fehleranfälligkeit beim Nachrichtenempfang weiter sinken. Durch eine verbesserte Konzeption des Sende-/Empfangsmechanismen könnten solche Fehlzustände sogar vollständig vermieden und der Kommunikationsdienst dadurch optimiert werden. Abschließend kann man sagen, daß der entworfene Kommunikationsdienst seine Aufgaben bezüglich der sicheren Nachrichtenaustausch erfüllt. Es wird sich allerdings erst durch zukünftige Anwendungen im Internet zeigen, in wie weit sich die gemessene Zeiten für eine "real time" Kommunikation der Agenten eignen. Der Einsatz der Sicherheitsmechanismen ist jedoch immer mit gewissen Kosten (Zeit, Leistung) verbunden. Es bleibt zu hoffen, daß die ständig wachsende Rechenleistung den Aufwand dieser Zusatz-Operationen.

Teil III

Resümee

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

Das Agentensystem SeMoA stellt eine Entwicklung dar, in der den Anstrengungen der Forscher am Fraunhofer-IGD Sicherheit als zentraler Punkt des Systems zugrunde liegt. Das Ziel dabei ist es ein funktionsfähiges System aufzubauen, das den meisten heutigen und manchen potenziellen zukünftigen Sicherheitsbedrohungen stand hält. In dieser Arbeit wurde deshalb ein weiterer wichtiger Aspekt der Sicherheit in SeMoA behandelt, nämlich die Sicherheit bei der Agentenkommunikation. Bisher war es in SeMoA lediglich möglich, Nachrichten in unsicherer Klar-Text-Form zu übertragen. Dieses ist für eine ernsthafte Anwendung der SeMoA-Agenten in einer unsicheren Umgebung wie dem Internet nicht ausreichend. Aus diesem Grund wurde als Ziel dieser Arbeit der Entwurf eines auf einem bestimmten Sicherheitsmechanismus aufbauenden Konzeptes für sichere Kommunikation und seine Implementation und Integration in SeMoA definiert.

Hierzu war es im ersten Teil der Arbeit (Kapitel 2 bis 4) nötig, die Grundlagen, der mit der Thematik der Arbeit verbundenen Bereiche, zu erläutern. Hier wurden die bedeutendsten Begriffe der Mobile-Agenten-Welt, der Kommunikation zwischen Mobilen Agenten und der Sicherheit erörtert. Angefangen mit der Beschreibung von möglichen Bedrohungen, hin bis zu den Sicherheitsmechanismen, wurde bei der Einführung der Grundbegriffe besonderes Augenmerk auf die Sicherheit in der Kommunikation gerichtet.

Das Konzept für die sichere Kommunikation wurde detailliert in Kapitel 5 beschrieben. Hier wurden zuerst, die an das Konzept gestellten Anforderungen vorgestellt. Als nächstes wurden die Sicherheitslösungen, durch dessen Anwendung das Konzept realisiert werden sollte, ausführlich diskutiert. Aufgrund der Vorteile gegenüber den anderen Mechanismen ist die Wahl auf die Absicherung durch den SSL-Mechanismus gefallen. Der Faktor, der zu dieser Entscheidung beigetragen hatte, war vor allem das Verhältnis zwischen Kosten und Nutzen bzw. die Einfachheit der Realisierung im Vergleich zu relativ hoher Sicherheit. Dieses spiegelte sich besonderes in der Unabhängigkeit der Realisierung von anderen Mechanismen und die gute Unterstützung seitens Java (JSSE) wider. Aus diesem Grund fiel die Entscheidung für die Realisierung der sicheren Kommunikation auf einem SSL-gesicherten Sicherheitskanal. Das

Konzept wurde daraufhin, unter Berücksichtigung der von SSL bzw. JSSE benötigten Voraussetzungen und der in SeMoA bereits realisierten Mechanismen für die Kommunikation, entwickelt. Zum Zweck der sicheren Kommunikation wurde ein neues, auf SSL basierendes Protokoll eingeführt und ein aus zwei komplementären Komponenten (Client und Server) bestehender Dienst zum Aufbau einer sicheren Verbindung und der Übertragung der Nachrichten zwischen zwei Agentenplattformen, entworfen. Außerdem wurde in der Konzeptbeschreibung die Notwendigkeit einer Filter-Pipeline für Nachrichten errötet. Außerdem ist eine kurze Beschreibung vergleichbarer Arbeiten auf diesem Gebiet Teil dieses Kapitels.

Wie die einzelnen Klassen des Dienstes zur sicheren Kommunikation implementiert worden sind, wurde im Kapitel 6 erläutert. Durch detaillierte Beschreibung ihrer Methoden wurde hier besonderes auf den Funktionsumfang der Klassen eingegangen. Bei der Realisierung wurde vor allem darauf geachtet, daß diese auf bestehenden Mechanismen aufbaut und eine spätere Erweiterung des Kommunikationmechanismus möglich ist. Die zur Realisierung von SSL benötigten Klassen (`SSLMasterImpl` und `X509TrustMaster`) wurden auf eine Weise implementiert, daß sie die Benutzung derselben SSL-Mechanismus den anderen Diensten ermöglichen. Anschließend wurde in dem Kapitel die Integration der implementierten Klassen, wie auch die Funktionalität der sicheren Kommunikation in SeMoA beschrieben.

Um die Implementation auf ihre Tauglichkeit in SeMoA zu prüfen, wurde ein Testszenario für die Agentenkommunikation entwickelt. Die Kommunikation wurde daraufhin ausführlich und mit verschiedener Parametrisierung getestet. Die Ergebnisse des Testens, wie eine Beschreibung des Testverfahrens wurden in Kapitel 7 gegeben. Das Testen hat die Einsatzfähigkeit des implementieren sicheren Kommunikationdienstes erwiesen.

8.2 Zukünftige Entwicklungsmöglichkeiten

Da die in dieser Arbeit vorgestellte Implementierung der sicheren Kommunikation, nicht als eine endgültige und komplette Lösung für Kommunikation zu verstehen ist, sondern nur als eine Möglichkeit, die Nachrichten auf dem sicheren Wege zu übertragen, werden noch einige Änderungen und Weiterentwicklungen an dem Kommunikationsmechanismus nötig sein.

Eigenimplementierung von SSL

Die Implementierung des Mechanismus für sichere Kommunikation basiert auf JSSE, das mit JDK 1.4 mitgeliefert wird. JSSE ermöglicht eine Realisierung der SSL-Mechanismen, die auf SUN-eigenen Kryptowerkzeugen basieren. So ist es bei JSSE möglich nur den von *SUN* mitgelieferten SunJSSE-Kryptoprovider und die damit verbundenen Sicherheitsmechanismen zu benutzen. Obwohl es nach *SUN* eine Möglichkeit gäbe andere Kryptowerkzeuge zu benutzen (siehe z.B. [**?**]), wird dieses von JSSE nicht zugelassen. Einige Mechanismen bei SeMoA, wie z.B. ATLAS basieren aber auf anderen Kryptoprovidern (bei ATLAS auf dem *CDC-Provider*[1]). Deshalb ist es nötig eine eigene Implementierung des SSL-Mechanismus zu

[1] Der CDC-Provider ist eine von der TU-Darmstadt entwickelte Java-Kryptobibliothek. CDC stellt die meisten heute verfügbaren Kryptowerkzeuge bereit, und ist kompatibel zum Java-Framework JCA/JCE.

realisieren, der mehr Freiheit in dieser Hinsicht zulassen würde.

Integration von Simple Host Information Protocol

Das Simple Host Information Protocol (SHIP) ist eine von den Entwicklern von SeMoA angedachte Lösung für den Austausch von wichtigen Informationen zwischen Agentenplattformen. SHIP stellt somit ein Protokoll dar, mit dem es möglich sein sollte, wichtige Informationen wie z.B. Kommunikationsparameter (Kommunikationsmöglichkeit, benutzte ACLs, Kommunikationsports, usw.) auszutauschen. Dadurch sollte die im Abschnitt **??** vorgestellte Adressierung der Kommunikationspartnern für Agenten vereinfacht werden, indem die Angaben über dem Host des empfangenden Agenten und sein Kommunikationsport automatisch durch die von SHIP ausgetauschten Informationen ergänzt werden. Die Unterstützung von SHIP muß noch in die entsprechenden Kommunikationsmechanismen Integriert werden.

Optimierung und Weiterentwicklung der Kommunikationsmechanismen

Die durch das Testen der implementierten Lösung in den Vordergrund gerückte Fehleranfällig des Sende-/Empfangmechanismen sollte durch eine Optimierung entweder völlig vermieden oder zumindest möglichst gering gehalten werden. Hierbei sollte überlegt werden, ob die Fehlerbehandlung vollständig dem Agenten überlassen wird, oder einige Aufgaben in solchen Fällen der Plattform übertragen werden. So könnte eine Plattform z.B. mehrmals versuchen eine Nachricht zu übermitteln (bei einer fehlgeschlagenen Zustellung) bzw. die Nachricht direkt zum nächsten Host weiter senden (forwarding), falls sich der Agent nicht mehr auf der Plattform befindet.

Es wurden bisher nur Basis-Mechanismen für den Nachrichtenaustausch also auf die unterster Ebene des Kommunikationdienstes bei SeMoA realisiert. Die Interpretation wie auch das Management der Nachrichten wird momentan noch dem kommunizierenden Agenten überlassen. Es sollten noch Mechanismen für die Interpretation der Nachrichteninhalte, z.B. ein ACL-Interpreter, entwickelt werden.

Außerdem ist die Realisierung von Signalisierungsmechanismen beim Nachrichtenempfang denkbar, die eine direkte Reaktion der Agenten ermöglichen würden. Weiterhin wäre noch eine genaue Definition einer Sicherheitspolitik wichtig, die z.B. bestimmen würde, mit welchen Sicherheitsparametern, in welchen Situationen kommuniziert werden dürfte.

Teil IV

Anhang

Anhang A

Abkürzungsverzeichnis

ACC	Agent Communication Channel (FIPA)
ACL	Agent Communication Language
API	Application Programming Interface
CA	Certification Authority
CORBA	Common Object Request Broker
CMS	Cryptographic Message Syntax (PKCS #7)
DES	Data Encryption Standard
DoS	Denial of Service (Angriff)
E-mail	Electronic Mail
FIPA	Foundation for Intelligent Physical Agents
GUI	Graphical User Interface
IETF	Internet Engineering Task Force
IIOP	Internet Inter-ORB Protocol (CORBA)
IP	Internet Protocol
IPSec	Internet Protocol Security
ISO	International Standard Organisation
JAR	Java Archive
JCA	Java Cryptography Architecture

JCE	Java Cryptography Extension
JKS	Java Key Store
JSSE	Java Secure Socket Environment
JVM	Java Virtual Machine
KI	Künstliche Intelligenz
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
KSE	Knowledge Sharing Effort
MAC	Message Authentication Code
MASIF	Mobile Agent System Interoperability Facility
NIST	National Institute of Standards and Technology (USA)
OMG	Object Management Group
PKCS	Public-Key Cryptography Standards
PKI	Public Key Infrastructure
RFC	Request for Comment
RMI	Remote Methode Invocation
RPC	Remote Procedure Call
SeMoA	Secure Mobile Agents
SHA	Secure Hash Algorithm
S/MIME	Secure Multipurpose Internet Mail Extension
SSL	Secure Socket Layer
Tcl	Tool Command Language
TCP/IP	Transfer Control Protocol/Internet Protocol
TLS	Transport Layer Security
UML	Unified Modelling Language
URL	Universal Resource Locator
VPN	Virtual Private Network
WWW	World Wide Web
XML	Extensible Markup Language

Anhang B

Abbildungen und Tabellen

Datei	Ergänzungen
rc.global	setenv SSL enable setenv AGENT_COMM enable setenv COMM_LOG_FILE ${semoa.etc}/${semoa.conf}/comm.log setenv TIME_LOG_FILE ${semoa.etc}/${semoa.conf}/time.log
whatis.conf	SSL_MASTER = /security/sslmaster SSL_TRUSTMANAGER = /security/ssltrustmanager PODS = /comm/inbox/pods PODS_CLIENT = /comm/outbox/pods MESSAGE_FILTERS_IN = /comm/filters/in MESSAGE_FILTERS_OUT = /comm/filters/out
rc.security	java DE.FhG.IGD.semoa.bin.Providers -add JSSE publish DE.FhG.IGD.semoa.net.SSLMasterImpl ${whatIs:SSL_MASTER} publish DE.FhG.IGD.semoa.security.X509TrustManager ${whatIs:SSL_TRUSTMANAGER}
rc.network	publish -detach -spawn -proxy none -key ${whatIs:PODS} -class DE.FhG.IGD.semoa.comm.Pods ; -setPort ${PODS_PORT} -setCapacity 10 -setMax 65535 -run publish DE.FhG.IGD.semoa.comm.PodsClient ${whatIs:PODS_CLIENT} publish DE.FhG.IGD.semoa.comm.AddTimeFilter ${whatIs:MESSAGE_FILTERS_OUT}/1.addtime publish -proxy none -key ${whatIs:MESSAGE_FILTERS_IN}/2.gettime -class DE.FhG.IGD.semoa.comm.GetTimeFilter ; -setLog ${TIME_LOG_FILE}

Tabelle B.1: Ergänzungen in Initialisirungsdateien

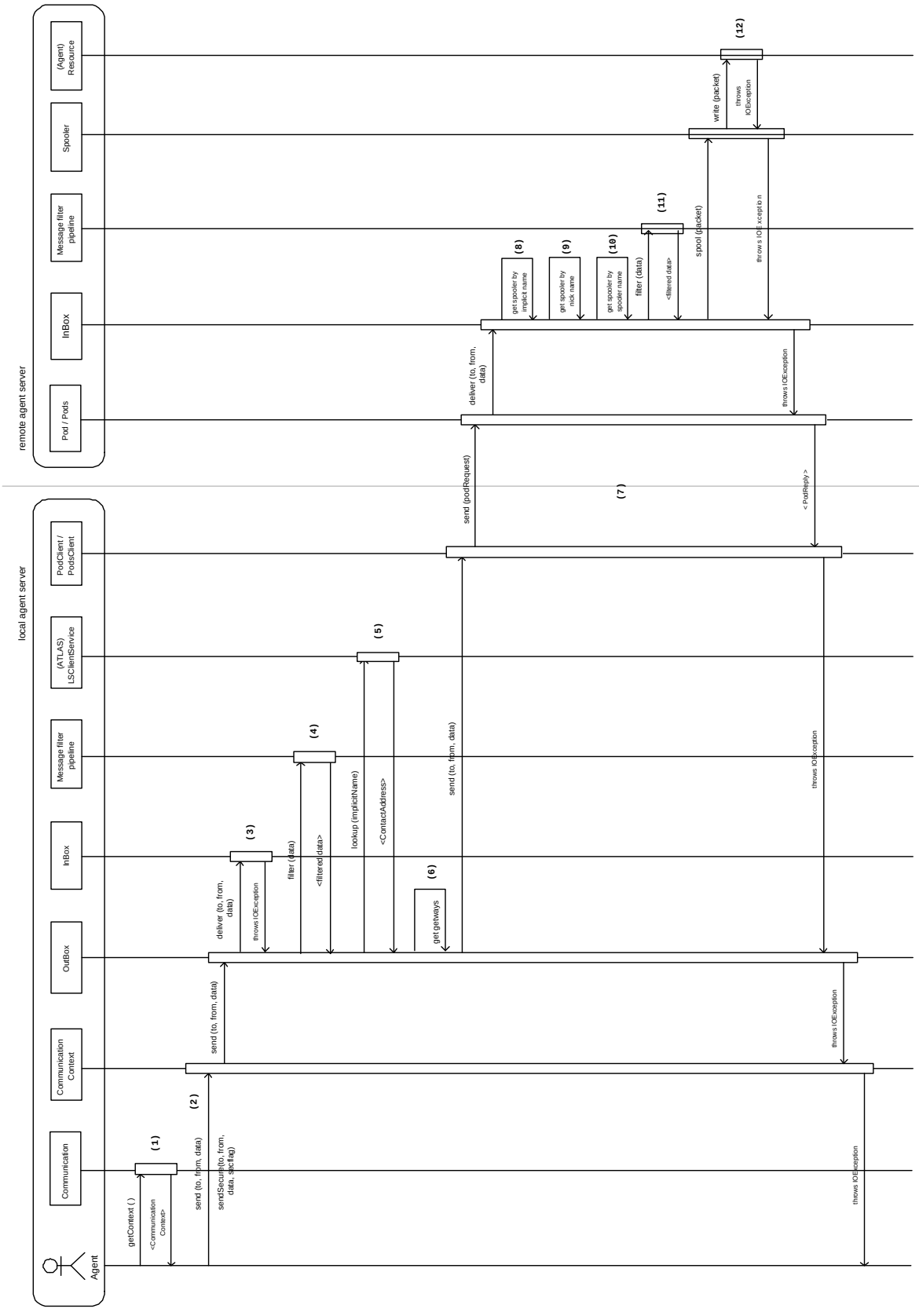

Abbildung B.1: Sichere Kommunikation in SeMoA

Literaturverzeichnis

[1] Fabio Bellifeme, Agostino Poggi, and Giovanni Rimassa. Jade - a fipa-compliant agent framework. Technical report, CSELT and University of Parma, 1999. Internet-Dokument URL: http://sharon.cselt.it/projects/jade/papers/PAAM.pdf.

[2] Brockhaus, editor. *Der Brockhaus Computer und Informationstechnologie*. Bibliographisches Institut & F.A. Brockhaus AG, Mannheim, 2002.

[3] Ingo Busse. *Mobile Agents in Communications*. PhD thesis, Technische Universität Berlin, February 1999.

[4] David Chess, Colin Harrison, and Aaron Kershenbaum. Mobile agents: Are they a good idea? In *Mobile Object Systems: Towards the Programmable Internet, Lecture Notes in Computer Science*, volume 1222. Springer-Verlag, 1997.

[5] Netscape Communications. Introduction to ssl, 1998. Internet-Dokument URL: http://developer.netscape.com/docs/manuals/security/sslin/contents.htm, Netscape Communications Corp.

[6] Tim Dierks and Christopher Allen. Tsl protocol version 1.0, 1999. Internet-Dokument URL: http://www.ietf.org/rfc/rfc2246.txt, RFC 2246, IETF, Network Working Group.

[7] Xinyu Feng, Jiannong Cao, Jian Lü, and Henry Chan. An efficient mailbox-based algorithm for message delivery in mobile agent systems. In *Lecture Notes in Computer Science*, volume 2240. Springer-Verlag, 2001.

[8] Tim Finnin, Richard Fritzson, Don McKey, and Robin McEntire. Kqml as an agent communication language. In *Proceedings of the 3rd International Conference on Information an Knowledge Management (CKIM'94)*, Gaithersburg, MD, USA, 1994. ACM Press.

[9] Tim Finnin, James Mayfield, and Chelliah Thirunavukkarusu. A security architecture for (kqml) agent communication language. In *Intelligent Information Agents Workshop at Fourth International Conference on Information and Knowledge Management (CIKM'95)*, Baltimore, USA, November 1995.

[10] Stan Franklin and Art Graesser. Is it an agent, or just a programm? a taxonomy for autonomus agents. In *Proceedings of the ECAI, Workshop on Agent Theories, Architectures and Languages*, Budapest, Hungary, August 1997. Springer-Verlag.

[11] Alan Freier, Philip Karlton, and Paul Kocher. Ssl protocol version 3.0, 1996. Internet-Dokument URL: http://wp.netscape.com/eng/ssl3/draft302.txt, Netscape Communications Corp., Transport Layer Security Working Group.

[12] Otfried Fries, Andreas Fritsch, Volker Kessler, and Birgit Klein. *Sicherheitsmechanismen*. REMO-Arbeitsberichte, Band 2. R. Oldenbourg Verlag, München, Wien, 1993.

[13] Michael Genesereth and Richard Fikes. *Knowledge Interchange Format (KIF) Reference Manual*. Stanford University Logic Group, Stanford, CA, USA, version 3, technical report logic-92-1 edition, Januar 1992.

[14] Rüdiger Grimm. Sicherheit und datenschutz in der informationstechnik. Vorlesungsskript, J.W.Goethe-Universität Frankfurt, Institut für Informatik. März 1998.

[15] Rüdiger Grimm. *Sicherheit für offene Kommunikation*. Sicherheit in der Informations- und Kommunikationstechnik, Band 4. BI-Wissenschaftsverlag, Leipzig, Wien, Zürich, 1994.

[16] IETF S/MIME Working Group. S/mime working group homepage, 2002. Internet-Dokumenten URL: http://www.imc.org/ietf-smime/index.html, Internet Mail Consortium.

[17] Fritz Hohl. *Sicherheit in Mobile-Agenten-Systemen*. PhD thesis, Fakultät für Informatik der Universität Stuttgart, April 2001.

[18] Fritz Hohl. The Mobile Agent List, Mai 2002. Internet-Dokument URL: http://mole.informatik.uni-stuttgart.de/mole/mal.html.

[19] Wayne Jansen. Countermeasures for mobile agent security. In *Computer Communications, Special Issue on Advances Research and Application of Network Security*, November 2000.

[20] Nicholas R. Jennings and Michael Wolldridge. Intelligent agents: Theory and practice. In *Knowledge Engineering Review 10*, January 1996.

[21] Heechelon Jeon, Charles Petrie, and Mark R. Cutkosky. Jatlite: A java agent infrastructure with message routing. In *IEEE Internet Computing, Vol 4., No. 2*. IEEE Computer Society, March 2000.

[22] RSA Laboratories. Public-key cryptography styndards, pcks #1-#15, 2002. Internet-Dokumenten URL: http://www.rsa.security.com/rsalabs/pkcs/index.html, RSA Security Inc.

[23] Yannis Labrou, Tim Finnin, and Yun Peng. The current landscape of agent communication languages. In *Intelligent Systems, volume 14, number 2*. IEEE Computer Society, 1999.

[24] Yannis Labrou, Tim Finnin, and Yun Peng. The interoperability problem: Bringing together mobile agents and agent communication language. In *Proceedings of the 32nd Hawaii International Conference on System Science*, Maui, Hawaii, USA, 1999. IEEE Computer Society.

[25] SUN Microsystems. *Java Cryptography Architecture (JCA) API Specification & Reference*. Sun Microsystems Inc., August 2002. Internet-Dokument URL: http://java.sun.com/j2se/1.4/docs/guide/security/CryptoSpec.html.

[26] SUN Microsystems. *Java Secure Socket Extension (JSEE) Reference Guide*. Sun Microsystems Inc., Januar 2002. Internet-Dokument URL: http://java.sun.com/j2se/1.4/docs/guide/security/jsse/JSSERefGuide.html.

[27] Dejan Milojcic, Markus Breugst, Ingo Busse, John Campbell, Stefan Covaci, Barry Friedman, Kazuya Kosaka, Danny Lange, Kouichi Ono, Mitsuru Oshima, Cynthia Tham, Sankar Virdhagriswaran, and Jim White. Masif, the omg mobile agent system interoperability facility. In *Second International Workshop, MA'98*, Stuttgart, Germany, September 1998.

[28] Scott Oaks. *Java Security*. Second Edition, O'Reilly & Associates, Inc., Sebastopol, California, 2001.

[29] Paul O'Brien and Richard Nicol. Fipa - towards a standard for software agents. In *BT Technology Journal, Towards engineering intelligent systems, Volume 16:3*, Suffolk, England, July 1998.

[30] Foundation of Intelligent Physical Agents. Fipa communicative act library specification. Technical report, Federation on Intelligent Physical Agents, 2001. Internet-Dokument URL: http://www.fipa.org/specs/fipa00037/XC00037H.html.

[31] Jan Peters. Namensdienste für mobile agenten systeme. Master's thesis, Technische Universität Darmstadt, Dezember 2000.

[32] Stefan Poslad, Phil Buckle, and Rob Hadingham. The fipa-os agent platform: Open source for open standards. In *5th International Conference on Practical Application of Intelligent Agents and Multi-Agent Systems (PAAM200)*, Manchester, UK, April 2000.

[33] Volker Roth. *The $SeMoA^{TM}$ Code Conventions*. Fraunhofer IGD, Darmstadt, Germany, November 2000.

[34] Volker Roth, Ulrich Pinsdorf an Jan Peters, Patric Kabus, and Roger Hartmann. *$SeMoA^{TM}$ Developer's Guide*. Fraunhofer IGD, Darmstadt, Germany, March 2002.

[35] Volker Roth and Mehrdad Jalali. Concepts and architecture of a security-centric mobile agent server. In *Proc. Fifth International Symposium on Autonomous Decentralized Systems (ISADS 2001)*, Dallas, Texas, U.S.A., März 2001. IEEE Computer Society.

[36] Volker Roth and Ulrich Pinsdorf. *$SeMoA^{TM}$ Installation an Configuration Guide*. Fraunhofer IGD, Darmstadt, Germany, May 2001.

[37] Tomas Sander and Christian F. Tschudin. Protecting mobile agents against malicious hosts. In Giovanni Vigna, editor, *Mobile Agents and Security, Lecture Notes in Computer Science, Volume 1419*. Springer-Verlag, March 1998.

[38] Robert Silverman. A cost-based security analysis of symmetric and asymmetric key lengths. Technical report, RSA Laboratories, RSA Security Inc., 2000. Bulletin #13 (revised 2001), Internet-Dokument URL: http://www.rsasecurity.com/rsalabs/bulletins/bulletin13.htm.

[39] William Stallings. *Cryptography and Network Security: Principles and Practice*. Second Edition, Prentice Hall, Upper Saddle River, New Jersey, 1999.

[40] Cisco Systems. Internet protocol security (ipsec) white paper, 2000. Internet-Dokument URL: http://www.cisco.com/warp/public/cc/so/neso/sqso/eqso/ipsec_wp.pdf, Cisco Systems Inc.

[41] Juan Jim Tan, Leonid Titkov, and Constantinos Neophytou. Securing multi-agent platform communication. In *Second International Workshop on Security of Mobile Multiagent Systems (SEMAS-2002) at 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, Bologna, Italien, Juli 2002.

[42] Stephen Wade. *An Investigation into the use of the Tuple Space Paradigm in Mobile Computing Environment*. PhD thesis, Computing Depertment of Lancaster University, England, September 1999.

[43] Dirk Westhoff. *Ein dezentrales Agentensystem unter Berücksichtigung von mehrseitiger Sicherheit*. PhD thesis, Fakultät für Informatik der FernUniversität-Gesamthochschule in Hagen, Oktober 2000.

[44] Min Zhang, Ahmed Karmouch, and Roger Impey. Adding security features to fipa agent platforms. In *Mobile Agents for Telecommunication Applications, Third International Workshop (MATA 2001)*, Montreal, Canada, August 2001. Springer-Verlag.