Technische Universität Darmstadt
Fachbereich Informatik

Fraunhofer Institut für
Graphische Datenverarbeitung


**Diplomarbeit**

# Namensdienste für Mobile Agenten Systeme

von

**Jan Peters**
Matrikelnummer 370358

Technische Universität Darmstadt
Fachbereich Informatik
Fachgebiet Graphisch-Interaktive Systeme
Wilhelminenstraße 7
64283 Darmstadt

Betreuer: Dipl.-Inform. Volker Roth

Prüfer: Prof. Dr.-Ing. J.L. Encarnação

**Aufgabenstellung für die Diplomarbeit des**
**Herrn cand.-Inform. Jan Peters**
**Matrikel-Nr. 370358**

## Thema: "Namensdienste für Mobile Agenten Systeme"

Mobile Software Agenten sind Programme, die zusammen mit Zustandsinformationen und Daten autonom in einem Netzwerk wandern können. Dazu ist eine Infrastruktur aus Agentenservern erforderlich, die den Transport der Agenten zwischen den verschiedenen Wirtssystemen übernimmt. Um mobile Agenten in Multi-Agenten Anwendungen zu koordinieren und den Austausch von Nachrichten zwischen solchen Agenten unabhängig von deren Position zu ermöglichen, wird ein *Namensdienst* benötigt, der die Positionswechsel der Agenten verfolgt, und auf dessen Grundlage eine transparente Weiterleitung von Nachrichten erfolgen kann. Bislang existiert kein Konzept für einen solchen Dienst, das den besonderen Anforderungen von Mobile Agenten Systemen genügt (hohe Änderungsraten) und auch für große Konföderationen von Agentensystemen skalierbar ist.

Aufgabe dieser Diplomarbeit ist es zu untersuchen, wie solche Namensdienste für Mobile Agenten Systeme beschaffen sein können. Ausgehend von gebräuchlichen Namensdiensten sind die weitergehenden Anforderungen von Mobile Agenten Systemen an solche Dienste zu formulieren, wobei auch Sicherheitsanforderungen berücksichtigt werden sollen. Auf Basis der Voruntersuchungen sollen anschließend Konzepte für einen Namensdienst für Mobile Agenten Systeme entwickelt und untersucht werden. Die Untersuchung schließt die Simulation oder Implementierung eines oder mehrerer geeigneter Ansätze ein.

Darmstadt, den 11.04.2000

**Betreuer:**
Dipl.-Inform. Volker Roth

Prof. Dr.-Ing. J. Encarnação

Darmstadt, Dezember 2000

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

Jan Peters

## Danksagung

An dieser Stelle möchte ich meinen Eltern danken, die mir das Studium ermöglicht haben, aber auch meiner Schwester. Es ist die beste Familie, die ich mir vorstellen kann. Wenn ich Rat, Unterstützung oder etwas Ablenkung brauchte, war sie für mich da.

Weiterhin möchte ich meiner Freundin und den lieben Freunden und Kommilitonen danken, die mich die Zeit über begleitet, und gerade während den letzten Monaten mit viel Geduld eher meine Abwesenheit als Anwesenheit erduldet haben, aber auch mit manchen Anregungen unterstützten.

Dank gebührt auch all denjenigen, die mir ermöglicht haben, drei Monate in den USA am Center for Research in Computer Graphics an meinen Diplom arbeiten zu können. Aus dieser Zeit nehme ich neben unvergesslichen Erfahrungen und der Verbesserung meiner Sprachkenntnisse auch einige sehr gute Freundschaften mit, die mir über den Aufenthalt in Providence hinaus erhalten geblieben sind.

Und selbstverständlich danke ich meinem Betreuer Volker Roth für seine gute Betreuung und die Möglichkeit an solch einem interessanten und aktuellen Forschungsthema arbeiten zu können, das mich über sechs Monate lang auf Trab hielt, aber auch sehr begeistert hat.

Jan Peters Darmstadt, Dezember 2000

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| ACL | Agent Communication Language |
| ACM | Association for Computing Machinery |
| API | Application Program Interface |
| ARPA | Advanced Research Projects Agency |
| ASN.1 | Abstract Syntax Notation One |
| BER | Basic Encoding Rules (für ASN.1) |
| BIND | Berkley Internet Name Domain |
| BOOTP | Boot Protocol |
| CBC | Cypher Block Chaining |
| CER | Canonical Encoding Rules (für ASN.1) |
| CLSID | Class Identifier (unter Microsoft Windows) |
| CORBA | Common Object Request Broker Architecture (der OMG) |
| DCE | Distributed Computing Environment (der OSF) |
| DCOM | Distributed Component Object Model |
| DER | Distingushed Encoding Rules (für ASN.1) |
| DES | Data Encryption Standard |
| DHCP | Dynamic Host Configuration Protocol |
| DN | Distinguished Name |
| DNS | Domain Name System |
| DSA | Digital Signature Algorithm |
| FIPA | Foundation for Intelligent Physical Agents |
| FTP | File Tranfer Protocol |

| | |
|---|---|
| GUI | Graphic User Interface |
| GUID | Global Unique Identifier |
| HTTP | Hypertext Transfer Protocol |
| IDE | Integrated Development Environment |
| IEC | International Engineering Consortium |
| IEEE | Institute of Electrical and Electronics Engineers |
| IETF | Internet Engineering Task Force |
| IMAP | Internet Message Access Protocol |
| ISO | Information Systems Organization |
| ITU | Internet Telecommunications Union |
| IP | Internet Protocol |
| JAR | Java Archive |
| JCA | Java Cryptography Extensions |
| JCE | Java Cryptography Architecture |
| JDK | Java Development Kit |
| KIF | Knowledge Interchange Formalism |
| KQML | Knowledge Query and Manipulation Language |
| LAN | Local Area Network |
| LDAP | Lightweight Directory Access Protocol |
| LDIF | LDAP Data Interchange Format |
| LPC | Local Procedure Call |
| MAC | Message Authentification Code |
| MASIF | Mobile Agent System Interoperability Facilities Specification (der OMG) |
| MD5 | Message Digest (Einweg Hashfunktion von Rivest) |
| NFS | Network File System |
| NIC | Network Information Center |
| NIS | Network Information Service |
| OID | Object Identifier |

| | |
|---|---|
| OSF | Open Software Foundation |
| OSI | Open Systems Interconnection |
| OMG | Object Management Group |
| PER | Packed Encoding Rules (für ASN.1) |
| PC | Personal Computer |
| POP | Post Office Protocol |
| PKCS | Public-Key Cryptography Standard |
| QoS | Quality of Service |
| RFC | Request for Comment |
| RMI | Remote Method Invokation |
| RPC | Remote Procedure Call |
| RSA | Public-Key Algorithmus nach Rivest, Shamir und Adleman |
| SHA | Secure Hash Algorithm |
| SMTP | Simple Mail Transfer Protocol |
| SSL | Secure Socket Layer |
| SSP-Chain | Chain of Stub-Scion Pairs |
| TCP | Transmission Control Protocol (verbindungsorientiert) |
| UDP | User Dategram Protocol (verbindoungslos) |
| UML | Unified Modeling Language |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| URN | Uniform Resource Name |
| WINS | Windows Internet Name Service |
| WWW | World Wide Web |

# Kapitel 1

# Einleitung

## 1.1 Motivation

Gerade in letzten Jahren erlebt das World Wide Web eine rasante Entwicklung. Neben vielen Firmen, die sich im Internet „nur" repräsentieren, sind ganze (oft kommerzielle) Bereiche hinzugekommen, die über das Netzwerk Informationen und Dienstleistungen anbieten und verstärkt in neue Technologien investieren. E-Buisiness und E-Commerce sind nur zwei Schlagworte, die im Bereich der Informationstechnologie in aller Munde sind.

Des weiteren ist die mobile Telekommunikation zu erwähnen, die in der heutigen Zeit immer mehr Verbreitung findet. Ob es die immer intelligenter werdenden Endgeräte wie Mobiltelefone oder Terminplaner sind, oder um den Laptop als Informationszentrale und mobilen Arbeitsplatz handelt. Sie sind bereits prägend für das Arbeitsverhalten der Benutzer und gemein ist ihnen die Möglichkeit, über den permanenten oder eher kurzzeitigen Zugang zu einem lokalen Netzwerk und über dieses ins Internet, Aufgaben erledigen zu können, die durch den Benutzer initiiert werden.

Hinzu kommt dann auch noch der Begriff des *Verteilten Rechnens*, der ein Szenario beschreibt, in dem mehrere Programme, oder allgemeiner gesehen mehrere Objekte, auf verschiedenen Rechnern in einem Netzwerk Teilaufgaben erfüllen. Unter Ausnutzung lokaler Ressourcen und dem Datenaustausch über gegebene Kommunikationsstrukturen können sie dann der Lösung einer gemeinsamen Aufgabe dienen.

Diese drei aktuellen Strömungen in der Informatik (und Industrie) gehen immer mehr ineinander über, und in diesem Kontext stellt das Konzept des mobilen Software-Agenten ein fast perfekt auf die Bedingungen angepasstes Hilfsmittel dar, das seinem Benutzer Arbeit erleichtert. Ein mobiler Software-Agent ist ein Programm, das zusammen mit Zustandsinformation und Daten autonom in einem Netzwerk wandern kann. Allerdings wird dafür eine Infrastruktur aus Agentenservern nötig, welche die Verwaltung dieser Agenten auf einzelnen Rechnersystemen und deren Transport zwischen verschiedenen Rechnersystemen übernimmt.

Um mobile Agenten in Multi-Agenten Anwendungen zu koordinieren und den Austausch von Nachrichten zwischen solchen Agenten unabhängig von deren Position zu ermöglichen, wird

ein *Namensdienst* benötigt, der die Positionswechsel der Agenten verfolgt, und auf dessen Grundlage eine transparente Weiterleitung von Nachrichten erfolgen kann.

Bislang existiert kein Konzept für einen solchen Dienst, das den besonderen Anforderungen von Mobile Agenten Systemen genügt. Neben den hohen Änderungsraten, die bei einem Konzeptentwurf beachtet werden müssen, stellt im Besonderen die Skalierbarkeit des Modells auch für große Konföderationen von Agentensystemen eine wichtige Anforderung dar. Skalierbarkeit bezieht sich dabei auf die Anzahl der beteiligten Objekte, die Dimension der Verteilung dieser Objekte und damit deren Distanz untereinander, sowie die Anzahl der Organisationen die Kontrolle auf Subsysteme ausüben. Sie hat vor Allem Auswirkungen auf die Verlässlichkeit, Leistungsstärke und den Organisationsaufwand des Gesamtsystems [54].

## 1.2 Zielsetzung

Im Rahmen dieser Arbeit soll nun die Frage behandelt werden, wie Namensdienste für Mobile Agenten Systeme beschaffen sein können. Ausgehend von gebräuchlichen Namensdiensten für relativ statische Strukturen, wie der Rechner- und Benutzerkonstellation in LAN (Intranet innerhalb von Firmen) bzw. WAN (über Firmengrenzen hinausgehende Netzwerke wie z.B. dem Internet), und spezielleren Namensdiensten, die bereits eine gewisse Unterstützung für mobile Objekte bieten, sollen die Anforderungen an solche Dienste formuliert werden. Dabei sind auch Sicherheitsanforderungen zu berücksichtigen.

Auf Basis dieser Voruntersuchungen soll anschließend ein Modell für einen Namensdienst für Mobile Agenten Systeme entwickeln werden. Nach der Implementierung eines Prototyps sollen in einer konkreten Simulations- oder Testumgebung Daten gesammelt werden, um den gewählten Ansatz abschließend zu diskutieren.

## 1.3 Typographie und Terminologie

Klassennamen und Datentypen sowie Quelltextauszüge sind in `Schreibmaschinenschrift` gesetzt. Um die für einen Absatz relevanten Schlüsselwörter oder Satzteile bzw. Begriffsdefinitionen hervorzuheben, wird ein serifenloser Schriftsatz verwendet. Im Gegensatz dazu lassen sich neu eingeführte Begriffe im Text an ihrer *kursiven* Darstellung erkennen. Abkürzungen werden bei ihrem erstmaligem Auftreten explizit erklärt und dann anschließend ohne besondere Hervorhebung verwendet.

Gerade in der Informatik sind viele englische Fachwörter in Gebrauch. Sofern sich diese akzeptabel übersetzen lassen, wird das in dieser Arbeit auch getan. Beschreibt ein englischer Begriff den geschilderten Sachverhalt allerdings besser bzw. hat er direkten Bezug auf Komponenten oder Vorgänge des beschriebenen oder zu entwickelnden Modells, so wird er in seiner ursprünglichen Form belassen. In diesem Fall wird er bei jedem Auftreten *kursiv* gesetzt und für die Konjugation oder Nutzung im Plural nach den Regeln der englischen Rechtschreibung und Grammatik behandelt. Das heißt, dass diese Begriffe klein geschrieben werden, auch wenn sie als Nomen in Gebrauch sind, und im Besonderen keine Genitiv-Form existiert. Eine Ausnahme bilden die im Kontext der Informatik feststehenden, englischen

Ausdrücke, die in ihrer ursprünglichen Form teilweise auch groß geschrieben werden. Sie werden behandelt, wie deutsche neu eingeführte Begriffe (siehe oben). Bei der Kombination zwei verschiedensprachiger Wörter, wird kein Bindestrich benutzt, d.h. die einzelnen Wörter werden getrennt nach den genannten Konventionen dargestellt.

In den Abbildungen und in den Spezifikationen im Anhang werden ohne spezielle Hervorhebungen durchgängig englischsprachigen Begriffe verwendet. Der Quellcode des zu implementierenden Prototyps wird streng nach den *SeMoA*$^{TM}$ *Code Conventions* [76] entwickelt und englisch dokumentiert.

Werden in dieser Arbeit Begriffe häufig verwendet, dessen Bedeutung im normalen Sprachgebrauch nicht eindeutig ist, so findet sich im Glossar in Anhang D eine kurze Beschreibung, die dessen Bedeutung im Kontext dieser Arbeit klar definiert.

Bleibt noch zu erwähnen, dass diese Arbeit nach den Regeln der neuen deutschen Rechtschreibung verfasst wird.

## 1.4 Implementierungsumgebung

Die Implementierung des Prototyps erfolgt durch das *Java*$^{TM}$ *Software Developer Kit (SDK)* von *Sun Microsystems Inc.* [34], das in der aktuellen Version 1.3 Verwendung findet. *Kawa*$^{TM}$ von *Tek-Tools Inc.* [35] dient unter einem Windows NT System in der Version 4.10a als *Integrated Development Environment (IDE)*, das neben einer übersichtlichen Darstellung der Klassenhierarchie und einem komfortablen Editor auch integrierte Hilfe- und Test-Funktionalität zur Verfügung stellt. Sowohl zum anfänglichen Design der Klassenstrukturen des Prototyps als auch zur Erzeugung von UML- und Sequenz-Diagrammen, wird das *Together*$^{TM}$ *Control Center* von *TogetherSoft* [87] in der Version 4.1 eingesetzt. Die eigentliche Evaluation des Prototyps finden dann in einem heterogenen LAN mit integrierten Windows- und Solaris-Systemen statt.

## 1.5 Struktur der Arbeit

Im Anschluss an die hier dargelegte Aufgabenstellung und die Rahmenbedingungen der Arbeit stellt *Kapitel 2* die Grundlagen von Agentensystemen und Namensdiensten vor, auf denen diese Arbeit aufbaut. Die Problemstellung wird erörtert und aufgrund einer erstellten Anforderungsliste werden verwandte Arbeiten und Systeme diskutiert.

In *Kapitel 3* wird das Modell eines Namensdienstes entwickelt, das die Anforderungen im Sinne der Aufgabenstellung erfüllt. *Kapitel 4* und *Kapitel 5* beschreiben anschließend die Implementierung des Funktionsmusters dieses Modells und geben die ermittelten Testergebnisse wieder. Abschließend gibt das *Kapitel 6* eine Zusammenfassung und Bewertung der in dieser Arbeit gefundenen Lösungen wieder und stellt einen Ausblick auf mögliche Erweiterungen und Anwendungsszenarien dar.

Ein Überblick über die bei Protokolldefinitionen verwendete Syntaxnotation ASN.1 und die Agenten-Kommunikationssprachen KQML und ACL finden sich im *Anhang*. Darüber hin-

aus werden dort auch die für das Funktionsmuster benötigten Schnittstellenspezifikationen angeführt.

# Kapitel 2

# Grundlagen und Anforderungsanalyse

In diesem Kapitel wird ein Überblick über Agenten und Agentensysteme im Allgemeinen gegeben und dann konkret auf die Struktur von SeMoA als Referenz der Mobile Agenten Plattform eingegangen, die ich für die Evaluierung meiner Arbeit verwende. Nachdem die Notwendigkeit von Namensdiensten im Kontext verteilter Systeme dargestellt und in diesem Bezug grundlegende Konzepte für die Lokalisierung von mobilen Objekten erläutert wurden, gehe ich auf die besonderen Anforderungen eines solchen Dienstes für Mobile Agenten Systeme ein. Dabei stehen neben der Anforderung, den hohen Änderungsraten der Systemkonfiguration zu genügen, Aspekte der Skalierbarkeit und Sicherheit im Vordergrund.

Auf Basis dieser Vorarbeiten wird anschließend versucht, existierende Namens- und Verzeichnisdienste den dargestellten Lokalisierungskonzepten zuzuordnen und deren Implementierungen im Hinblick auf Verwendbarkeit im Sinne der Aufgabenstellung zu diskutieren. Darüber hinaus stelle ich Konzepte, Algorithmen und spezielle Protokolle für die Lokalisierung und die Organisation von mobilen Objekten bzw. den Nachrichtenaustausch dar. Dabei wird wiederum analysiert, inwiefern sich diese Modelle auf die konkreten Anforderungen eines Namensdienstes für Mobile Agenten Systeme übertragen lassen.

## 2.1 Agentensysteme

Der Begriff *Agent* ist in der letzten Jahren zum Modewort avanciert, das als Bezeichnung für verschiedene Erscheinungen im Bereich der Informatik verwendet wird. Agenten sind bereits seit Ende der 70er Jahre aus dem Gebiet der künstlichen Intelligenz bekannt, wurden seitdem aber auch im Bereich der Betriebssysteme untersucht. Durch die Popularität des Internets in der Gegenwart liegt das Interesse jedoch immer mehr auf der Entwicklung von sogenannten *mobilen Agenten* in verteilten Systemen. Wobei anzumerken ist, dass die Definitionen eines Agenten in diesen drei Bereichen ineinander übergehen. Die angesprochenen mobilen Agenten entsprechen übrigens den sogenannten *mobilen Software-Agenten*, die im nächsten Abschnitt noch genauer definiert werden.

### 2.1.1 Definition und Klassifikation von Agenten

Das eigentliche Wort „Agent" stammt aus dem Lateinischen und bezeichnet einen Handelnden. Im Kontext von Computersystemen definieren Woolridge und Jennings [47] einen *intelligenten* Agenten über folgende Eigenschaften:

**Autonomie:** Agenten operieren ohne direkten Einfluss eines anderen und verfügen über eine Art Kontrolle über ihre Aktionen und ihren internen Zustand.

**Soziales Verhalten:** Agenten interagieren mit anderen Agenten bzw. dem Menschen über eine Art von Agenten-Kommunikationssprache.

**Reaktivität:** Agenten nehmen ihre Umgebung wahr und reagieren auf Umgebungsveränderungen.

**Proaktivität:** Agenten agieren nicht einfach nur als Reaktion auf ihre Umgebung, sondern sind fähig zielgerichtetes Verhalten an den Tag zu legen, indem sie Initiative ergreifen.

Betrachtet man die Gesamtheit der intelligenten Agenten nach obiger Definition, so lassen sich Klassifizierungen finden, die sich je nach Agententyp durch weitere bzw. speziellere Eigenschaften auszeichnen. Die folgende Unterteilung in Agentenklassen entspricht weitgehend dem Artikel von Nwana [57]:

**Kollaborierende Agenten** versuchen in Verhandlung mit anderen Agenten ihr Ziel zu erreichen. Sie arbeiten rational und autonom in einer offenen Umgebung, in der sich auch andere Agenten befinden.

**Schnittstellen-Agenten** kommunizieren im Gegensatz zu den kollaborierenden Agenten nicht mit Agenten sondern vor Allem mit dem Benutzer. Dabei wird versucht durch Imitation aus dessen Verhalten zu lernen. Beschleunigt wird der Lernvorgang mittels *Feedback* und Instruktionen von Seiten des Benutzers.

**Mobile Agenten** besitzen die Fähigkeit, sich in einem Netzwerk zu bewegen, und somit ihre Aufgaben auf verschiedenen Rechnern bearbeiten zu können. Dadurch, dass sie Ressourcen jeweils lokal nutzen, reduzieren sie die Netzlast. Des weiteren zeichnet sie die Eigenschaft aus, ihren Weg selber bestimmen zu können, um ihre Aufgabe dadurch optimal zu lösen.

**Informations-Agenten** können innerhalb der mittlerweile unüberschaubaren Informationsflut großer Netzwerke wie dem Internet als Werkzeug zur Verwaltung dieser Daten dienen. Dabei übernehmen sie die Aufgabe der Informationsfilterung bzw. Manipulation und Sammlung von Informationen.

**Reaktive Agenten** besitzen kein Modell ihrer Umwelt sondern reagieren auf Reize, die von der Umwelt auf sie ausgeübt werden. Sie sind daher sehr einfach aufgebaut und finden nur ein eingeschränktes Einsatzgebiet, vor Allem in der Simulation von Objekten und Subjekten.

**Hybride Agenten** vereinen die Eigenschaften der oben erwähnten Agententypen.

**Smarte Agenten** existieren noch nicht in der Realität, sondern stellen die Idealform eines Agenten dar, der autonom ist, mit anderen Agenten kooperiert und in der Lage ist selbständig zu lernen.

Die sogenannten *Software-Agenten* stellen nun eine Teilmenge der intelligenten Agenten dar. Für eine Definition sei hierbei wiederum auf den Artikel von Nwana [57] hinzuweisen. Er beschreibt einen Software-Agenten als ein Stück Programmcode, das in der Lage ist, durch genaues Einhalten der Vorgaben seines Inhabers in dessen Namen eine Aufgabe zu erfüllen.

Der Vollständigkeit halber sollte erwähnt werden, dass neben den Software-Agenten auch *Hardware-Agenten* als Untergruppe der intelligenten Agenten existieren. Ihre Funktionalität ist durch das Zusammenspiel von Hardware-Komponenten gegeben, sie sind besonders in der Automobilbranche und der industriellen Fertigung zu finden. Im folgenden werde ich mich allerdings ausschließlich auf Software-Agenten beziehen.

Eine Agenten- bzw. Multiagentensystem stellt nun die Umgebung mit den in ihr enthaltenen Agenten dar. Dieses Agentensystem ist nicht notwendigerweise, und besonders im Fall der mobilen Agenten nicht, auf einen Rechner beschränkt.

Der Begriff *Agentensystem* befindet sich noch auf einer recht hohen Abstraktionsebene: Geht man von einem oder mehren kooperierenden Agenten aus, die direkt auf einem bzw. mehreren Rechnersystemen zur Ausführung kommen, so stellt das Agentensystem nur eine Art Hüllensystem um die Agenten, deren Schnittstelle zum Rechnersystem und deren Kommunikationskanäle dar. Heutzutage beinhaltet ein Agentensystem allerdings meistens eine zusätzliche Softwareebene, welche die Agenten von den darunter liegenden Rechnersystemen abkapselt und eine Verwaltungsstruktur mit klar definierte Schnittstellen bereitstellt.

Je nach Agententyp, der in einem Agentensystem zur Ausführung gebracht werden soll, unterscheiden sich die Agentensysteme in den angebotenen Diensten. Neben einer Verwaltungsstruktur, die normalerweise für das Starten, Beenden und die Kontrolle der Agenten während der Laufzeit zuständig ist, werden meist klar spezifizierte Schnittstellen zur Außenwelt (zu Rechnerressourcen oder zum Benutzer) oder zur Inter-Agentenkommunikation bereitgestellt.

### 2.1.2 Was zeichnet einen mobilen Agenten aus ?

Ich möchte den Begriff des mobilen Agenten im Gegensatz zur Definition von Nwana vorerst etwas erweitern und werde mich im Folgendem in meiner Arbeit auf diese erweiterte Definition beziehen, wenn ich von mobilen Agenten rede. Dies hat vor Allem den Grund, dass sich das Verständnis von mobilen Agenten heutzutage im Allgemeinen und für die SeMoA-Plattform (siehe Abschnitt 2.2) im Speziellen verändert hat.

Geht man von Nwanas Klassifizierung aus, so sind mobile Agenten im heutigen Sinn immer noch als Teilmenge der intelligenten Software-Agenten zu sehen, beinhalten aber bereits hybride Ansätze. Sie können sich nicht nur auf selbstbestimmten Pfaden durch ein Netzwerk bewegen und bei der Ausführung auf Rechnern lokale Ressourcen nutzen, sondern sind durchaus schon in der Lage auf ihrem Weg durch definierte Schnittstellen zu lokalen Datenbanken

Informationen zu sammeln und auszutauschen. Aufgrund dieser Informationen beeinflussen sie den weiteren Ablauf ihres Programms und ihren Pfad durch das Netzwerk.

Des weiteren wird intensiv daran gearbeitet, Modelle für den sicheren Nachrichtaustausch zwischen Agenten zu entwickeln (innerhalb einer Rechnerumgebung, aber auch über das Netzwerk), und darauf aufbauend eine standardisierte Agenten-Kommunikationssprache (siehe Abschnitt B) zum Wissensaustausch aufzusetzen.

Zur Ermöglichung dieser Ziele spielt die Agentenplattform eine immer größer werdende Rolle. Zum einen kapselt sie die Agenten durch definierte Schnittstellen von den teilweise heterogenen Rechnerstrukturen der Wirtsysteme ab und zum anderen kann sie einen Transportmechanismus zur Verfügung stellen, der die eigentliche Migration und damit die wichtigste Fähigkeit mobiler Agenten für diese so transparent wie möglich gestaltet.

Bevor ich allerdings näher auf die Funktion der Agentenplattform eingehe, beschreibe ich kurz, was nötig ist, um einem Agenten während seines Lebenszyklus die Möglichkeit zu geben, auf verschiedenen Wirtsystemen (den *Agentenservern*) zur Ausführung zu kommen, und dadurch den Vorgang der Migration:

Will ein Agent nach der Ausführung auf einem bestimmten Server diese Umgebung verlassen, um dann auf einem anderem Server seine Arbeit an diesem Punkt wieder aufzunehmen, so darf er nicht wirklich terminieren. Sein Kontrollfluss wird unterbrochen und im Idealfall auf der Zielserver genau an diesem Punkt wieder fortgesetzt. Es ist unbedingt nötig, dass dafür nicht nur sein Programmcode auf den neuen Server transportiert wird, sondern auch sein interner Zustand vor dem Transport gespeichert und anschließend auf dem Zielsystem wiederhergestellt wird. In der Praxis wird dies erreicht, indem die Klasseninformation und der Zustand aller seiner Objekte in einen transportablen Datenstrom (einer Folge von Bytes) umgewandelt wird, das sogenannte *Serialisieren*. Dieser Datenstrom wird auf dem Zielserver dann wieder in die entsprechenden Objekte zurück transformiert (*deserialisiert*), bevor der Agent an definierter Stelle wieder gestartet wird.

Die angesprochene größtmöglichste Transparenz der Migrationsmechanismen gegenüber dem Agenten wird dadurch erreicht, dass die bei dem Vorgang dem Agenten zugeteilte Aufgabe lediglich auf die Angabe einer gültigen Zieladresse reduziert wird, bevor er seine Prozesse beendet. In der Praxis wird dies durch einen speziellen Methodenaufruf oder das Ausfüllen einer Datenstruktur mit den erforderlichen Angaben erreicht, die oft als *passport* oder *ticket* bezeichnet wird.

Die Aufgabe der Agentenplattform besteht dabei nun aus folgenden Schritten:

- Der Agent wird als Datenpaket über einen Transportkanal in Empfang genommen.
- Seine Komponenten werden in der lokalen Umgebung installiert.
- Bei der Deserialisierung wird der Zustand der Datenstrukturen wiederhergestellt.
- Die Ausführung des Agenten wird an definierter Stelle gestartet.

- Gegebenenfalls werden dem Agenten durch das Angebot von Diensten gewisse Ressourcen und Informationen zur Verfügung gestellt.
- Im Gegenzug kann es dem Agenten möglich sein, ebenfalls Dienste zu registrieren, die dann von dem Agentenserver bzw. von anderen Agenten genutzt werden können.
- Beendet der Agent seine Prozesse mit vorausgehender Angabe eines Migrationswunsches wird der Agent serialisiert und zur gewünschten Zieladresse geschickt.

Die Agentenplattform setzt durch dieses Vorgehen eine feste Struktur der mobilen Agenten voraus. Falls der Benutzer einen Agenten für eine spezifische Plattform programmieren möchte, muss er sich also an diese vorgeschriebene Struktur halten. Dies spiegelt sich in Restriktionen bei der Wahl der Programmiersprache oder in der geforderten Implementierung von gewissen Schnittstellen wieder. Aus diesem Grund ist eine sehr hardwareorientierte Programmiersprache wie Assembler zur Agentenprogrammierung ungeeignet. Im Gegenteil, manche Agentensysteme liefern eine abstrakte, speziell auf die Plattform ausgelegte Agenten-Programmiersprache gleich mit (siehe auch Abschnitt 2.6.2) bzw. ermöglichen durch vorgefertigte Programmstrukturen, Agenten im Baukastenprinzip zusammenzusetzen.

Mittlerweile gibt es viele Forschungsgruppen die sich mit dem Entwurf von Plattformen für mobile Agenten beschäftigen und Prototypen nach verschiedensten Anforderungsmustern entwickelt haben. Zwei Umfangreiche Listen finden sich in [18] und [43].

### 2.1.3 Einsatzmöglichkeiten mobiler Agenten

Es sei bemerkt, dass Plattformen für mobile Agenten normalerweise auch statische Agenten zulassen, die lokal Aufgaben erfüllen können. Da ein Agent in der Regel nicht gezwungen wird, nach gewisser Zeit zu terminieren bzw. einen Migrationswunsch anzumelden, könnte er sich darauf beschränken, in der lokalen Ausführungsumgebung gewisse Dienste anzumelden oder durch das Nutzen bestimmter Dienste Informationen zu sammeln. Darüber hinaus wäre eine Interaktion mit einem Benutzer denkbar, der dadurch Kontrollfunktionen zur Verfügung gestellt bekäme.

Will man allerdings die Einsatzmöglichkeiten mobiler Agenten beschreiben, so sollte man auch die Vorzüge in Anspruch nehmen, welche diese Modell bietet. Als ausschlaggebendste Eigenschaft eines mobilen Agenten ist wohl zu nennen, dass er autonom und ohne ständige Verbindung zum Heimatserver bzw. seinem Besitzer in dessen Auftrag Aufgaben erfüllen kann.

Dies wird auch insofern immer wichtiger, als dass in der heutigen Zeit immer mehr mobile Endgeräte Verbreitung finden, die nur zeitweise über die Ankopplung an lokale Netze oder über Funkverbindung mit der Außenwelt Kontakt aufnehmen. Unabhängig davon, ob für dieses oft vom Benutzer geprägte Verhalten nun Kosten-, Zeit-, oder andere Gründe genannt werden, mobile Agenten fügen sich wunderbar in dieses Verhaltensmuster ein.

Der mobile Agent kann durch eine einmalige Verbindung zu einem lokalen oder globalen Netzwerk (wie dem Internet) auf seinen Weg geschickt werden. Es ließe sich nun vorstellen,

dass dieser nach Beenden seiner Aufgabe auf einem „befreundetem", Server das Ergebnis, z. B. in Form von gefilterten Informationen, hinterlegt und dieses bei einer weiteren Verbindung zum Netzwerk abgerufen werden kann.

Im folgenden wird kurz noch ein Überblick über konkrete Anwendungsszenarien angeführt:

**Informationsbeschaffung:** Der Benutzer beauftragt einen mobilen Agenten mittels angegebener Kriterien im Netzwerk nach Informationen zu suchen und diese nach Rückkehr auf die Heimatserver in ansprechender Form und gefiltert zu präsentieren. Gerade bei der Informationsflut wie sie im Internet auf dezentralen Datenbanken vorhanden ist, könnte dieses Vorgehen viel Zeit ersparen. Der Pfad des Agenten durch das Netzwerk kann dabei entweder vorher durch den Benutzer statisch festgelegt werden, oder vom Agenten durch Kooperation mit anderen Agenten oder den Agentenservern frei gewählt werden.

**Kommerzielle Anwendungen:** Der mobile Agent besucht verschiedene Produktanbieter und vergleicht nach den Maßstäben seines Benutzers die Angebote. Er evaluiert eigenständig nach Forderungen wie Maximalpreis, Qualität oder Ausstattung und tätigt anschließend den Einkauf bei dem Anbieter, der den Ansprüchen des Benutzers am besten nachkommt. Ein interessantes Thema das in diesem Bezug noch einiger Diskussion bedarf ist die Zahlungsweise, und ob der Benutzer in diesem letzten Schritt noch interaktiv eingreifen sollte bzw. muss.

**Personalisierte Dienste:** Mittels eines erstellten Persönlichkeitsprofils des Benutzers wertet der mobile Agenten Informationen aus und filtert diese selbsttätig nach den ihm bekannten Interessengebieten des Benutzers. Diese Informationen können z. B. durch regelmäßige Nachforschungen auf Informationsservern oder durch das Abonnieren von Informationen gesammelt werden. Dabei könnte es sich um Neuerscheinungen, Programminformationen zu Veranstaltungen oder günstigen Angeboten handeln.

**Delegation von routinemäßige Aufgaben bzw. Fernwartung:** Bieten Rechnersysteme Agenten über entsprechende Agentenserver die benötigten Schnittstellen an, so kann der mobile Agent routinemäßige Arbeiten bzw. Wartungen auf entfernten Rechnersystemen durchführen, ohne den Benutzer und Initiator dabei übermäßig in Anspruch zu nehmen. Der Benutzer kann dann im Fehlerfall über die Probleme bzw. auch im Normalfall durch ein Zusammenfassung des Ergebnisses informiert werden.

**Workflow Assistenten:** Ein Workflowsystem zeichnet sich dadurch aus, dass mehrere Instanzen an einer Gesamtaufgabe arbeiten, wobei die einzelnen Arbeitsschritte allerdings die Synchronisation und Kooperation dieser Instanzen voraussetzen. Dies kann z. B. auf die Projektarbeit in einer größeren Abteilung einer Firma übertragen werden. Der mobile Agent könnte hierbei die Aufgabe der automatischen Informationsweitergabe und dem Initiieren von Meetings erfüllen bzw. sich um andere organisatorische Aufgaben kümmern.

### 2.1.4 Sicherheitsproblematik bei mobilen Agenten

Das Modell des mobilen Agenten birgt einige Sicherheitsrisiken. Sowohl für den Agenten als auch für den Anbieter eines Agentenservers: Der Agent kann normalerweise einem Besitzer

zugeordnet werden und kommt während seines Lebenszyklus' auf vorher definierten oder dynamisch ermittelten Pfaden durch das Netzwerk auf verschiedenen „fremden" Servern zur Ausführung. Die Anbieter von Agentenservern nehmen „fremde" Agenten in Empfang und starten diese auf ihrem Rechnersystem, ohne vorher zu wissen, welche Intension ihr Besitzer bei dessen Programmierung hatte. Betrachtet man die oben angesprochenen Anwendungsszenarien, wird dieses noch deutlicher, und die Sicherheitsprobleme lassen sich ziemlich konkret benennen.

Man sollte übrigens nicht außer Acht lassen, dass neben gezielten Angriffen auf Agent bzw. Agentenserver mit bösartiger Intension auch unbeabsichtigtes Fehlverhalten von Agent bzw. Agentenserver den gleichen oder einen ähnlichen Effekt haben könnte.

Des weiteren tritt gerade beim Transport der Agenten im Internet von Agentenserver zu Agentenserver ein Sicherheitsrisiko auf, das von einer dritten Partei ausgeht. In diesem Fall passieren die Agenten beim Transport durch das Netz auch Rechnersysteme, die an der Infrastruktur des Agentensystems eigentlich unbeteiligt sind. Durch die Möglichkeit des direkten Zugriffs auf die transportierten Daten erhöht sich damit die Anzahl der potentiellen Angreifer.

In diesem Abschnitt möchte ich nun die Sicherheitsprobleme in zwei Kategorien aufgeteilt benennen und anschließend stichpunktartig Lösungsmöglichkeiten darstellen.

Zuerst betrachte ich die Risiken für den Agentenserver. Dabei ist zu beachten, dass ein Agentenserver im Prinzip nur eine Softwareebene darstellt, welche die Ausführungsumgebung der Agenten von dem darunter liegendem Rechnersystem abkapselt. Auf diesem Rechnersystem können durchaus weitere Agentenserver installiert sein bzw. durch andere Programmpakete sensitive Daten des Anbieters des Agentenservers verarbeitet werden. Der Agentenserver stellt in der Regel Dienste zur Verfügung, die es dem Agenten ermöglichen auf Ressourcen des Rechners zuzugreifen oder Informationen auszutauschen. Außerdem lässt sich nicht verheimlichen, dass der Agent natürlich im Speicher des Rechnersystems ausgeführt wird, dessen Prozessorleistung in Anspruch nimmt und gegebenenfalls in der Lage ist während der Ausführung Informationen auf dem lokalen Speichermedium abzulegen.

Ob nun durch bösartigen Willen oder durch fehlerhafte Programmierung kann der Agent nun vor Allem zwei Risiken darstellen. Zum einen könnte er im engeren Sinne die Dienste des Agentenservers bzw. im weiteren Sinne Ressourcen des gesamten Rechnersystems auf eine Weise in Anspruch nehmen, die es den anderen Agenten bzw. auf dem Rechnersystem befindlichen Programmpaketen nicht mehr ermöglicht weiterhin ihrer Aufgabe nachzukommen (*Denial of Service (DoS) attacks*). Zum anderen könnte der Agent die Möglichkeit bekommen sensitive Daten anderer Agenten bzw. Programmpakete auszuspähen, zu manipulieren oder zu vernichten (*attacks on privacy*).

Der Inhaber des Agentenservers hat nun einige Möglichkeiten, sich gegen diese Gefahren abzusichern:

- Die richtige Wahl der Agenten-Programmiersprache bzw. die Sicherheitspolitik dieser Programmiersprache ist ausschlaggebend. Dadurch können im Voraus gewisse Basis-

Schutzmechanismen eingeführt werden. Dahinter steckt auch die Idee, eine Agenten-Programmiersprache zur Verfügung zu stellen, die gewisse sicherheitskritische Zugriffe erst gar nicht ermöglicht.

- Durch saubere Programmierung und eindeutige Spezifaktionen der Schnittstellen zwischen Agentenserver und Agent, kann die Möglichkeit von ungewolltem Zugriff auf Fremdressourcen stark vermindert werden. Dabei spielt die Abkapselung des Agenten von den Ausführungsumgebungen anderer Agenten, dem Agentenserver oder anderen Programmpaketen auf dem Rechnersystem eine große Rolle.

- Verlangt der Agentenserver eine Signatur der Agenten, die auf dem Transportkanal über das Netzwerk den Agentenserver erreichen, wird es möglich, den wahren Besitzer des Agenten zu identifizieren und damit seine Vertrauenswürdigkeit einzuschätzen.

- Nach der Ankunft eines Agenten, aber noch vor dessen Ausführung, kann durch inhaltsbasierte Filter das zukünftige Verhalten des Agenten abgeschätzt und ihm dadurch gegebenenfalls die Ausführung komplett verweigert werden. Diese Filter basieren meist auf der statistischen Analyse des Programmcodes der Agenten.

- Die ständige Überwachung des Agenten in seinen Aktionen während der Ausführung, könnte ein sofortiges Eingreifen bei ungewollten bzw. sicherheitskritischen Handlungen ermöglichen. Durch einen interpretativen Ansatz bei der Agenten-Programmiersprachen ließen sich dabei Zugriffe auf sicherheitsrelevante Systemressourcen schon durch den Interpreter reglementieren.

Nun werden die Risiken für den Agenten besprochen. Zuerst ist noch einmal zu erwähnen, dass die Aufgabe eines Agenten normalerweise direkt von seinem Besitzer vorgegeben wird und der Agent dann nach seinem Start versucht, diese Aufgabe zielgerichtet zu erfüllen. In manchen Fällen ist es bereits unerwünscht, dass die Aufgabenstellung offengelegt wird. In anderen Fällen ist es wichtiger, dass die im Zuge dieser Aufgabenstellung gesammelten Informationen geschützt werden. Ein weiterer Aspekt ist, dass der Agent zur Erfüllung seiner Aufgabe bereits vom Benutzer sensitive Daten mit auf den Weg bekommen kann, die unter keinen Umständen in falsche Hände geraten dürfen (man denke dabei an die oben beschriebenen kommerziellen Anwendungen oder personalisierte Dienste). Angriffe auf die genannten Szenarien fallen alle unter die Kategorie *attacks on privacy*. Wiederum besteht allerdings auch die Möglichkeit von *Denial of Service attacks*. Durch die Manipulation des Agentencodes oder gar der Terminierung eines Agenten kann dieser daran gehindert werden, seine Aufgabe korrekt auszuführen. Des weitern könnte durch einen manipulierten Datenaustausch zwischen Agent und Agent bzw. zwischen Agent und Agentenserver eine Vortäuschung von falschen Tatsachen stattfinden, dass den Agenten zu ungünstigen Entscheidungen zwingt.

Abgesehen von der unabsichtlichen Veränderung des Agentencodes oder gar der versehentlichen Terminierung des Agenten durch ein fehlerhafter Agentenserver bzw. Probleme beim Transport, treten die anderen genannten Sicherheitsprobleme eigentlich hauptsächlich durch bösartiges Handeln eines Angreifers auf. Dabei stehen dem Anbieter eines Agentenservers natürlich weitaus mehr Möglichkeiten offen, als einem nur durch den Agententransport über sein Rechnersystem involvierten Dritten.

Der Besitzer eines Agenten hat nun folgende Möglichkeiten, sich gegen diese Gefahren abzusichern:

- Die grundlegende Methode zum Schutz sensitiver Daten ist die Verschlüsselung dieser Daten. Dadurch ist es dem Agenten möglich, Teile der mitgeführten Informationen nur gewissen Personengruppen bzw. Agentenservern zur Verfügung zu stellen.
- Durch eine digitale Signatur des Besitzers für statische Teile des Agenten, wie dem Programmcode bzw. Initialinformationen, lassen sich Manipulationen an diesen Daten leicht feststellen, und darüber hinaus lässt sich der Besitzer als eindeutiger Urheber dieser Daten identifizieren.
- Im Gegenzug kann der Agentenserver die von ihr gelieferten Informationen, die der Agent z. B. als Ergebnis beim Benutzer präsentieren soll, ebenfalls signieren, um sich dadurch selbst als Urheber zu identifizieren, bzw. zusätzlich verschlüsseln um sie ausschließlich dem Besitzer des Agenten zugänglich zu machen
- Kritische Entscheidungen, die der Agent zu treffen hat, sollten so weit möglich auf neutralen Boden, d.h. auf einem vertrauenswürdigem Agentenserver, getroffen werden.
- Unterstützt das Modell eines Agentensystems die Lokalisierung von bzw. Kommunikation mit Agenten, so kann der Besitzer regelmäßig direkt Informationen über das Verhalten des Agenten einholen und somit dessen korrekte Ausführung aktiv kontrollieren.
- Ein weiterer sehr interessanter Ansatz, der die Sicherheit von Agenten gegenüber bösartigen Agentenservern verbessert, wird in einem Artikel von Volker Roth sehr gut beschrieben [74]. Sein Vorschlag ist es, kritische Informationen oder die Entscheidungsfähigkeit auf zwei kooperierende Agenten zu verteilen, die sich in verschiedenen Domänen im Netzwerk befinden, sich durch Nachrichtenaustausch aber gegenseitig kontrollieren können. Die Wahrscheinlichkeit, dass Manipulationen an einem der beiden Agenten unbemerkt bleibt ist in diesem Fall sehr gering.

Ob und in welchem Maße diese Sicherheitsrisiken für Agent oder Agentenserver bestehen oder von Relevanz sind, hängt natürlich von der Anwendung und den Anforderungen an das gegebene Agentensystem ab. Betreibt man allerdings ein Agentensystem, dass im Gegensatz zu dem recht vertrauenswürdigen Grenzen eines kleinen Forschungsnetzes im Internet Anwendung finden soll und dort in gewissen Grenzen verschiedensten Parteien öffentlich zur Verfügung gestellt werden soll, so sollte man sich durchaus Gedanken machen, alle Beteiligten vor den genannten Gefahren zu schützen.

## 2.2 Die SeMoA-Plattform

In diesem Abschnitt wird nach dem allgemeinen Überblick über Agentensysteme nun die SeMoA-Plattform (Secure Mobile Agents) als eine konkrete Implementierung einer Agentenplattform vorgestellt, die im Rahmen einer Forschungsarbeit des Fraunhofer Instituts für Graphische Datenverarbeitung in Darmstadt entwickelt wurde.

Meine Arbeit, und damit die Entwicklung eines sicheren und skalierbaren Namensdienstes für Mobile Agenten Systeme, nutzt die SeMoA-Plattform als Referenzsystem zur Evaluierung und könnte das Agentensystem zukünftig um diese, noch fehlende Funktionalität erweitern. Aus diesem Grund möchte ich das in [79] von Volker Roth und Mehrdad Jalali beschriebene Basiskonzept für einen sicheren Mobile Agenten Server in den nächsten Abschnitten vorstellen und, auch im Kontext des letzten Kapitels, getroffene Implementierungsentscheidungen für die aktuelle Version des System klassifizieren und erläutern.

Es ist allerdings anzumerken, dass sich die SeMoA-Plattform als Experimentiersystem immer noch im Entwicklungsstadium befindet und eine Umstrukturierung des Kernsystems zur noch konsequenteren und klareren Umsetzung des Basiskonzeptes und der Designziele für die nähere Zukunft geplant ist. Neben einigen Anwendungsszenarien, die auf der Basis von SeMoA entwickelt und getestet werden, liegt ein anderer Schwerpunkt auf der Implementierung von Modulen, die als Erweiterung der Basisfunktionalität dienen sollen und wahlweise je nach Anwendungsszenario zusätzlich zum SeMoA-Server installiert werden können.

### 2.2.1 Architektur von SeMoA

Bereits der Name Secure Mobile Agents weißt darauf hin, dass die Sicherheit bei der Konzeption dieses Agentensystems im Fordergrund stand. In welcher Hinsicht und mit welchen Mitteln dabei folgende Szenarien bedacht wurden, wird weiter unten beschrieben (siehe Abschnitt 2.2.2):

- Sicherheit für den Agenten gegenüber anderen Agenten
- Sicherheit des Agentenservers gegenüber einem bösartigen Agenten
- Sicherheit des Agenten gegenüber einem bösartiger Agentenserver
- Sicherheit von Agent und Agentenserver gegenüber Angriffen aus dem Netzwerk

Neben der Forderung nach Sicherheit, sind allerdings weitere wichtige Designkriterien aufzuzählen, die den Entwurf des Agentenservers entscheidend geprägt haben:

- Minimales Kernsystem
- Einfache Schnittstellen
- Modularität
- Flexibilität
- Leichte Erweiterbarkeit
- Qualitativ hochwertiger Programmcode

Abgesehen von diesen Forderungen soll der SeMoA-Server natürlich auch den in Abschnitt 2.1.2 angesprochenen Aufgaben einer Agentenplattform nachkommen.

Zur Implementierung des SeMoA-Servers und als Programmiersprache für die Agenten wurde von den Entwicklern die Sprache Java gewählt, die momentan in der Version 1.3 zum Einsatz kommt. Damit wird zum einen recht einfach die erwünschte Plattformunabhängigkeit erzielt, die mitunter ausschlaggebend für die Flexibilität dieses Agentensystems ist, und zum anderen konnte der kryptographische Schutzmechanismus des SeMoA-Servers auf der bereits vorhandenen *Java Cryptography Architecture (JCA)* und den *Java Cryptography Extensions (JCE)* aufbauen, womit eine leichte Erweiterbarkeit um neue kryptographische Algorithmen zur Verschlüsselung und digitalen Signatur erreicht wird. Außerdem ist in Java die Unterstützung des *class loading* schon vorhanden, die das Installieren von Agenten auf dem Server erleichtert. Auf der anderen Seite wird aber auch nicht verschwiegen, dass noch einige Angriffsmöglichkeiten (vor Allem DoS Angriffe) möglich sind, die sich gerade aufgrund der speziellen Struktur von Java ergeben.

Dieses Agentensystem versucht soweit möglich auf existierenden Standards aufzubauen und dadurch attraktiv auf einen späteren Anwender von SeMoA zu wirken. Für den Transport der Agenten wird neben einem einfachen RAW-Protokoll bereits HTTP unterstützt. Durch die später beschriebene Modularität der Plattform ist es dann allerdings leicht möglich die Unterstützung von SMTP, POP, IMAP oder FTP aufzusetzen. Der kryptographische Schutzmechanismus baut wie bereits erwähnt auf JCA/JCE auf, wobei die internen Verwaltungsstrukturen für Verschlüsselung und digitale Signatur konform mit den Spezifikationen von PKCS#7 [55] und PKCS#8 [56] sind. Sobald ein Nachrichtendienst zur Kommunikation zwischen Agenten bzw. zwischen Agent und Benutzer (siehe Abschnitt 3.1.3) realisiert ist, ließen sich darauf Agenten-Kommunikationssprachen wie KQML und ACL (siehe Anhang B) oder die *Remote Method Invokation (RMI)* aufsetzen.

Im Folgenden werden nun die wichtigsten Komponenten und Schnittstellen der SeMoA-Plattform erläutert:

### Services und Registry

In der *registry* des SeMoA-Servers werden alle Dienste (`Service`) registriert, die von dem Agentenserver angeboten werden, wobei jeder Dienst einer bestimmten Sicherheitsebene zugeordnet ist. Dies erlaubt später eine differenzierte Zugriffskontrolle auf die einzelnen Funktionen. Neben der Basisfunktionalität können beim Starten des Servers bereits automatisch Informationsdienste installiert werden, die aufgrund der Tatsache, dass sich ihre Klassenstruktur auf dem lokalen Dateisystem befindet, als vertrauenswürdig eingestuft werden, und deswegen eine privilegierte Position einnehmen können. Diese sogenannten *plugin services* (`PluginService`) können z. B. durch den direkten Zugriff auf eine Datenbank Informationen liefern und aufnehmen.

Aus der Sicht eines Agenten sind es genau diese Dienste, die einen Agentenserver attraktiv machen. Ein Agent kann eine Liste aller zur Verfügung stehenden Dienste in einer bekannten Sicherheitsstufe anfordern und einen der registrierten Dienste in Anspruch nehmen, sofern er dazu berechtigt ist. Darüber hinaus ist es ihm auch möglich, während seiner Ausführungszeit eigene Dienste zu registrieren und anschließend wieder abzumelden. Der Kontakt zwischen

Agenten untereinander kann innerhalb des Servers ebenfalls nur durch die Nutzung von angemeldeten Diensten stattfinden.

Diese Organisation ist auch ein ausschlaggebender Punkt dafür das die Basisfunktionalität eines SeMoA-Servers leicht durch neue Module, die z. B. durch statische Agenten als Dienst registriert werden, erweitert werden kann.

#### Transportmechanismus

Der Transportmechanismus in SeMoA, der für die Migration der Agenten zuständig ist, besteht aus zwei Portalen (dem `InGate` und dem `OutGate`), die wiederum in der Form von Diensten realisiert sind: Beide Portale implementieren Module für unterstützende Transportprotokolle und kontrollieren über diese den Netzwerkverkehr in und aus dem Server.

Gelangt ein Agent, durch ein unterstütztes Protokoll gekapselt, an das eingehende Portal, so sucht das `InGate` die *registry* in einer bestimmten Sicherheitsebene nach Filtern ab, denen die Datenpakete nacheinander übergeben werden. Nach Transformation der Daten bzw. inhaltlicher Untersuchung hat jeder Filter die Möglichkeit den Agenten zu akzeptieren oder abzulehnen, und nur wenn der Agent alle Filter passiert hat, wird er auch installiert und gestartet. Terminiert ein Agent, so wird das `OutGate` getriggert, dass wiederum den Agenten durch eine Reihe von gefundenen Filtern schickt, bevor er an seine nächste Zieladresse geschickt wird, und auch nur dann, falls das gewünschte Transportprotokoll unterstützt wird. Diese Zieladresse wird dem sogenannten `Ticket` entnommen, einer Datenstruktur, die der Agent vor seiner Terminierung mit den entsprechenden Daten füllen muss.

#### EventReflector

Der `EventReflector` stellt eine weitere Möglichkeit dar, innerhalb des SeMoA-Server auf Ereignisse zu reagieren. Falls ein Agent oder ein Dienst einen entsprechenden `EventListener` installiert hat, kann er dann im gegebenen Fall auf bestimmte Ereignisse direkt reagieren, die von andern Diensten ausgelöst wurden. Auf diese Art und Weise kann z. B. ein statischer Agent über die Ankunft eines neuen Agenten informiert werden.

#### Agenten

Die Agenten selber laufen auf dem Agentenserver innerhalb einer gekapselten Ausführungsumgebung ab, indem sie in einer eigenen `ThreadGroup` gestartet werden und ihren eigenen `ClassLoader` besitzen. Sie haben Zugriff auf ein mit ihnen assoziiertes Verzeichnis auf dem lokalen Dateisystem, in dem sich alle benötigten Klassen, Konfigurationsparameter und mitgeführte veränderlichen Daten befinden. Diese Struktur repräsentiert den Agenten während seiner Migration durchs Netzwerk (siehe auch [78]). Für den Transport wird diese Verzeichnisstruktur dann in ein sogenanntes *Java Archiv (JAR)* umgewandelt und als Datenpaket verschickt. Die wichtigste Schnittstelle zu dieser Verzeichnisstruktur und zur Außenwelt, in diesem Fall dem Agenten-Server mit seinen Diensten, stellt das `Environment` dar, dass der Agenten-Server für jeden Agenten einrichtet.

Wird der Agentenserver nach dem Herunterfahren durch den Benutzer wieder gestartet, so besteht die Möglichkeit alle in ihrer Ausführung unterbrochenen Agenten ebenfalls zu reaktivieren (*respawn*).

### 2.2.2 Sicherheitspolitik

Im letzten Abschnitt wurde deutlich, wie die einzelnen Komponenten des Agentenservers miteinander interagieren und dass dem Benutzer von SeMoA durch diesen Aufbau ein leistungsfähiges Agentensystem zur Verfügung gestellt wird. Dieses kann leicht auf die jeweiligen Bedürfnisse angepasst und durch das Hinzufügen von Modulen, in Form von Agenten die Dienste registrieren, erweitert werden. In diesem Abschnitt wird jetzt aber noch einmal etwas detaillierter auf manche kryptographische Mechanismen eingegangen.

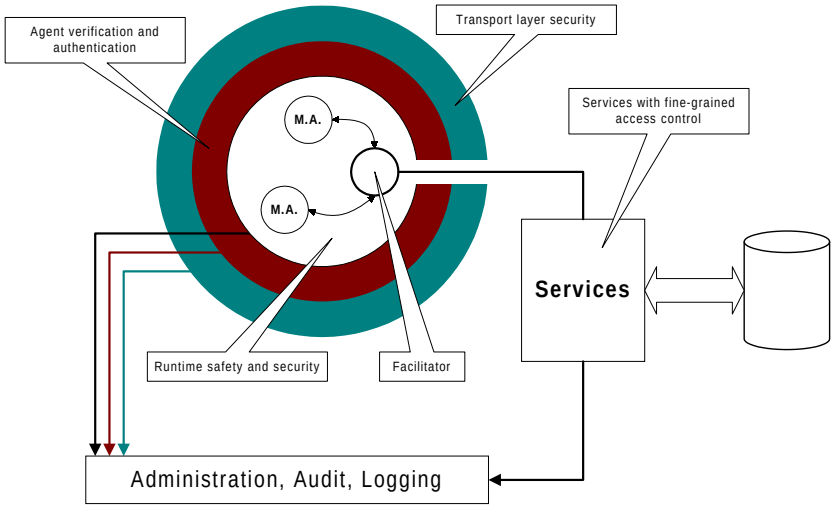


Abbildung 2.1: Die Sicherheitsmechanismen der SeMoA-Plattform

Die Sicherheitsmechanismen der SeMoA-Plattform, die auch in Abbildung 2.1 zu erkennen sind, lassen grundsätzlich sich in drei Ebenen einteilen: Durch Verschlüsselung lassen sich in der Transportebene Informationen vor unbefugten Zugriff sicher (*transport layer security*). Digitale Signatur ist in der darunter liegenden Sicherheitsebene das Mittel, um die Herkunft von Informationen zu identifizieren und personengebundene Berechtigungen zu vergeben (*agent verification and authentification*). Erreicht ein Agent die dritte Ebene, und wird damit tatsächlich in der Umgebung eines Agentenservers ausgeführt, so treten die Laufzeit-Sicherheitsmechanismen in Kraft (*runtime safety and security*), die zum größten Teil schon im vorigen Abschnitt beschrieben worden sind.

Um die Umsetzung der Sicherheit in den ersten beiden Ebenen zu erläutern muss noch einmal die Struktur eines Agenten beim Transport und das Verhalten von `InGate` und `OutGate` betrachtet werden bzw. die Vorkehrungen, die schon vorher beim Kreieren eines Agenten vom Benutzer getroffen werden müssen:

Die Struktur eines Agenten besteht bei SeMoA prinzipiell aus zwei Teilen, was sich auch in der Repräsentation des Agenten als Verzeichnisstruktur im Dateisystem wiederspiegelt: Einem *statischen Teil*, der die Klassenstruktur und feste initiale Konfigurationsparameter enthält und einem *veränderlichem Teil*, in dem der interne Zustand und mitgeführte Informationen gespeichert werden.

Um den Agenten vor ungewollten Manipulationen am Programmcode zu schützen, signiert der Besitzer einmal zu Anfangs den statischen Teil des Agenten. Damit lässt er sich später auch als tatsächlichen Besitzer identifizieren. Des weiteren steht im offen, Teile des veränderlichen Teil mit den öffentlichen Schlüsseln bestimmter Agentenserver zu verschlüsseln und diese Informationen somit nur bestimmten Benutzergruppen zur Verfügung zu stellen (siehe auch Abschnitt 2.1.4). Näheres über die auf diese Weise ermöglichte Zugriffskontrolle und die nötige Schlüsselverwaltung und Struktur des Agenten findet sich in [78].

Der zuvor serialisierte und auf diese Weise geschützte Agent wird nun zum Transport in ein erweitertes JAR-Archiv gepackt, in dem auch die benötigten Zertifikate und kryptographischen Daten nach PKCS7 [55] gespeichert werden [77].

Erreicht dieser Agent das `InGate` eines SeMoA-Servers, so durchläuft er zuerst den `VerifyFilter`, der die Signaturen überprüft und damit die Datenintegrität sicherstellt. Anschließend versucht der `DecryptFilter` mit dem privaten Schlüssel des Agentenservers, die für diesen Server gedachten Verzeichnisse zu entschlüsseln. Verlässt der Agent den Server, so können Teile der Klartextinformationen durch den `EncryptFilter` für einen gewissen Empfänger verschlüsselt werden. Anschießend wird der gesamte Agent vom `SignFilter` signiert.

Durch dieses Modell lassen sich nun einfach gesehen die folgenden vier Zugriffsrechte auf Daten vergeben, die der Benutzer verwenden kann um gegebenenfalls sensitive Daten vor ungewolltem Zugriff zu schützen [77].

- read only durch Signatur vom Besitzer
- read/write mit Bestätigung durch Signatur des Agentenservers
- group read only durch Verschlüsselung vom Benutzer
- group read/write

Abgesehen von der impliziten Sicherheit, die durch die Basisfunktionalität von SeMoA also bereits erzwungen wird und die meisten in Abschnitt 2.1.4 angesprochenen Angriffsmöglichkeiten verhindert, kann der Benutzer auch noch explizit Zugriffsrechte für Teilinformationen des Agenten vergeben. Darauf aufbauend hat der Benutzer die Möglichkeit, eigene Sicherheitsprotokolle einzuführen, sei es durch die Installation weiterer Filter beim Transportmechanismus, oder durch *secret sharing* bzw. Interagentenkommunikation im Netzwerk [74].

## 2.3 Namensdienste

Die Aufgabe dieser Arbeit ist es, einen Namensdienst für Mobile Agenten Systeme zu entwerfen und für die im letzten Abschnitt beschriebene SeMoA-Plattform in Form eines Prototyps

umzusetzen und zu evaluieren. Um die Notwendigkeit eines Namensdienstes zu verstehen und sich über die erweiterten Anforderungen durch die spezielle Aufgabenstellung klar zu werden, gebe ich in diesem Abschnitt nun erst einmal einen kurze Überblick über die bisherige Entwicklung der Namensdienste.

Anhand des DNS-Protokolls als Beispiel erläutere ich die Anforderungen an einen Namensdienst im heute gebräuchlichem Sinn, um im Abschnitt 2.4 konkret auf die erweiterten Anforderungen eingehen zu können, die bei einem Namensdienst für mobile Objekte wichtig sind. Im Anschluss versuche ich zudem verschiedene Implementierungsvarianten zu klassifizieren, um die in den Abschnitten 2.5 und 2.6 vorzustellenden Dienste und Protokolle einordnen zu können.

### 2.3.1 Entstehung der Namensdienste

Der Begriff des *Namensdienstes* kam parallel mit der Vernetzung von Rechnersystemen auf: Denkt man dabei an eine Konstellation von über ein Netzwerk miteinander verbundenen Workstations, an denen sich verschiedene Benutzer einloggen und daraufhin Applikationen starten können, die über dieses Netzwerk miteinander kommunizieren sollen, so lässt sich das leicht erklären.

Um in diesem Szenario die gewollte Kommunikation über das Netzwerk zu ermöglichen und regulieren zu können, müssen gewisse Informationen wie Maschinenadressen, Benutzernamen, Kennwörter oder Zugriffserlaubnisse bekannt sein. In kleinen Netzwerken reichte es aus, eine Kopie diese Informationen z. B. in Form einer Datei auf jeder Workstation zu speichern und diese bei gegebenen Struktur-Änderungen manuell zu aktualisieren.

Doch mit wachsender Größe der Netzwerke kam immer mehr die Notwendigkeit auf, diese Daten zentral zu verwalten, um zum einen die Administration des Netzwerkes, die meist in der Hand einer oder weniger Personen liegt, zu vereinfachen und zum anderen die Konsistenz dieser Daten immer noch gewährleisten zu können.

Diese zentrale Verwaltung der Daten wurde in den ersten Systemen dadurch realisiert, dass die verschiedenen Workstations z. B. über ein gemeinsames Dateisystem (man denke an die Verzeichnisstruktur von UNIX) Zugriff auf Dateien hatten, die sich an einer ganz bestimmten Stelle in diesem Dateisystem finden ließen und die benötigten Informationen enthielten. Änderte sich in diesem Modell die Netzstruktur, so musste der Administrator diese Änderungen nur noch an einer Stelle aktualisieren.

Die Weiterentwicklung dieser Idee war die Einführung einer Softwareebene die *lookup* und *update* Anfragen auf diesen Daten entgegen nahm und beantwortete. Die Daten konnten dadurch auch in einer Datenbank gespeichert werden, auf die dann mittels einer fest definierten Schnittstelle zugegriffen werden konnte. Außerdem stellte diese Softwareebene eine gewisse Kapselung der Daten dar, die den Zugriff mittels Authentifikation und Autorisation regulieren konnte.

Der Schritt hin zu einer Client-Server Architektur, bei der Clients über klar spezifizierte Protokolle auf die Daten eines Servers zugreifen können, ist dann nur noch klein. Durch

die Verpackung der Serveranfragen in Datenpakete, die über das Netzwerk transportiert werden können, wird dabei eine deutliche physikalische Trennung von Client und Server möglich.

Einer der Hauptaspekte war und ist immer noch die Zuordnung eines beschreibenden Namens zu der, meist durch eine Folge von Ziffern repräsentierten, physikalischen Adresse einer Ressource im Netzwerk, sei es nun die Ethernet- oder die IP-Adresse einer Workstation im LAN bzw. dem Internet. Der Dienst, der diese Aufgabe erfüllt, wird nun treffender Weise als *Namensdienst* bezeichnet.

Der Grund, einer Netzwerkressource neben ihrer physikalischen Adresse noch einen Namen zu geben, ist dabei zum einen die einfachere Handhabung für den Benutzer. Diesem wird es leichter fallen, sich einen Namen zu merken, der je nach Wahl viel eher mit der Ressource assoziiert wird, als eine willkürliche Ziffernfolge. Zum ergibt sich durch die Aktualisierbarkeit der Bindung von Namen zu Adressen Flexibilität: Verändern sich Netzwerkstrukturen und Maschinenadressen, so können die Namen beibehalten werden, umgekehrt ist es aber auch möglich bei gleichbleibender Netzwerkstruktur die Namen neu zu vergeben.

### 2.3.2 Anforderungen an ein herkömmlichen Namensdienst

Auf den verschiedenen Rechnersystemen entstanden im Zuge dieser Entwicklung Protokolle wie WINS, NetInfo, NIS/NIS+ oder DNS, die alle immer noch im Einsatz sind (siehe auch Abschnitt 2.5).

1983 wurden die ersten Standardisierungsideen zu dem *Domain Name System (DNS)* veröffentlicht. Parallel dazu wird in dem Artikel von Lampson 1986 DEC's Entwurf des *Global Name Service (GNS)* erörtert [40], das DNS sehr ähnlich ist. Mittlerweile ist das DNS-Protokoll, dessen wichtigste Aufgabe immer noch die Bindung von einem Domänennamen an eine IP-Adresse ist, durch die weite Verbreitung des TCP/IP-basierten Internets, das wohl bekannteste Protokoll zur Namensauflösung im Internet, aber auch im LAN-Bereich von Firmennetzwerken. Es wurde mit der Zeit erweitert und erfüllt dadurch heute immer noch weitgehend zufriedenstellend seine Aufgabe.

Aus diesem Grund werde ich anhand dieses Protokolls jetzt die ursprünglichen und heute wichtig gewordenen Anforderungen und Designziele erläutern, die für einen globalen Namensdienst im Internet gelten:

**Verteilte Speicherung der Daten:** Bedingt durch die große Datenmenge und Frequenz der Anfragen in einem globalen Netz soll eine Aufteilung der Daten auf verteilte Rechnersysteme möglich sein. Es soll keine Notwendigkeit bestehen und auch vermieden werden, eine Kopie der gesamten Datenbank erstellen zu müssen. Dies ist dadurch möglich, dass man das System z. B. hierarchisch strukturiert und dadurch die Delegation der Zuständigkeiten an verschiedene Systeme erlaubt.

**Konsistenter Namensraum:** Die Zuordnung von Namen zu konkreten Netzwerkressourcen (bzw. deren Adressen) soll eindeutig sein. Bei gleichen Anfragen soll unabhängig von deren Ausgangsposition immer die gleiche Antwort zurückgegeben werden, und der Name

sollte wenn möglich weder Netzwerkidentifikatoren und Adressen noch *routing* Informationen beinhalten. In der Realität werden Namen allerdings häufig auf mehrere IP-Adressen abgebildet, um z.B. Toleranz gegenüber vereinzelten Rechnerausfällen zu ermöglichen.

**Unabhängigkeit von gegebenen Strukturen:** Um den Namensdienst unabhängig von darunter liegenden Netzstrukturen zu halten, ist die Unterstützung mehrerer Transportprotokolle bei Transaktionen zwischen Client und Server nötig (z. B: TCP und UDP).

**Verlässlichkeit:** Durch Replikation der Daten auf getrennten Rechnern, soll eine gewisse Sicherheit gegen Netzausfälle garantiert werden.

**Autonomie:** Administratoren sollen in ihrem Autorisationsbereich unabhängig von andern Änderungen vornehmen können.

### 2.3.3 Grundlegende Konzepte

Allen Namensdiensten gemein ist die Möglichkeit, Suchanfragen (*lookup*) stellen zu können. Dabei wird allerdings unterschieden, ob dies über direkten Zugriff auf eine Datei bzw. Datenbank, den Aufruf einer lokalen Schnittstellenfunktion oder das Senden einer Nachricht an einen Namenserver geschieht.

Bei den Änderungsanfragen (*update*) unterscheidet man ebenfalls verschiedene Stufen. Der Administrator kann in manchen Fällen gezwungen sein, Änderungen manuell vornehmen zu müssen. Daneben existieren die drei bereits beim *lookup* unterschiedenen Methoden. Außerdem ist besonders beim *update* manchmal die Möglichkeit der Authentifikation und Autorisation des Anfragestellers wichtig.

Beim Namensraum wird unterschieden, ob dieser flach oder hierarchisch organisiert wird. Damit entscheidet sich meistens auch, ob die Daten zentral auf einem Server oder dezentral und verteilt auf mehreren Servern gespeichert wird.

Betrachtet man im Besonderen bewegliche Objekte im Netzwerk, stellt sich beim *update*, über die Frage nach dem *wie* hinaus, auch die Frage, *wer* und an *wen* die *update* Anfragen gestellt werden. Hält man sich auch das Modell der mobilen Agenten aus Abschnitt 2.1 vor Augen, so ergeben sich die folgenden vier *update* Szenarien [75]:

**Direct Response** In diesem Fall benachrichtigt das mobile Objekt bei seinem Positionswechsel jedes mal direkt den Homeserver über seinen neuen Standort.

**Buffered Response** Die Position des mobilen Objektes wird hier zentral, aber bei einem speziell als Namensserver ausgewiesenem Computer im Netzwerk gespeichert, indem dieser vom den mobilen Objekten benachrichtigt wird.

**Forward References** Anstatt die Positionsdaten des mobilen Objektes in einer Datenbank zentral zu speichern, merkt sich jeder Computer, auf dem das Objekt verweilt hat, den nächsten Aufenthaltsort, zu dem es unterwegs war. Dadurch entsteht ausgehend vom Homeserver ein Pfad, der bei der aktuellen Position des Objekts endet.

**Searching** Sind die Server bekannt, die das Objekt in seinem Lebenszyklus besuchen will, so kann eine Suche durch Anfragen an die entsprechenden Server gestartet werden, um das Objekt zu finden.

Die Benachrichtigung eines Namensservers kann in diesen Fällen übrigens vom Objekt selbst, gegebenenfalls aber auch von der Ausführungsumgebenen ausgehen.

Darüber hinaus kann man bei der Verbreitung von Protokollnachrichten im Netzwerk zwischen einem verbindungsorientierten Transport (z. B. über TCP) und einem verbindungslosen Transport (z. B. über UDP) unterscheiden. In kleinen lokalen Netzwerken ließen sich auch *broadcast* Anfragen vorstellen, mit denen Server ausfindig gemacht werden.

Natürlich kann es bei all diesen Unterscheidungsmerkmalen auch hybride Ansätze geben.

### 2.3.4 Verwendung eines Namensdienstes im weiteren Sinn

Hinter der Zuordnung eines Namens zu einer Ressource im Netz steht vor Allem auch die Idee, eine Ressource im Netz lokalisieren zu können. Dies wiederum bietet mehrere mögliche Anwendungsszenarien, die direkt auf dieser Funktionalität aufbauen können.

**Tracing:** Die Lokalisierung von Objekten ist vor Allem dann interessant, wenn diese ihre physikalische Position im Netz häufiger verändern. Unter Tracing versteht man dann die Aufzeichnung des Pfads, den diese Objekte bei dem Wandern durch das Netzwerk nehmen.

**Messaging:** Ist die aktuelle Position eines Objektes erst einmal bekannt, so wäre der nächste Schritt, mit diesem Objekt durch den Austausch von Nachrichten zu kommunizieren.

**Controling:** Mit der Möglichkeit der Kommunikation mit Objekten im Netz ist dann auch an die (Fern-)Steuerung dieser Objekte zu denken.

Diese drei Szenarien werden besonders in Kapitel 3 noch einmal aufgegriffen und detailliert diskutiert, um den Sinn eines Namensdienst für Mobile Agenten Systeme zu erklären.

## 2.4 Anforderungen und Problemstellung

Die Anforderungen an einen Namensdienst, die bereits in Abschnitt 2.3.2 für das Domain Name System formuliert wurden, besitzen eine gewisse Allgemeingültigkeit. Dabei wird allerdings von einem Szenario ausgegangen, dem relativ statische Netzstrukturen zugrunde liegen. Damit ist die Anzahl der *lookup* Anfragen um ein Vielfaches größer als das der *update* Anfragen. Der Namensdienst dient dort hauptsächlich der Zuordnung eines Hostnamens zu seiner physikalischen IP-Adresse.

Die Aufgabe dieser Arbeit (siehe auch Abschnitt 1.2) ist es, einen sicheren Namensdienst für Mobile Agenten Systeme zu entwerfen. Damit weist das Ausgangsszenario schon grundlegende Unterschiede zu obigen auf: Es sollen mobile Agenten lokalisiert werden und damit mobile Objekte, die relativ häufig ihre Position wechseln, ohne dabei einem bestimmten,

vorher absehbaren Migrationsmuster zu entsprechen. Die Lebenszeit dieser Objekte ist im Vergleich zu der Erreichbarkeit der meisten Hosts im Internet sehr kurz. Damit ist die Anzahl der *update* Anforderungen an den Namensserver ziemlich groß und teilweise vielleicht sogar höher als die Anzahl der *lookup* Anforderungen. Der Namensdienst dient in diesem Fall der Zuordnung eines eindeutigen Bezeichners für einen Agenten zu der kodierten Positionsangabe (z. B. durch eine URL).

Durch das veränderte Ausgangsszenario kommen neue Anforderungen hinzu und es fallen andere weg, bzw. liegt das Hauptaugenmerk einfach auf anderen Anforderungen. Im Folgenden werden diese neuen Anforderungen, eingeteilt in vier Kategorien, dargestellt und dienen in den beiden Abschnitten 2.5 und 2.6 dann als Grundlage für die Diskussion der dort vorgestellten Namensdienste und Protokolle.

### 2.4.1 Allgemeine Anforderungen

- Das Modell des Namensdienstes für Mobile Agenten Systeme soll im weitesten Sinne generisch und damit auch unabhängig von SeMoA einsetzbar sein.
- Durch die hohe Mobilität der Agenten ist es notwendig, Anfragen schnell bearbeiten bzw. beantworten zu können, bevor die Information wieder veraltet ist.
- Wegen der hohen Änderungsfrequenz und der geringen Lebenszeit der Agenten sollte an zeitliche Vorgaben für die Gültigkeit der Einträge gedacht werden.

### 2.4.2 Skalierbarkeit

- Die Kommunikationslast und das Speicheraufkommen sollten verteilt werden, d.h. das Protokoll sollte eine Vielzahl von Namensservern unterstützen.
- Der Namensdienst sollte sowohl in kleinen, abgekoppelten Netzwerken als auch im Internet zu installieren sein, d.h. die Anzahl der aktiven Namensserver sollte regulierbar sein.
- Das Format des eindeutigen Bezeichner eines Agenten sollte einen Namensraum zur Verfügung stellen, der ausreichend groß ist.

### 2.4.3 Sicherheit

- Das Protokoll sollte die transportierten Informationen möglichst geheim halten und auf diese Weise Angriffe auf die *privacy* durch das Abhören der übermittelten Nachrichtenpakete erst gar nicht ermöglichen.
- Besonders die Anfragebearbeitung des Namensservers aber auch das Protokoll an sich sollten möglichst robust gegen DoS Angriffen sein.
- Autorisation bei *update* Anfragen sollte bedacht werden. *Lookup* Anfragen sollen dagegen grundsätzlich jedem möglich sein.

### 2.4.4 Fehlertoleranz

- Im Modellentwurf sollte die Möglichkeit vorgesehen werden, dass auch in durch *routing* Probleme oder absichtlich abgekoppelten Netzwerksystemen der Namensdienst nicht komplett zusammenbricht.
- Namensserver sollten nach einem Neustart, ob nach Rechnerabsturz oder beabsichtigten Herunterfahren, auf Anfragen wieder einen konsistenten Zustand des Netzwerksystems und in ihm befindlichen Agenten zurückgeben.

## 2.5 Existierende Namens- und Verzeichnisdienste

In diesem Abschnitt werden nun einige konkrete Namens- und Verzeichnisdienste vorgestellt. Nach der Beschreibung deren Struktur und die Einordnung in die in Abschnitt 2.3.3 aufgestellten Kategorien, werden diese Modelle in Hinblick auf deren Verwendbarkeit für die Aufgabenstellung nach den in Abschnitt 2.4 gestellten Anforderungen diskutiert.

Neben dem in Abschnitt 2.3 vorgestellten Modell des Namensdienstes ist dabei die Definition eines Verzeichnisdienstes etwas allgemeiner zu fassen: Gemein ist beiden Diensten die Zuordnung von Informationen zu einem eindeutigem Namen oder Bezeichner. Diese Informationen können auch beim Verzeichnisdienst auf mehrere Server verteilt, und der Zugriff auf diese durch bestimmte Protokolle gewährleistet werden. Normalerweise unterscheidet sich allerdings die Art der gespeicherten Informationen. Im Gegensatz zum Namensdienst, der Informationen zu Netzwerkressourcen verwaltet, ist ein Verzeichnisdienst eher als strukturierte Datenbank zu sehen, die jegliche Art von Informationen speichert. Die Verwaltungen von persönlichen Daten (Kontaktmöglichkeiten wie Adresse, Email, Zimmernummer) von Firmenmitarbeitern wäre hierfür als Beispiel zu nennen.

Die *International Telecommunication Union (ITU)* hat ihre Vorstellungen von einem Verzeichnisdienst in den Spezifikationen X.500 (und folgenden) festgehalten [65].

Einen guten Überblick über die benötigten Protokolle bei der Namensauflösung, besonders in Windows Netzwerken, deren Zusammenspiel und Konfiguration bekommt man in ausführlicher Form in [38] präsentiert.

Da sich viele Komponenten bei den verschiedenen Namediensten wiederholen, wird das Domain Name System etwas detaillierter beschrieben und bei den folgenden Abschnitten hauptsächlich Unterschiede dargestellt.

### 2.5.1 DNS

Der *Domain Name System (DNS)* stellt einen hierarchisch organisierten Namenraum für Netzwerkressourcen dar und ist durch das Internet zu dem mittlerweile bekanntesten Namensdienst für TCP/IP-Netzwerke geworden. Da sich die Anforderungen über die Zeit verändert

haben, wurde auch die Spezifikation von DNS mehrmals überarbeitet und um einige Komponenten ergänzt. Die Spezifikation des heute gültige Standards findet sich in [50] und [51]. Einen guten Überblick, Beschreibungen und Spezifikationen von Erweiterungen finden man online unter [81].

### Struktur

Das komplette System dieses Namensdienstes besteht aus den folgenden Komponenten:

**Domain Name Space und Resource Records** Der Namensraum ist in Form eines Baumes mit einer fest definierten Wurzel (*root*) und darunter liegenden Knoten organisiert. Der *domain name* baut sich aus, durch Punkte voneinander getrennten, Bezeichnern auf. Diese Bezeichner (von rechts nach links gelesen) geben den Pfad von der Wurzel des Baumes zu genau einem Knoten wieder, in dem nun die zum Domain Name zugeordneten Informationen in Form von sogenannten *resource records* gespeichert werden.

Der Namensdienst ordnet einem *domain name* also eindeutig eine Reihe von *resource records* zu, die je nach Typ verschiedene Informationen enthalten können. Der *domain name* an sich kann nun einen ganzen Teilbaum, einen Rechner im Internet oder einen Benutzer bezeichnen. Die in der Basisspezifikation des DNS vorgesehenen *resource record* Typen speichern vor Allem Informationen, welche die Kommunikation mit identifizierten Rechnern oder den Mailkontakt mit identifizierten Benutzern ermöglichen.

Ursprünglich wurden diese *resource records* in sogenannten *masterfiles* eines Namensservers gespeichert und vom Administrator gegebenenfalls manuell verändert. Aus diesem Grund existiert zu jedem *resource records* Typ auch eine textuelle Repräsentation.

**Name Server** Im Gegensatz zur abstrakten Struktur des *domain name space*, bei dem für jeden Bezeichner genau eine Knoten vorgesehen ist, werden die den *domain names* zugeordneten Informationen auf den sogenannten *Namensservern* zusammengefasst.

Auf der abstrakten Ebene wird der *domain name space* in sogenannte *Domänen* eingeteilt, die einem kompletten Teilbaum des Namensraums entsprechen. Betrachtet man allerdings die tatsächliche Namensserver-Infrastruktur, so spiegelt diese eher eine Einteilung in Zonen wieder.

Eine Zone entsteht innerhalb von Domämen durch Schnitte zwischen belieben, adjazenten Knoten der Baumstruktur, die eine Gruppe von miteinander verbunden Knoten zusammenfassen. Der im Namensraumbaum der Wurzel nächste Knoten innerhalb der Zone identifiziert diese Zone. Eine Zone enthält die *resource records* für alle enthaltenen Knoten und ebenfalls in einem bestimmten *resource record* gespeicherte Verweise auf zuständige Namensserver für alle Subzonen. Dabei wird eine Zone aus Sicherheitsgründen mindestens auf zwei Namensservern gespeichert, wobei ein Namensserver durchaus die Informationen zu mehreren Zonen enthalten kann. Die Replikation einer Zone entspricht also dem Transfer von *resource records* in Form von einzelnen Nachrichten oder dem gesamten *masterfile*.

**Resolver** Ein *resolver* extrahiert auf Benutzeranfragen hin die gewünschten Informationen vom Namensserver. Er muss fähig sein, zumindest einen Namenserver kontaktieren zu können. Enthält dieser Namensserver nicht die gewünschten Informationen, so wird die

Anfrage rekursiv durch den Namensserver bzw. iterativ durch den Resolver weitergereicht, bis der angeforderte gewünschte *resource record* gefunden wird. Um dieses Vorgehen nicht bei jeder *lookup* Anfrage wiederholen zu müssen, werden Anfrageergebnisse und die für bestimmte Domänen zuständigen Namensserver im *cache* gehalten, was die Bearbeitungsgeschwindigkeit der Anfragen erheblich beschleunigt. Eine Applikation nutzt diese Funktionalität normalerweise direkt durch den Aufruf von Methoden einer definierten Schnittstelle. Sie muss das sogenannte *domain protocol*, das zur Kommunikation mit einem Namensserver übers Netzwerk benutzt wird, also nicht verstehen.

**Domain Protocol** Das *domain protocol* zur Kommunikation mit Namensservern über das Netzwerk definiert Nachrichtentypen für Benutzeranfragen bzw. Antworten, die über UDP bzw. TCP übertragen werden können. Dabei folgen vor Allem bei den Antworten die angefragten *resource record* einfach im Klartext einem speziellen Nachrichtenkopf.

**(Security) Extensions** Mit steigenden Anforderungen an DNS wurden einige Erweiterungen eingeführt, die sich besonders in Form von zusätzlichen *resource record* Typen und weiteren Nachrichtentypen für das *domain protocol* bemerkbar machen. Zum einen ist es mittlerweile möglich, Änderungen an den *resource records* auch dynamisch durch *update* Anfragen vorzunehmen [90] [13]. Zum anderen wurde neue *resource records* definiert, die *public keys* und digitale Signaturen [9] wie auch Zertifikate [14] speichern können. Dabei wird die Einführung von *public key* und *signature resource records* auch dazu benutzt, die durch das *domain protocol* möglichen Transaktionen zu authentisieren und autorisieren.

Komplexe *Usecases*, die das Zusammenspiel der verschiedenen Komponenten noch detaillierter erläutern, finden sich in [51]. Darüber hinaus wird allgemein die Namensauflösung für TCP/IP Netzwerke und Konfiguration von DNS-Namensservern, mit weiterer Differenzierung bei der Funktionalität von Name Servern, in [21] beschrieben.

### Analyse

Das Domain Name System baut auf der relativ statischen Rechnerstruktur des Internets auf und verwaltet dort mit der aktuellen Infrastruktur Hosts in der Größenordnung von $10^8$ bezüglich deren Anzahl. Durch den hierarchischen Namensraum und die Nutzung von Replikation wird die Kommunikationslast und das Speicheraufkommen verteilt und zudem eine gewisse Fehlertoleranz gegenüber vereinzelten Namensserver-Ausfällen gewährleistet. Dabei entstehen voneinander unabhängige Administrationsbereiche, in denen Modifikationen meist manuell vorgenommen werden. *Caching* der Adressen zuständiger Namensserver und bereits erhaltener Anfrageergebnisse stellt in diesem Fall ein sehr effektives Mittel dar, um bei erneuten Anfragen den Weg von Hierarchieebene zu Hierarchieebene bei der Suche nach der angefragten Information abzukürzen bzw. sogar direkt antworten zu können.

Die zu verwaltenden Objekte bei Mobile Agenten Systemen bewegen sich allerdings in der Regel ohne festes Migrationsmuster global im gesamten Netzwerk. Dadurch ist es nicht mehr sinnvoll, durch Mechanismen wie Zoneneinteilungen Lokalität ausnutzen zu wollen. Unter der Voraussetzung einer viel höheren *update* zu *lookup* Rate wird darüber hinaus die Aktualität von Anfrageergebnissen immer wichtiger. Es kann nicht in Kauf genommen werden, Abfrageergebnisse aus einem *cache* wieder zu verwenden. Außerdem ist die Synchronisation

von Namensservern für die Replikation bei DNS aus dem gleichen Grund nicht schnell genug. Des weiteren sind die *update* Möglichkeit der verwalteten Daten durch das Vorhandensein gekapselter Administrationszonen zu beschränkt und restriktiv.

### 2.5.2 NIS/NIS+

Unabhängig von DNS ist für die Verwaltung und Administration kleinerer Netzwerke vor Allem unter UNIX-ähnlichen Plattformen der *Network Information Service (NIS)* entstanden, der mittlerweile durch NIS+ ersetzt wurde [22]. Im Gegensatz zu DNS, dessen ursprüngliche Intension nur die Umsetzung von sprechenden Namen für Netzwerkressourcen in IP-Adressen war, verwaltet der Network Information Service Informationen zur Verwaltung von Benutzergruppen, Netzwerkdiensten und der eigentlichen Netzwerkstruktur auch unterhalb der IP-Ebene. Die Anbindung eines NIS/NIS+-Netzwerkes bzw. einer NIS/NIS+-Domäne an das Internet geschieht wiederum über DNS.

#### Struktur

Bevor es einen Namensdienst gab, war es nötig, die benötigten Informationen zur Netzwerkadministration auf jedem Rechner bzw. an einem speziellen Ort im gemeinsam benutzten UNIX-Dateisystem in einer Datei zu speichern. NIS ersetzt diese Struktur nun durch eine Client-Server Struktur zur Zentralisation dieser Daten, wobei es wie bei DNS ebenfalls aus Sicherheitsgründen Replikationsserver gibt. Die Daten werden in den sogenannten *NIS-Maps* gespeichert, zweispaltigen Tabellen mit jeweils einem suchbaren Schlüssel- und einem zugeordneten Datenwert. Eine NIS-Domäne beinhaltet nun eine NIS-Map mit den enthaltenen Informationen über Rechner- und Benutzergruppen und den in diesem Rahmen angebotenen Netzwerkdiensten. Im Gegensatz zu DNS ist diese aber nicht hierarchisch strukturiert sondern flach.

Um größere Netzwerkstrukturen zu unterstützen wurde NIS+ eingeführt, das ähnlich wie das UNIX-Verzeichnissystem hierarchisch strukturiert ist und NIS ablöst. Der NIS+-Namensraum wird vergleichbar mit DNS in mehrere Domänen aufgeteilt, wobei Benutzer einer Domäne auch Zugriff auf Daten über Domänengrenzen hinweg haben. In NIS+ werden die Informationen im Gegensatz zu NIS in Multispaltentabellen einer relationalen Datenbank gespeichert. Mehrere Spalten einer Tabelle können suchbare Schlüsselwerte enthalten, womit auch inverse Suche ermöglicht wird. Außerdem wurde eine durch Authentifikation und Autorisation gesteuerte Zugriffskontrolle auf Tabellen-, Zeilen- und Spaltenebene eingeführt. Der Benutzer kann bei der Anforderung von Informationen explizit zwischen NIS, NIS+, DNS oder den lokalen Dateien wählen.

Erwähnenswert ist übrigens auch die Organisation der Serverstruktur bei NIS/NIS+: Zu aller erst wird versucht angefragte Daten im Speicher zu halten, um Zugriffe auf die Festplatte zu vermeiden. Bei NIS wurde die Datensicherung auf die Replikationsserver noch als Transfer der gesamten Datenbank umgesetzt. NIS+-Daten werden inkrementell an die Replikationsserver weitergegeben, indem der Hauptserver alle Transfers mit einem Zeitstempel versehen in einem *logfile* festhält und von Zeit zur Zeit an die Replikationsserver weitergibt. Ist der Hauptserver durch einen Ausfall nicht verfügbar, so können *lookup* Anfragen von den

Replikationsservern zwar noch beantwortet werden, *update* Anfragen sind dann allerdings nicht möglich.

#### Analyse

Der Network Information Service verwaltet eine Vielzahl von Informationen zur Netzwerkadministration. Durch die Einführung von NIS+ wurde, mit der Dezentralisierung der Datenbank, dem Aufheben von Restriktionen bei der Spezifikation der Datenstrukturen, der Einführung von Autorisation bei *update* Anfragen und der Verbesserung des Replikationsmechanismus, eine bessere Skalierbarkeit des Systems gewährleistet.

Nichts desto trotz ist dieses System für die Administration relativ statischer Strukturen in kleineren Netzwerke vorgesehen. Der Schwerpunkt liegt viel eher auf der Bereitstellung einer komfortablen Administrationsumgebung, und damit der Verwaltung von relativ komplexen Datenstrukturen, als auf der Bereitstellung eines schnellen und sicheren Namensdienstprotokolls, dass die einfache Zuordnung von Name zu Position ermöglicht.

### 2.5.3 NetInfo

Was NIS/NIS+ für UNIX-ähnliche Plattformen darstellt, ist *NetInfo* für *NeXTSTEP*, *OpenStep for Mach* und *Mac OS X Server*. Es verwaltet administrative Informationen, aber auch das NFS oder lokale Ressourcen wie lokale Drucker. Einen Überblick über die Funktion von NetInfo findet sich online unter [42], [6] und [5].

#### Struktur

NetInfo ist eine hierarchisch-verteilte Datenbank: Daten werden in einem Verzeichnisbaum gespeichert, wobei jedes Verzeichnis sogenannte *properties* enthalten kann. Ein Property hat einen Namen und kann mehrere Werte speichern. Zugriff auf diese Daten ist nur durch spezielle Tools möglich.

Jeder Rechner der einen NetInfo-Dienst startet, stellt gleichzeitig auch einen NetInfo-Server dar, da jeder Rechner eine lokale Domäne darstellt, welche die lokalen Rechnerressourcen beinhaltet. In einem NetInfo-Netzwerk ist aber auch mindestens eine Netzwerkdomäne vorhanden. Der Rechner, der diese verwaltet, enthält zu der Datenbank für die lokalen Daten also noch eine zweite.

*Lookups* werden entweder durch Schnittstellen zum NetInfo-Dienst oder direkt durch die *NetInfoAPI* initiiert. *Lookup* Anfragen werden dabei normalerweise erst lokal und dann schrittweise in höheren Hierarchieebenen bearbeitet, es ist aber auch möglich *lookup* Anfragen direkt an bestimmte Hierarchieebenen zu stellen.

Wird ein Rechner mit einem NetInfo-Dienst neu gestartet, so wird dieser über das *binding protocol* dynamisch in die Hierarchiestruktur eingebunden. Dafür sorgt der sogenannte *binding daemon*, der durch *broadcast* Nachrichten vorerst versucht, einen möglichen Vater in der Hierarchie zu finden, um sich bei diesem anschließend als Kind zu registrieren.

Wegen der Wichtigkeit der Daten für die einzelnen Rechner wird empfohlen regelmäßig Kopien der Domänendaten zu machen. Dies muss allerdings manuell geschehen und unter Wahrung der Konsistenz: Es dürfen während der Replikation der Datenbank also keine *update* Zugriffe bearbeitet werden.

#### Analyse

Nach NIS/NIS+ liegt der Schwerpunkt bei NetInfo noch mehr auf der Verwaltung von Ressourcen, die sich lokal auf einem Rechnersystem befinden, als auf der Zusammenarbeit mehrerer Rechnersysteme innerhalb eines Netzwerkes. Aus diesem Grund wird auch mehr Wert auf eine lokale Schnittstelle gelegt, die den Zugang zur lokalen Datenbank ermöglicht, als auf optimierte Protokolle als Schnittstelle zwischen verteilten Datenbanken oder dessen Administration.

Durch die gegenüber der Aufgabenstellung dieser Arbeit sehr eingeschränkten Anforderungen steht die Skalierbarkeit dieses Ansatzes im Hintergrund und damit auch die Verwendbarkeit für diese Arbeit. Die dynamische Einordnung von einzelnen Rechnersystemen in eine gewisse Hierarchie durch das *binding protocol* stellt allerdings einen interessanten Ansatz dar.

### 2.5.4 DHCP

Das *Dynamic Host Configuration Protocol (DHCP)* ist die Weiterentwicklung des *BOOT-Protocol (BOOTP)* und stellt damit eigentlich keinen eigenständigen Namensdienst dar. Es dient eher der Konfiguration von Rechnern, zur dynamischen Einbindung dieser in ein WINS- bzw. DNS-strukturiertes Netzwerksystem, das auf der TCP/IP-Protokollfamilie basiert. Somit lässt sich dieses Protokoll mit dem *binding protocol* von NetInfo aus Abschnitt 2.5.3 vergleichen. Innerhalb von Firmennetzwerken ermöglicht DHCP die dynamische Vergabe von IP-Adressen und vereinfacht dadurch die Arbeit der Netzwerkadministratoren. Diese Organisation der IP-Adressen-Vergabe hebt sich somit auch von den globalen und recht statischen Strukturen des DNS im Internet ab und erleichtert besonders mobilen Endgeräten die kurzzeitige Anbindung an lokale Netzwerke und über diese ans Internet.

Die Spezifikation von DHCP wird in [11] und [63] gegeben. Weitere Informationen finden sich auch in [38].

#### Struktur

Um die oben genannten Aufgaben zu erfüllen, müssen auf den einzelnen Rechnern sogenannte *DHCP-Clients* installiert sein. Bei Neustart der Rechners wird versucht, Kontakt zu einem *DHCP-Server* aufzubauen. Ist der Kontakt erst mal aufgebaut, können dem in das Netzwerk einzubindendem Rechner die Adressen von Gateway, DNS-Server, WINS-Server und Informationen über Domänenname und WINS-Knotentyp übermittelt werden. Anschließend wird dem Rechner dynamisch eine IP-Adresse und die entsprechende Subnetzmaske (*subnet mask*) zugewiesen.

Wie der Name schon sagt liegt der Schwerpunkt bei DHCP auf dem Protokoll. Aus diesem Grund wird jetzt kurz das Vorgehen beschrieben, um den Kontakt mit einem DHCP-Server aufzunehmen und dann eine IP-Adresse zu leasen.

- Der DHCP-Client sendet eine *DHCP discover* Nachricht via *broadcast* an alle Rechnern des LAN. Falls sich in diesem LAN ein sogenannter *BOOTP relay agent* befindet, kann dieser die empfangende Nachricht auch über die Grenzen des LAN an einen DHCP-Server in einem anderen Subnetz weiterleiten.
- Alle DHCP-Server, die diese Nachricht empfangen, antworten mit einer *DHCP offer* Nachricht an den Absender
- Der DHCP-Cient nutzt nun die erste Antwort, um seinen Ansprechpartner für das weitere Vorgehen zu identifizieren. Dadurch wird nebenbei auch gewährleistet, dass der DHCP-Server mit der momentan besten Verbindung (Datenübertragungsgeschwindigkeit) kontaktiert wird.
- Im weiteren Vorgehen wird über die DHCP-Nachrichten *request*, *ack*, *decline*, *nack* und *release* die Vergabe der IP-Adresse ausgehandelt und die Konfigurationsparameter übertragen.
- Die Vergabe der IP-Adressen wird *Leasing* genannt, da der DHCP-Server dessen Gültigkeit zeitlich einschränken kann. Auch wenn dem DHCP-Client also bereits einmal eine IP-Adresse zugewiesen wurde, muss dieser die Adresse bei jedem Neustart bzw. nach Ablauf der Gültigkeitsdauer erneut beantragen.

Im eigentlichen Sinne ist DHCP also kein Namensdienst, allerdings sind Aspekte eines Informationsdienstes vorhanden: Ein DHCP-Client wendet sich an einen unbekannten DHCP-Server, falls dieser ausfindig gemacht werden kann, und erhält von diesem Informationen. Die Art dieser Information wird in diesem Fall allerdings vom Server beschrieben und nicht explizit von Client angefragt.

#### Analyse

Das Dynamic Host Configuration Protocol beschränkt sich auf die dynamische Zuteilung von Konfigurationsparameter an anfragende Rechnersysteme innerhalb eines LAN, wobei ein DHCP-Client einen DHCP-Server durch *broadcast* Nachrichten im Subnetz ausfindig macht. Insofern hat es nur wenig mit einem Namensdienst im Sinne der Aufgabenstellung gemein. Gerade diese Idee werde ich allerdings in Kapitel 3 noch einmal aufgreifen, um einen Proxy-Server innerhalb eines LAN ausfindig zu machen und gegebenenfalls in die Namensdienststruktur einzubinden.

### 2.5.5 LDAP

Das *Lightweight Directory Access Protocol (LDAP)* ist ein Client-Server Protokoll für den Zugriff auf Verzeichnisdienste. Ursprünglich wurde es als *frontend* für *X.500 Server* (Spezifikation in X.500-X.521 [65]) verwendet, es kann allerdings auch mit alleinstehenden bzw.

anderen X.500 ähnlichen Verzeichnisdiensten in Einsatz gebracht werden. Das Protokoll definiert Transport- und Format-Nachrichten, die benutzt werden um Informationen aus dem Verzeichnisdienst abzufragen oder zu verändern, und ist mittlerweile zum Internet-Standard erhoben worden. Durch LDAP wird also nicht der eigentliche Verzeichnisdienst definiert.

Die Spezifikation von LDAP(v3) findet man in [46], wobei die genaue Darstellung der im Protokoll benötigten Attribute mit der entsprechenden Kodierung in [92] nach ASN.1 Manier (siehe Anhang A) gegeben wird.

Unter [33] und [4] findet man eine tiefergehende Einführung in das Thema.

#### Struktur

Das ursprüngliche Protokoll zum Zugriff auf X.500 Verzeichnisse nutzt noch den kompletten *OSI Protocol Stack*. Im Gegensatz dazu basiert LDAP auf dem bekannteren und „leichtgewichtigerem" TCP/IP Protocol Stack, ist *string* basiert und zudem unter Kontrolle der *Internet Engineering Task Force (IETF)*. Darüber hinaus vereinfacht es einige X.500 Zugriffsoperationen.

Wie auch bei Namensdiensten besteht hier die Zuordnung zwischen einem Schlüssel und den dazugehörigen Informationen. In diesem Fall wird der Schlüssel durch einen *Distinguished Name (DN)* [45] repräsentiert, der einen Datensatz eindeutig identifiziert.

Die 9 Basisfunktion von LDAP lassen sich in drei Gruppen einteilen: Die Abfragefunktionen *search* und *compare*, die Modifikationsfunktionen *add*, *delete*, *modify* und *modify DN* (*rename*) und die Funktionen *bind*, *unbind* und *abandon*, die unter Verwendung von Authentifikation und Autorisation eine Verbindung zum Verzeichnisserver aufbauen und Berechtigungen für Abfrage bzw. Modifikation der Datensätze vergeben.

Mittlerweile wird LDAP in Version 3 verwendet: Im Gegensatz zu LDAP(v2) ermöglicht LDAP(v3), Zertifikate in DER-Kodierung zu übermitteln (siehe Anhang A). Außerdem werden seit dieser Version alle Attribute in UTF-8 Kodierung des Unicode-Zeichensatzes übertragen. LDAP(v3) zeichnet sich auch durch seine Erweiterbarkeit in der Funktionalität der Protokoll-Nachrichten aus.

Durch die Einführung eines LDAP URL Formats ([32]) wird die Möglichkeit gegeben, direkt durch die Angabe einer URL den Protokolltyp LDAP festzulegen und *lookup* Anfragen zu formulieren.

Für den Austausch von Datenbankinformationen zwischen LDAP-basierten Verzeichnisdiensten dient das *LDAP Data Interchange Format (LDIF)* [29]. Dabei besteht eine 1-zu-1 Korrelation zwischen den LDAP Protokoll Nachrichten und der Spezifikation der sogenannten *change records*. Dieses Dateiformat wird für Import und Export ausgewählter Datensätze aber auch zum Kopieren von gesamten Datenbanken verwendet.

**Analyse**

Das Lightweight Directory Access Protocol stellt ein standardisiertes Protokoll als Schnittstelle zwischen einem Client und einem Verzeichnisserver dar, das durch die eindeutige Spezifikation in ASN.1 relativ unabhängig von den darunter liegenden Netzwerkprotokollen und beteiligten Rechnersystemen ist. Der unterschiedliche Zugriff auf Verzeichnisdaten bzw. dessen Modifikation wird durch die verschiedenen Nachrichtentypen ermöglicht. Außerdem stellt das LDAP URL Format darüber hinaus eine interessante Alternative dar, *lookup* Anfragen an Verzeichnisserver zu stellen.

Die Tatsache, dass LDAP-Anfragen vorerst eine Authentifikation und Autorisation beim Verzeichnisserver durch die oben genannten Nachrichten *bind*, *unbind* und *abandon* als eine Art Verbindungskontrolle benötigen, senkt die Bearbeitungsgeschwindigkeit der Anfragen etwas ab. Ein Namensdienstprotokoll für Mobile Agenten Systeme wird aber durchaus ähnlichen Gesichtspunkten genügen müssen. Deswegen werden einige dieser Ansätze auch für diese Arbeit in modifizierter Form Verwendung finden.

## 2.6 Spezielle Systeme für mobile Objekte

Die im letzten Abschnitt vorgestellten Systeme zur Namensgebung und Lokalisierung von Objekten beruhen meist auf der Voraussetzung von relativ statischen Strukturen innerhalb der betrachteten Netzwerke.

Mittlerweile liegt der Schwerpunkt vieler Forschungsgruppen aber immer mehr darauf, verteilte Systeme zu entwerfen, die auch den Umgang mit mobilen Objekte erlauben. Dem Benutzer soll ein System zur Verfügung gestellt werden, mit dem er möglichst einfach mobile Objekte kreieren und auf deren Basis weitverteilte Applikationen entwerfen kann.

Namens- und Kommunikationsdienste sollen zur Verfügung stehen und möglichst transparent benutzt werden können. Ein Sicherheitsmodell ist erwünscht. Unabhängigkeit von Hardware, Betriebssystem und Netzwerk wird gefordert. Der Benutzer möchte nicht an die Notwendigkeit von Lastverteilung, Ressourcenkontrolle oder Fehlertoleranz als Voraussetzungen für die Skalierungsfähigkeit des Systems denken.

Die Tatsache, dass dabei nicht mehr nur von einzelnen Rechnern oder Benutzern die Rede ist, sondern eine Vielzahl von Objekten pro Benutzer verwaltet werden muss, erhöht die Größenordnungen in denen diese Systeme noch skalieren müssen. Vor Allem aber die Forderung nach Mobilität stellt eine neue Voraussetzung dar, die beim Erfüllen der Anforderungen bedacht werden muss. Es wird teilweise nötig, bestehenden Algorithmen und Protokolle für verteilte Systeme zu überdenken bzw. komplett auszutauschen.

In diesem Abschnitt werden nun einige dieser Systeme vorgestellt und deren Struktur erläutert. Dabei wird speziell auf die Namensgebung und die Lokalisierung von Objekten eingegangen. Anschließend wird wiederum analysiert, in wie weit die verwendeten Mechanismen auch für den Entwurf eines sicheren Namensdienst für Mobile Agenten Systeme genutzt werden können.

Gerade der beim Globe-Projekt im folgenden Abschnitt vorgestellte Lokationsdienst wurde unter Berücksichtigungen vieler für die Skalierbarkeit verteilten Systeme wichtiger Aspekte

entworfen und erweitert. Aus diesem Grund wird dieses System sehr ausführlich beschrieben. Anschließend werden weitere Systeme, Algorithmen und Protokolle vorgestellt, bevor in Abschnitt 2.8 der Artikel zur Sprache kommt, der direkt zur bearbeiteten Aufgabenstellung dieser Arbeit geführt hat.

### 2.6.1 Das Globe-Projekt

Im Rahmen des *Globe-Projekts* [44] wird eine skalierbare Infrastruktur für weltweit verteilte Systeme bereitgestellt, die auf der Basis von mobilen Objekte den Entwurf von verteilten Applikationen ermöglichen soll. Prozesse kommunizieren und interagieren dabei durch gemeinsam benutzte mobile Objekte.

Der interne Zustand des mobilen Objekts kann durch Replikation und Partitionierung physikalisch auf mehrere Rechner verteilt werden. Art der Zustandsverteilung, Kommunikationsprotokolle, Replikationsstrategien und Migrationmechanismen werden vom Objekt selbst durch die jeweilige Implementierung vorgegeben, allerdings hinter Schnittstellen verborgen.

Die Infrastruktur des Globe-Projekts stellt einige Basisdienste zur Verfügung, die z. B. auch die Anbindung eines Prozesses an den ein Objektes ermöglichen, um anschließend dessen Funktionalität zu nutzen. Diese Anbindung geschieht über die Kontaktadresse des Objektes, die wiederum durch eine Netzwerkadresse und ein Protokoll eindeutig bestimmt ist.

Ein Schwerpunkt bei der Arbeit am Globe-Projekt war die Entwicklung eines weltweit skalierbaren Namens- und Lokationsdienstes für mobile Objekte [88], der in [89] mit Implementierungsvorschlägen noch detaillierter beschrieben wird.

#### Dienstemodell

Betrachtet man die im Internet sonst übliche Zuordnung einer URL zu einer Ressource (siehe Abschnitt 2.5.1), um mobile Objekte zu benennen, so lassen sich gleich zwei Probleme erkennen: Zum einen ist durch diese Namensgebung keine Replikation möglich. Zum andern wird die URL gleichzeitig zur Identifikation von Ressourcen und für den Zugriff auf diese genutzt. Diese beiden Vorgänge haben aber unterschiedliche Anforderungen.

Um eine bessere Skalierbarkeit beim Umgang mit mobilen Objekten zu erreichen, setzt Globe auf eine Namensgebung, die jeden Hinweis auf die Objektposition verbirgt. Dies wird durch eine zweistufige Namenshierarchie erreicht, indem im Gegensatz zu einem herkömmlichen Namensdienst (siehe Abschnitt 2.5), beim Globe-Projekt ein Name also nicht direkt zu einer Adresse aufgelöst wird. Ein Namensdienst übernimmt hier die Aufgabe, Objektnamen einem global eindeutigem, ortsunabhängigen *Objekt Identifier* zuzuordnen. Ein Lokationsdienst übernimmt dann im zweiten Schritt die Zuordnung dieses Objekt Identifier zu einer Kontaktadresse.

#### Namensdienst

Zur Identifikation eines Objektes wird in [26] ein für den Menschen einprägsamer, sogenannter *Human Friendly Name (HFN)* vorgeschlagen, der im Gegensatz zu den in DNS gebräuchli-

chen URLs ortsunabhängig ist. Als erste Lösung wird die Zuordnung von HFNs zu Objkt-Identifiern durch das DNS vorgeschlagen, indem ein HFN auf einen regulären Domänenbezeichner abgebildet und der *Objekt Identifier* in einem besonderen *resource record* (siehe Abschnitt 2.5.1) gespeichert wird.

In einem weiteren Vorschlag [25] wird für die Zukunft allerdings auch ein neuer Namensdienst für globale, verteilte Systeme vorgeschlagen, der durch Replikation von Verzeichnissen Lokalität ausnutzt und darüber hinaus einen größeren Namensraum unterstützt.

#### Lokalisierungsdienst

Die Aufgabe des Lokalisierungsdienstes bei Globe ist es nun, dem Objekt-Identifier eine Kontaktadresse zuzuordnen: Der in [88] und [89] vorgestellte skalierbare Lokationsdienst basiert auf einem hierarchischem Suchbaum. Das weltweite Netzwerk wird dafür in ineinander verschachtelte Regionen (ähnlich den Domänen bei DNS) eingeteilt, die jeweils mit einem Verzeichnisknoten assoziiert sind. Es existiert eine ausgezeichnete Wurzel, von der aus Pfade in die kleinsten Regionen an den Blättern des Baumes führen.

Die Kontaktadresse eines Objektes wird nun in einem vorerst beliebigen Verzeichnisknoten auf dem Weg von der Wurzel bis in die kleinste Region, die das Objekt beinhaltet, gespeichert, wobei dieser Pfad von der Wurzel bis zum eigentlichen Speicherort der Kontaktadresse mit Zeigern aufgebaut wird. Dabei kann es zu einem Objekt wegen der Replikation mehrere Kontaktadressen geben. Außerdem werden diese in jedem Fall ortabhängig gespeichert.

Das heißt aber auch, dass der Wurzelknoten mindestens einen Zeiger für jedes Objekt speichern muss, was ein sehr hohes Speicher- und Kommunikationsaufkommen mit sich zieht. Eine Lösung ist die Implementierung des Wurzelverzeichnisses durch mehrere physikalisch getrennte Verzeichnisse auf verschiedenen Rechnern, die gleichmäßig auf der Erde verteilt sind. In [23] wird beschrieben, wie sich diese Idee bei spezieller Einteilung der Erde in Regionen umsetzen lässt und dabei Lokalität ausgenutzt wird.

Bei lookup Anfragen wird diese Baumstruktur vom Anfrageort Richtung Wurzel verfolgt, bis entweder der erste Zeiger oder direkt die Kontaktadresse des gesuchten Objekts gefunden wird. Wird ein Zeiger gefunden, so kann der Pfad wieder in Richtung der Blätter bis zur eigentlichen Kontaktadresse verfolgt werden. Dies garantiert, dass die Kontaktadresse des physikalisch nächstgelegenen Objekt zurückgebeben wird. Im schlechtesten Fall ist ein Objekt immer durch das Verfolgen der Zeiger vom Wurzelknoten aus lokalisierbar.

Bei update Anfragen werden Kontaktadressen auf ähnliche Weise ergänzt oder gelöscht. Dies geschieht durch eine RPC-Aufrufkette, die vom Ursprungsort der *update* Anfrage ausgeht und anschließend auch wieder dahin zurückkehrt.

Um nach einem möglichen Ausfall eines Verzeichnisrechners die Konsistenz des Systems wiederherzustellen, wird in [24] ein einfacher Algorithmus vorgestellt der darauf basiert, beim Aufbau der RPC-Aufrufkette durch den Suchbaum Richtung Wurzelverzeichnis die Änderungen im Speicher durchzuführen und diese erst anschließend beim Rückweg permanent zu speichern. Wird ein Rechner in der RPC-Aufrufkette nach einem Ausfall erneut gestartet, so bittet er seine Kinder einfach, die ausstehenden Modifikationen erneut auszuführen.

Betrachtet man noch einmal den Verzeichnisknoten im Suchbaum, in dem die eigentliche Kontaktadresse eines Objektes gespeichert wird: Je näher dieser am Wurzelknoten liegt, desto größer ist die Wahrscheinlichkeit dafür, dass das Objekt nach einer Migration immer noch in der durch den Verzeichnisknoten „abgedeckten" Region liegt und sich damit der Speicherort der Kontaktadresse nicht verändern muss. Dafür ist die RPC-Aufrufkette bei Anfragen in der Regel länger. In [2] wird nun beschrieben, wie man unter Beobachtung der Migrationsmuster von Objekten die optimale Speicherposition im Suchbaum ausfindig macht. Dann kann durch effektives *caching* eine weitere Leistungssteigerung hervorgerufen werden.

#### Analyse

Das Globe-Projekt stellt ein gut durchdachtes Konzept eines Systems für miteinander kommunizierende, mobile Objekte dar, was die Skalierbarkeit betrifft. Durch die Bereitstellung eines Namensdienstes *und* eines Lokationsdienstes konnten diese Dienste getrennt voneinander auf die jeweiligen Anforderungen hin optimiert werden. Diese 2-teilung der Dienste basiert auch auf der Einführung eines eindeutigen Objekt-Identifiers für die Objektlokalisierung, der keine Ortsabhängigkeiten besitzt. Der Namensdienst ermöglicht nun darüber hinaus eine benutzerfreundliche Bezeichnung für mobile Objekte. Diese Ansätze lassen sich auch auf die Lokalisierung von Agenten in einem Mobile Agenten System übertragen und spielen deswegen bei der Modellentwicklung in Kapitel 3 eine nicht unwesentliche Rolle.

Der eigentliche Lokationsdienst ist allerdings unter der Annahme entworfen und optimiert worden, dass sich zum einen bei der Bewegung mobiler Objekte im Netzwerk Migrationsmuster erkennen lassen und zum anderen durch Replikation von Objekten Lokalität ausnutzen und dadurch der Kommunikationsaufwand minimieren lässt. Diese Annahmen treffen in der Regel aber beide nicht auf mobile Agenten zu. Des weiteren wurde die Sicherheit des Lokationsdienstes gegenüber Angriffen aus dem Netzwerk komplett außer Acht gelassen.

### 2.6.2 Normadic Pict

*Nomadic Pict* [95] basiert auf dem sogenannten *Nomandic $\pi$-calculus* [94] und stellt eine Agenten-Programmiersprache dar, die unter anderem die Kommunikation zwischen migrierenden Agenten ermöglicht.

Die Sprache teilt sich in zwei Ebenen auf: Die *low level* Ebene umfasst Funktionen für das Kreieren von Agenten und deren Migration zwischen *sites* bzw. die ortsabhängige Kommunikation mit und zwischen Agenten über *channels*. Auf der *high level* Ebene wird eine ortsunabhängige Kommunikation ermöglicht.

Dabei stellen die *sites* jeweils Instanzen des Normadic Pict Laufzeitsystems dar. Agenten besitzen einen eindeutigen Namen und sind jederzeit einer bestimmten *site* zugeordnet. In der Form von ausführbaren Code bestehen sie aus Normadic Pict Prozessen.

Programme in Nomadic Pict werden kompiliert, indem die high-level Anteile vorerst in low-level Programme übersetzt und dann vom Runtime-System ausgeführt werden.

### Lokalisierungs-Algorithmus

Diese Infrastruktur basiert vor Allem auf folgenden zwei Algorithmen: Beim Central Server Algorithmus existiert ein einziger Sever, der die aktuelle *site* jedes Agenten registriert. Die Agenten synchronisieren sich mit diesem vor und nach Migrationen. Ortsunabhängige Nachrichten können dann über diesen Server verschickt werden. Der Forwarding Pointer Algorithmus besitzt einen *daemon* auf jeder *site*. Wenn ein Agent von einer *site* weg migriert, so hinterlässt er einen Zeiger auf die *site*, zu der er migriert. In diesem Fall werden Nachrichten über die *daemons* weitergeleitet.

Im ersten Fall ist der *central server* Kommunikationsengpass für alle Nachrichten zwischen *sites* und jede Nachricht braucht genau zwei *hops*. Beim zweiten Algorithmus wird der genannte Engpass zwar behoben, allerdings brauchen die Nachrichten in der Regel mehrere *hops*.

Der Query Server Algorithmus sieht als Optimierung der benötigten *hop* Anzahl für Nachrichten auf jeder *site* einen *daemon* vor, der vorerst die aktuelle Position des Zielagenten über den *central server* ermittelt, und die Nachricht dann direkt versendet. Falls der Agent in dieser Zeit weiter migriert ist, wird die Nachricht wieder zurückgeschickt, um einen erneuten Versuch zu starten. Dadurch braucht die eigentliche Nachricht zwar nur einen *hop*, allerdings ist das Aufkommen von Kontrollnachrichten um einiges höher.

Der Query Server with Caching versucht diese Problem zu beheben: Falls ein *daemon* eine Nachricht an einen Agenten empfängt, der sich nicht mehr auf dieser *site* befindet, so wird diese Nachricht an den Agenten an den *central server* geschickt und über diesen weitergeleitet. Darüber hinaus wird eine Nachricht zum *cache update* an den Ausgangsdaemon geschickt.

Als weitere Verbesserung wird vorgeschlagen, lange Zeigerketten beim *Forwarding Pointer Algorithmus* zum Beispiel zu kürzen oder einen serverlosen Algorithmus vorzusehen, bei dem sich zusammenarbeitende Agenten für den Nachrichtenaustausch selber synchronisieren.

### Analyse

Die Kommunikation zwischen Agenten ist bei Nomadic Pict in der Form von ortsabhängigen bzw. ortsunabhängigen Befehlskonstrukten in die Agenten-Programmiersprache eingebunden, wobei die Agenten in einem flachen Namensraum mit eindeutigen Bezeichnern identifiziert werden. Für die Lokalisierung und Kommunikation bei der ortsunabhängigen Variante, finden in verknüpfter Form implizit die beiden Konzepte *buffered response* und *forward references* (siehe Abschnitt 2.3.3) Verwendung.

Das Hauptaugenmerk liegt hier klar auf der Nachrichtenzustellung an Agenten und nicht auf Sicherheitskriterien bei der Agentenlokalisierung bzw. Fehlertoleranz gegenüber Ausfällen von einzelnen Rechnern im System. Da das Modell des *buffered response* in diesem Fall nur einen Namensserver vorsieht, wird es auch nur in kleinen Netzwerken zu verwenden sein können und im Internet nicht skalieren.

### 2.6.3 Das Aleph Toolkit

Das Aleph Toolkits [31] ist ein in Java implementiertes System für gemeinsam genutzte, verteilte Objekte.

Ein mobiles Objekt kann eine Datei, ein Prozess oder eine Datenstruktur sein. Kopien dieser Objekte können im Aleph Toolkit von sogenannten *Processing Elements (PE)*, jedes für sich eine *Java Virtual Machine*, bei Bedarf im *cache* gehalten werden. Ein *directory manager* muss nun die Position und den Status der Objektkopien aufzeichnen und sie gegebenenfalls invalidieren bzw. von PE zu PE verschieben.

Die Schnittstelle zu einem *communication manager* ermöglicht eine Punkt-zu-Punkt Kommunikation über Nachrichten. Darüber hinaus implementiert das System einen *event manager* und einen *transaction manager*.

#### Lokalisierungsdienst

Um mobile Objekte zu lokalisieren besitzt dieses System zwei verschiedene verteilte Verzeichnisprotokolle: Das eine ist ein *home based protocol*, in dem ein fester Knoten im Netz die Position und den Status der verteilten Objekte verfolgt und registriert. Das zweite (*Arrow Distributed Directory Protocol*) basiert auf einem simplen Pfadumkehr-Algorithmus, und schlägt das erstere in der Geschwindigkeit teilweise um Längen [93].

Im Home Directory Protocol trägt das, dem globalen Objekt fest zugeordnete *home* PE, dafür die Verantwortung für *lookup* und *update* Anfragen. Wenn die Anzahl der PE steigt oder ein Objekt häufig verwendet wird, so wird die zugeordnete PE schnell zum Synchronisations-Flaschenhals. Ist das Objekt im Netzwerk weit von seiner *home* PE entfernt, so steigt der Kommunikationsaufwand stark an.

Das Arrow Distributed Directory Protocol geht von einem aufspannenden Baum aller PE aus. Jede PE speichert pro Objekt einen Verzeichniseintrag in Form eines Zeigers, der entweder auf die eigene PE oder auf einen Nachbarn im Baum zeigen kann. Verweist der einem Objekt zugeordnete Zeiger auf die eigene PE, so befindet das Objekt entweder auf dieser PE oder wird bald dort zu finden sein. Verweist dieser Zeiger auf eine Nachbar-PE, so bedeutet das, dass sich das Objekt in dessen Teilbaum befindet. Durch diesen Verzeichniseintrag weiß die PE also nur in welcher Richtung sich das registrierte Objekt befindet.

Bewegungen von Objekten innerhalb des PE-Baumes können nun durch einen einfachen Pfad-Algorithmus verfolgt werden [10]. In diesem Algorithmus geschieht die Synchronisation und Navigation in einem Schritt.

Den Autoren nach ist noch zu zeigen, ob eine geeignete Kombination der beiden genannten Protokolle zu weiteren Verbesserung in der Bearbeitungsgeschwindigkeit von Anfragen führt.

#### Analyse

Für die Lokalisierung von Objekten im Aleph-Toolkit wurden unabhängig voneinander das Konzept *direct response* und als Variante des *forward references* das *Arrow Distributed Directory Protocol* implementiert. Für die Lokalisierung eines Objektes mittels *direct response*

muss zu jedem Objekt die Adresse der Ursprungs-PE bekannt sein und die Ursprungs-PE darüber hinaus permanent verfügbar sein. Vor allem der zweite Punkt ist zu restriktiv. Die Voraussetzungen für den zweiten Lokalisierungsdienst sind noch unrealistischer. In diesem Fall muss nämlich die gesamte Netzwerkstruktur bekannt und statisch sein, und sich in einen aufspannenden Baum aus PE umwandeln lassen. Jede PE im Netzwerk müsste für jedes Objekt einen Verzeichniseintrag speichern. Dieses Modell lässt sich nur in recht kleinen Netzwerken umsetzen. Außerdem wurden auch hier weder Sicherheitsaspekte noch Aspekte der Fehlertoleranz in Betracht gezogen.

### 2.6.4 CORBA

Die *Common Object Request Broker Architecture (CORBA)* [16] [60] stellt eine herstellerunabhängige Architektur und Infrastruktur dar, die es Applikationen erlaubt über Netzwerke hinweg zusammen zu arbeiten. Durch die Nutzung des *Internet Inter-ORB Protocol (IIOP)* ist es einem CORBA-basierten Programm fast unabhängig von Hersteller, Computer, Betriebssystem, Programmiersprache oder Netzwerk möglich mit einem anderen CORBA-basierten Programm zu kommunizieren. Dies wird durch sehr strikte Schnittstellendefinitionen für jedes beteiligte Objekt in der *Interface Definition Language* der *Objekt Management Group (OMG IDL)* möglich, wobei die eigentliche Implementierung der Objekte vor dem System verborgen bleibt.

#### Struktur

Neben dem *objekt request broker* als Kern von CORBA, der den Objekten ermöglicht Anfragen zu stellen und Antworten zu empfangen, werden durch *object services* eine Sammlung von Schnittstellen und Objekten bereitgestellt, die Basisfunktionalität wie z. B. das Kreieren, Löschen, Kopieren und Verschieben von Objekten erlaubt [59]. Darunter befindet sich auch ein Namensdienst.

In CORBA geschieht die Zuordnung eines Namens zu einem Objekt immer relativ zu einem sogenannten *Namenskontext*. Der Namenskontext ist ebenfalls ein Objekt, das diese Zuordnungen enthält, und in dem der Name eines Objektes eindeutig sein muss. Ein Name besteht aus einer bzw. mehreren Komponenten, wobei die letzte Komponente das eigentliche Objekt identifiziert und die vorangegangenen einen Namenskontext.

Die eigentliche Implementierung des Namensdienstes kann applikationsabhängig sein oder auf bestehenden Namensdiensten aufbauen. In einem verteilten System sind Graphen von Namenskontexten vorstellbar.

#### Analyse

Ein Namensdienst für die Bezeichnung von Objekten innerhalb von CORBA wird zwar unterstütz und erlaubt auch eine Hierarchiebildung durch Namenskontexte, bei der vorgeschriebenen Namensgebung wird allerdings die Position des Objektes bereits in dem Bezeichner kodiert, der zur Kontaktaufnahme bekannt sein muss. Unter der Anforderung, dass sich Objekte „frei" bewegen können sollen, skaliert dieser Ansatz nicht mehr.

### 2.6.5 DCOM

Das *Distributed Component Object Model (DCOM)* von Mircosoft [48] ist eine Erweiterung von COM. Es erlaubt über die DCOM Laufzeitumgebung die Kommunikation zwischen Clients und lokalen oder auf einem entfernten Server befindlichen Objekten. Diese Objekte müssen dafür bestimmte Schnittstellen implementieren.

Im Gegensatz zu COM, in dem diese Kommunikation durch *Local Procedure Calls (LPC)* geschieht, wird bei DCOM das *Object-RPC (ORPC)* Protokoll benutzt, das *Remote Procedure Calls (RPC)* kapselt und über das Netzwerk sendet. Dadurch wird auch eine gewisse Plattform- und Lokations-Unabhängigkeit gewährleistet.

Der Zugriff auf Objekte wird durch Auhtentifikation und Autorisation gesteuert, indem starker Gebrauch der von Windows NT angebotenen Sicherheitsstrukturen, wie z. B. den *Access Control Lists*, gemacht wird.

#### Struktur

Um die Verbindung mit einem Objekt aufzubauen wird lokal der Dienste-Kontrollmanager von DCOM aufgerufen [49]. Dieser lokalisiert das Objekt gegebenenfalls auf einem entfernten Server, initiiert dessen Kreierung, falls noch keine Instanz des Objektes existiert, und liefert dann einen Zeiger auf eine Schnittstelle zurück, die Zugriff auf die Objektfunktionalität gewährt.

Dazu werden Objekte im COM durch *Global Unique Identifier (GUID)*, die sogenannten *Class IDs (CLSID)*, identifiziert. Befindet sich bei DCOM das Objekt auf einen entfernten Server, so muss zusätzlich der Servername bekannt sein. Dieser kann dafür explizit als Parameter angegeben werden oder wird aus der lokalen Registrierungsdatei ausgelesen.

#### Analyse

DCOM unterstützt keinen Lokationsdienst im eigentlichen Sinne. Zur Identifizierung eines Objektes muss dessen CLSID und die Adresse des Servers bekannt sein, auf dem es sich befindet. DCOM versteckt in verteilten Systemen lediglich den zusätzlichen Kommunikationsaufwand hinter der Definition von RPC-Schnittstellen vor dem Benutzer.

### 2.6.6 Ein verteilter Verzeichnisdienst für mobile Agenten

In [52] wird ein uniformer Algorithmus für einen verteilten Verzeichnisdienst für statische und mobilen Agenten entwickelt. Der Algorithmus basiert auf den Annahmen, dass keine Kompatibilität mit einer bestehenden Infrastruktur gewährleistet werden muss und keine zentralisierte Kontrolle erwünscht ist. Wegen der geplanten Nutzung in Anwendungsebene wurde viel Wert auf Zuverlässigkeit gelegt. Nach der formalen Definition des Algorithmus und der benötigten Datenstrukturen folgt in dem genannten Artikel aus diesem Grund auch ein Korrektheitsbeweis.

Ausgegangen wird von einem generischen Agentensystem, bei dem Agenten von Agentenserver zu Agentenserver migrieren und diese untereinander Nachrichten austauschen können. Die Agentenmigration wird dabei nicht atomar wahrgenommen.

#### Struktur

Der verteilte Verzeichnisdienst sieht nun auf jedem Agentenserver eine Tabelle vor, in der die Positionen von Agenten gespeichert werden können. Migriert ein Agent von einem Ausgangs- zu einem Zielserver, so muss der Agent anschließend den Ausgangsserver durch eine Bestätigungs-Nachricht (*ack*) über seine neue Position informieren.

Falls ein Agent einen Server besucht, auf dem er bereits war und diesen anschließend wieder verlässt kann es zu einer „Wettlauf" zwischen zwei Bestätigungs-Nachrichten kommen, falls die erste Nachricht zu diesem Zeitpunkt noch nicht angekommen ist. Erreichen die beiden Nachrichten diesen Server in falscher Reihenfolge entsteht ein unerwünschter Zeiger-Zyklus. Um das zu vermeiden wird ein Mobilitätszähler für jeden Agent eingeführt, der bei jeder Migration erhöht wird und mit den Bestätigungs-Nachrichten mitgeschickt wird.

Um einen Agenten nun zu lokalisieren wird vorausgesetzt, dass der Kreierungsort des Agenten bekannt ist, um von dort aus seinen Weg über die Zeigerketten verfolgen zu können. In einer zentralisierten Lösung müsste ein spezieller Verzeichnisserver für jeden Agenten dessen Kreierungsort speichern. In ein verteilten Lösung würde der Kreierungsort im Namen des Agenten kodiert.

Als Optimierung werden Informations-Nachrichten (*inform*) eingeführt, mit ein Agentenserver anderen Agentenservern spontan über die Position eines Agenten informieren kann. Dadurch können die Zeigerketten gekürzt werden. In kleinen verteilten Systemen wäre das *broadcasting* dieser Nachrichten eine realistische Lösung. Außerdem wäre vorstellbar, dass der Agent bei jeder Migration zusätzlich zum letzten Server auch $n$ vorherige Server auf seinem Weg über seine aktuelle Position informiert.

Damit dieser Algorithmus in großen Netzwerken skalieren kann, werden unbenötigte Einträge in den Positionstabellen auf jedem Agentenserver von Zeit zu Zeit gelöscht. Dies könnte z. B. durch einen Zeitstempel geschehen, der in regelmäßigen Abständen überprüft wird. Außerdem sollten die Transportprotokolle einen zuverlässigen Nachrichtenaustausch ohne Nachrichtenverlust garantieren.

#### Analyse

Der hier beschriebene Lokationsdienst für mobile Agenten entspricht dem in Abschnitt 2.3.3 dargestellten Konzept der *forward references*, wobei die Ketten aus Vorwärtsreferenzen gegebenenfalls abgekürzt werden können. Nachteilig ist die Tatsache, dass zur Lokalisierung von Objekten deren Ausgangsserver bekannt und darüber hinaus erreichbar sein muss. Die Kommunikationslast und der Speicherbedarf wird zwar gleichmäßig verteilt, dafür müssen an einen Agenten adressierte Nachrichten abgesehen von Abkürzungen den gleichen Weg nehmen und ihn dann auch einholen. Gerade im Internet und damit der möglichen Überbrückung großer

Distanzen von Agentenserver zu Agentenserver besteht dann bei jedem Abschnitt des Pfades zum Agenten die Gefahr von Verzögerungen bzw. Rechnerausfällen an Knotenpunkten erneut.

## 2.7 Weitere Algorithmen für mobile Objekte

In diesem Abschnitt werden nur kurz ein paar weitere Algorithmen vorgestellt, die für die Lokalisierung von und die Kommunikation mit mobilen Objekten entworfen wurden:

**(1)** In [3] wird ein benutzerfreundliches (human readable) Namensschema vorgeschlagen. Es basiert auf der *Mobile Agent System Interoperability Facilities Specification (MASIF)* [15], dem Standardisierungsvorschlag eines Mobile Agenten Systems der *Object Management Group (OMG)*. Diese Spezifikation beschreibt ein Mobile Agenten System, das sich in mehrere Regionen (*regions*) aufteilt, die wiederum mehrere Agentenserver (*agent systems*) beinhalten, auf denen in bestimmten Kontexten (*places*) Agenten ausgeführt werden können. Folgende Konvention bei der Namensgebung verspricht Transparenz, Unabhängigkeit von der Agentenposition und eine relativ freie Namenswahl für den Agentenbesitzer: `AgentName = agent:<LocalName>@<RegionName>`. Dabei muss der frei wählbare `LocalName` innerhalb der Ursprungsregion `RegionName`, in welcher der Agent erstmals gestartet wurde, eindeutig sein. Um Agenten in dem beschriebenen System zu lokalisieren, ist pro Region ein Namensserver vorgesehen. Ausgehend von der Ursprungsregion speichern diese jeweils einen Verweis auf den Namensserver der nächsten Region, in welche der Agent mirgriert ist. Außerdem wird bei der Migration eines Agenten versucht, auf dem Namensserver seiner Ursprungsregion einen zusätzlichen, direkten Verweis auf die aktuelle Region zu aktualisieren. Innerhalb der aktuellen Region wird der Agent durch *broadcast* Nachrichten an alle enthaltenen Agentenserver lokalisiert. *Caching* bereits angefragter Agentenpositionen verkürzt darüber hinaus den zu verfolgenden Pfad bei *lookup* Anfragen. Als weitere Optimierungsmöglichkeit wird die Nutzung von DNS (siehe Abschnitt 2.5.1) für die Lokalisierung der Ursprungsregion angesprochen, welche durch den Emailadressen-ähnlichen Aufbau des Agentennamens nahe liegt. Wurde ein Agent im Zuge eines Kommunikationsversuchs lokalisiert, so wird dessen Migration verhindert, bis die erwartete Nachricht an ihn ausgeliefert wurde. Ein *timeout* soll dabei das Warten auf eine nicht stattfindende Interaktion abwehren. Dieses *Broadcast-Search-and-Path-Chase* genannte Protokoll beinhaltet wiederum das Konzept der *forward references* (siehe Abschnitt 2.3.3) und birgt deswegen auch die bereits in der Analyse von Abschnitt 2.6.3 und 2.6.6 angesprochenen Nachteile. Die vom Autor des Artikels angesprochene Möglichkeit der Integration von DNS in den Lokationsprozess ist aber durchaus interessant.

**(2)** Ein verlässliches Protokoll zur Nachrichtenauslieferung, Agentenmigration und Verfolgung von Agentenpositionen wird in [80] dargestellt. Voraussetzung hierfür ist eine balancierte Baumstruktur von sogenannten Gateway-Servern, die jeweils für eine Gruppe von Agentenservern mit eindeutigen Adressen zuständig sind, und dadurch eine Aufteilung der Agentenserver in Domänen hervor rufen. Alle Übermittlungen, ob inter-gruppen oder intra-gruppen Transfers von Nachrichten bzw. Agenten, müssen über diese Gateways laufen. Unter dieser Annahme wird nun bei jeder der folgenden Operationen eine Beschränkung der benötigten

*hops* auf $O(log\ n)$ bezüglich der Anzahl der Gateway-Server zugesichert: Agentenkreierung, Agententerminierung, Agentenmigration bzw. Nachrichtenauslieferung. Jedes Gateway speichert einen Eintrag für jeden Agent, der über ihn versendet wird. Die dynamische Erzeugung der Baumstruktur geschieht dadurch, dass ein neuer Gateway-Server als Vaterknoten mehrere bereits operierende Gateways als Kinder unter sich vereint. Bis auf einen *root* Knoten einer zusammenhängenden Baumstruktur kennen nach diesem als *joining* bezeichnetem Vorgang zum einen alle Gateways ihren Vater, zum anderen übernimmt ein Vater mit seinen Kindern auch deren kompletten Satz an Einträgen. Der Nachteil an diesem Verfahren ist allerdings der, dass der *root* Knoten einer zusammenhängenden Baumstruktur die Einträge zu sämtlichen vorhanden Agenten speichern muss und dadurch die Skalierbarkeit eingeschränkt wird. Außerdem ist die obere Grenze $O(log\ n)$ für die Anzahl der *hops* bei den oben genannten Operationen nicht unbedingt ein erwünschtes Optimum.

**(3)** In [53] wird ein Mechanismus vorgestellt, durch den die zuverlässige Kommunikation mit hochmobilen Agenten gewährleistet werden soll. Eine Kombination aus den beiden aus Abschnitt 2.3.3 bekannten Konzepten *forwarding* und *broadcasting* soll die Zustellung von Nachrichten an Agenten nach dem Prinzip *at-least-once* garantieren. Das Netzwerk wird mit der zuzustellenden Nachricht an einen Agenten „geflutet". Allerdings basiert diese Idee auf der Annahme, dass sich der zu kontaktierende Agent in einem dem Algorithmus „bekannten" und fixen Bereich des Netzwerks befindet. Wenn der Agent diesen Bereich verlässt, muss sein neuer Aufenthaltsort in diesen bekannten Bereich mit aufgenommen werden. Für das Internet sind diese Annahmen unrealistisch. Außerdem kann dieser Algorithmus wegen dem hohen Kommunikationsaufkommen und dem Speicherbedarf für zu puffernde Nachrichten für große Anzahlen von Agenten und Rechnern im Netzwerk nicht skalieren.

**(4)** Ketten von sogenannten *Stup-Scion* Paaren (SSP Chains) [83] stellen eine robuste Möglichkeit dar, Objekte über Vorwärtsreferenzen zu lokalisieren und auf dem gleichem Weg direkt auf ihre Schnittstellen zuzugreifen. Als Seiteneffekt von Objektzugriffen können die Referenzketten gegebenenfalls abgekürzt werden. Außerdem unterstützt diese Modell azyklisches *Garbage Collection* nicht mehr benötigter SSPs.

**(5)** In [7] wird das Prinzip des Location Independent Invokation (LII) beschrieben, dass durch eine Kombination von Vorwärtsreferenzen und globalen Namensdienst ermöglicht mobile Objekte zu lokalisieren. Auf den globalen Namensdienst wird genau dann zurückgegriffen, wenn das *lookup* zu *update* Verhältnis zu groß wird. Die Last wird in diesem Modell automatisch gleichmäßig verteilt.

**(6)** Wandern kommunizierende Objekte in einem verteilten System von Host zu Host, so wird die Lokalisierung oft über einen sogenannten *Diffusionsbaum* gewährleistet, an dessen Wurzel der *home host* steht. Das heißt jeder Host auf dem Migrationspfad eines Objektes speichert eine Referenz auf den nachfolgenden Host. *Zombies* sind in diesem Kontext Hosts, die nicht mehr auf ein gewisses Objekt zugreifen müssen, sondern nur noch dafür da sind den Diffusionsbaum für dieses Objekt aufrecht zu erhalten. In [62] nun wird formal ein Algorith-

mus (Graceful Disconnection) beschrieben, mit dem sich diese Zombies aus der Referenzkette entfernen lassen, ohne die Verbindung von home-Host zum Objekt zu unterbrechen.

**(7)** In [61] wird der Mobility Layer für IP definiert. Unter Wahrung der Abwärtskompatibilität soll hierdurch ermöglicht werden mobile Hosts über nur eine URL und damit eine IP-Adresse anzusprechen. Das wird dadurch erreicht, dass jeder mobile Host einen sogenannten *home agent* besitzt, der als Proxy alle an den mobilen Host gerichteten Nachrichten getunnelt weiterleitet. Eine Diskussion der Anwendbarkeit findet sich in [84].

#### Zusammenfassende Analyse für die Algorithmen (4)-(7)

SSP Ketten, LLI und dem angesprochenen Algorithmus des Graceful Disconnection gemein ist, dass ein mobiles Objekt durch ein Kette von Vorwärtsreferenzen lokalisiert wird. In allen Fällen wird zwar versucht diese Kette auf verschiedene Weisen möglichst kurz zu halten, um das mobile Objekt in wenigen Schritten lokalisieren bzw. erreichen zu können, doch beginnt diese Referenzkette wie auch beim in Abschnitt 2.6.6 beschriebenen Namensdienst für mobile Agentensysteme immer beim Besitzer des Objekts. Es muss also garantiert werden, dass der Ursprungsrechner des Objektes permanent verfügbar ist, um das Objekt zu lokalisieren, und dessen Adresse darüber hinaus auch noch veröffentlicht werden, um dritten den Zugriff auf das Objekt zu ermöglichen. Dieser Aspekt muss übrigens auch beim Mobility Layer für IP beachtet werden. Für ein Mobiles Agenten System sollte das nun aber keine Voraussetzung sein.

## 2.8 Ein skalierbarer und sicherer globaler Namensdienst für mobile Agenten

In [75] entwickelt Volker Roth eine erste Idee für einen globalen Namensdienst für mobile Agenten, der auch im Internet noch skalieren soll und gleichzeitig Sicherheit gegenüber Angriffen aus dem Netzwerk bietet. Er geht speziell auf die Anforderungen ein, die sich durch ein Mobiles Agenten System ergeben:

Vor Allem die hohe Migrationsrate der Agenten verlangt nach einer Lösung, die leistungsstark genug ist, um nach dem Registrieren einer Agentenposition darauf folgende Anfragen schnell genug beantworten zu können, bevor der Agent seine Position erneut verändert hat.

Nach der Diskussion von vier generischen Modellen zur Positionsverfolgung mobiler Objekte (siehe auch Abschnitt 2.3.3) fällt die Wahl auf das Prinzip des buffered response. Darauf aufbauend wird dann ein Protokoll als Schnittstelle zwischen Agentenserver und Namensserver entwickelt, dass *update* Anfragen in einer Art kapselt, die möglichst viele Angriffsszenarien (sowohl Angriffe auf die *privacy* als auch DoS Attacken) verhindert.

#### Struktur

Die Kommunikationslast und das Speicheraufkommen des Namensdienstes soll auf mehrere Namensserver verteilt werden können, wobei dessen Anzahl skalierbar bleiben soll.

Die Zuordnung zwischen Agent und dem für die Lokalisierung verantwortlichem Namensserver geschieht dabei im Gegensatz zu DNS (Abschnitt 2.5.1) oder Globe (Abschnitt 2.6.1) nicht abhängig von dem aktuellen Aufenthaltsort des Agenten und damit dynamisch, sondern fix bei Kreierung des Agenten.

Das Protokoll für *update* Anfragen an den Namensserver wird in fünf Schritten entwickelt. In all diesen Szenarien ist es die Aufgabe des Agentenservers, die benötigten Daten an den Namensserver zu übermitteln:

- Das trivialste Protokoll sieht bei jeder Migration des Agenten die Klartextübermittlung der aktuellen Agentenposition an einen, bei dessen Kreierung vom Besitzer festgelegten, Namensserver vor. Der Besitzer des Agenten muss in diesem Fall die Zuordnung Agent $\rightarrow$ Namensserver ebenfalls im Klartext veröffentlichen bzw. dem Agenten als Information mit auf den Weg geben.
- Im zweiten Schritt wird durch eine Signatur dieser Daten gewährleistet, dass die Entscheidung des Besitzers nicht unbemerkt verändert werden kann.
- Als weitere Sicherheitsmaßnahme wird mit sogenannten *Cookies* (in diesem Fall Zufallszahlen) gearbeitet, die dem Agenten wie auch dem Namensserver bei der initialen Positionsregistrierung als Information mitgegeben werden. Bei folgenden *update* Anfragen wird kontrolliert, ob der Sender diese autorisierende Information kennt. Gleichzeit wird das alte Cookie durch ein neues ersetzt.
- Eine alternative Zuordnung zwischen Agent und Namensserver wird durch die Einführung des Hashcodes eines Agenten festgelegt. Hierbei wird durch eine kryptographische Hashfunktion der Hashcode des statischen Teil eines Agenten gebildet, die diesen, mit vernachlässigbar kleinen Abstrichen in Hinblick auf die Kollisionswahrscheinlichkeit der verwendeten Hashfunktionen, eindeutig bezeichnet. Die ersten Bits dieses Codes spezifizieren nun den Namensserver, der für den Agenten zuständig ist. Je nach Anzahl der zur Auswahl des Namensservers betrachteten Bits kann die Anzahl der Namensserver reguliert werden.
- Im letzten Schritt werden durch Signatur des statischen Teils des Agenten und Verwendung des Cookies bei *update* Anfragen an den Namensserver wiederum die vorher angesprochenen Sicherheitsmaßnahmen eingeführt.

**Analyse**

Stark an diesen Artikel von Volker Roth angelehnt und von den gleichen Anforderungen ausgehend, wurde dann auch die Aufgabenstellung zu dieser Arbeit formuliert (siehe Abschnitt 1.2). Aus diesem Grund werden einige der in diesem Abschnitt erwähnten Ansätze bei der Modellentwicklung des Namensdienstes für Mobile Agenten Systeme in Kapitel 3 aufgegriffen und gegebenenfalls modifiziert oder verbessert.

# Kapitel 3

# Modellentwicklung

Will man einen neuen Informationsdienst für ein spezielles System entwerfen, so sollte man den Kontext nicht außer Acht lassen, in dem dieser Dienst zur Anwendung kommt. Die folgenden Fragen (angelehnt an [54]) ziehen sich als roter Faden durch die Modellentwicklung eines Informationsdienstes und sind ausschlaggebend für die interne Struktur des Dienstes sowie dessen Schnittstellen nach außen:

- Wer benutzt den Dienst ?
- Welche Daten sollen gespeichert werden bzw. anzufragen sein ?
- Auf welchen Diensten bzw. Datenstrukturen kann dieser Dienst aufsetzen ?
- Welche Dienste bzw. Softwareschichten sollen auf dem Dienst aufbauen ?
- Wie wird sich die Anfragefrequenz bei wachsendem Gesamtsystem verhalten ?
- Welcher Anteil dieser Anfragen muss von einem zentralen Server bearbeitet werden ?
- Ist Replikationen ein geeignetes Mittel, um die Sicherheit zu erhöhen ?
- Kann durch Replikationen Lokalität ausgenutzt werden ?
- Kann die Anzahl der beteiligten Server nachträglich verändert werden.
- Welchen Komponenten stellen Flaschenhälse bei der Kommunikation dar ?
- Wovon hängt die Datenmenge ab, die von einzelnen Servern verwaltet werden muss ?
- Wie oft verändern sich gespeicherte Informationen ?
- Wovon hängt die Bearbeitungszeit für Anfragen ab ?
- Welcher *update* Mechanismus wird genutzt ?
- Ist es sinnvoll, Anfrageergebnisse zu zwischenzuspeichern (*caching*) ?
- Ist es nötig, Informationen in bestimmten Zeitintervallen zu invalidieren ?
- Wie groß sind diese Zeitintervalle ?
- Ist eine einzige Autorität für die Administration nötig ?

Mit diesen Fragen im Hinterkopf und unter Beachtung der bereits im letzten Kapitel formulierten Anforderungen wird in diesem Kapitel nun das Modell eines sicheren Namensdienstes für Mobile Agenten Systeme entwickelt. Nach der Einordnung in den entsprechenden Kontext und einer differenzierteren Betrachtung des Namensdienstes an sich, wird die eigentliche Lokalisierung von mobilen Agenten beschrieben. Dazu werden die benötigten Komponenten aufgeführt und deren Schnittstellen zueinander dargestellt. Nach der Identifizierung möglicher Engpässe bzw. Sicherheitslücken, wird ein sicheres Protokoll für *lookup* bzw. *update* Anfragen entwickelt. Anschließend werden einige Angriffsszenarien erörtert und diskutiert, in wieweit das Gesamtkonzept in Punkto Sicherheit dagegen standhält.

Das zu entwickelnde Modell sollte so generisch wie möglich bleiben, um eine spätere Nutzbarkeit für verschiedene Anwendungsszenarien zu gewährleisten. Der Entwurf geschieht deswegen vorerst noch relativ abstrakt und unabhängig von der Art des Agentensystems. Die Aspekte und Anpassungen, die bei der konkreten Implementierung eines Funktionsmodells und der Einbindung in die SeMoA-Plattform zu beachten sind bzw. die detaillierte Objektstruktur des Prototyps, werden anschließend separat in Kapitel 4 erörtert.

## 3.1 Ein 3-geteiltes Dienstemodell

Der zu entwickelnde Namensdienst soll im Sinne der Aufgabenstellung ermöglichen, mobile Agenten zu lokalisieren, die häufig von Agentenserver zu Agentenserver migrieren. Der tatsächliche Pfad, den die Agenten dabei durch das Netzwerk beschreiten wird hauptsächlich durch deren Aufgabe bestimmt und hängt damit von den initialen Vorgaben des Besitzers und darüber hinaus von dynamischen Entscheidungen auf Grund von lokal gewonnenen Informationen und der unterliegenden Netzinfrastruktur ab. In der Regel sind keine besonderen Migrationsmustern zu beobachten. Aus diesem Grund erscheint es sinnvoll, die Agenten durch einen eindeutigen Bezeichner zu identifizieren, der im Gegensatz zu den meisten in Abschnitt 2.5 betrachteten hierarchischen Namensräumen keine ortsabhängigen Informationen enthält.

Die Aufgabe eines Agenten, wird ausschließlich durch seinen Programmcode und die angesprochenen initialen Parameter des Besitzers bestimmt, wobei diese Daten während des gesamten Lebenszyklus des Agenten statisch sind. Wenn man die Eindeutigkeit eines Agentenbezeichners nun auf die Aufgabenstellung des bezeichneten Agenten zurückführt, liegt es nahe, die genannten statischen Daten zur Identifikation desselben zu nutzen. Es ist allerdings unrealistisch, bei jeder *lookup* Anfrage den kompletten statischen Teil eines Agenten als Parameter zu übergeben. Eine Lösung für dieses Problem stellt deswegen die Verwendung einer kryptographischen Hashfunktion dar (siehe auch [82]), die auf den statischen Teil des Agenten angewendet wird und einen *bit string* fester Länge zum Ergebnis hat. Dieser *bit string*, der sogenannte Hashcode des Agenten, identifiziert den Agenten nun eindeutig im Kontext kryptographischer Hashfunktionen und kann von jedem gebildet werden, der den Agenten kurzzeitig im Besitz hat, d.h. von jedem Agentenserver, auf dem der Agent zur Ausführung kommt. Die einzige Voraussetzung für dieses Vorgehen ist ein Agentenstruktur, welche die Trennung zwischen statischen und veränderlichen Anteilen des Agenten während seines gesamten Lebenszyklus ermöglicht.

Um nun auch Außenstehenden, die nie im Besitz des eigentlichen Agenten waren und niemals in dessen Besitz kommen werden, die Lokalisierung eines Agenten zu ermöglichen, kann der Besitzer den Hashcode seines Agenten durch einen weiteren Dienst veröffentlichen. Dies kann und sollte in diesem Fall durchaus durch ortsbehaftete Bezeichner geschehen, die auch den Namen des Besitzers beinhalten könnten, was den Grund hat, dass sich der Mensch einen sprechenden Namen für einen Agenten leichter merken kann, als einen *bit string*. Darüber hinaus wird durch die Kombination von ortsgebundenem Bezeichner, Namen des Besitzers und/oder aufgabenabhängigem Agentenbezeichner als Agentenname die Assoziation mit dem eigentlichen Agenten erleichtert. Es wird also eine weitere Zuordnung von einem benutzerfreundlichem Agentennamen zu dem vorher angesprochenen Hascode des Agenten als Zwischenschritt für dessen Lokalisierung nötig.

Die eigentliche Lokalisierung eines Agenten kann man dadurch nun auf die Zuordnung des Agenten-Hashcodes zu der aktuellen Position des Agenten eingrenzen. Der dafür zuständige Dienst hat also die Aufgabe, auf die Anfrage nach einem Hashcode, eine *Kontaktadresse* zurückzugeben, unter der der Agent erreichbar ist.

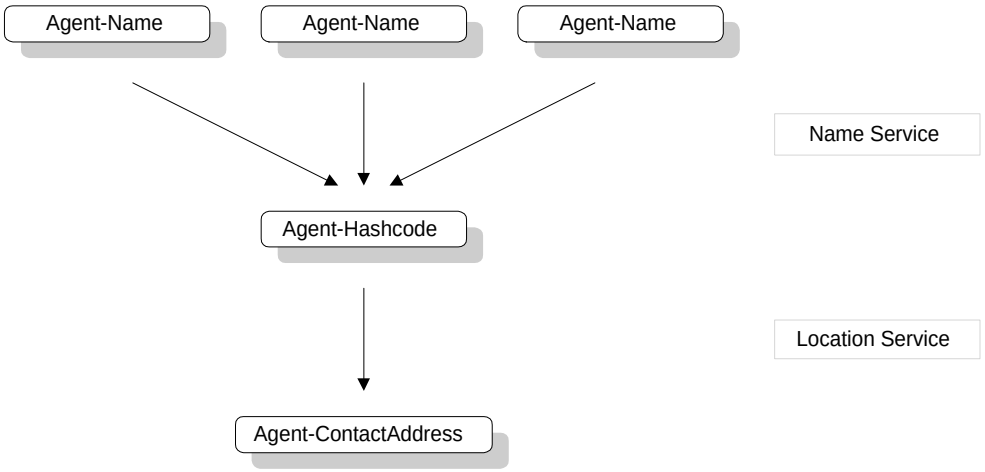


Abbildung 3.1: Die Aufgabe von Namens- und Verzeichnisdienst

In Abbildung 3.1 wird diese Aufteilung und damit die Trennung der Dienste noch einmal verdeutlicht: Ein Namensdienst hat die Aufgabe, Agentennamen, die nicht eineindeutig sein müssen, auf den Agenten-Hashcode abzubilden, der den Agenten dann allerdings eineindeutig identifiziert. Der Lokationsdienst ermittelt nun auf Grund des Hashcodes des Agenten dessen aktuelle Position im Netzwerk und gibt diese in Form einer Kontaktadresse zurück. Obwohl es im Rückblick auf die letzten Kapitel verwirrend erscheint, so ist die Differenzierung der Dienste und dadurch die abgeänderte Definition des Begriffs *Namensdienst* im Hinblick auf den noch folgenden Modellentwurf durchaus gerechtfertigt.

Kommen wir noch einmal auf die Erreichbarkeit eines Agenten unter einer Kontaktadresse zurück. Die Frage, die bis jetzt noch nicht beantwortet wurde, ist die nach dem Zweck der Kontaktadresse. Der Grund, der uns zur Lokalisierung eines Agenten bewegt, ist neben der passiven Beobachtung die aktive Interaktion (siehe auch Abschnitt 2.3.4). Als Grundlage

dafür benötigen wir eine Adresse, mit der ein sogenannter *Nachrichtendienst* die Kommunikation mit dem Agenten über Nachrichten ermöglicht, die sogenannte Kontaktadresse. Auf der Basis dieses Nachrichtendienstes sind dann vielfältige Anwendungsszenarien möglich.

Diese Dreiteilung des Dienstemodells in Namensdienst, Lokationsdienst und Nachrichtendienst kann übrigens auch dadurch begründet werden, dass die Anforderungen an die verschiedenen Dienste stark voneinander abweichen und auf diese Weise eine Optimierung jedes einzelnen Dienstes für sich möglich wird.

Die Aufgabe dieser Arbeit beschränkt sich auf die Lokalisierung eines Agenten. Da dieser Dienst allerdings nicht ohne den entsprechenden Kontext betrachtet werden kann, möchte ich jetzt kurz einen Überblick über die Aufgaben der drei Komponenten des von mir definierten 3-geteilten Dienstemodell geben, bevor in Abschnitt 3.2 der eigentliche Lokationsdienst entwickelt wird.

### 3.1.1 Name Service (NS)

Der Namensdienst ist in diesem Fall für die Zuordnung von Agentennamen zu einem Agenten-Hashcode zuständig. Im Gegensatz zu der Zuordnung von einem Agenten-Hashcode zu aktuellen Agenten-Kontaktadressen hängt die Änderungsfrequenz der Zuordnungen und damit die *update* Frequenz in diesem Fall nicht von der Migrationsrate des Agenten sondern von seinem Lebenszyklus ab, und ist somit bereits um einen gewissen Faktor kleiner als beim Lokationsdienst. Zudem wird die Intension hinter der Nutzung des Namensdienstes meistens folgende sein: Ein Agent bietet, unabhängig davon ob er statisch oder mobil ist, einen gewissen Dienst an und möchte diesen einer breiten Gruppe von Anwendern untern einen sprechenden Namen zur Verfügung stellen. Diese Intension sollte dann auch die kontinuierliche Verfügbarkeit über einen längeren Zeitraum implizieren. Geht man nun von diesen Annahmen aus, d.h die benötigten Zuordnungen können als relativ statisch angesehen werden und werden auch nur für eine kleine Untergruppe der im Gesamtsystem vorhandenen Agenten benötigt, so lässt sich durchaus auf bestehende Namensdienste zurückgreifen (siehe Abschnitt 2.5).

Vorstellbar ist zum Beispiel die Nutzung von DNS: Der Agentenname kann als Domänenname kodiert werden, der durch die Kombination der Domäne, auf die der Besitzer des Agenten Zugriff hat, und einem Agentenbezeichner entsteht. Der Agenten-Hashcode ließe sich dann in einem *resource record* vom Typ `TXT` als *byte string* speichern.

Für einen Agenten, der einen Dienst zum Finden von Bildern mit bestimmten Merkmalen anbietet, und der auf dem Host `host.domain.net` gestartet wird, ist unter dieser Annahme sogar der ausführliche Agentenname `ImageRetrievalByCharacteristic-Agent.domain.net` vorstellbar. Der Zugeordnete Hashcode kann in dem `TXT` *resource record* des für die Domäne `damain.net` zuständigen DNS-Namensservers unverändert gespeichert werden.

Es ist allerdings noch zu evaluieren, ob die oben angesprochenen Annahmen tatsächlich der Realität entsprechen und dadurch die Nutzung von DNS auch eine realistische Lösung darstellt.

### 3.1.2 Location Service (LS)

Der Lokationsdienst hat die Aufgabe einen durch einen Hashcode eindeutig identifizierbaren Agenten im Netzwerk zu lokalisieren und als Antwort auf *lookup* Anfragen eine Kontaktadresse zurückzugeben.

Wie sich auch in Abschnitt 2.6 erkennen lässt, birgt das oft verwendete Konzept der *forward references* bei der Lokalisierung mobiler Objekte einige Nachteile, die den entwickelten Anforderungen entgegen stehen. Aus diesem Grund fällt die Wahl für das zugrundeliegende Konzept dieses Lokationsdienstes auf das Prinzip des *buffered response*. Das heißt, ein existierender Lokationsserver muss von den Agentenservern durch *update* Anforderungen über die aktuelle Positionen der im Umlauf befindlichen Agenten informiert werden.

Eine detaillierte Entwicklung des Lokationsdienstes folgt ab Abschnitt 3.2.

### 3.1.3 Message Service (MS)

Der Nachrichtendienst hat die Aufgabe, Nachrichten von einem Programm bzw. einem Agenten zu einem anderen Agenten zu transportieren. Die Kontaktadresse eines Agenten spezifiziert dabei einen Agentenserver als aktuellen Aufenthaltsort des Agenten und ermöglicht dann in Kombination mit dem Hashcode des Agenten den Nachrichtentransfer.

Dafür kann die Kontaktadresse einer URL mit angegebenen Protokoll, Rechneradresse und Port entsprechen, die eine Socketverbindung zu dem Nachrichtendienst innerhalb des Agentenservers ermöglicht. Bei einem etwas allgemeineren Ansatz wäre auch vorstellbar, dass die URL statt eines Nachrichtendienstes einen allgemeinen Kontrolldienst des Agentenservers beschreibt, der die ankommenden Nachrichten filtert und gegebenenfalls Zugriff auf Kontrollfunktionen und die Informationsdienste des Agentenservers ermöglicht. Auf diese Weise ließen sich auch KQML bzw. ACL als Kommunikationsprotokolle (siehe Anhang B) oder eine Implementierung für die *Remote Method Invokation* (siehe auch [86]) auf der Basis von Nachrichten realisieren.

Ein weiterer Aspekt, der beim Entwurf eines Nachrichtendienstes beachtet werden muss, ist die Art und Weise, in der Nachrichten angenommen werden. Auf diesen Punkt wird jetzt vor Allem deswegen detaillierter eingegangen, weil er direkt mit dem Entwurf des Lokationsdienstes zu tun hat:

- Nachrichten könnten von dem Agentenserver nach Empfang gepuffert werden, bis der entsprechende Agent die an ihn adressierte Nachricht über eine definierte Schnittstelle des Server abruft. Dies ist besonders dann sinnvoll, wenn der Lokationsdienst die neue Position des Agenten kurz vor seiner Migration schon beim Lokationsserver registriert und Nachrichten auf dem Zielserver bereits während der Migration für den bald ankommenden Agenten gepuffert werden. Damit würde die Erreichbarkeit des Agenten optimiert. Dieses Konzept bietet allerdings die Möglichkeit einer DoS Attacke: Da der Zielserver nicht weiß, ob und wann ein Agent eintrifft, muss er alle eintreffenden Nachrichten puffern. Nun könnte ein Angreifer die Kontaktadresse eines Agentenservers mit „sinnlosen" Nachrichten überhäufen und dadurch gegebenenfalls den Nachrichtendienst überlasten, wodurch reguläre Nutzer keinen Zugriff mehr auf diesen Dienst

hätten. Es könnte natürlich auch passieren, dass ein Agent beim Transport verloren geht bzw. durch im Voraus erkannte Netzwerkprobleme eine Weile lang in eine Sende-Warteschlange gestellt wird und dadurch reguläre Nachrichten an ihn auf dem Zielserver in der Zwischenzeit die Empfangspuffer füllen.

- Ein alternatives Modell des Nachrichtendienstes könnte Agenten, die sich tatsächlich auf dem Server befinden und vorher die Bereitschaft angemeldet haben Nachrichten zu empfangen, durch einen Art *event* Mechanismus (vergleichbar mit dem in Abschnitt 2.2.1 beschriebenen) direkt über eintreffende Nachrichten benachrichtigen. Falls sich der durch eine Nachricht adressierte Agent nicht auf dem Server befindet bzw. nicht bereit ist Nachrichten zu empfangen, so wird diese Nachricht sofort gelöscht. In diesem Modell muss mit einem gewissen Nachrichtenverlust während den Migrationsphasen der Agenten gerechnet werden. Eine Sicherung gegen diesen vom Sender vorerst unbemerktem Nachrichtenverlust lässt sich nur durch bidirektionalen Nachrichtenverkehr auf einem geöffneten Kanal zum Zielagenten bzw. durch die Auswertung von asynchronen Ergebnisnachrichten vom Agenten (im positiven Sinn) bzw. des Nachrichtendienstes (im negativen Sinn) erreichen.
- Geht man davon aus, dass die Position eines Agenten erst nach dessen Migration, von dem Zielserver beim Lokationsserver registriert wird, so könnte der Ursprungsserver während der Migrationsphase des Agenten weitere Nachrichten an diesen empfangen. Um diese Nachrichten nicht einfach verfallen zu lassen, wären das einmalige forwarding der Nachrichten an den Zielserver in Kombination mit der dortigen Pufferung bzw. das fortlaufende *forwarding* an den Zielserver in bestimmten Intervallen, bis der Agent die Nachricht in Empfang nehmen kann, mögliche Lösungsvorschläge. Wird während des gesamten Agententransport und bis zu dessen erneuten Ausführung nach abgeschlossener Migration die Verbindung zwischen Ursprungs- und Zielserver offen gehalten, so könnten Statusmeldungen ausgetauscht werden, die diesen Vorgang optimieren würden.

Wird mit Nachrichtenpuffern gearbeitet, sollte die Möglichkeit einer DoS Attacke nicht außer Acht gelassen werden. Es empfiehlt sich auf jeden Fall, ungelesene Nachrichten nach dem Verstreichen eines bestimmten Zeitintervalls zu löschen.

Für den folgenden Entwurf des Lokationsdienstes in dieser Arbeit wird nun davon ausgegangen, dass der Agentenserver die Position eines Agenten sofort durch eine *update* Anfrage veröffentlicht, nachdem sein Hashcode ermittelt werden konnte. Es wird nicht vorausgesetzt, dass die Verbindung bei der Migration noch bis zur eigentlichen Ausführung des Agenten auf dem Zielserver offen gehalten wird.

## 3.2 Der Location Service

Wie bereits im letzten Abschnitt erläutert, besteht die Struktur des Lokationsdienstes also aus einer Client-Server-Architektur, in der ein Server die *update* bzw. *lookup* Anfragen der Clients entgegen nimmt und bearbeitet. Durch eine update Anfrage wird eine Zuordnung von Agenten-Hashcode zu Agenten-Kontaktadressen gespeichert, aktualisiert oder gelöscht. Eine lookup Anfrage, bei der ein Agenten-Hashcode als Parameter übergeben wird, sollte dann

im günstigen Fall die Agenten-Kontaktadresse des entsprechenden Agenten und damit seine aktuelle Position zum Ergebnis haben.

Um dies zu realisieren, muss auf jedem Agentenserver ein Client des Lokationsdienstes installiert sein, der durch den Server dazu veranlasst wird die Position der auf ihr zur Ausführung kommenden Agenten an den Server weiterzugeben. Darüber hinaus sollte eine auch den Agenten zugängliche Schnittstelle des Clients ermöglichen, *lookup* Anfragen zu stellen. Der Server des Lokationsdienstes muss nun in der Lage sein, ankommende Anfragen bearbeiten und in seiner internen Datenbank Änderungen vornehmen zu können.

Der Flaschenhals beim Speicher- und Kommunikationsaufkommen stellt eindeutig der Server des Lokationsdienstes dar. Um den Ansatz skalierbar zu gestalten ist in dieser Hinsicht eine Aufspaltung der Zuständigkeiten bezüglich der Verwaltung der Agentenpositionen also dringend erforderlich. Dass heißt, es muss mehrere Lokationsserver geben die physikalisch möglichst gut im Netzwerk verteilt werden sollten, und die Agenten müssen einem für sie zuständigem Lokationsserver zugeordnet werden. Dabei ist es aus verschiedenen Gründen (siehe auch Abschnitt 3.1) nicht sonderlich vorteilhaft Lokalität ausnutzen zu wollen oder sogar eine dynamische Zuordnung vergleichbar der im Globe-Projekt (siehe Abschnitt 2.6.1) anzuwenden.

Eine Lösung stellt nun die Nutzung des Agenten-Hashcodes dar. Dieser kann nämlich neben der Identifikation eines Agenten auch dazu genutzt werden, um den zuständigen Lokationsserver ausfindig zu machen (siehe Abschnitt 2.8):

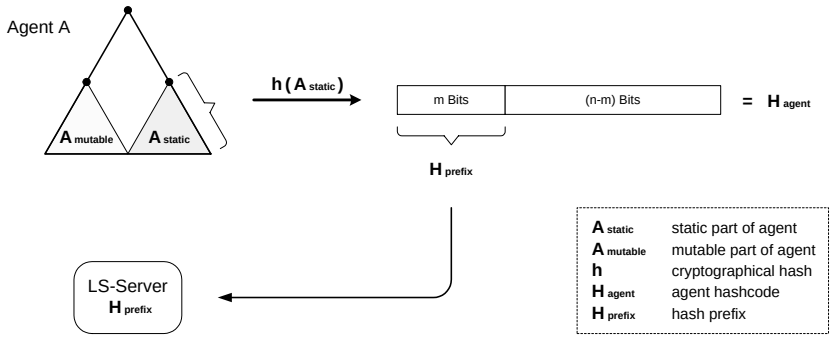


Abbildung 3.2: Ermittlung des Hashpräfixes zur Wahl des zuständigen Lokationsservers

Liefert die kryptographische Hashfunktion einen Agenten-Hashcode der Länge `n` Bits, so lassen sich wie in Abbildung 3.2 die ersten `m` Bits als Hashpräfix dafür nutzen, den zuständigen Lokationsserver zu bestimmen. In diesem Fall wären dann $2^m$ Lokationsserver für die Gesamtheit der im Umlauf befindlichen Agenten zuständig.

Diese Methode der Lokationsserver-Wahl hat nun einige Vorteile: Zum einen ist die Verteilung der Agenten auf die zuständigen Lokationsserver und damit auch die Verteilung des Speicheraufkommens und der Kommunikationslast durch die Nutzung einer kryptographischen Hashfunktion bei der Hashcodeberechnung gleichmäßig. Zum anderen kann diese Zuordnung nicht im Nachhinein manipuliert werden, da sie durch den statischen Teil des Agenten selbst bestimmt wird und für jeden nachvollziehbar ist. Außerdem lässt sich dieser Ansatz bei

steigender Last durch eine erhöhte Anzahl von im Umlauf befindlichen Agenten sehr gut skalieren. Durch die Wahl der Bits die für den Hashpräfix relevant sind, lässt sich die Anzahl der Lokationsserver nämlich recht unproblematisch regulieren.

Der Prototyp des in dieser Arbeit entwickelten Lokationsdienstes wird im Kontext des Mobile Agenten Systems SeMoA auf die Anfrage eines Agenten-Hashcodes also die Position des dadurch eindeutig identifizierten Agenten in Form einer Agenten-Kontaktadresse zurück geben. Es ist aber durchaus vorstellbar, dass dieser Dienst auch in anderen Systemen eingesetzt wird, die ähnliche Anforderungen an einen Lokationsdienst stellen. Aus diesem Grund wird in den folgenden Abbildungen und Tabellen, in denen Datentypen benannt werden, bereits eine nicht so spezifische Bezeichnung gewählt: Der Agenten-Hashcode wird in dem Datentyp ImplicitName gespeichert und die Agenten-Kontaktadresse wird allgemeiner als ContactAddress bezeichnet.

### 3.2.1 Komponenten des Location Service im Überblick

Nachdem im letzten Abschnitt das Prinzip des Lokationsdienstes an sich beschrieben wurde, folgt in diesem Abschnitt die Darstellung seiner Struktur im Netzwerk mit allen benötigten Komponenten und deren Aufgaben im Zusammenspiel miteinander. Dabei wird zum Teil schon recht detailliert auf interne Strukturen bzw. Sicherheitsaspekte eingegangen. Das „LS" bei den englischsprachigen Komponentennamen steht im folgenden für „Location Service".

Vorher sollen aber auch noch die Gründe dafür genannt werden, warum bei diesem Lokationsdienst auf die Replikation der Lokationsserver und das Zwischenspeichern von Anfrageergebnissen (*caching*) auf der Client-Seite verzichtet wurde:

Gründe für die Replikation wären die Erhöhung der Datensicherheit durch Redundanz bzw. bei weit verteilten Replikaten, die mögliche Ausnutzung von Lokalität für die Clients bei ihren Anfragen. Die Erhöhung der Datensicherheit kann bei Bedarf allerdings auch durch interne Mechanismen der entsprechenden Datenbank erreicht werden, und die weite Verteilung von Replikaten ist deswegen nicht sinnvoll, weil dann komplizierte Synchronisationsalgorithmen nötig wären, die zudem auch noch leistungsstark genug sein müssten, um mit der hohen Änderungsfrequenz der Daten mithalten zu können. Aus letzterem Grund ist auch das Zwischenspeichern von Anfrageergebnissen nicht sinnvoll. Bei einer hohen Änderungsfrequenz der zu speichernden Daten ist nämlich auch davon auszugehen, dass sich die Ergebnisse auf gleiche Anfragen häufig ändern. Durch einen Cache auf Client-Seite würden dann unbemerkt falsche Ergebnisse zurück geliefert.

In Abbildung 3.3 ist ein erster Überblick über die beim Lokationsdienst beteiligten Komponenten gegeben. Die drei Bereiche Agenten-Server, LAN und Internet sind mit Sicherheits- oder Vertrauensbereichen gleichzusetzen, die bei der späteren Betrachtung der Kommunikationsschnittstellen eine wichtige Bedeutung erlangen. Nur die beiden Komponenten LS-InfoGUI und LS-StrorageDB fehlen in diesem Diagramm, da sie eine etwas gesonderte Rolle spielen.

Bevor nun die Beschreibung der einzelnen Komponenten folgt wird noch einmal ein neuer Begriff eingeführt. Bis jetzt wurde von *lookup* und *update* Anfragen an den Lokationsdienst

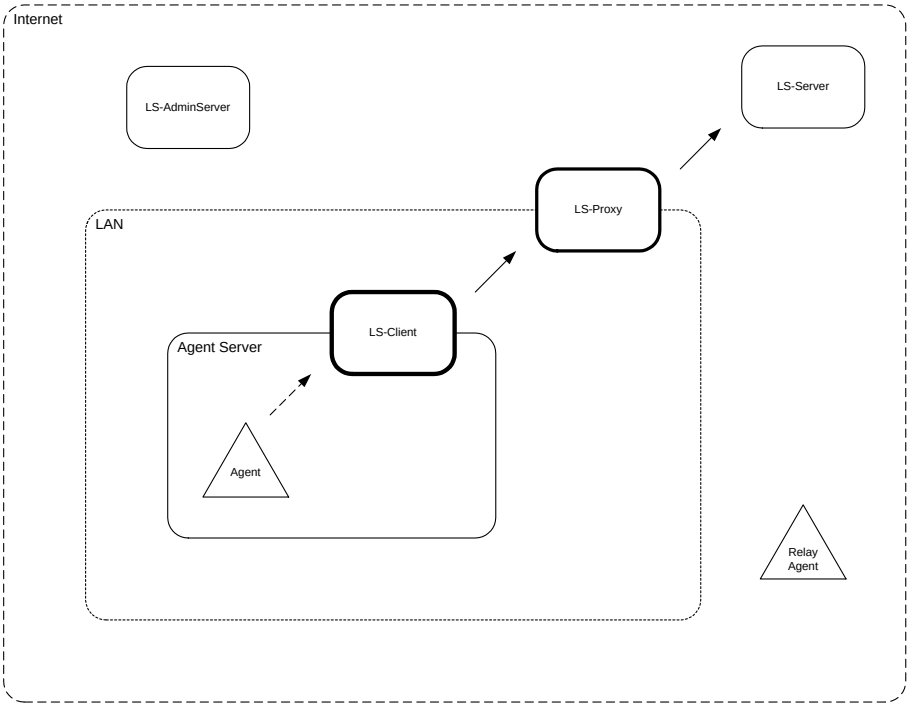


Abbildung 3.3: Die Sicherheitsbereiche für die Komponenten des Lokationsdienstes

gesprochen. Im folgenden wird zwischen den lesenden *lookup* und den schreibenden *register* Anfragen unterschieden. Eine register Anfrage umfasst dabei neben der Aktualisierung einer Hashcode/Kontaktadressen-Zuordnung (*update*) auch dessen initiale Eintragung (*init*) und abschließende Löschung (*delete*).

### LS-Server

Ein LS-Server ist jeweils für alle Agenten mit einem bestimmten Hashpräfix (in dem sie identifizierenden Agenten-Hashcode) verantwortlich und speichert die entsprechenden Hashcode/Kontaktadressen-Paare dieser Agenten in seiner lokalen Datenbank. Darüber hinaus beantwortet er alle *lookup* Anfragen, sofern er über die entsprechenden Daten verfügt, und dies möglichst parallel zu anderen Anfragen.

Um aus Sicherheitsgründen gegebenenfalls verschlüsselte Anfragen zuzulassen, wird dem LS-Server ein *public key / private key* Schlüsselpaar zugeordnet. Um die Möglichkeit zu haben, Angriffsversuche im Nachhinein zurück verfolgen zu können, sollten alle *register* Anfragen in einem logfile gespeichert werden.

Alle Einträge in der Datenbank werden mit einem Zeitstempel versehen und nach dem Verstreichen einer bestimmten Zeitspanne wieder gelöscht, falls sie bis dahin nicht vom LS-Client durch eine refresh Anfrage erneut bestätigt wurden. Diese Zeitspanne kann durchaus im Bereich von Tagen liegen. Der Grund für dieses Vorgehen ist zum einen der Schutz gegen eine DoS Attacke, bei der durch eine große Anzahl von „unsinnigen" *update* Anfragen die

Datenbank gefüllt wird, ohne dass diese Eintragungen jemals wieder gelöscht würden. Ein anderer Grund liegt in dem Aufbau des Schnittstellenprotokolls LSP verborgen, mit dem der LS-Server angesprochen werden kann (siehe Abschnitt 3.2.3). Es sollte möglich sein, in einer Refresh-Anfrage gleich mehrere Eintragungen auf einmal zu bestätigen, indem eine Liste von Hashcodes für die entsprechenden Eintragungen übergeben werden.

Der Hashpräfix, für den der Server zuständig ist, wird übrigens nicht von dem Server selbst gewählt sondern von einer anderen Instanz zugewiesen (siehe LS-AdminServer bzw. Initialisierungsdatei von LS-Proxy und LS-Client). Dies muss bei dem Empfang von Anfrage-Nachrichten berücksichtigt werden. Der Vorteil der dadurch entsteht wird bei der Beschreibung des LS-AdminServers näher erläutert.

Damit der LS-Server seine Arbeit aufnehmen kann, muss eine Schnittstelle zum LS-Proxy bzw. LS-Client definiert werden, welche die Anfragen *lookup*, *register* (*init*, *update*, *delete*) und *refresh* über das Internet als schwächsten Vertrauensbereich (siehe Abbildung 3.3) behandelt. *Lookup* Anfragen sollen dabei jedem möglich sein, *register* Anfragen nur autorisierten Komponenten (siehe Abschnitte 3.2.2 und 3.2.3).

Darüber hinaus sollte eine lokale Schnittstelle existieren, die im stärksten Vertrauensbereich zur Administration der Serverdatenbank bzw. zur Kontrolle durch das LS-InfoGUI benutzt werden kann.

### LS-Proxy

Der LS-Proxy stellt bei der Kommunikation zwischen LS-Client und LS-Server eine mögliche Zwischeninstanz dar, die sich zusammen mit den anfragenden LS-Clients in dem Vertrauensbereich des LAN-Subnetzes befinden muss und dadurch als Gateway ins Internet betrachtet werden kann. Falls die Kontaktadresse bei einer *register* Anfrage einen Agentenserver im gleichen LAN identifiziert, speichert der Proxy unabhängig von einem Hashpräfix grundsätzlich alle Hashcode/Kontaktadressen-Zuordnungen in seiner internen Datenbank. Dadurch erhält der LS-Proxy gewissermaßen die Zuständigkeit für das gesamte Subnetzwerk.

Wie man auch in Abbildung 3.4 erkennt, nimmt der LS-Proxy also ausschließlich Anfragen von LS-Clients aus dem gleichen LAN an. Sofern sich die durch eine *lookup* Anfrage angeforderte Kontaktadresse in seiner Datenbank befindet, beantwortet er diese Anfrage direkt, andernfalls wird diese dem zuständigem LS-Server weitergegeben. *Register* Anfragen werden im Gegensatz dazu *immer* auch an der zuständigen LS-Server weitergegeben (*write-through*).

Jetzt wäre folgende Situation vorstellbar: Der LS-Proxy enthält in seiner Datenbank die Kontaktadresse eines Agenten. Migriert dieser nun zu einem Agentenserver, der sich nicht mehr innerhalb des LAN befindet, wird seine neue Position durch diesem Server direkt bzw. über den LS-Proxy des entsprechenden LAN beim zuständigen LS-Server registriert. Eine *lookup* Anfrage an den Ursprungsproxy würde dann fälschlicherweise noch die alte Kontaktadresse des Agenten zum Ergebnis haben.

Um dieses Fehlverhalten zu verhindern muss der LS-Proxy eine zusätzliche Funktion (*invalidate*) unterstützen: Wird von dem Agentenserver festgestellt, dass ein Agent bei der Migration das LAN verlässt, so ist sie dafür verantwortlich dieses mittels einer *invalidate*

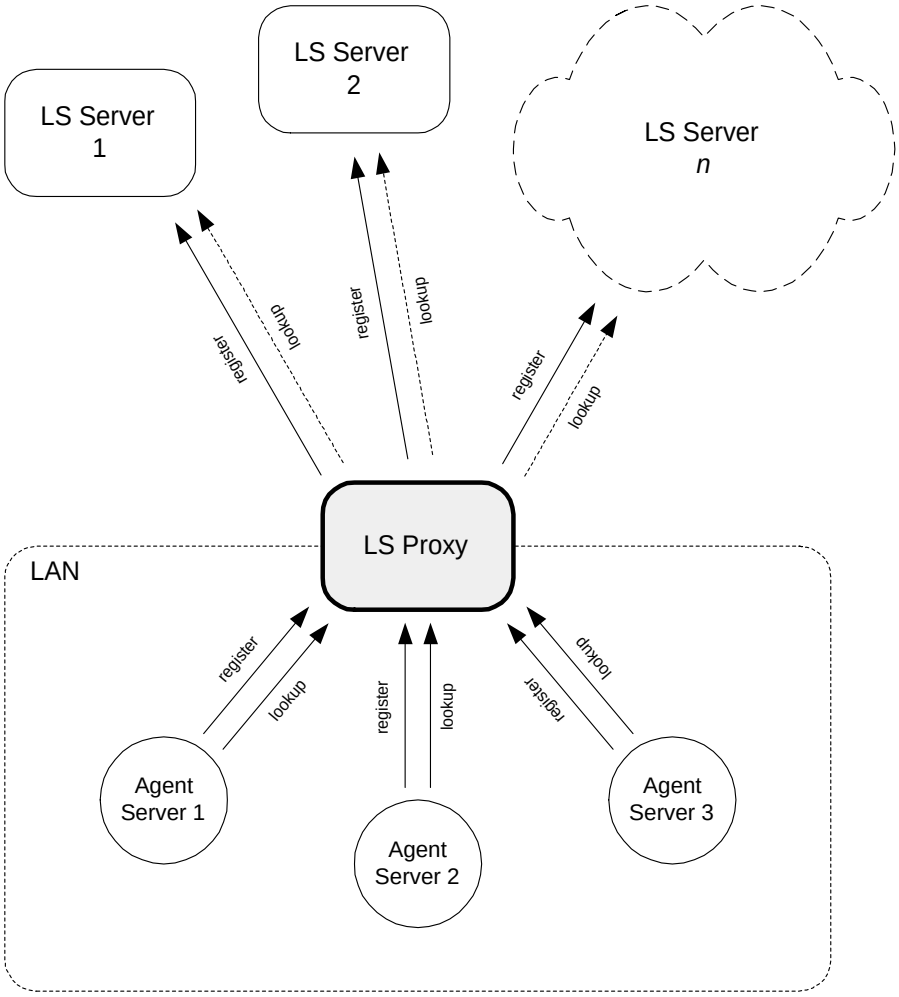


Abbildung 3.4: Die Funktion des LS-Proxy

Anfrage über den LS-Client auch dem LS-Proxy mitzuteilen, damit sich dessen Datenbank anschließend immer noch in einem konsistenten Zustand befindet.

Eine andere aber nicht so sichere Alternative wäre ein *timeout* für die Zuordnungen in der Datenbank des LS-Proxy. Dieses sollte aber aus den gleichen Gründen wie beim LS-Server auf jeden Fall eingeführt werden. Außerdem sollte auch der LS-Proxy *register* Anfragen in *logfiles* festhalten.

Für das Konzept des LS-Proxy wird davon ausgegangen, dass sich nur ein Proxy innerhalb eines LAN befindet. Es wird angenommen, dass in einem LAN ein Proxy ausreicht, um die auftretende Last zu bewältigen. Um zu gewährleisten, dass nicht zwei LS-Proxy in einem LAN installiert werden bzw. um die LS-Clients über die Anwesenheit eines LS-Proxy zu informieren beinhaltet der Proxy einen multicast daemon, der entsprechende *broadcast* Nachrichten aus dem Subnetz positiv mit einer Kennung und seiner Adresse beantwortet.

Eine Alternative zu diesem Vorgehen, wäre das Vorhandensein mehrerer *potentieller* LS-

Proxy, die untereinander aushandeln, welcher nun für das LAN verantwortlich ist (Prinzip des voting). Fällt der aktuelle LS-Proxy aus, so kann aus den anderen potentiellen LS-Proxy sofort ein neuer gewählt werden. Die LS-Clients werden in diesem Modell vom Wahlmechanismus direkt über das jeweils aktuelle Ergebnis informiert werden (siehe hierzu auch Kapitel 4).

In diesem Modell benötigt der LS-Proxy also eine Schnittstelle zum LS-Client, die über die regulären Anfragen an einen LS-Server hinaus auch eine *invalidate* Anfrage unterstützen muss. Wie auch beim LS-Server ist für den LS-Proxy ebenfalls eine lokale Schnittstelle zur Administration und für ein LS-InfoGUI vorgesehen. Damit gegebenenfalls Kontakt mit den entsprechenden LS-Servern aufgenommen werden kann, muss der Proxy wissen, welches Hashpräfix welchem LS-Server zugeordnet ist. Dies geschieht entweder durch die Konfiguration mittels einer Initialisierungsdatei oder durch den LS-AdminServer (siehe unten).

Die Einführung dieser Komponente hat nun besonders zwei Gründe: Zum einen sichert sie dem Lokationsdienst im LAN eine gewisse Autonomität zu, d.h. dieser kann auch abgekoppelt vom Internet in einem kleinen abgeschlossenem System mit den darin enthaltenen Agenten und Agentenservern weiterhin seine Aufgabe erfüllen. Zu dieser Situation kommt es z.B. bei Netzwerkproblemen. Zum anderen wird dadurch die Bearbeitungsgeschwindigkeit bei Anfragen des LS-Clients verbessert.

#### LS-StorageDB

Wie in Abbildung 3.5 zu erkennen ist, stellt LS-StorageDB keine eigentliche Komponente dar, sondern eine einheitliche Schnittstelle mit der auf die jeweils implementierte Datenbank zugegriffen werden kann:

Da bei jeder Anfrage an LS-Proxy bzw. LS-Server auf die Datenbank zugegriffen werden muss, sollte die zur Implementierung gewählte Datenbankstruktur genug Leistung bieten (siehe auch Kapitel 4).

| Datentyp | Beschreibung |
|---|---|
| ImplicitName | Der Hashcode des statischen Teil eines Agenten |
| `ContactAddress` | Die aktuelle Kontaktadresse des Agenten der durch den `ImplicitName` identifiziert wird |
| `Cookie` | Das aktuelle Cookie, das für das Kommunikationsprotokoll (siehe Abschnitt 3.2.3) benötigt wird |
| `Timestamp` | Der Zeitpunkt der letzten Eintragsänderung |

Tabelle 3.1: Die Struktur der Datenbankeinträge für LS-Proxy bzw. LS-Server

In Tabelle 3.1 ist die Struktur der Daten zu finden, auf die durch die LS-StrotageDB im Rahmen eines LS-Proxy bzw. LS-Server zugegriffen werden müssen.

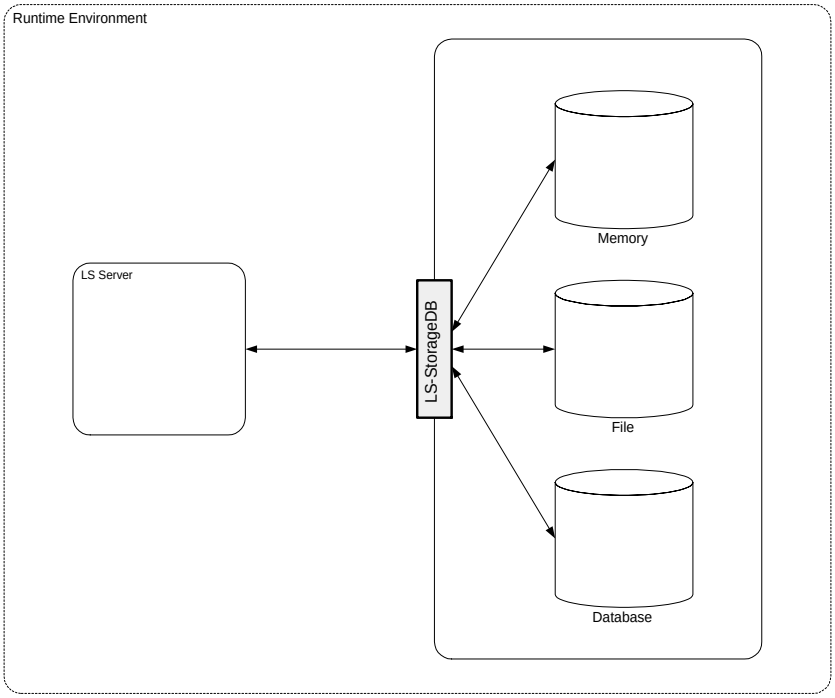


Abbildung 3.5: Die Schnittstelle zwischen LS-Server und LS-StorageDB

### LS-Client

Der LS-Client bildet innerhalb eines Agentenservers die Schnittstelle zum Lokationsdienst. Er nimmt *lookup*, *register*, *refresh* und *invalidate* Anfragen entgegen und leitet diese an die entsprechenden Komponenten des Lokationsdienstes weiter. Dies geschieht auf den drei verschiedenen Ebenen *local*, *proxy* und *global*:

```
lookup.local(ImplicitName)
+-lookup.proxy(ImplicitName)
  +-lookup.global(ImplicitName)

register.proxy(ImplicitName,ContactAddress)
+-register.global(ImplicitName,ContactAddress)

refresh.proxy(ImplicitNameList)
refresh.global(ImplicitNameList)

invalidate.proxy(ImplicitName)
```

Abbildung 3.6: LS-Client Schnittstellen gegenüber dem Agentenserver

Durch `lookup.local` wird die interne Datenbank des Agentenservers durchsucht. Befindet sich der durch den Hashcode identifizierte Agent lokal auf dem Agentenserver, so kann die Antwort auf die Anfrage sofort gegeben werden. Andernfalls wird die Anfrage an den LS-Proxy weitergegeben, sofern dieser lokal im LAN installiert ist (dies entspräche einem direkten Aufruf von `lookup.proxy`). Ist kein LS-Proxy installiert, wird der dem Hashcode des Agenten zugeordnete LS-Server angefragt (entspricht einem `lookup.global`).

Ähnlich ist das Vorgehen bei den *register* Anfragen: Durch `register.proxy` wird eine Hashcode/Kontaktadressen-Zuordnung beim LS-Proxy aktualisiert und von diesem gleichzeitig an den zuständigen LS-Server weitergeleitet (äquivalent mit `register.global`).

Durch die `refresh` Schnittstelle können die Einträge bei LS-Proxy und LS-Server regelmäßig bestätigt werden. Diese Anfragen beziehen sich durch die Angabe einer Liste von Hashcodes gleich auf mehrere Einträge. Dabei muss allerdings bedacht werden, dass für den LS-Server die Einträge nach Hashpräfixen sortiert und dann an den jeweils zuständigen LS-Server weitergeleitet werden müssen. Da die Zeitintervalle, in denen Die Einträge in den Datenbanken der LS-Proxy bzw. LS-Server bestätigt werden müssen, unterschiedlich lang sein können, sollte der LS-Client auch zwei getrennte *daemons* zur Erledigung dieser Aufgabe initiieren.

Die `invalidate` Schnittstelle erfüllt die beim LS-Proxy bereits beschriebene Funktion.

Durch die entsprechenden Schnittstellenfunktionen können die verschiedenen Ebenen auch direkt angesprochen werden, dies sollte allerdings vermieden werden, wenn man einen im LAN installierten LS-Proxy nutzen möchte. Wird der LS-Proxy nämlich nur von einigen LS-Clients im LAN für *register* Anfragen verwendet, so können bei *lookup* Anfragen an den LS-Proxy unter Umständen veraltete (falsche) Ergebnisse zurück gegeben werden (vergleichbar mit einer unterlassenen *invalidate* Anfrage).

Wie auch der LS-Proxy muss der LS-Client Kontakt mit den LS-Servern aufnehmen können und braucht deswegen deren Adressen im Netzwerk. Dies geschieht ebenfalls durch die Konfiguration durch eine Initialisierungsdatei bzw. Anfragen an den LS-AdminServer.

Bei der Initialisierung eines neuen LS-Clients versucht dieser, einen im LAN möglicherweise installierten LS-Proxy mittels entsprechenden *broadcast* Nachrichten zu lokalisieren. Auch der LS-Client sollte *register* Anfragen in *logfiles* speichern.

#### LS-AdminServer

LS-CLients und LS-Proxy benötigen für die Behandlung von Anfragen die Adressen und gegebenenfalls die *public keys* (siehe Abschnitt 3.2.3) der entsprechenden LS-Server. Es ist vorgesehen, dass diese Konfigurationsdaten beim Initialisieren dieser Komponenten aus einer Initialisierungs-Datei eingelesen und im Speicher gehalten werden. Das Modell des in dieser Arbeit entwickelten Lokationsdienstes erlaubt aus Gründen der Skalierbarkeit allerdings, dass die Anzahl der LS-Server nachträglich verändert werden kann, womit sich auch die Verantwortung der einzelnen LS-Server für die Hashpräfixe verändert. Diese Änderungen müssten dann manuell in den Initialisierungsdateien aller LS-Clients und LS-Proxy angepasst werden.

Eine Alternative hierzu soll der LS-AdminServer bietet. Seine Aufgabe ist, die Anzahl der relevanten Bits des Agenten-Hashcodes für die Zuordnung zum entsprechenden LS-Server und

darüber hinaus auch alle dadurch möglichen Zuordnungen von Hashpräfixen zu den Adressen und den *public keys* des entsprechenden LS-Server zu speichern.

Ändert sich diese Daten nun durch eine Veränderung der LS-Server-Infrastruktur, so müssen diese Änderungen nur noch beim LS-AdminServer festgehalten werden. Die Rolle von LS-Client, LS-Proxy und LS-Server sieht dabei jetzt folgendermaßen aus:

- Bei der Initialisierung dieser Komponenten werden die angesprochenen Daten vorerst aus der lokal gespeicherten Initialisierungsdatei gelesen und für die Dauer der Ausführung im Speicher gehalten.
- Falls sich die LS-Server-Infrastruktur nicht verändert, sollte der reibungslose Betrieb gewährleistet sein.
- Wird nun die Infrastruktur der LS-Server verändert, sollten diese Änderungen auch beim LS-AdminServer eingetragen werden.
- Die LS-Server müssen sich in regelmäßigen Abständen, nach dem für sie relevanten Hashpräfix beim LS-AdminServer, erkundigen und dadurch Änderungen in der Infrastruktur bemerken.
- Anfragen von LS-Client bzw. LS-Proxy mit dem falschen Hashpräfix als Parameter sollten mit einer entsprechenden Antwort abgelehnt werden.
- Erhält nun ein LS-Client bzw. LS-Proxy eine solche Antwort auf eine gestellte Anfrage, so informieren sie sich ebenfalls beim LS-AdminServer über die dem entsprechenden Hashcode zugeordneten Daten und stellen die Anfrage dann erneut an den ermittelten LS-Server.
- LS-Client bzw. LS-Proxy aktualisieren nun in regelmäßigen Abständen und/oder beim Herunterfahren die Initialisierungsdatei mit den im Speicher befindlichen Daten.

Auf diese Weise werden die Veränderungen der LS-Server-Infrastruktur, die übrigens nicht sehr häufig auftauchen sollten, bei Bedarf nach und nach an die LS-Clients und LS-Proxy weitergegeben. Da die Initialisierungs-Datei regelmäßig aktualisiert wird, ist es für dies Komponenten auch nicht nötig, den gesamten Datensatz bei einem Neustart jedes mal komplett vom LS-AdminServer anzufordern. Dadurch wird vermieden, dass der LS-AdminServer seinerseits zu einem neuen Kommunikations-Flaschenhals wird. Voraussetzung für die Einführung eines LS-AdminServers ist allerdings, dass *zumindest seine Adresse* bei der Initialisierung bekannt ist.

Für die Realisierung des LS-AdminServers kann ein DNS-Namensserver dienen: Die Infrastruktur der LS-Server ist relativ statisch und alle benötigten Informationen lassen sich in vorhandenen *resource record* Typen des DNS speichern (siehe Abschnitt 2.5.1).

Wählt man als DNS-Domäne für den LS-AdminServer z.B. `LS-AdminServer.org`, so kann man die Daten des LS-Server, der für den Hashpräfix `123f5a` zuständig ist, in den *resource records* des Eintrags `123f51.LS-AdminServer.org` folgendermaßen speichern:

Unter `info.LS-AdminServer.org` ist dann im *resource record* des Typs `TXT` die Länge des relevanten Hashpräfixes in Bits angegeben. Theoretisch ist durch die Nutzung von DNS

| RR-Typ | zu speichernde Informationen |
|---|---|
| `A` | IP-Adresse des zum Hashcode zugeordneten LS-Server |
| `WKS` | Vom LS-Server unterstütze Portadresse für Anfragen über das Internet |
| `KEY` | *Public key* zur Verschlüsselung von Anfragen an den LS-Server |

Tabelle 3.2: Nutzung der DNS *resource records* für die Realisierung des LS-AdminServers

sogar eine autorisierte Änderung dynamisch von externen Rechnersystemen möglich (siehe Kapitel 2.5.1).

#### LS-InfoGUI

Die LS-InfoGUI stellt eine graphisches Benutzerschnittstelle dar, mit der man auf der einen Seite die Aktivitäten auf LS-Proxy und LS-Server darstellen und auf der anderen Seite aktiv *register* Anfragen initiieren kann.

Sie wird lokal, und damit im größten Vertrauensbereich, zusammen mit den entsprechenden Komponenten installiert und benötigt eine besondere Schnittstelle zu diesen Komponenten. Es sollte möglich sein, durch eine sogenannte *list* Anfrage mit einem Zeitstempel als Übergabeparameter inkrementell alle nach einem bestimmten Zeitpunkt auf dem LS-Proxy bzw. LS-Server registrierten Agenten zurück zu bekommen. Darüber hinaus sollte ein direkter Zugriff auf die LS-StorageDB machbar sein. Vorstellbar wäre auch eine Darstellung der *logfile* Daten bzw. ein durch SSL gesichertes Protokoll zur *Fernanalyse* statt der lokalen Schnittstelle.

Wird die LS-InfoGUI zusammen mit einem LS-Client installiert sollte dem autorisierten Benutzer ermöglicht werden, manuell Anfragen auf allen drei Ebenen (local, proxy und global) stellen zu können.

Ein alternatives bzw. ergänzendes Modell für die Realisierung des LS-InfoGUI ist das polling: LS-Proxy und LS-Server müssen dafür einen Mechanismus zur Verfügung stellen, der von sich aus in der Lage ist, über jede interne Veränderung inkrementell informieren zu können. Das LS-InfoGUI muss sich dann nach seiner Installation bei den genannten Komponenten registrieren, um die gewünschten Informationen zu erhalten.

Die LS-InfoGUI vereinfacht auf diese Weise also den Test und gleichzeitig die Administration des Lokationsdienstes.

#### LS-RelayAgent

Der LS-RelayAgent ist eigentlich keine tatsächliche Komponente des Lokationsdienstes. Er kann allerdings dafür verwendet werden, den Lokationsdienst sicherer gegen Attacken auf die privacy zu machen:

Wie man in Abbildung 3.7 erkennen kann, ist der LS-RelayAgent ein Agent der auf einem bestimmten Port im Internet auf Nachrichtenpakete wartet und diese dann an eine spezifizierte

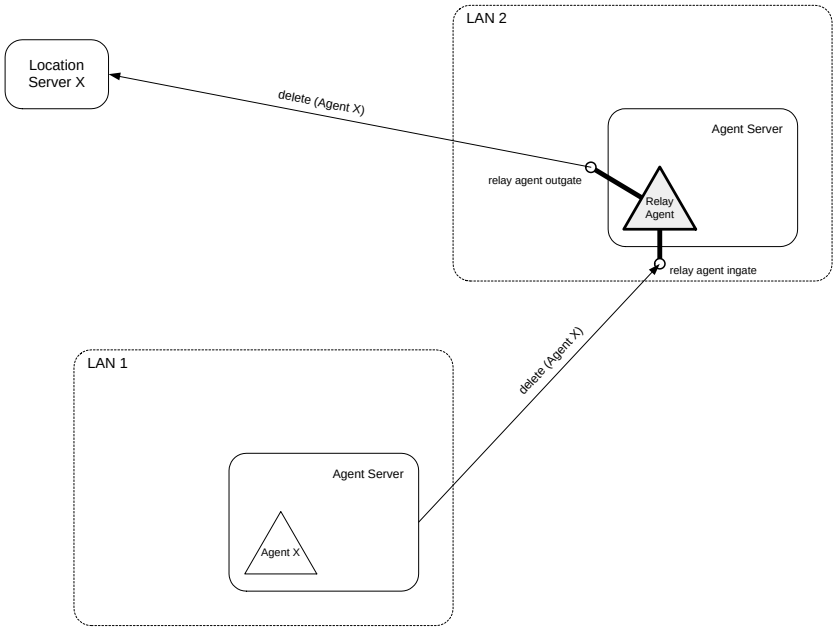


Abbildung 3.7: Die Funktion des LS-RelayAgent

Zieladresse weiterleitet. Im dargestellten Fall leitet er eine *delete* Anfrage eines Agentenservers an den entsprechenden LS-Server weiter, wobei die wahre Absenderadresse verschleiert wird.

Geht man davon aus, dass ein Besitzer seine Agenten nach Erfüllung ihrer Aufgabe und Präsentation der Ergebnisse in der Regel auf seinem eigenem Agentenserver terminieren lässt, könnte der Besitzer des LS-Server dabei Rückschlüsse auf die Identität des Besitzers ziehen. Dies wird durch den LS-RelayAgent verhindert.

### 3.2.2 Schnittstellendefinition

In folgender Abbildung 3.8 werden jetzt alle Schnittstellen zwischen den einzelnen Komponenten im Überblick dargestellt und anschließend im Einzelnen betrachtet definiert. Die Pfeilrichtungen geben dabei an, von welcher Komponente die initiale Kommunikation jeweils ausgeht.

Dabei werden auch die in Abbildung 3.3 bereits dargestellten Sicherheitsbereiche vor Allem bei der Sicherung der Kommunikationskanäle beachtet. Die Kommunikationskanäle über das Internet betrachtet, ist übrigens zu bemerken, dass die Entscheidung letztendlich klar für TCP und damit einem verbindungsorientierten Transportprotokoll fiel. Die zu übertragenden Kommunikations-Nachrichten könnten durchaus in einem einzigem UDP-Datenpaket mit weniger *overhead* für den Paketkopf übertragen werden. Damit wäre auch ein Mechanismus zur Erhaltung der Sequenzfolge bei aufeinander folgendenden Datenpaketen nicht nötig, der in TCP integriert ist. Allerdings bietet TCP darüber hinaus eine Fehlerkorrektur die gegenüber UDP einen wesentlich geringeren Paketverlust bei Übertragungen im Inter-

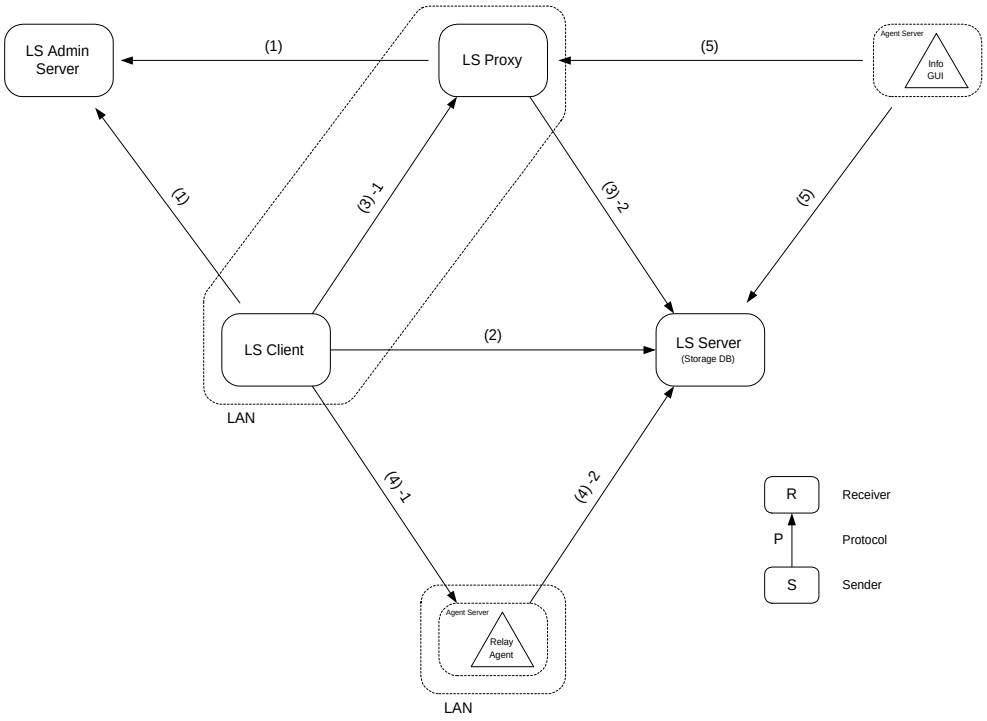


Abbildung 3.8: Die Schnittstellen zwischen den Komponenten des Lokationsdienstes

net ermöglicht. Im LAN wird UDP als verbindungsloses Transportprotokoll dagegen wieder interessant.

Bei der Schnittstellendefinition werden nun folgende Punkte betrachtet:

- Die Vertraulichkeit der zu übertragenen Informationen (Informationspolitik)
- Das Übertragungsmedium (Sicherheitsbereich)
- Das Kommunikationsaufkommen (niedrig/hoch/sehr hoch)
- Die für die Übertragung verwendeten Datentypen
- Das zu verwendende Protokoll

Die verschiedenen Anfragemethoden, die bei der Implementierung entweder als Nachrichtentyp des Protokolls bzw. als lokale Schnittstellenfunktion umzusetzen sind, werden wie folgt auf generische Weise in mathematischer Notation angegeben:

***Anfrage:*** $Parameter1 \times Parameter2 \rightarrow R\ddot{u}ckgabeWerte$

Erklärung

Bei einer lokalen Schnittstellenfunktion entspricht diese Syntax dem Aufruf der Methode `Anfrage` mit den Übergabeparametern `Parameter1` und `Parameter2`, die `RückgabeWerte`

zurück gibt. Wird die Anfragemethode als Nachricht eines Protokolls implementiert, so sollte die Nachricht `Anfrage` in diesem Fall die Daten `Parameter1` und `Parameter2` als Information enthalten. Es wird dann eine Nachricht mit der Information über die `RückgabeWerte` als der Antwort erwartet.

Anschließend an die Schnittstellendefinitionen folgen noch einmal alle verwendeten Datentypen mit Beschreibung im Überblick.

### (1) LS-Client / LS-Proxy → LS-AdminServer

Sofern sich durch die Umstrukturierung der LS-Server-Infrastruktur Veränderungen bei den Parametern der LS-Server ergeben haben, können die LS-Clients und LS-Proxy über diese Schnittstelle die aktuellen Parameter anfordern. Diese beim LS-AdminServer gespeicherten Informationen sollen jedem zugänglich sein und bedürfen deswegen keinerlei Schutzmechanismen, die im Protokoll zu berücksichtigen wären. Im Gegensatz zur Abfrage sollte die Veränderung dieser Daten allerdings nur autorisierten Personen erlaubt sein. Um diese Schnittstelle zu realisieren, eignet sich nun das *domain protocol* sehr gut (siehe auch Abschnitt 2.5.1).

***getHashPrefixLength:*** $\emptyset \rightarrow HashPrefixLength$

Anfrage der momentan relevanten Länge des Hashpräfixes für die Zuordnung zwischen Agent und zuständigem LS-Server.

***getServerIP:*** $HashPrefix \rightarrow ServerIP$

Anfrage der IP-Adresse des für den Hashpräfix verantwortlichen LS-Server.

***getServerPort:*** $HashPrefix \rightarrow ServerPort$

Anfrage des Ports des für den Hashpräfix verantwortlichen LS-Server.

***getServerPublicKey:*** $HashPrefix \rightarrow ServerPublicKey$

Anfrage des *public keys* des für den Hashpräfix verantwortlichen LS-Server.

| | |
|---|---|
| Informationspolitik<br>Übertragungsmedium<br>Kommunikationsaufkommen | öffentlich / jedem zugänglich<br>Internet (schwächster Vertrauensbereich)<br>niedrig |
| Verwendetes Protokoll | *Domain protocol* (siehe Abschnitt 2.5.1) |
| Verwendete Datentypen | `HashPrefix, HashPrefixLength, ServerIP, ServerPort, ServerPuplicKey` |

Tabelle 3.3: Kurzbeschreibung der Schnittstelle (1)

### (2) LS-Client → LS-Server

Die Schnittstelle zwischen LS-Client und LS-Server ist die wichtigste des Lokationsdienstes. Über diesen Kommunikationskanal werden sowohl die sensibelsten als auch die Daten mit der

höchsten Änderungsfrequenz transportiert. Aus diesem Grund wurde ein eigenes Protokoll, das *Location Service Protocol (LSP)*, entworfen, dass sowohl auf Bearbeitungsgeschwindigkeit bei Anfragen als auch auf Sicherheit hin optimiert wurde (siehe Abschnitt 3.2.3). Durch LSP werden sowohl alle Anfrage- als auch Antwort-Nachrichten definiert, die nötig sind, um die relevanten Daten des Lokationsdienstes in beide Kommunikationsrichtungen zu übertragen. Verschlüsselung dient in der sicheren Variante dieses Protokolls ($LSP_{secure}$) unter anderem dem Schutz vor Attacken auf die *privacy* bei dem Transfer der Daten durch den schwachen Vertrauensbereich des Internets. Allerdings ist es unter bestimmten Umständen, die in Schnittstellendefinition (3) besprochen werden, teilweise nicht möglich, die Daten zu verschlüsseln. Aus diesem Grund existiert auch eine Variante von LSP, in der die Nachrichten im Klartext versendet werden ($LSP_{plain}$).

***lookup:*** $ImplicitName \rightarrow ReplyState \times ContactAddress$

Anfrage der aktuellen Kontaktadresse des Agenten, der durch den gegebenen Agenten-Hashcode identifiziert wird.

***register:*** $ImplicitName \times ContactAddress \rightarrow ReplyState$

Änderungs-Anfrage (*init*, *update*, *delete*) für einen existierenden Eintrags im LS-Server.

***refresh:*** $ImplicitNameList \rightarrow ReplyState$

Bestätigung aller Einträge, dessen ImplicitName in der ImplicitNameList enthalten ist.

| | |
|---|---|
| Informationspolitik<br>Übertragungsmedium<br>Kommunikationsaufkommen | geheim<br>Internet (schwächster Vertrauensbereich)<br>hoch |
| Verwendetes Protokoll | Location Service Protocol ($LSP_{secure}$)<br>über eine Socketverbindung (siehe Abschnitt 3.2.3) |
| Verwendete Datentypen | `ImplicitName, ImplicitNameList, ContactAddress` |

Tabelle 3.4: Kurzbeschreibung der Schnittstelle (2)

Gegeben den Fall, dass sich LS-Client und anzusprechender LS-Server in der selben Laufzeitumgebung befinden, so beschränkt sich die Schnittstellendefinition auf einen Satz von Methoden, die lokal und damit im stärksten Vertrauensbereich aufgerufen werden. Hier ist es deswegen nicht notwendig, sich viele Gedanken um Geschwindigkeit zu machen, und die Sicherheit hängt vor Allem vom umgebenden Laufzeitsystem (im Speziellen dem Agentensystem) ab.

### (3) LS-Client → LS-Proxy → LS-Server

Bei diesem Kommunikationskanal von LS-Client zu LS-Server stellt der LS-Proxy eine Zwischeninstanz dar, welche die transportierten Daten sowohl passiv verarbeitet, als auch aktiv in die Kommunikation eingreift. Dadurch teilt sich der Kommunikationsweg in zwei Teilabschnitte (3)-1 und (3)-2 auf. Davon abgesehen, werden aber die gleichen Informationen

übermittelt, die auch bei der vorherigen Schnittstellendefinition (2) von Relevanz waren. Aufgrund der Struktur von LSP, die in Abschnitt 3.2.3 detailliert beschrieben wird, und der Tatsache, dass der LS-Proxy fähig sein muss, die empfangenen Daten zu interpretieren, ist es allerdings nötig, dass diese Daten im Abschnitt (3)-1 zwischen LS-Client und LS-Proxy im Klartext übertragen werden. Aus diesem Grund wird auch eine Version des Location Service Protcols definiert, die unverschlüsselte Nachrichten vorsieht ($LSP_{plain}$). Das Protokoll für den Abschnitt (3)-2 zwischen LS-Proxy und LS-Server, dessen Schnittstellendefinition mit der vorherigen übereinstimmt, ist dann wieder die sichere Variante von LSP ($LSP_{secure}$). Der LS-Proxy hat also auch die Aufgabe das Protokoll zu wechseln, falls Anfragen vom LS-Client auch an den LS-Server weitergegeben werden.

Auf dem Hintergrund, dass der Kommunikationskanal (3)-1 zum LS-Proxy hin komplett im LAN des LS-Clients enthalten ist, befinden wir uns nach Abbildung 3.3 noch im mittleren Vertrauensbereich. Will man die Komponente LS-Proxy beim Lokationsdienst verwendet, so muss man also jeden Teilnehmer und damit jeden Benutzer mit umfassendem Zugriff auf ein Rechnersystem innerhalb dieses LAN in Hinblick auf die Anwendung als vertrauenswürdig genug einstufen können, um $LSP_{plain}$ zu benutzen. Dies fällt in den begrenzten Ausdehnungen eines LAN-Subnetzes, dass physikalisch zumeist durch eine Firma oder sogar Abteilung eingegrenzt wird, allerdings noch um einiges leichter als im Internet mit seiner Anonymität. Die Eingrenzung des genannten Kommunikationskanals auf das umgebende LAN-Subnetz wird übrigens von Seiten des LS-Proxy dadurch gewährleistet, dass die IP-Absendeadresse von Anfragen vor deren Bearbeitung durch die Subnetzmaske (*subnet mask*) mit dem aktuellen Subnetz verglichen wird.

Vor der folgenden Beschreibung der Schnittstelle (3)-1 zwischen LS-Client und LS-Proxy ist außerdem zu bemerken, dass durch die Sonderstellung des LS-Proxy zum LSP die bis jetzt noch nicht erwähnte Funktion *invalidate* hinzukommt. Die Schnittstelle (3)-2 ist mit der Schnittstelle (2) aus dem letzten Abschnitt identisch.

***lookup:*** $ImplicitName \rightarrow ReplyState \times ContactAddress$

Anfrage der aktuellen Kontaktadresse des Agenten, der durch den gegebenen Agenten-Hashcode identifiziert wird.

***register:*** $ImplicitName \times ContactAddress \rightarrow ReplyState$

Änderungs-Anfrage (*init*, *update*, *delete*) für einen existierenden Eintrags im LS-Server.

***invalidate:*** $ImplicitName \rightarrow ReplyState$

Entfernt einen Entrag aus der Datenbank des LS-Proxy, der nicht mehr aktuell ist.

***refresh:*** $ImplicitNameList \rightarrow ReplyState$

Bestätigung aller Einträge, dessen ImplicitName in der ImplicitNameList enthalten ist.

Vergleichbar mit der Schnittstelle (2) ist in diesem Fall die Möglichkeit, dass LS-Client und LS-Proxy in der selben Laufzeitumgebung gestartet werden. Für diesen Fall existiert ebenfalls ein Satz von äquivalenten Methoden die lokal aufgerufen werden können.

| | |
|---|---|
| Informationspolitik<br>Übertragungsmedium<br>Kommunikationsaufkommen | geheim<br>LAN (mittlerer Vertrauensbereich)<br>sehr hoch |
| Verwendetes Protokoll | Location Service Protocol ($LSP_{plain}$)<br>über eine SocketVerbindung (siehe Abschnitt 3.2.3) |
| Verwendete Datentypen | `ImplicitName, ImplicitNameList, ContactAddress` |

Tabelle 3.5: Kurzbeschreibung der Schnittstelle (3)-1 (zur Datenübertragung)

Neben der beschriebenen Schnittstelle zum Datenaustausch während des Betriebs des Lokationsdienstes existiert zwischen LS-Client und LS-Proxy noch eine zweite Schnittstelle, die der Initialisierung dient. Wie in Abschnitt 3.2.1 unter LS-Proxy bereits angesprochen, soll LS-Clients durch *broadcast* Nachrichten innerhalb des LAN ermöglicht werden, einen existierenden LS-Proxy zu lokalisieren und in die Lokationsdienst-Struktur einzubinden. Darüber hinaus soll auf dem gleichen Weg garantiert werden, dass pro Subnetz nur ein LS-Proxy aktiv ist. Das hierfür nötige Protokoll (*Location Service Proxy Discover Protocol (LSPDP)*) wird hier beschrieben:

***discover:*** $\emptyset \rightarrow \emptyset$

*broadcast* Nachricht im Subnetz zum Ermitteln eines vorhandenen LS-Proxy.

***offer:*** $ProxyIP \rightarrow \emptyset$

Antwortnachricht mit IP-Adresse auf Discover(), mit der sich ein Proxy zu erkennen gibt.

| | |
|---|---|
| Informationspolitik<br>Übertragungsmedium<br>Kommunikationsaufkommen | öffentlich<br>LAN (mittlerer Vertrauensbereich)<br>niedrig |
| Verwendetes Protokoll | Location Service Proxy Discover Protocol (LSPDP)<br>über einen UDP-Multicastport |
| Verwendete Datentypen | `ProxyIP` |

Tabelle 3.6: Kurzbeschreibung der Schnittstelle (3)-1 (zur Initialisierung)

### (4) LS-Client → LS-RelayAgent → LS-Server

In diesem Fall stellt der LS-RelayAgent bei dem Kommunikationskanal von LS-Client zu LS-Server eine Zwischeninstanz dar. Im Gegensatz zu Schnittstelle (3) ist diese Komponente im Bezug auf die transportierten Informationen allerdings ausschließlich passiv. Die einzige Aufgabe des LS-RelayAgents ist die Weiterleitung von Nachrichten des LSP an den entsprechenden LS-Server und dabei die Verschleierung der eigentlichen Absenderadresse.

Um dies zu realisieren, könnte man LSP-Nachrichten zusammen mit der Zieladresse des LS-Server noch einmal kapseln, ohne den eigentlichen Inhalt der LSP-Nachricht zu verändern, und dann an den LS-RelayAgenten senden. Der LS-RelayAgent hat nun die Aufgabe, den

empfangenen Daten die ursprüngliche Adresse des LS-Server zu entnehmen, und ohne Betrachtung des Inhalts, die LSP-Nachricht zu identifizieren und an diese Adresse weiterzuleiten.

#### (5) LS-InfoGUI → LS-Proxy / LS-Server

Zusätzlich zu den Schnittstellenmethoden, die der LS-Server einem lokal installierten LS-Client bereits zur Verfügung stellt, sollte auf einer gesonderten Sicherheitsstufe eine weitere Methode definiert sein, die bei Übergabe eines Zeitstempels alle Einträge in einer Liste zurück gibt, die nach diesem Zeitpunkt aktualisiert wurden. Die gleiche Schnittstelle muss auch ein LS-Proxy dem LS-InfoGUI zur Verfügung stellen.

***lookup:*** $ImplicitName \rightarrow ReplyState \times ContactAddress$

Anfrage der aktuellen Kontaktadresse des Agenten, der durch den gegebenen Agenten-Hashcode identifiziert wird.

***register:*** $ImplicitName \times ContactAddress \rightarrow ReplyState$

Änderungs-Anfrage (*init*, *update*, *delete*) für einen existierenden Eintrags im LS-Server.

***list:*** $Timestamp \rightarrow ReplyState \times EntryList$

Anfrage aller Einträge, die seit dem angegebenen Zeitpunkt verändert wurden.

***refresh:*** $ImplicitNameList \rightarrow ReplyState$

Bestätigung aller Einträge, dessen ImplicitName in der ImplicitNameList enthalten ist.

| | |
|---|---|
| Informationspolitik<br>Übertragungsmedium<br>Kommunikationsaufkommen | geheim<br>Agentensystem (stärkster Vertrauensbereich)<br>niedrig |
| Verwendetes Protokoll | lokale Methodenaufrufe<br>(lokal innerhalb der Ausführungsumgebung) |
| Verwendete Datentypen | `ImplicitName, ImplicitNameList,`<br>`ContactAddress, Timestamp` |

Tabelle 3.7: Kurzbeschreibung der Schnittstelle (5)

#### Verwendete Datentypen und deren Bedeutung

### 3.2.3 Entwicklung eines sicheren Protokolls

Nach den Schnittstellendefinitionen, wird hier nun mit dem Location Service Protcol (LSP) das Protokoll entwickelt, dass neben den eigentlichen Komponenten maßgeblich für die Funktion des Lokationsdienstes verantwortlich ist. Es wird in Abbildung 3.8 an den Punkten (2), (3) und (4) und als Äquivalent in Form von lokalen Schnittstellenmethoden auch an Punkt

| Datentyp | Bedeutung |
|---|---|
| `ContactAddress` | Kontaktadresse (URL des Agentenservers mit Port), die im Zusammenhang mit dem ImplicitName dazu verwendet werden kann, mit dem dadurch identifizierten Agenten zu kommunizieren. |
| `ImplicitName` | Hashcode (*bit string*) des statischen Teil eines Agenten, der als eindeutiger Bezeichner für diesen verwendet wird bzw. einen Eintrag im LS-Server oder LS-Proxy identifiziert. |
| `ImplicitNameList` | Eine Liste von ImplicitNames, mit der gleich mehrere Agenten bzw. Einträge im LS-Server oder LS-Proxy identifiziert werden können. |
| `EntryList` | Eine Liste von (`ImplicitName`,`ContactAddress`)-Tupeln, die Einträge im LS-Server oder LS-Proxy repräsentieren. |
| `HashPrefix` | Der Anteil des ImplicitNames, der für die Zuordnung zum verantwortlichen LS-Server verwendet wird. |
| `HashPrefixLength` | Länge des Hashpräfixes. |
| `ProxyIP` | IP-Adresse eines LS-Proxy. |
| `ReplyState` | Rückgabewert, der den Ausführungsstatus einer Protokollmethode. |
| `ServerIP` | IP-Adresse eines LS-Server. |
| `ServerPort` | Vom LS-Server unterstützer Port für das LSP. |
| `ServerPuplicKey` | *Public key* eines LS-Server, der zur Verschlüsselung beim $LSP_{secure}$ verwendet wird. |
| `Timestamp` | Zeitstempel. |

Tabelle 3.8: Für die Schnittstellendefinition benötigte Datentypen und ihre Bedeutungen

(5) eingesetzt. Durch dieses Protokoll sollen sowohl die grundlegenden *lookup* und *register* Anfragen des Lokationsdienstes für den Zugriff auf die bei LS-Proxy und LS-Server gespeicherten Einträge abgedeckt werden, als auch die bereits weiter oben angesprochenen, erweiterten Anfragen zur Kommunikation mit LS-Proxy bzw. vom LS-InfoGUI aus. Hier sind noch einmal alle bereits in Abschnitt 3.2.2 definierten Anfragen, die für das LSP relevant sind, im Überblick:

***lookup:*** $ImplicitName \rightarrow ReplyState \times ContactAddress$

***register:*** $ImplicitName \times ContactAddress \rightarrow ReplyState$

***invalidate:*** $ImplicitName \rightarrow ReplyState$

***refresh:*** $ImplicitNameList \rightarrow ReplyState$

***list:*** $Timestamp \rightarrow ReplyState \times Entrylist$

Bis auf `register`, werden alle Anfragen einfach durch einen entsprechende Anfrage-Nachrichtentyp zur Übermittlung der Parameter und einen Antwort-Nachrichtentyp zur Rückgabe der Antwort realisiert (siehe Anhang C.1).

Das Hauptaugenmerk liegt allerdings auf der Entwicklung eines in sich stimmigen und sicheren Protokolls für die register Anfragen, da nur durch sie die eigentlichen Einträge in den Datenbanken der LS-Server bzw. LS-Proxy verändert werden können. Begonnen wird nun

mit der Beschreibung von $LSP_{plain}$, bei dem alle Nachrichten zunächst im Klartext übertragen werden.

#### $\mathbf{LSP_{plain}}$

Dieses Protokoll basiert auf der Idee, dass nur autorisierte Instanzen fähig sein sollen, bestehende Einträge beim LS-Server zu verändern. Kommen wir also noch einmal auf das grundlegende Szenario von einem Agenten zurück, der von Agentenserver zu Agentenserver migriert und dessen aktuelle Position einem Lokationsserver übermittelt werden soll. Der Agentenserver, auf dem der Agent zur Ausführung kommt, besitzt durch diesen auch seinen Hashcode, und ist dadurch in der Lage eine *register* Anfrage stellen zu können. Der erste Eintrag bei einem Lokationsserver wird durch den Agentenserver generiert, auf dem der Agenten das erste Mal zur Ausführung kommt. Anschließend sollte nur der Server in der Lage sein, den entsprechenden Eintrag zu verändern bzw. diesen bei Terminierung des Agenten zu löschen, auf dem sich der Agent gerade befindet.

Angelehnt an das Modell aus Abschnitt 2.8 wird diese Idee nun dadurch realisiert, dass bei jeder *register* Anfrage an den Lokationsserver neben dem Agenten-Hashcode und der Agenten-Kontaktadresse auch zwei *bit strings*, mit übertragen werden. Der erste *bit string* wird mit einem beim Lokationsserver gespeichertem Wert verglichen, um die Anfrage zu autorisieren, der zweite stellt den gültigen Vergleichswert für die nächste *register* Anfrage dar, der jedes mal vom Anfragesteller zufällig generiert wird (das sogenannte *Cookie*). Im Detail sieht der Ablauf folgendermaßen aus:

- Da vor der ersten *register* Anfrage beim Lokationsserver noch kein Eintrag für den betreffenden Agenten existiert, ist auch noch kein Vergleichswert zur Autorisation vorhanden. Der Agentenserver überträgt ein beliebiges, *aktuelles* Cookie, generiert aber ein für die nächste Anfrage gültiges Cookie, das der Lokationsserver mit dem Eintrag zu diesem Agenten in seiner Datenbank speichert.
- Zusätzlich wird dieses Cookie im veränderlichen Teil des Agenten gespeichert und auf diese Weise als Autorisationsschlüssel mit dem Agenten an den nächsten Agentenserver weitergegeben.
- Bei allen folgenden *register* Anfragen, wird das aktuelle Cookie zur Autorisation aus dem veränderlichen Teil des Agenten entnommen. Außerdem wird ein neues Cookie generiert, das durch die Anfrage sowohl dem Lokationsserver mitgegeben wird, als auch das im Agenten gespeicherte Cookie für die nächste Anfrage ersetzt.
- Erhält der Lokationsserver eine Anfrage mit einem ungültigem Cookie, so lehnt er diese Anfrage durch eine entsprechende Fehlermeldung ab.

Wird für die Cookies bei der Implementierung ein Datentyp mit großer Anzahl von Bits gewählt, so ist es fast unmöglich den Cookie zu erraten. Um vor Allem das Vorgehen im folgenden $LSP_{secure}$ zu vereinfachen, sollte die Länge des Cookies in Bits der Länge des Ergebnisses der kryptographischen Hashfunktion entsprechen, die sowohl für die Berechnung des Agenten-Hashcodes (siehe Abbildung 3.2) als auch innerhalb $LSP_{secure}$ verwendet wird.

Ein geeigneter Kandidat wäre der *Secure Hash Algorithm (SHA)* [82], mit dem die Anzahl der Bits auf 160 festgelegt würde: Dadurch ergäbe sich die Wahrscheinlichkeit ein Cookie zu erraten zu $\frac{1}{2^{160}} \approx 10^{-48}$.

Dadurch wird nun gewährleistet, dass nur Anfragen der Instanzen autorisiert werden, die mit dem Agenten auch das aktuelle Cookie in ihrem Besitz haben. Die veränderte *register* Anfrage sieht also folgendermaßen aus:

***register:***

$$ImplicitName \times ContactAddress \times NewCookie \times CurrentCookie \rightarrow ReplyState$$

Wie bereits in Abschnitt 3.2.1 erwähnt, soll es durch dieses Protokoll sowohl möglich sein einen neuen Eintrag zu erstellen, als auch einen bestehenden Eintrag zu verändern bzw. zu löschen. Dies kann nun durch die geeignete Wahl der Parameter wie folgt geschehen, ohne den eigentlichen Nachrichtentyp für die Anfrage verändern zu müssen.

**init:** register(ImplicitName, ContactAddress, NewCookie, `null`)

**update:** register(ImplicitName, ContactAddress, NewCookie, CurrentCookie)

**delete:** register(ImplicitName, `null`, `null`, CurrentCookie)

Damit der Lokationsserver diese Anfragen eindeutig unterscheiden kann, darf der Wert `null` nicht im normalen Definitionsbereich der entsprechenden Parametertypen vorkommen.

#### $\mathbf{LSP_{secure}}$

Durch das oben beschriebene Protokoll ist es also niemandem ohne Kenntnis des aktuellen Cookies möglich, einen Eintrag beim Lokationsserver zu verändern. Da die *register* Anfragen allerdings im Klartext über das Internet übertragen werden, ist es durchaus denkbar, dass ein Angreifer über dessen Rechnersystem die Anfrage-Nachricht beim Transport geleitet wird, in den Besitz eines solchen Cookies kommt. Er wäre dadurch in der Lage, die nächste Anfrage stellen zu können und durch die Angabe eines eigenen, neuen Cookies gleichzeitig jedem anderen diese Möglichkeit zu entziehen. Dieser Angriff fällt allerdings sofort auf. Es besteht aber auch die Möglichkeit, durch Manipulation der Kontaktadresse in der Anfrage-Nachricht ohne den Austausch des Cookies, unbemerkt einen falschen Eintrag im Lokationsserver registrieren zu lassen.

Ein sicheres Protokolls muss also das Cookies während des Transports verbergen und dem Lokationsserver darüber hinaus die Möglichkeit bieten, die Integrität der transportierten Informationen zu überprüfen.

Das folgende Protokoll erfüllt nun genau diese beiden Anforderungen und ist darüber hinaus immer noch recht leistungsstark. Es ist außerdem nicht erforderlich, neben dem aktuellen Cookie ein zusätzliches, gemeinsames Geheimnis zwischen Anfragesteller und Lokationsserver einzuführen:

**(1)** Wei beim $LSP_{plain}$ generiert der Anfragesteller ein neues Cookie.

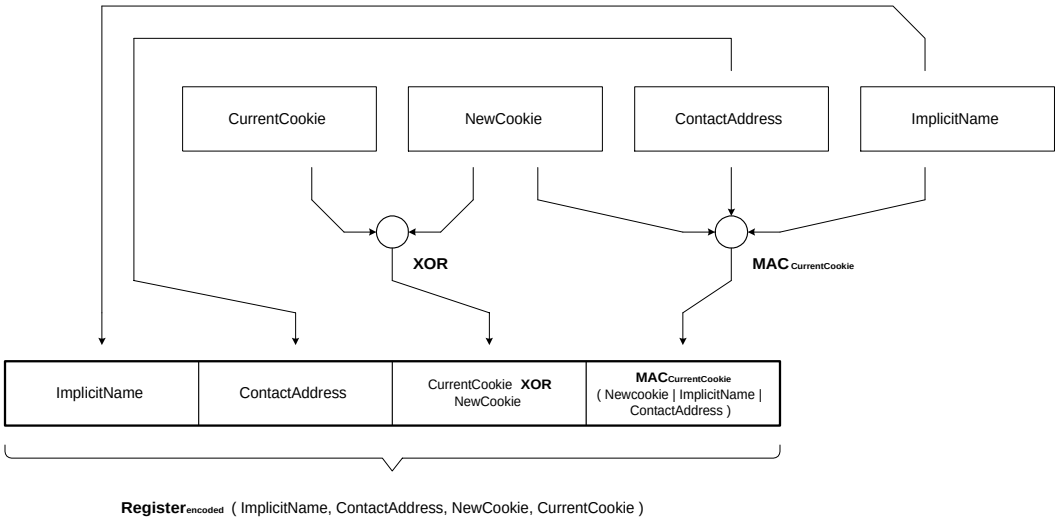


Abbildung 3.9: Format der kodierten *register* Anfrage in $LSP_{secure}$

**(2)** Das neue Cookie wird durch `XOR` ($\oplus$) mit dem aktuellen Cookie verknüpft.

**(2)** Durch einen *Message Authentification Code (MAC)* und dem aktuellen Cookie als Initialisierungsparameter wird aus dem Agenten-Hashcode, der Kontaktadresse des Agenten und dem neuen Cookie eine Art Prüfsumme der Anfrage erzeugt.

**(3)** Der Anfragesteller stellt die Anfragenachricht nun wie in Abbildung 3.9 in dieser Reihenfolge aus Agenten-Hashcode, Agenten-Kontaktadresse, dem Ergebnis der XOR-Verknüpfung und der MAC Prüfsumme zusammen, und sendet diese an den entsprechenden Lokationsserver.

**(4)** Der Lokationsserver ermittelt wie beim $LSP_{plain}$ durch den im Klartext übertragenen Agenten-Hashcode den entsprechenden Eintrag in der Datenbank und liest den zugeordneten aktuellen Cookie aus. Falls er keinen solchen Eintrag findet, lehnt er die Anfrage mit einer entsprechenden Fehlermeldung ab.

**(5)** Mit diesem Cookie extrahiert er durch eine wiederholte XOR-Verknüpfung das neue Cookie.

**(6)** Er besitzt nun alle Informationen, um den MAC selber zu berechnen und sein Ergebnis mit der übermittelten Prüfsumme zu vergleichen. Stimmen diese beiden Werte überein, so bearbeitet er Anfrage ordnungsgemäß. Andernfalls erhält der Anfragesteller eine Antwortnachricht mit entsprechende Fehlerstatus.

Die XOR-Verknüpfung erscheint bei der Verwendung zum Verschlüsseln im Kontext der Kryptographie als Schwachstelle. Doch selbst wenn ein potentieller Angreifer mehrere aufeinanderfolgende Anfragen bezüglich ein und desselben Agenten an den zugeordneten Lokationsserver abfängt, ist es ihm nicht möglich das jeweils aktuelle oder neue Cookie zu ermitteln.

Und ohne das Wissen über das aktuelle Cookie ist er auch nicht in der Lage den „passenden" MAC zu berechnen, falls er einen oder mehrere der vorangehenden Teile der Nachricht manipuliert hat.

Dieses Protokoll hat allerdings noch eine Schwachstelle: Bei der initialen *register* Anfrage (*init*) besteht noch kein gemeinsames Geheimnis zwischen Anfragesteller und Lokationsserver. Aus diesem Grund müsste für die XOR-Verknüpfung und die Berechnung des MAC in diesem Fall ein fest definiertes Cookie verwenden werden. Fängt ein Angreifer allerdings diese erste Anfrage-Nachricht an den Lokationsserver ab, so hat er, wie der Lokationsserver auch, die Möglichkeit, folgende Anfragen zu entschlüsseln. Die erste Anfrage an den Lokationsserver muss also auf eine besondere Weise verschlüsselt werden.

Hierfür eignet sich nun asymmetrische Verschlüsselung. Jedem Lokationsserver wird ein *public key / private key* Paar zugeordnet, wobei der *public key* veröffentlicht wird. Der Anfragesteller kann nun bei der ersten Anfrage unter Verwendung des *public key* des entsprechenden Lokationsservers die Nachricht verschlüsseln. Nur dieser Lokationsserver ist dann in der Lage, die Nachricht zu entschlüsseln und den ersten gültigen Cookie zu entnehmen, der dann wie oben beschrieben bei der nächsten Verschlüsselung Verwendung findet. Das Format dieser *register* Anfrage sieht nun folgendermaßen aus:

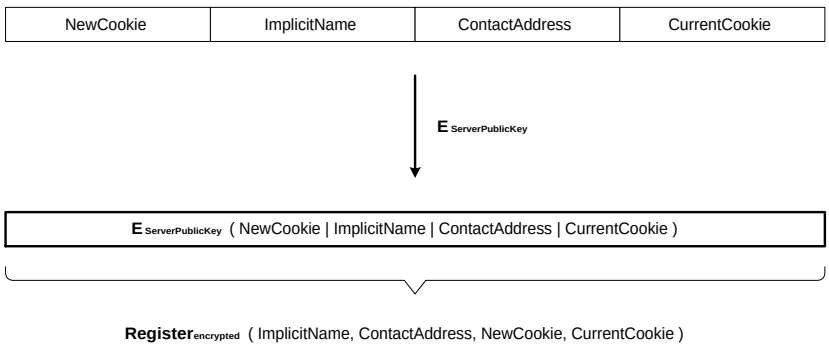


Abbildung 3.10: Format der verschlüsselten *register* Anfrage in $LSP_{secure}$

Nach PKCS#7 stellt $E_{ServerPublicKey}$ in Abbildung 3.10 eine hybride Verschlüsselungsfunktion dar [55] [82]. Um einer sogenannten *known plaintext attack* vorzubeugen, wird beim Verschlüsseln der Modus *Cypher Block Chaining (CBC)* verwendet und das jedes Mal zufällig generierte neue Cookie dem erratbaren Agenten-Hashcode bzw. der Agenten-Kontaktadresse vorangestellt.

Nun wird auch verständlich, warum für die in Abschnitt 3.2.2 beschriebene Schnittstelle (3)-1 zwischen LS-Client und LS-Proxy nicht $LSP_{secure}$ verwendet wird. Zum einen kann der LS-Proxy die *init* Anfrage nicht entschlüsseln, da er den geheimen Schlüssel des entsprechenden LS-Server nicht kennt. Zum anderen könnte er auch folgende *update* bzw. *delete* Anfragen nur dann entschlüsseln, wenn er jede Positionsänderung des Agenten und die damit verbundene Anfrage registrieren würde, und dadurch im Besitz des aktuellen Cookies wäre. Das ist allerdings bei dem Konzept des LS-Proxy nicht unbedingt vorgesehen. Die Anfrage

des LS-Clients an den LS-Proxy geschieht also vorerst durch $LSP_{plain}$. Der LS-Proxy ist daraufhin im Besitz aller nötigen Informationen, um den weiteren Transport der Anfrage an den LS-Server nun mittels $LSP_{secure}$ zu sichern.

Das LSP setzt übrigens nicht voraus, dass alle *register* Anfragen an den LS-Server (Schnittstellen (2), (3)-2 und (4)) mittels $LSP_{secure}$ gestellt werden. Unterstützt ein LS-Client bzw. LS-Server keine Verschlüsselung, so kann durchaus $LSP_{plain}$ genutzt werden. Wird allerdings nur eine Anfrage nicht verschlüsselt, so ist die durch das $LSP_{secure}$ „erkaufte" Sicherheit verloren.

Ebenfalls aus Sicherheitsgründen sollte vom LS-Server übrigens getestet werden, ob eine bei *update* Anfragen übergebene Kontaktadresse tatsächlich existiert. Dies könnte durch einen *lookup* über DNS erreicht werden, der die dazugehörige IP-Adresse ermitteln. Ein *reverser* Loopup dieser IP-Adresse (ebenfalls über DNS) sollte dann mit der Kontaktadresse übereinstimmen. Andernfalls wird die *update* Anfrage abgelehnt.

### Umgang mit Cookies von Seiten des Agentensystems

Sowohl bei $LSP_{plain}$ als auch bei $LSP_{secure}$ ist der Besitz des aktuellen Cookies ist also zwingend erforderlich, um eine *register* Anfrage stellen zu können. Wird die Sequenzkette der *register* Anfragen unterbrochen bzw. geht das aktuelle Cookie auf andere Weise verloren, so hat man keine Möglichkeit mehr den entsprechenden Eintrag beim LS-Server zu verändern bzw. zu löschen. Aus diesem Grund sollte man sorgsam mit diesen *bit strings* umgehen.

Eine mögliche Gefahrenquelle stellt eine Fehlfunktion während der *register* Anfrage dar: Zum einen könnte der Agentenserver durch eine Fehlfunktion den Dienst versagen, was einen Neustart des Server und Informationsverlust aller im Speicher befindlichen Daten zur Konsequenz hätte. Zum anderen wäre es möglich, dass während dem Transport der Anfrage-Nachricht ein Problem auftaucht und diese den Lokationsserver deswegen nie erreicht. Um diesen Gefahren vorzubeugen, die einen Cookieverlust zur Folge haben könnten, sollte auf Seite des Agentenservers bei *register* Anfragen folgendes Schema eingehalten werden:

**(1)** Generierung eines neuen Cookies und Sicherung in einer Datei

**(2)** *Register* Anfrage mit aktuellen Cookie aus dem Agenten und dem neu generierten

**(3)** Die Antwort-Nachricht vom Lokationsserver abwarten

**(4)** Überschreiben des Cookies im Agent mit dem neu generierten

**(5)** Löschen des Cookies aus der Datei

Falls durch eine Fehlfunktion ein Neustart des Agentenservers notwendig wurde, wird nun versucht, alle Agenten mit den Cookies aus ihrem veränderlichem Teil erneut anzumelden. Sofern der Lokationsserver die letzte Anfrage nicht erhalten hat (bei einer Unterbrechung vor Schritt (3)), müsste diese Anfrage erfolgreich sein. Wenn dies nicht der Fall ist, wird die *register* Anfrage mit dem Cookie aus der Sicherungsdatei wiederholt. Damit der Agentenserver auf diese und andere Situationen adäquat reagieren kann, müssen im LSP allerdings auch differenzierte Antwort-Nachrichten vorgesehen sein (siehe Definition von LSP in Anhang C.1).

Ein ähnliches Problem entsteht, wenn Agenten bei der Migration verloren gehen bzw. es dem Zielserver nicht möglich ist sie zu installieren. In diesem Fall würden *verwaiste* Einträge bei den Lokationsservern entstehen, die nicht gelöscht werden könnten, da das aktuelle Cookie fehlt. Auch aus diesem Grund sollte bei LS-Proxy und LS-Server, neben den bereits in Abschnitt 3.2.1 genannten Gründen, ein *timeout* für gespeicherte Einträge vorgesehen werden.

### 3.2.4 Angriffsmöglichkeiten spezieller Instanzen im System

In diesem Abschnitt werden die Gefahren abgeschätzt und analysiert, die von einzelnen Komponenten der Lokationsdienstes ausgehen könnten, falls sich diese im Besitz eines Angreifers befinden.

#### Agentenserver als Angreifer

- Der Angreifer, der im Besitz eines Agentenservers ist, kann Agenten absichtlich sterben lassen. Dies ist immer möglich und nicht zu vermeiden. Unabhängig davon, ob der Angreifer die Kontaktadresse zu dem Agenten beim entsprechenden Lokationsserver nun ordnungsgemäß löscht oder nicht, lässt sich durch die *logfiles* des LS-Server zumindest aber herausfinden, welche Route der Agent bis dahin genommen hat. Steht der Besitzer des Agenten darüber hinaus in regelmäßigem Kontakt, so kann der bösartige Agentenserver noch leichter identifiziert und in Zukunft gemieden werden.
- Speichert der Agentenserver einen ungültigen Cookie im veränderlichen Teil des Agenten, so ist kein Zielserver auf der zukünftigen Route des Agenten mehr in der Lage, seine Position beim Lokationsdienst zu aktualisieren. Dies wird allerdings sofort bemerkt. Falls der Agent für Problemfälle z.B. die Daten zur Benachrichtigung seines Besitzers per Email mit sich führt, kann dieser umgehend informiert werden, und durch die *logfiles* des LS-Server wiederum versucht werden den bösartige Agentenserver zu identifizieren.

Gerade in diesen beiden Fällen ist zu erkennen, welchen Nutzen das Führen von *logfiles* auf den LS-Servern hat. Allerdings ist es für einen Agentenserver immer noch möglich durch eine *register* Anfrage eine Kontaktadresse anzugeben, die nicht mit der eigenen übereinstimmt. Bisher ist im Modell des Lokationsdienstes nur vorgesehen, die angegebene Kontaktadresse auf ihre physikalisch Existenz hin zu prüfen (siehe Abschnitt 3.2.3). Das heißt der Angreifer könnte in den oben angesprochenen Szenarien die Spuren in den *logfiles* dadurch verwischen, dass er den Pfad des Agenten durch „verfälschte" Anfragen um einige Agentenserver erweitert.

Aus diesem Grund sollte in einer zukünftigen Version des Lokationsdienstes und besonders des LSPs vorgesehen werden, die bei eine *register* Anfrage übermittelte Position eines Agenten bestätigen zu lassen, ndem z. B. über die Kontaktadresse angefragt wird, ob sich der entsprechende Agent tatsächlich auf dem angegebenen Agentenserver befindet.

**Lokationsserver als Angreifer**

- Der Refresh-Mechanismus stellt kein großes Sicherheitsrisiko dar. Ein möglicher Angreifer wäre durch diesem Mechanismus nur in der Lage Eintragungen in der Datenbank des Servers zu halten, bis eine reguläre *delete* Anfrage gestellt wird. Er würde dem Besitzer des Agenten dadurch eher einen Gefallen tun, als ihm zu schaden.

- Die Positionsangaben eines Agenten werden in Verbindung mit dem Hashcode seinen ganzen Lebenzyklus über bei dem selben LS-Server veröffentlicht. Ein Angreifer im Besitz dieses Lokationsservers ist also in der Lage den Pfad des Agenten durch das Netzwerk zu verfolgen. Darüber hinaus erhält er aber keinerlei Informationen. Der Besitzer eines Agenten kann seine eigene Position verschleiern, indem er die letzte *register* Anfrage von seinem Server aus (*delete* ohne Angabe einer Kontaktadresse) über einen LS-RelayAgent zum LS-Server sendet (siehe Abschnitt 3.2.1). In diesem Fall ist es dem Angreifer nicht mehr möglich, beobachtete Pfade einer Benutzeradresse zuzuordnen. Und Hashcodes bzw. Cookies sagen ebenfalls nichts über den Agenten, seine Aufgabe oder den Besitzer aus. Auf diese Weise wird einer Privacy-Attacke erst gar keine Chance gegeben.

- Falls vom Lokationsserver Daten durch *register* Anfragen angenommen werden, diese bei *lookup* Anfragen aber nicht bzw. nur in verfälschter Form wieder zurückgegeben werden, kann man von DoS sprechen. Die Richtigkeit der Daten kann allerdings leicht überprüft werden und dann ist der Angreifer sofort identifiziert.

**Transporthost im Internet als Angreifer**

- Beim Transport von *register* Anfragen im Klartext, kann jeder Host, über den Die Anfrage beim Transport geleitet wird, mitlesen bzw. die Anfrage sogar manipulieren. Durch den Austausch des aktuellen Cookies, ist er in der Lage weitere, eigentlich reguläre, Anfragen zu verhindern (DoS Attacke). Aus diesem Grund wurde $LSP_{secure}$ entwickelt, das den geschilderten Angriff vereitelt.

**Außenstehender Angreifer**

- Sofern der Agentenbesitzer den Hashcode seines Agenten nicht veröffentlicht, hat ein außenstehender Angreifer ohne den Agenten sehr schlechte Chancen, den richtigen Hashcode zu erraten. Außerdem könnte er selbst mit diesem Wissen ohne den aktuellen Cookie Einträge weder ändern noch löschen.

- Einem Angreifer wäre es allerdings möglich, eine Reihe von zufällig erzeugten Hashcodes neu zu registrieren (*init*), falls noch keine Einträge existieren, und dadurch zum einen die Datenbank eines LS-Server zu füllen und zum anderen Einträge mit diesem Hashcode für andere zu sperren. Werden diese Einträge allerdings nicht in regelmäßigen Abständen bestätigt, so entfernt sie der LS-Server nach Ablauf einer gewissen Zeit wieder aus der Datenbank. Außerdem haben Benutzer von Agenten die Möglichkeit durch Veränderungen am statischen Teil des Agenten (Hinzufügen von zufälligen Bytefolgen) den Hashcode gegebenenfalls noch zu verändern, falls erkannt wird, dass der für den

ursprünglichen Hashcode zuständige Lokationsserver momentan nicht erreichbar bzw. der Hashcode bereits verwendet wird.

- Es sollte nicht verschwiegen werden, dass jeder öffentlich zugängliche Dienst im Internet, auf dem Daten gespeichert werden können, als geheimer Nachrichtendienst missbraucht werden kann.

### 3.2.5 Mögliche Fehlerquellen durch Ausfälle von Komponenten

**Ausfall des Agentenservers**

- Werden die, durch den Ausfall in ihrer Ausführung unterbrochenen, Agenten bei dem Neustart des Agentenservers nicht wieder gestartet, d.h. sind sie „unfreiwillig" terminiert worden, so besteht keine Möglichkeit mehr den Positionseintrag bei einem existierendem LS-Proxy bzw. dem entsprechendem LS-Server zu löschen. Diese Einträge werden allerdings nach gewisser Zeit automatisch durch den *timeout* Mechanismus gelöscht.
- Unterstützt der Agentenserver den Neustart in ihrer Ausführung unterbrochener Agenten, so wird nach dem Neustart der Agentenplattform zuerst versucht die Positionseinträge aller aktiven Agenten mit einer *refresh* Anfrage zu aktualisieren. Das benötigte Cookie wird dem Agenten bzw. der Sicherungsdatei (siehe Abschnitt 3.2.3) entnommen. Ist der Eintrag durch ein *timeout* bereits gelöscht worden, so kann mittels eines neu generierten Cookies eine neue *init* Anfrage gestellt werden.
- Der Cache der die Informationen über die LS-Server enthielt, muss nach einem Neustart des LS-Client ebenfalls wieder durch die Initialisierungsdatei aktualisiert werden bzw. nach und nach über den LS-AdminServer angefragt werden.

**Ausfall des LS-Proxy**

- Der Ausfall eines LS-Proxy geht einher mit dem Verlust aller dort gespeicherten Einträge. Dies ist allerdings nicht sonderlich tragisch, sondern vergleichbar mit einem kalten Cache, der langsam wieder gefüllt werden muss um seine Effizienz wieder zu erlangen.
- Falls sich in einem Netzwerk mehrere potentielle LS-Proxy befinden, kann durch *voting* dynamisch sofort ein Ersatz ermittelt werden, andernfalls muss auf die Vorteile des LS-Proxy solange verzichtet werden bis er wieder einsatzfähig ist, und Anfragen des LSP werden vorerst wieder direkt an die LS-Server gestellt. Die Funktion des Lokationsdienstes wird dadurch aber nicht grundlegend behindert.

**Ausfall des LS-Server**

- Der Ausfall eines LS-Server geht wie beim LS-Proxy einher mit dem Verlust aller dort gespeicherten Einträge. In diesem Fall hat das allerdings weitergehende Folgen. Ist ein LS-Proxy im Subnetz installiert, in dem sich der Agent gerade bewegt, so können dort Anfragen immer noch beantwortet werden. Anfragen von einem Agentensystem außerhalb dieses Subnetzes können allerdings nicht beantwortet werden.

- In diesem Fall sollte so schnell wie möglich versucht werden, den betreffenden LS-Server wieder einsatzfähig zu bekommen bzw. einen Ersatzserver zu installieren. Wird ein Ersatzserver mit einer neuen URL gestartet so müssen alle LS-Client und LS-Proxy Module neu konfiguriert werden: Wird ein LS-AdminServer verwendet, so ist selbst dies kein großes Problem; finden Initialisierungsdateien Verwendung, so wird die Neukonfiguration weit aufwendiger.

- Nachdem ein LS-Server für den entsprechenden Agenten-Hashcode wieder aktiv ist, können die Einträge bei der nächsten *update* Anfrage einfach wieder neu initialisiert werden und enthalten dann auch das aktuelle Cookie. Im Falle eines statischen Agenten, muss der LS-Client den Fehlerfall beim der nächsten *refresh* Anfrage erkennen und von sich aus den Eintrag erneut durch eine *init* Anfrage initialisieren.

# Kapitel 4

# Prototypimplementierung

In diesem Kapitel wird nun die Umsetzung des in Kapitel 3 beschriebenen Modells des Lokationsdienstes im Rahmen eines ersten Prototyps beschrieben. Nach einem Überblick über die Struktur des Prototyps mit den enthaltenen Komponenten und Modulen folgt die konkrete Darstellung der Schnittstellen und Klassenstrukturen. Es werden Implementierungsentscheidungen erörtert und verwendete Datenstrukturen beschrieben.

Anhand von Sequenzdiagrammen und Statustabellen werden die Bearbeitungsschritte beim Empfang von *lookup* bzw. *register* Anfragen erläutert und beschrieben nach welchem Algorithmus die einzelnen Module bereits selber auf auftretende Fehler reagieren, um die Fehlertoleranz des Gesamtsystems zu verbessern.

Darüber hinaus wird die Integration dieses Funktionsmodells an das Mobile Agenten System SeMoA (siehe Abschnitt 2.2) dargestellt und eine kurze Übersicht über Interaktion zwischen Lokationsdienst und SeMoA-Komponenten gegeben.

## 4.1 Überblick über die Struktur des Prototyps

Das im Rahmen dieser Arbeit entwickelte Funktionsmodell stellt einen lauffähigen Prototyp des in Kapitel 3 entwickelten Lokationsdienstes dar. Es implementiert die drei Basiskomponenten `LSServer`, `LSClient` und `LSProxy`, die unabhängig voneinander auf verschiedenen Rechnersystemen installiert werden können, und verfügt über eine dem beschriebenen LS-InfoGUI ähnliche Komponente `LSGui`, die dem Benutzer ermöglicht den Inhalt der Datenbanken von LS-Proxy und LS-Server lokal auf dem entsprechenden Rechnersystem anschaulich darzustellen und im direkten Zugriff über dieses Modul zu manipulieren. Die Initialisierung von LS-Client und LS-Proxy mit den Daten über die Infrastruktur der LS-Server geschieht in dieser Version des Lokationsdiesntes noch ausschließlich über Initialisierungsdateien, die beim Starten der jeweiligen Komponente ausgewertet werden. Eigene Module wurden entwickelt, die das *logging* von Informationen ermöglichen (`Log2Stream`) und das Cookie-Management übernehmen (`CookieManager`). Das *Location Service Protocol* wurde mit Unterstützung der Sicherheitsoptionen in der Version $LSP_{secure}$ vollständig umgesetzt. Die Schnittstelle

der internen Datenbank für LS-Proxy bzw. LS-Server wird von einer Komponente implementiert, welche die nötigen Einträge in einer im Speicher gehaltenen Datenstruktur ablegt (`MemoryDB`).

Ist beim Starten des Agentenservers der LS-Client aktiviert, so übernimmt er sofort selbsttätig die Aufgabe, folgende innerhalb dem Agentenserver auftretenden Ereignisse zu registrieren und entsprechend darauf zu reagieren: Das Kreieren bzw. der Empfang eines neuen Agenten, die Migration von Agenten auf andere Server und dessen Terminierung. Dem Agentenserver wird außerdem eine Schnittstelle zur Verfügung gestellt, die *lookup* Anfragen an den Lokationsdienst ermöglicht, um Agentenpositionen zu erfragen.

## 4.2 Allgemeine Implementierungsentscheidungen

SeMoA ist aus Gründen der Portabilität und der Sicherheit vollständig in Java$^{TM}$ realisiert. Die Portabilität des Codes spielt im Bereich der Mobilen Agenten eine wichtige Rolle, da der Programmcode der Agenten, in einem heterogenen Netzwerk auf verschiedenen Rechnersystemen zur Ausführung kommen soll. Hier sind die Stärken von Java$^{TM}$ in Bezug auf die Plattformunabhängigkeit von großem Vorteil. Auch die Sicherheitsphilosophie von Java ist, wenn auch nicht perfekt, so doch ausgereifter als bei anderen Programmiersprachen. Die *Virtual Machine* kontrolliert den Zugriff auf Ressourcen des Betriebssystems, es gibt einen weitreichenden Synchronisationsmechanismus und eine zuverlässige Typ-Identifikation zur Laufzeit. Zudem bietet das neue Sicherheitskonzept ab JDK 1.2 die Möglichkeit, diverse Ausführungs- und Zugriffsrechte bis hinunter auf Paket- und Klassenebene zu vergeben [28]. Aus diesen Gründen ist SeMoA vollständig in Java$^{TM}$ implementiert, und daher wurde auch die Implementierung dieser Arbeit in Java$^{TM}$ vorgenommen.

### 4.2.1 Integration externer Klassenbibliotheken

Da die Arbeit am Fraunhofer Institut für Graphische Datenverarbeitung erstellt wurde, finden sich die entwickelten Java-Klassen in *java packages* mit dem kennzeichnenden Präfix `DE.FhG.IGD`. Die Implementierung des Lokationsdienstes umfasst mehrere Klassen die auf drei Pakete aufgeteilt sind (siehe auch Anhang C.2.1): `DE.FhG.IGD.atlas.core` beinhaltet die Basiskomponenten des Lokationsdienstes mit den benötigten Modulen. In `DE.FhG.IGD.atlas.lsp` befinden sich alle für das *Location Service Protocol* benötigten Strukturen. Allgemeine Hilfsklassen, die für diese Arbeit entwickelt wurden, finden sich unter `DE.FhG.IGD.atlas.util`. Die Komponente `LSGui` wurde ebenfalls in einem eigenen *package* (`DE.FhG.IGD.atlas.ui`) abgelegt. Atlas steht in diesem Kontext übgrigens für Agent Tracking and Location Service.

Abgesehen von den genannten, neu entwickelten Java-Paketen und den im JDK 1.3 enthaltenen Klassenbibliotheken, die für das `LSGui` auch die `javax.swing` Bibliothek enthalten müssen, baut der Lokationsdienst auch noch auf weiteren externen Klassenbibliotheken auf (siehe auch Anhang C.2.2 und C.2.3). Diese ermöglichen die Anbindung an SeMoA, die Implementierung des LSP in ASN.1 und die vorgesehene Unterstützung von $LSP_{secure}$ durch kryptographische Algorithmen:

**SeMoA** - ist die Klassenbibliothek des SeMoA Projekts in dessen Rahmen das hier entwickelte System eingesetzt wird. Sie bietet die Möglichkeit, in SeMoA integrierbare *services* zu entwickeln und auf Ereignisse zu reagieren, die durch Agenten innerhalb des Agentenservers ausgelöst werden. Außerdem wird der in SeMoA bereits integrierte `KeyMaster` zur Schlüsselverwaltung, der konfigurierbare `AbstractServer`, auf dem LS-Proxy und LS-Server aufbauen, und die Schnittstelle `Vicinity` zu einem *multicast daemon* zur Wahl des aktiven LS-Proxy innerhalb des LAN verwendet.

**codec** - stellt eine Implementierung von ASN.1 und die Umsetzung einiger kryptographischer Standards zur Verfügung, die bei der Entwicklung des LSP Verwendung fanden.

**newa8jce** - enthält die Implementierung einiger kryptographischer Algorithmen, die für $LSP_{secure}$ benötigt werden (siehe auch Abschnitt 4.2.3).

**jce** - die *Java Cryptography Extensions* werden ebenfalls für $LSP_{secure}$ benötigt.

### 4.2.2 Verwendete Datentypen

In Tabelle 4.1 finden sich alle Basisdatentypen in Java und ASN.1, die bei Umsetzung des LSP verwendet wurden (siehe Abschnitt C.1).

| Datentyp des Lokationsdienstes | in Java | in ASN.1 |
|---|---|---|
| `ContactAddress` | `URL` | `OCTET STRING` |
| `ImplicitName` | `byte[]` | `OCTET STRING` |
| `Cookie` | `byte[]` | `OCTET STRING` |
| `ReplyState` | `int` | `INTEGER` |
| `Timestamp` | `long` | `INTEGER` |
| `Entry` | `LSPEntry` | `SEQUENCE {ImplicitName in, ContactAddress ca, Cookie c, Timestamp t}` |
| `ImplicitNameList` | `List` | `SET OF ImplicitName` |
| `EntryList` | `List` | `SET OF Entry` |

Tabelle 4.1: Bei der Implementierung verwendete Datentypen in Java bzw. ASN.1

### 4.2.3 Verwendete kryptographische Algorithmen

Wie bereits erwähnt, wurden bei der Implementierung des Lokationsdienstes einige kryptographische Algorithmen verwendet. Welche Algorithmen verwendet und zu welchem Zweck sie eingesetzt wurden findet sich in folgender Übersicht und in Tabelle 4.2.

**Generierung des Agenten-Hashcodes** Wie in Abschnitt 3.2 beschrieben, verwendet der Lokationsdienst den Hashcode des statischen Teils eines Agenten zu dessen eindeutiger Identifizierung als Bezeichner. Zur Berechnung dieses Wertes wird der *SecureHashAlgorithm* als *message digest* verwendet, der einen *bit string* der Länge 160 (20 Bytes) zum Ergebnis hat.

**Generierung des Cookies** Zur Autorisierung von *register* Anfragen im Rahmen des *Location Service Protocol* (siehe Abschnitt 3.2.3) wird ein zufällig generierter *bit string* als Cookie benötigt. Diesen liefert uns ein *SecureRandomGenerator* der basierend auf dem *SecureHashAlgorithm* beliebig lange *byte strings* generieren kann.

**Verschlüsselte *register* Anfragen** Um den Inhalt einer *register* Anfrage des LSP, wie in Abschnitt 3.2.3 beschrieben, nur einem bestimmten LS-Severn zugänglich zu machen, wird diese Anfrage bei der initialen Registrierung der Kontaktadresse eines Agenten verschlüsselt. Ein hybrides Verfahren aus symmetrischer und asymmetrischer Verschlüsselung kommt hier zum Einsatz. Vorerst wird bei jeder Anfrage ein *secret key* erzeugt. Mit diesem als Argument verschlüsselt *Tripel-DES* im Modus *CipherBlockChaining* den Inhalt der Anfrage. Dieser Operationsmodus erschwert eine *known plaintext attack*, sofern man kritischen Inhalt so positioniert, dass er nicht mit dem ersten, sondern erst mit einem der folgenden Blöcke verschlüsselt wird. Anschließend wird der *secret key* mit dem *public key* des LS-Server durch den asymmetrischen Verschlüsselungsalgorithmus *RSA* verschlüsselt und mit der Anfrage mit geschickt.

**Kodierte *register* Anfragen** Existiert durch eine initiale *register* Anfrage das aktuelle Cookie als gemeinsames Geheimnis zwischen LS-Client und LS-Server, so wird dieses bei folgenden Anfragen dazu verwendet, um mittels einer einfachen `XOR`-Verknüpfung das neue Cookie zu kodieren. Nun wird allerdings noch der *MessageAuthentificationCode* der Anfrage berechnet und mit geschickt, damit der LS-Server einen Integritätstest durchführen kann, um Manipulationen während des Transports zu bemerken.

| Algorithmus | Bezeichnung | Provider |
|---|---|---|
| *SecureRandomGenerator* | SHA1PRNG | SUN |
| *SecureHashAlgorithm* | SHA1 | SUN |
| *RSA* | RSA | SUN |
| *Trippel-DES* | DESede/CBC/PKCS5Padding | A8 |
| *MessageAuthentificationCode* | HmacSHA1 | A8 |

Tabelle 4.2: Bei der Implementierung verwendete kryptographische Algorithmen

## 4.3 Komponenten des Prototyps

Hier wird nun auf die Implementierung der einzelnen Module des Prototyps eingegangen, aus dem sich der Lokationsdienst zusammensetzt. `LSServer`, `LSClient` und `LSProxy` sind dabei eigenständige Komponenten, die durch eine existierende *main()* Methode direkt von der Kommandozeile aus mit entsprechenden Kommandozeilenparametern unabhängig voneinander gestartet werden können. Das `LSGui` Modul lässt sich ebenfalls von der Kommandozeile aus starten und sucht dann automatisch nach einer lokal installierten Schnittstelle zu LS-Proxy und/oder LS-Server. Die anderen aufgeführten Komponenten werden bei Bedarf implizit installiert und genutzt.

Die interne Klassenstruktur sowie Vererbungs- und Nutzungsbeziehung zwischen den Klassen der folgenden Komponenten wird durch UML-Diagramme verdeutlicht (siehe dazu auch [58], [73] und [91]).

### 4.3.1 LSP-Implementierung

Die Umsetzung des *Location Service Protocol* aus Abschnitt 3.2.3 geschah 1-zu-1 nach der entsprechenden Spezifikation in ASN.1 (siehe Anhang C.1). Wie man in Abbildung 4.1 erkennt, stellt `LSPRefresh` dabei einen Container für alle Anfragen dar, die im Kontext von LSP möglich sind. Durch die Angabe eines `requestType` Attributs wird die im `requestBody` aufgenommene Anfrage spezifiziert und typgerechte Kodierung bzw. Dekodierung zum Transfer der Anfrage über das Netzwerk ermöglicht. Eine fixes `VERSION` Attribut verhindert Probleme, die durch Versionsunterschiede des LSP auf Client- oder Server-Seite auftreten könnten. Die spezielle Behandlung der unterschiedlichen `LSPRegister` Anfragen geschieht gegebenenfalls gekapselt innerhalb den entsprechenden Datenstrukturen `LSPRegisterPlain`, `LSPRegisterEncrypted` bzw. `LSPRegisterEncoded`.

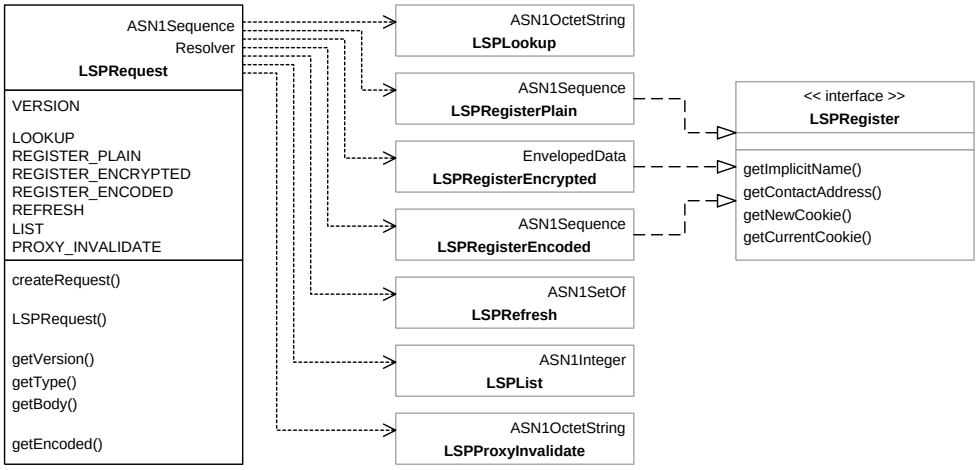


Abbildung 4.1: UML-Diagramm des LSPRequest

Vergleichbar zu `LSPRequest` stellt `LSPReply` einen Container für alle Antwortnachrichten des LS-Server bzw. LS-Proxy an den LS-Client dar (siehe Abbildung 4.2). Darüber hinaus ist hier allerdings auch vorgesehen, einen Statuscode anzugeben und bei Bedarf drei *error flags* unabhängig voneinander zu setzen oder zu löschen. Außerdem sind durch den `replyType` `STATE_ONLY` Antwortnachrichten ohne `replyBody` nur mit einem `replyState` möglich.

Die Struktur der möglichen Statuscodes ähnelt der Einteilung der HTTP Status Codes in Gruppen [64]. Dabei gibt es in diesem Fall fünf unterschiedliche Gruppen von Statuscodes, die folgende Bedeutung haben:

**Success** Die entsprechende Anfrage wurde erfolgreich empfangen, interpretiert und bearbeitet.

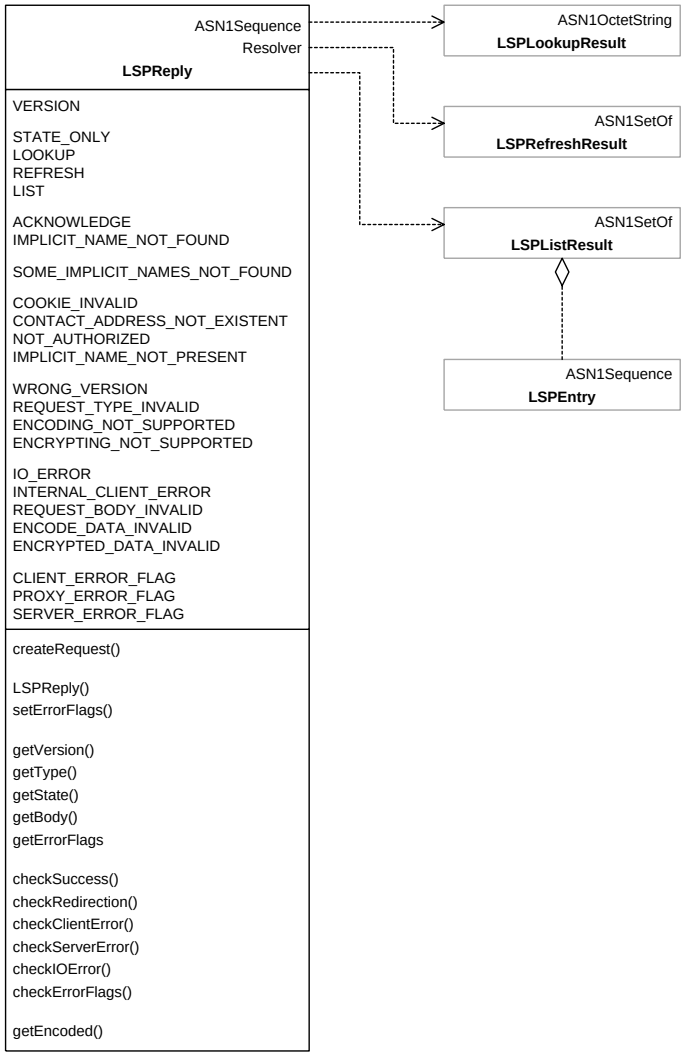


Abbildung 4.2: UML-Diagramm des LSPReply

**Redirection** Um die entsprechende Anfrage zufriedenstellend zu erfüllen, müssen vom Client gegebenenfalls weitere Schritte durchgeführt werden.

**ClientError** Die entsprechende Anfrage wurde erfolgreich empfangen und interpretiert, konnte allerdings nicht bearbeiter werden, da der Client bei der Angabe der Daten einen Fehler gemacht hat.

**ServerError** Die entsprechende Anfrage wurde erfolgreich empfangen und interpretiert, allerdings ist es dem Server durch eine unerwartete Situation nicht möglich, die Anfrage erfolgreich zu bearbeiten.

**IOError** Die entsprechende Anfrage wurde nicht empfangen oder nicht richtig interpretiert, da bei Transport, Kodierung oder Dekodierung ein Fehler aufgetreten ist.

Da die Zugehörigkeit des aktuellen Statuscodes von einem `LSPReply` zu einer der genann-

ten Gruppen direkt durch die jeweilige Methode (`checkSuccess()`, `checkRedirection()`, `checkClientError()`, `checkServerError()` oder `checkIOError()`) abgefragt werden kann, erleichtert sich die Fehlerbehandlung.

Außerdem habe ich mich dazu entschlossen tatsächlich alle auftretenden Erfolgs- bzw. Fehlerfälle einheitlich im Statuscode zu kodieren, anstatt wie sonst üblich bei Schnittstellenfunktionen der Basisdienste *java exceptions* zu generieren. Dies hat vor allem den Grund, dass alle auftretenden Fehler dann mit der `LSPReply` Datenstruktur auch über das Netzwerk zurück zum Client übertragen werden können.

Die drei *error flags* `CLIENT_ERROR_FLAG`, `PROXY_ERROR_FLAG` und `SERVER_ERROR_FLAG`, können dabei zusätzlich zu den Statuscodes angeben, bei welcher Komponenten des Lokationsdienstes bei einer Anfrage Fehler aufgetreten sind. Dies wird vor Allem dann interessant, wenn Anfragen kaskadiert vom LS-Client über den LS-Proxy zum LS-Server gesendet werden (siehe auch Abschnitt 4.4) und dann gegebenenfalls nur partiell bearbeitet werden können.

### 4.3.2 StorageDB

Die `StorageDB` Schnittstelle bietet dem Implementierer die Möglichkeit, die interne Struktur der Datenbank für LS-Proxy und LS-Server selber zu wählen und umzusetzen. Auch aus Geschwindigkeitsaspekten wurde für den Prototyp mit `MemoryDB` eine Implementierung gewählt, welche die benötigten Datenstrukturen direkt im Speicher hält. Es ist aber durchaus denkbar, durch diese Schnittstelle auch eine Anbindung des LS-Proxy bzw. LS-Server an ein leistungsstarke kommerzielle Datenbank als *backend* zu realisieren. Beim Start von LS-Proxy bzw. LS-Server wird die gewünschte Implementierung als Initialisierungsparameter übergeben.

Wie man in Abbildung 4.3 erkennt, gibt der Rahmen aus `StorageDB`, und `StorageDBEntry` ein paar Vorgaben über die Struktur der zu speichernden Einträge und die Funktionalität der Datenbank, die erfüllt werden müssen: So wird neben den Methoden zum Auslesen und Ändern der gespeicherten Daten auch ein Mechanismus verlangt, der Einträge selbsttätig löscht, sofern eine gewisse Zeitspanne (*timeout*) abgelaufen ist, ohne das sie aktualisiert wurden (siehe auch Abschnitt 3.2.1). `StorageDBImmutableEntry` ist eine bereits implementierte Klasse, die dem Anwendungsprogrammierer als Container für Einträge der Datenbank und dem Implementierer der Datenbank als Rückgabeparameter auf *lookup* Anfragen dienen kann. Da die Möglichkeit bestehen sollte, Anfragen parallel zu bearbeiten, müssen die Zugriffe auf die Datenbank synchronisiert werden.

Bei der Implementierung des Prototyps und der Umsetzung von `MemoryDB` im Speziellen wurden nun einige optimierenden Aspekte bedacht, die folgend dargestellt werden:

- Durch *init* Anfragen neu generierte Einträge vom Typ `MemoryDBEntry` werden im Speicher gehalten und sind somit fester Bestandteil der Datenbank. Der Zugriff wird nur über zwei Indizes realisiert, die durch eine `TreeMap`, nach Agenten-Hashcode bzw. Zeitstempel sortiert, Referenzen auf diese Einträge enthalten.

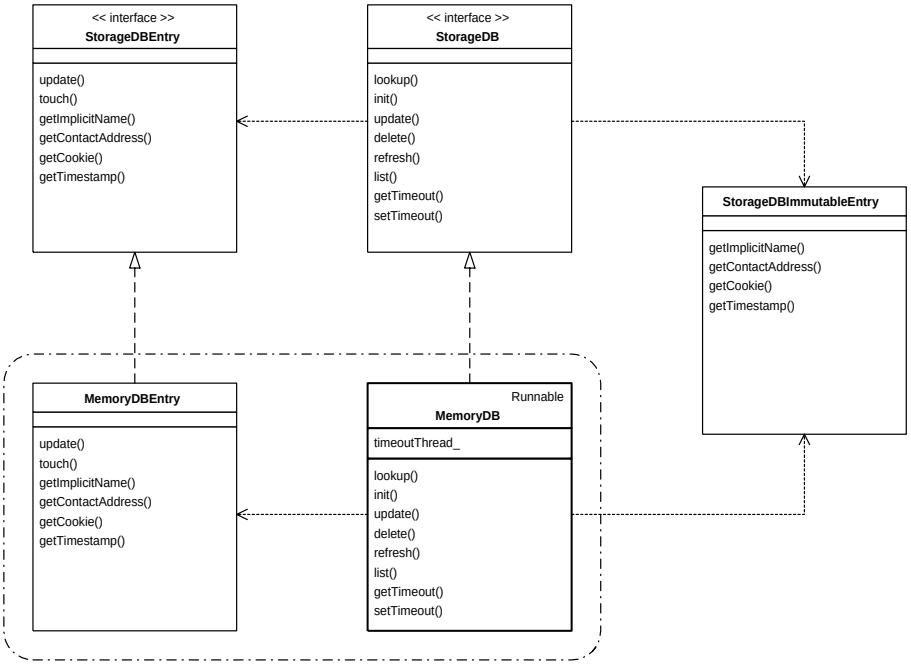


Abbildung 4.3: UML-Diagramm der Datenbankstruktur

- Eine Eintragsreferenz im ersten Index bleibt bis zur Löschung des Eintrags unverändert. Die Referenz auf den gleichen Eintrag im zweiten Index wird allerdings bei jeder *refresh* und *update* Anfrage gelöscht und unter dem neuen Zeitstempel der letzten Änderung abgelegt. Auf diese Weise wird der Aufwand für den Zugriff auf Einträge, der bei allen Methodenaufrufen der `StorageDB` Schnittstelle über den ersten Index geschieht, trotz Allem mit $O(log\ n)$ bezüglich deren Anzahl begrenzt.

- Der zweite Index wird nur für den internen *timeout* Mechanismus benötigt, der durch einen eigenen Thread realisiert ist: Dieser ermittelt mit dem Aufwand $O(1)$ den Eintrag mit dem niedrigstem Zeitstempel, also dem potentiell als nächstes zu löschendem Eintrag, und blockiert sich bis zum berechneten Zeitpunkt der Löschung. Nach seinem Erwachen ermittelt er wiederum den Eintrag mit dem niedrigsten Zeitstempel und löscht diesen, falls dessen *timeout* Zeit tatsächlich abgelaufen ist. Dies muss nicht der Fall sein, da in der Zwischenzeit eine *refresh* Anfrage bearbeitet worden sein kann. Anschießend blockiert er sich wieder, bis der nächste Eintrag potentiell zur Löschung ansteht.

- Da der Zugriff auf die beiden Indizes intern synchronisiert wird, ist die parallele Bearbeitung von Anfragen möglich.

### 4.3.3 LSServer

Der `LSServer` kann unabhängig von anderen Komponenten des Lokationsdienstes über die Kommandozeile gestartet werden. Dabei kann neben dem Port, auf dem er Anfragen empfangen wird, auch angegeben werden wie viele Anfragen parallel bearbeitet werden können

und wie groß eine Anfrage maximal sein darf, damit sie noch vom Server angenommen wird. Falls eine Anfrage die angegebene Maximalgröße überschreitet, so bricht die Bearbeitung ab, bevor darüber hinaus für diese Speicher allokiert wird. Dieses Vorgehen verhindert vor Allem die Möglichkeit von DoS-Attacken durch Anfragen „großer" Länge mit „sinnlosem" Inhalt.

Des weiteren lassen sich über die Kommandozeile ein *log file* mit entsprechendem *log level* angeben (siehe Abschnitt 4.3.10).

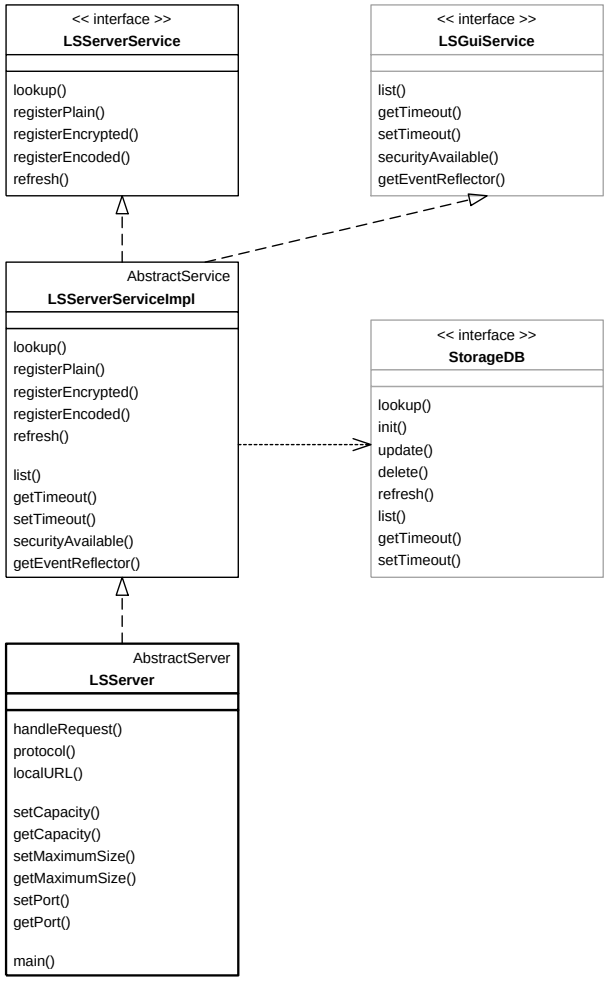


Abbildung 4.4: UML-Diagramm des LS-Servers

Wie man in Abbildung 4.4 erkennt, wird mit dem `LSServer` automatisch der entsprechenden Dienst `LSServerServiceImpl` installiert, auf den mittels der beiden Schnittstellen `LSServer-Service` und `LSGuiService` von „Außen" zugegriffen werden kann. Außerdem wird die entsprechende Datenbank initiiert. Standardmäßig handelt es sich dabei um die oben angesprochene, bereits implementierte Datenbank `MemoryDB`.

Wird der `LSServer` in der gleichen Ausführungsumgebung installiert wie ein `LSClient`, der auf diesen zugreifen möchte, so erkennt letzterer dies und nutzt die lokale Schnittstelle `LSServerService` direkt. Da davon ausgegangen wird, dass sich beide Komponenten bei

diesem Zugriff gemeinsam im größtmöglichsten Vertrauensbereich befinden, werden ursprünglich verschlüsselte bzw. kodierte *register* Anfragen des $LSP_{secure}$ (siehe Abschnitt 3.2.3) in diesem Fall durch Klartextparameter übermittelt, um die Bearbeitungsgeschwindigkeit zu erhöhen.

Die Schnittstelle `LSGuiService` dient dem Zugriff auf den Server mittels einer autorisierten Komponente wie dem `LSGui` (siehe Abschnitt 4.3.6). Wie man der Definition dieser Schnittstelle auch erkennt, ist es möglich sich die Referenz auf einen `EventReflector` des Servers zurückgeben zu lassen. Bei diesem lassen sich dann sogenannte `Listener` registrieren, die über den `EventReflector` Mechanismus vom Server immer dann informiert werden, wenn der Server aktiv den Status seiner internen Datenbank verändert. Nicht erfasst werden dadurch allerdings durch *timeout* direkt durch die Datenbank gelöschte Einträge.

Da bei dem implementierten Funktionsmodell die Komponente LS-AdminServer (siehe Abschnitt 3.2.1) noch nicht umgesetzt wurde, bearbeitet der `LSServer` grundsätzlich alle Anfragen die er erhält erst einmal unabhängig vom Hascode.

### 4.3.4 LSClient

Der `LSClient` wird in der Regel in einer gemeinsamen Ausführungsumgebung mit einem Agentenserver gestartet und kann wie der `LSServer` über die Kommandozeile konfiguriert werden. Zu den Konfigurationsparameters gehört wieder das *log file* mit entsprechendem *log level* und die Initialisierungsdatei, die Informationen über die Infrastruktur der LS-Server im Netzwerk enthält und von dem Modul `ServerInfo` eingelesen wird. Neben diesem wird auch das Modul `ProxyInfo` initiiert (Die Aufgabe dieser beiden Module wird weiter unten in den Abschnitten 4.3.8 und 4.3.9 noch näher erörtert).

Der eigentliche LS-Client hat eine ähnliche Struktur wie auch der LS-Server (siehe Abbildung 4.5). Mit dem `LSClient` wird auch wieder der entsprechende Dienst `LSClientServiceImpl` installiert, der über die Module `ServerInfo` und `ProxyInfo` alle Informationen erhält, die er benötigt, um über das Netzwerk mit einem existieredem Proxy oder dem entsprechendem Server Kontakt aufzunehmen, und mittels LSP eine Anfrage an diese zu senden.

Der `LSClient` stellt das Modul dar, das automatisch entsprechend auf Ereignisse reagiert, die innerhalb des Agentenservers durch Agenten ausgelöst werden. Darüber hinaus wird ein eigener Thread (`refreshThread_`) gestartet, der sich in regelmäßigen Abständen über die auf dem Agentenserver gerade in Ausführung befindlichen Agenten informiert und deren Einträge beim Proxy bzw. Server aktualisiert. Deswegen muss diese Klasse auch am ehesten auf die entsprechende Agentenplattform abgestimmt werden, in die der Lokationsdienst integriert werden soll. In wie weit dies für die SeMoA-Plattform nötig war, wird in Abschnitt 4.5 beschrieben.

Über die Schnittstelle `LSClientService` erhält der Agentenserver die Möglichkeit, explizit eine *lookup* oder *register* Anfrage zu stellen. Im Gegensatz zu dem relativ direkten Zugriff auf die Komponenten des Lokationsdienstes über die entsprechenden Methoden des Dienstes `LSClientServiceImpl`, die der `LSClient` intern nutzt, stehen dem Agentenserver nur die beiden allgemeinen Methoden `lookup()` und `register()` zur Verfügung. Diese erledigen

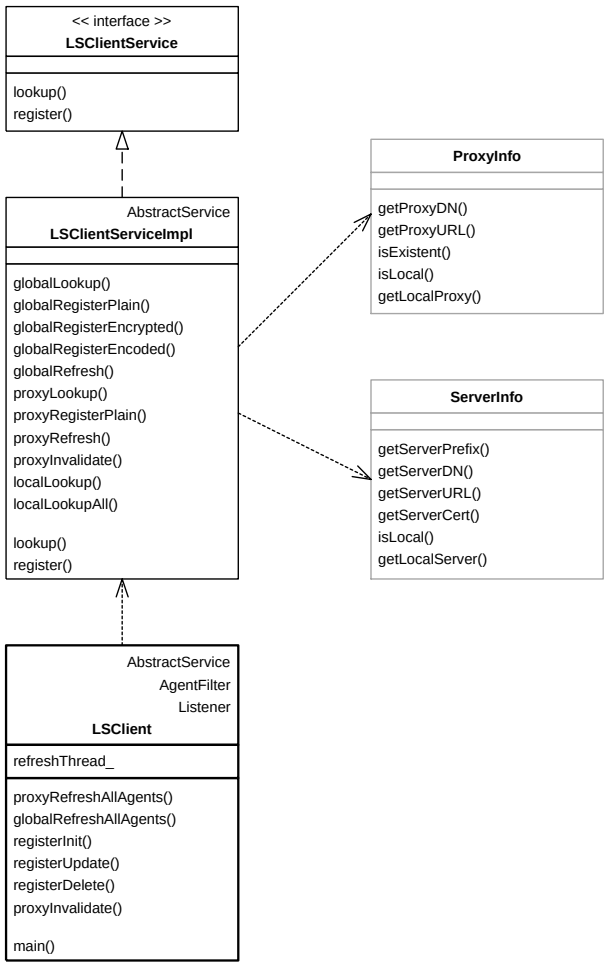


Abbildung 4.5: UML-Diagramm des LS-Clients

bereits einen großen Teil der Fehlerbehandlung und verfolgen dadurch auch eine einheitliche, von dem Agentenserver nicht veränderbare, Strategie (siehe Abschnitt 4.4). Durch diese Kapselung wird unter anderem die bestmöglichste Funktionsweise des Lokationsdienstes propagiert.

### 4.3.5 LSProxy

Da der `LSProxy` zwischen dem `LSClient` im gleichen LAN und dem `LSServer` im Internet die Rolle eines Gateways einnimmt, das sowohl Anfragen aus dem Netzwerk vergleichbar dem `LSServer` empfangen und in seiner internen Datenbank Änderungen vornehmen kann, als auch vergleichbar dem `LSClient` neue Anfragen initiieren muss, integriert dieser die bereits beschriebenen Dienstmodule `LSClientServiceImpl` und `LSServerServiceImpl` der beiden Komponenten. Aus diesem Grund vereinen die Kommandozeilenparameter, der ebenfalls über die Kommandozeile startbaren Komponente, die Optionen von `LSClient` und `LSServer`.

Darüber hinaus installiert der Proxy aber, wie in Abbildung 4.6 zu erkennen ist, auch den

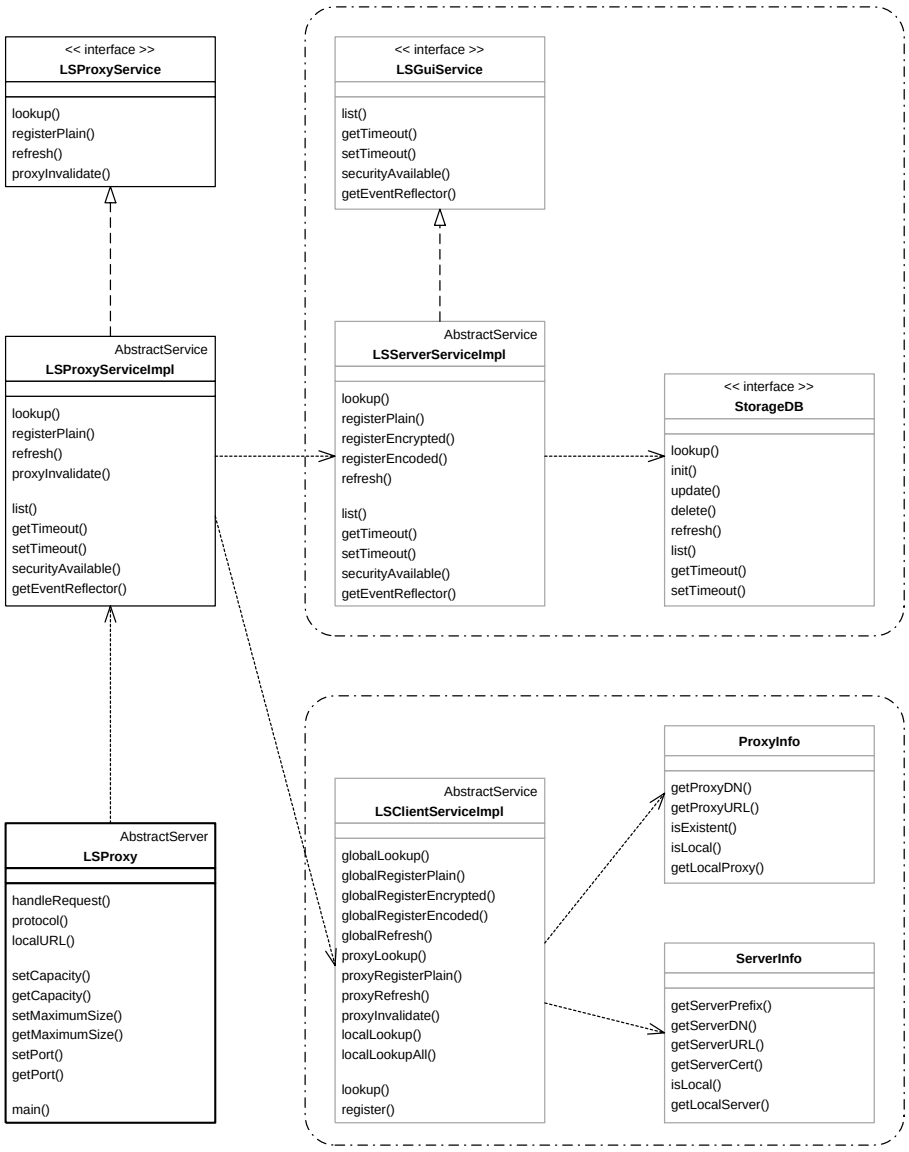


Abbildung 4.6: UML-Diagramm des LS-Proxy

eigenen Dienst `LSProxyServiceImpl`, der die proxy-spezifische Bearbeitung der Anfragen (siehe auch Abschnitt 4.4) vorsieht und auf die beiden eingebundenen Module delegiert.

Durch eine Instanz des Moduls `ProxyInfo` registriert sich der LS-Proxy im lokalen LAN als *potentiellen* Proxy für diese LAN und kann von diesem Zeitpunkt an zum *aktiven* Proxy gewählt (*voting*) werden (siehe Abschitt 4.3.9). Um zu ermitteln, ob eine empfangene Anfrage aus dem lokalen LAN stammt oder nicht, wird die eigene und die IP-Adresse des Senders einer Netzklasse zugeordnet (siehe Anhang C.6), die entsprechende Subnetz-Maske ermittelt und dadurch überprüft, ob beide Adressen im gleichen Subnetz liegen.

Wird der `LSProxy` in der gleichen Ausführungsumgebung installiert wie ein `LSClient`, der auf diesen zugreifen möchte, so erkennt letzterer das wie auch schon beim `LSServer` und

nutzt die lokale Schnittstelle `LSProxyService`. Außerdem wird über das integrierte Modul `LSServerServiceImpl` ebenfalls die Schnittstelle `LSGuiService` implementiert.

### 4.3.6 LSGui

Das `LSGui` wird über die Kommandozeile innerhalb der Ausführungsumgebung gestartet, in der auch von `LSProxy` und/oder `LSServer` installiert wurden, und bedarf keiner Kommandozeilenparameter zur Konfiguration. Werden die Schnittstellen `LSGuiService` und `LSProxyService` bzw. `LSServerService` der genannten Dienste lokal gefunden, so öffnet diese Komponente jeweils ein Fenster und nutzt diese Schnittstellen (siehe auch Abbildung 4.7), um den Inhalt der jeweiligen Datenbank graphisch in Tabellenform darzustellen. über ein Kontextmenu lassen sich einzelne Einträge direkt aktualisieren oder löschen und der *timeout* der jeweiligen Datenbank einstellen.

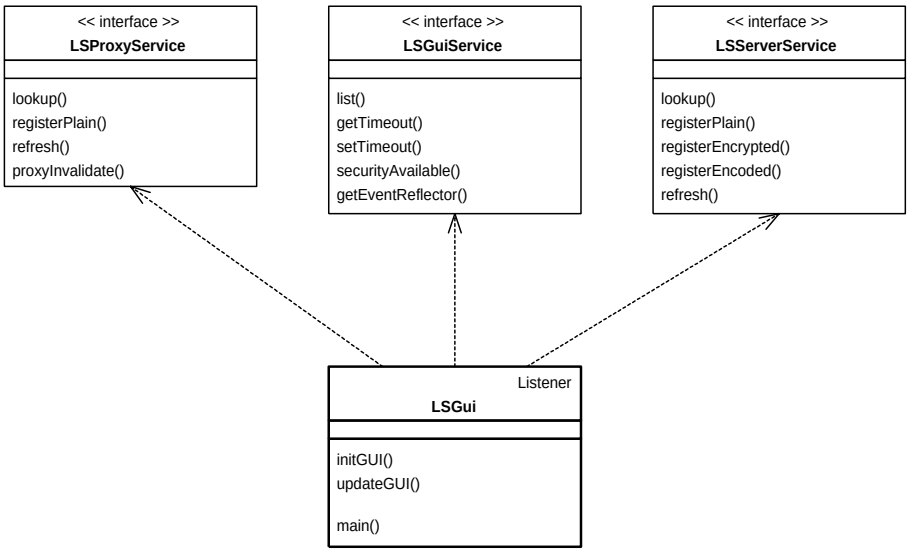


Abbildung 4.7: UML-Diagramm des LS-Gui

Außerdem implementiert diese Komponente einen `Listener`, den es bei dem `EventReflector` der entsprechenden Dienste registriert, um über Änderungen innerhalb der jeweiligen Datenbanken informiert zu werden. Auf solch ein Ereignis hin wird die Darstellung der Datenbank im entsprechenden Fenster aktualisiert.

### 4.3.7 CookieManager

Der `CookieManager` ist keine eigenständiges Komponente, sondern wird vom `LSClient` als Modul dazu verwendet, neue Cookies zu generieren und im Agenten zu speichern bzw. das aktuelle Cookie aus einem Agenten auszulesen. Dieses Modul kapselt alle Methoden die für den Umgang mit Cookies nötig sind (siehe Abbildung 4.8).

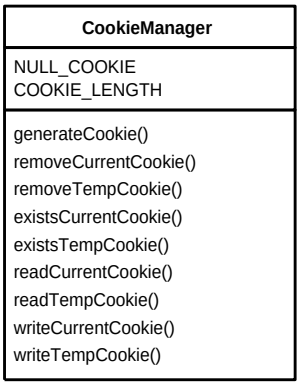


Abbildung 4.8: UML-Diagramm des CookieManager

Darüber hinaus definiert es die Länge eines Cookies, die in diesem Prototyp auf 20 Byte festgelegt wurde, und das NULL_COOKIE, das bei *register* Anfragen des LSP zur Unterscheidung zwischen *init*, *update* und *delete* verwendet wird (siehe Abschnitt 3.2.3).

### 4.3.8 ServerInfo

Das `ServerInfo` Modul kapselt die gesamte Funktionalität, die zur Lokalisierung der LS-Server im Netzwerk nötig ist und wird als Teil des *LSClientServiceImpl* Dienstes initiiert. Nachdem die Initialisierungsdatei mit den Daten über die Infrastruktur eingelesen wurde (siehe auch Anhang C.4), kann der Client über dessen Funktionalität (siehe Abbildung 4.9) bezüglich eines Agenten-Hashcodes den *Distinguished Name (DN)*, die Kontaktadresse und das Zertifikat mit dem *public key* des entsprechenden Severs abrufen. Darüber hinaus gibt `ServerInfo` auch die Dienstschnittstelle eines lokal installierten LS-Server zurück, falls vorhanden.

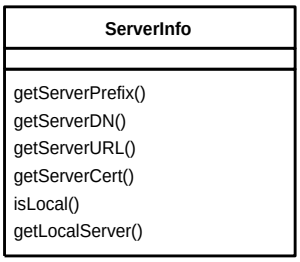


Abbildung 4.9: UML-Diagramm des ServerInfo

Die strikte Kapselung der beschriebenen Funktionalität in einem Modul ermöglicht später eine sehr einfache Integration der in Abschnitt 3.2.1 beschriebenen Komponente LS-AdminServer in den bestehenden Prototyp des Lokationsdienstes.

### 4.3.9 ProxyInfo

Was das `ServerInfo` Modul für die Lokalisierung der LS-Server bedeutet, stellt das `ProxyInfo` Modul für die Lokalisierung eines im lokalen LAN installierten LS-Proxy dar. Es wird ebenfalls als Teil des *LSClientServiceImpl* Dienstes initiiert.

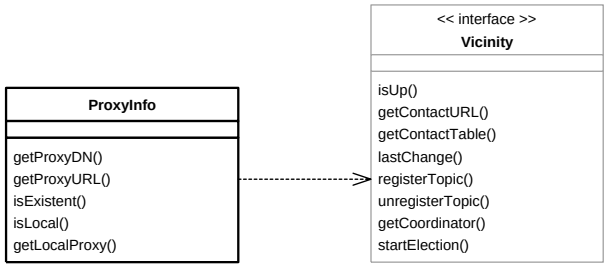


Abbildung 4.10: UML-Diagramm des ProxyInfo

Wie in Abbildung 4.10 zu erkennen ist ähnelt die Schnittstelle des Moduls stark der des `ServerInfo` Moduls, allerdings wird zur Lokalisierung des LS-Proxy ein Protokoll verwendet, das auf *multicast* Nachrichten basiert und dadurch ermöglicht, aus mehreren potentiellen LS-Proxy Komponenten im LAN jederzeit eine aktive zu wählen. Für diese Funktionalität ist ein *multicast daemon* zuständig, auf den über die Schnittstelle `Vicinity` zugegriffen werden kann.

### 4.3.10 Log2Stream

Aus den in Abschnitt 3.2.1 des letzten Kapitels angeführten Gründen ist es sinnvoll und wichtig, in der Lage zu sein Informationen zum Programmablauf bzw. auftretende Fehler bei LS-Client, LS-Proxy und LS-Server aufzuzeichnen (*logging*). Diese Aufgabe übernimmt das Modul `Log2Stream`, dessen Schnittstelle in Abbildung 4.11 dargestellt ist.

Verschiedene sogenannte *log level* ermöglichen nach folgendem Schema die Konfiguration, ob und wenn ja wie viele vom Programm erzeugte Nachrichten tatsächlich aufgezeichnet werden:

- Es existieren die *log level* `SILENCE`, `DEBUG`, `INFO`, `WARNING`, `ERROR` und `IMPORTANT` mit steigender Wertigkeit in dieser Reihenfolge.
- Jeder vom Programm aufzuzeichnenden Nachricht wird ein bestimmtes *log level* zugeordnet.
- Jeder Instanz dieses Moduls wird bei der Kreierung durch den Konstruktor eine bestimmte *log group* zugeordnet.
- Eine *log group* spezifiziert sowohl das aktuelle *log level*, als auch den *log stream*, der für die Aufzeichnung genutzt wird.

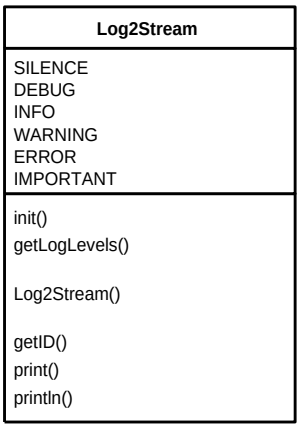


Abbildung 4.11: UML-Diagramm des Log2Stream

Unter diesen Voraussetzungen wird eine vom Programm generierte Nachricht genau dann aufgezeichnet, wenn ihr *log level* mindestens so hoch ist, wie das *log level*, welches der *log group* der Instanz zugeordnet ist, mit der die Nachricht aufgezeichnet werden soll.

Um Aufzeichnung von Nachrichten einer zusammengehörigen Modulgruppe nachträglich konfigurieren zu können, sollte jeder Instanz von `Log2Stream`, die von diesen Modulen zum Aufzeichnen von Nachrichten verwendet wird, bei der Kreierung die gleiche *log group* zugewiesen werden. Außerdem sollte jeder aufzuzeichnenden Nachricht von vorn herein ein vernünftiges *log level* zugewiesen werden.

Nun ist es möglich der entsprechende *log group* durch die statische Methode `init()` von `Log2Stream` vor der ersten Nutzung von einer beliebigen anderen Komponente in der gleichen Ausführungsumgebung ein bestimmtes *log level* und einen bestimmten *log stream* zuzuweisen.

Durch die freie Wahl des *log stream* kann sogar nach abgeschlossener Entwicklung der Modulgruppe entschieden werden, ob die aufzuzeichnenden Nachrichten auf dem Bildschirm ausgegeben, in eine Datei gespeichert oder über das Internet an einen Diagnose *host* übermittelt werden sollen.

Falls erwünscht generiert `Log2Stream` als Präfix für jede aufzuzeichnende Nachricht eine ID der folgenden Form, mit der sich die Nachrichten besser den Modulen zuordnen lassen: `[<loggroup>:<classname>]`

## 4.4 Fehlerbehandlung und Fehlertoleranz

Nachdem die einzelnen Module des Lokationsdienstes im letzten Abschnitt erläutert wurden, geben die hier aufgeführten Abbildungen noch einmal einen guten Überblick über das Zusammenspiel der Basiskomponenten und der darin enthaltenen Module. Außerdem wird die Strategie dargestellt, die bei *lookup*, *register* und *refresh* Anfragen verfolgt wird, um die Fehlertoleranz des Lokationsdienstes zu optimieren.

Es ist vorher noch zu bemerken, dass der LSProxy im Gegensatz zu *lookup* und *register* (siehe unten), die *refresh*, *proxyInvalidate* und *list* Anfragen ausschließlich auf der Basis seiner internen Datenbank bearbeitet und diese implizit nicht auch an den LSServer weiterleitet.

### 4.4.1 lookup()

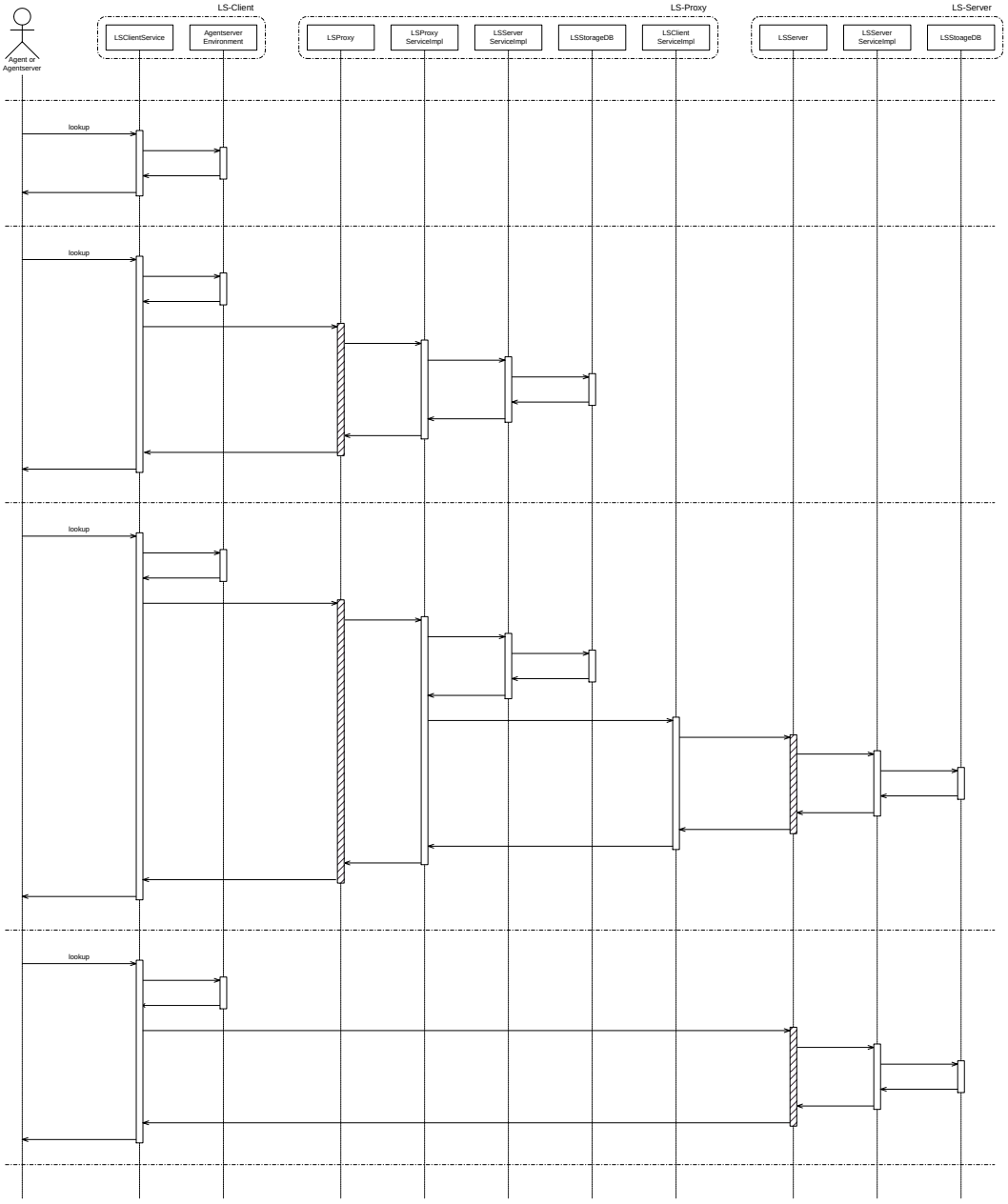


Abbildung 4.12: Sequenzdiagramm für die Bearbeitung von `LSClientService.lookup()`

In Abbildung 4.12 sind die Sequenzdiagramme für die verschiedenen möglichen Abläufe beim Aufruf der *lookup()* Methode über die `LSClientService` Schnittstelle dargestellt:

- Im ersten Fall wird die Kontaktadresse eines Agenten angefragt, der sich lokal auf dem Agentenserver befindet, auf dem die Anfrage initiiert wurde: Die Anfrage kann sofort mit den Daten aus dem Agentenserver beantwortet werden
- Im zweiten Fall wird die Kontaktadresse eines Agenten angefragt, der sich nicht auf dem lokalen Agentenserver befindet, aber auf einem, der sich im gleichen LAN befindet, und dessen Position auf dem für dieses LAN zuständigem LS-Proxy gespeichert ist: Nachdem auf dem Agentenserver keine Informationen über den Agenten gefunden wurden, wird der zuständige LS-Proxy angefragt. Dieser liefert die Kontaktadresse zurück und die Anfrage kann beantwortet werden.
- Im dritten Fall wird die Kontaktadresse eines Agenten angefragt, der sich nicht auf einem Agentenserver im gleichen LAN befindet: Nachdem der lokale Agentenserver und der LS-Proxy intern keine Daten über den Agenten gefunden haben, fragt der LS-Proxy automatisch den zuständigen LS-Server an und liefert sein Ergebnis an den LS-Client zurück.
- Im vierten Fall wird die Kontaktadresse eines Agenten angefragt, der sich nicht auf dem lokalen Agentenserver befindet. Außerdem ist im LAN kein LS-Proxy installiert bzw. *aktiv*: Nachdem der lokale Agentenserver keine Daten über den Agenten hat, fragt der LS-Client direkt den zuständigen LS-Server an.

Die beschriebene Strategie für *lookup* Anfragen sieht also vor, dass in der Reihenfolge *lokal*, *proxy*, *global* jeweils die nächste Instanz nur dann angefragt wird, wenn von der aktuellen Ebene kein positives Ergebnis zurückgegeben wurde. Diese gilt auch, wenn bei der Kommunikation mit der aktuellen Ebene ein Fehler aufgetreten ist. Durch dieses Verfahren wird Effizienz gewährleistet, geht man davon aus, dass bei der Kommunikation mit den Komponenten in der genannten Reihenfolge die Latenz und die Fehlerquote steigt. Darüber hinaus wird mit guter Wahrscheinlichkeit gewährleistet, dass der Lokationsdienst innerhalb eines durch Netzausfall begrenzten Systems für die darin enthaltenen Agenten noch funktionsfähig ist.

### 4.4.2 register()

In Abbildung 4.13 sind die Sequenzdiagramme für die verschiedenen möglichen Abläufe beim Aufruf der *register()* Methode über die `LSClientService` Schnittstelle dargestellt:

- Im ersten Fall läuft alles planmäßig: Die *register* Anfrage wird vom LS-Proxy bearbeitet und von diesem verschlüsselt bzw. kodiert an den entsprechenden LS-Server weiter geleitet, der die Anfrage ebenfalls bearbeiten kann.
- Im zweiten Fall kann der LS-Server die verschlüsselte bzw. kodierte *register* Anfrage nicht zufriedenstellend bearbeiten. Der LS-Proxy versucht deswegen, die gleiche Anfrage noch einmal im Klartext vom entsprechenden LS-Server bearbeiten zu lassen.
- Der dritte Fall entspricht dem ersten, abgesehen davon, das kein LS-Proxy lokalisiert werden kann und der LS-Client sofort den entsprechenden LS-Server kontaktiert.

- Der vierte Fall enspricht dem zweiten, wobei der LS-Client wiederum keinen LS-Proxy lokalisieren kann, sich direkt an den entsprechenden LS-Server wendet, dieser aber die verschlüsselte bzw. kodierte Anfrage nicht zufriedenstellend bearbeiten kann. Diesmal versucht der LS-Client die Anfrage im Klartext vom entsprechenden LS-Server bearbeiten zu lassen.

Die beschriebene Strategie für *register* Anfragen sieht also vor, dass die Daten in der Reihenfolge *proxy*, *global* an beide Komponenten weitergegeben wird falls vorhanden und die Anfrage an den LS-Server im Klartext wiederholt wird, falls dieser eine verschlüsselte bzw. kodierte Anfrage nicht zufriedenstellend bearbeiten kann. "Zufriedenstellend" bedeuten in diesem Fall, dass der zurückgegebene Statuscode nicht in die Gruppe `Success` oder `ClientError` fällt (siehe Abschnitt 4.3.1). In diesen beiden Fällen hätte eine Wiederholung der Anfrage im Klartext nämlich keinen Sinn.

Bei der *register* Anfrage ist es zwingend erforderlich, dass die Anfrage vom LS-Proxy auch an den LS-Server weitergegeben wird (*write-through*), da *lookup* Anfragen aus einem anderen LAN andernfalls kein oder ein falsches Ergebnis zur Folge hätten. Hier steht die Funktionstüchtigkeit im Vordergrund. Die Funktionstüchtigkeit des Lokationsdientes rechtfertigt im beschriebenen Fall übrigens auch die Wiederholung einer Anfrage an den LS-Server in Klartext und damit eine verminderte Sicherheit gegenüber Angreifern.

### 4.4.3 refresh()

Wie bereits in Abschnitt 4.3.4 beschrieben, versucht der LS-Client die Einträge beim LS-Proxy bzw. LS-Server in regelmäßigen Abständen zu aktualisieren. Das geschieht abhängig von der jeweiligen Komponente getrennt voneinander. Um die Fehlertoleranz gegenüber kurzzeitig auftretenden Netzwerkstörungen bzw. Nachrichtenverlust bei der Übertragung zu erhöhen, wurden für die Aktualisierungsintervalle 1/3 der *timeout* Zeit der jeweiligen Komponente gewählt. Dadurch können bis zu zwei *refresh* Anfragen verloren gehen, bevor die entsprechenden Einträge bei LS-Proxy bzw. LS-Server tatsächlich gelöscht werden.

Wenn der zu aktualisierende Eintrag durch *timeout* oder einen Neustart der jeweiligen Komponenten nicht mehr vorhanden ist, wird abhängig von dieser außerdem auf folgende Weise vorgegangen:

**LS-Proxy** Diese Tatsache wird ignoriert und von einem reinitialisieren des Eintrags durch eine folgende *register* Anfrage abgesehen, da der LS-Server immer noch einen gültigen Eintrag mit aktuellen Cookie enthalten könnte. Bei der Migration des zugehörigen Agenten auf einen anderen Agentenserver im gleichen LAN, wird der entsprechende Eintrag dann automatisch wieder erstellt (siehe Fehlerbehandlung bei `update(X→Y)` in Tabelle 4.3).

**LS-Server** Ist der zu aktualisierende Eintrag auf dem LS-Server nicht vorhanden, so wird der entsprechende Eintrag vorerst auf einem exstierendem LS-Proxy durch *proxyInvalidate* gelöscht und anschließend auf LS-Proxy und LS-Server durch eine folgende *register* Anfrage mit neuem Cookie reinitialisiert. Dies ist in diesem Fall möglich, da bei der *proxyInvalidate* Anfrage kein Cookie zu Autorisation benötigt wird.

### 4.4.4 Implizite Synchronisation von LS-Proxy und LS-Server

Durch Tabelle 4.3 wird die Fehlerbehandlung bei der *lookup* und *register* Anfrage noch einmal differenzierter dargestellt. Hier geht es vor Allem um das Verhalten des LS-Proxy nach Auswertung des Statuscodes vom LS-Server, und die Wahl des Statuscodes, der an den LS-Client zurückgegeben wird. Hierfür muss der Statuscode, der bei der eigenen Bearbeitung der Anfrage ermittelt wurde, mit dem als Antwort auf die Anfrage an den LS-Server ermittelten in geeigneter Form zu einem neuen zusammengefasst werden.

Tabelle 4.3 ist folgender Weise zu interpretieren:

| | |
|---|---|
| `lookup(X)` | Die Kontaktadresse des Agenten mit dem Hashcode `X` wird angefragt (dieser befindet sich nicht auf dem lokalem Agentenserver). |
| `-` | Es existiert kein Eintrag für diesen Agenten |
| `X` | Es existiert ein Eintrag für diesen Agenten |
| `?` | Bei der Anfrage an diese Komponenten ist ein `IOError` aufgetreten |
| `init(X)` | Ein Eintrag für den Agenten mit dem Hashcode `X` wird initialisiert. |
| `-` | Es existiert noch kein Eintrag für diesen Agenten |
| `Z` | Es existiert bereits ein Eintrag für diesen Agenten |
| `?` | Bei der Anfrage an diese Komponenten ist ein `IOError` aufgetreten |
| `update(X→Y)` | Der Eintrag für den Agenten mit dem Hashcode `X` wird durch den Eintrag Y aktualisiert. |
| `-` | Es existiert kein Eintrag für diesen Agenten |
| `X` | Es existiert ein Eintrag für diesen Agenten mit dem passenden Cookie |
| `Z` | Es existiert ein Eintrag für diesen Agenten mit einem anderen Cookie |
| `?` | Bei der Anfrage an diese Komponenten ist ein `IOError` aufgetreten |
| `delete(X)` | Der Eintrag für den Agenten mit dem Hashcode `X` wird gelöscht. |
| `-` | Es existiert kein Eintrag für diesen Agenten |
| `X` | Es existiert ein Eintrag für diesen Agenten mit dem passenden Cookie |
| `Z` | Es existiert ein Eintrag für diesen Agenten mit einem anderen Cookie |
| `?` | Bei der Anfrage an diese Komponenten ist ein `IOError` aufgetreten |

Den Werten in der Tabelle 4.3 liegt also, nach Anfragetyp aufgeschlüsselt, folgende Strategie zu Grunde:

**lookup** Hier reicht es aus, wenn einer der beiden Komponenten den Eintrag enthält, um ein positives Ergebnis zurückzuliefern. Tritt bei der Anfrage vom LS-Proxy an den LS-Server ein Ergebnis auf, dass nicht in die Gruppe `Success` (siehe Abschnitt 4.3.1) fällt, also kein eindeutiges Ergebnis auf die Anfrage ermittelt werden kann, so wird durch das `SERVER_ERROR_FLAG` ein Fehler bei der Verbindung zum LS-Server angezeigt und der Statuscode des LS-Proxy zurückgegeben. Tritt bereits bei der Verbindung zwischen LS-Client zu LS-Proxy ein Fehler auf, so wird das `SERVER_ERROR_FLAG` gesetzt.

**register** Wie bei der *lookup* Anfrage werden das `PROXY_ERROR_FLAG` bzw. das `SERVER_ERROR_FLAG` gesetzt, um Verbindungsfehler anzuzeigen, die nicht dem Statuscode entnommen werden können. Soweit die Komponenten LS-Proxy und LS-Server erreicht werden können,

wird die Anfrage auf beiden Komponenten, abhängig vom Inhalt ihrer internen Datenbank, aber unabhängig voneinander, bearbeitet. Folgende Fälle sind weiterhin interessant:

**init** Wird bei der Initialisierung einer Kontaktadresse für einen Agenten festgestellt, dass bereits ein Agent mit gleichem Hashcode existiert und bereits registriert wurde, so bleibt der Inhalt von LS-Proxy *und* LS-Server unverändert und `COOKIE_INVALID` wird zurückgegeben. Dieser Fall dürfte allerdings mit einer sehr geringen Wahrscheinlichkeit auftreten.

**update** Kann diese Anfrage nicht erfolgreich ausgeführt werden, weil noch kein entsprechender Eintrag existiert, so wird durch ein folgendes *init* versucht, den Eintrag mit dem gegebenen neuen Cookie neu zu initialisieren. Dies geschieht unabhängig davon, ob es sich dabei um LS-Proxy und/oder LS-Server handelt. Im Notfall sollte wenigstens eine der beiden Komponenten *lookup* Anfragen richtig beantworten können. Zurückgegeben wird in diesem Fall allerdings immer der Statuscode des LS-Proxy, da dieser bei *lookup* Anfragen im LAN Priorität hat. Das geschilderte Problem löst sich erst nach Terminierung des Agenten und dem *timeout* des entsprechenden Eintrags.

**delete** Falls auf LS-Proxy und/oder LS-Server ein entsprechender Eintrag gefunden wird und das aktuelle Cookie übereinstimmt, wird dieser gelöscht. Zurückgegeben wird in diesem Fall immer der Statuscode des LS-Proxy. Stimmt das aktuelle Cookie nicht überein, so wird der Eintrag gegebenenfalls durch sein *timeout* gelöscht.

Bei diesen Betrachtungen wird übrigens davon ausgegangen, dass Unterschiede im Inhalt der Datenbank von LS-Proxy bzw. LS-Server bezüglich eines Eintrags durch Netzwerkprobleme aufgetreten sind. Werden bei der Optimierung der Fehlertoleranz Aspekte der Sicherheit höher eingestuft, so müssen manche Entscheidungen anders gefällt werden. Unabhängig davon sind die in diesem Abschnitt angesprochenen Aspekte auf jeden Fall wichtige Optimierungsparameter für den laufenden Lokationsdienst. Dies betrifft vor Allem die Implementierung der beiden Module `LSClient` und `LSProxyServiceImpl` wie auch die beiden Funktionen der `LSClientService` Schnittstelle.

## 4.5 Integration des Lokationsdienstes in die Architektur von SeMoA

Nachdem die Implementierung des Prototyps bis jetzt weitgehend generisch in Bezug auf die Einbindung in eine Agentenplattform beschrieben wurde, wird in diesem Abschnitt jetzt abschließend erläutert, in welcher Form das Funktionsmodell des Lokationsdientes in die *SeMoA-Plattform 2* integriert wurde. Es wird beschrieben in wie weit er auf Strukturen der Plattform aufbaut und was nötig ist, um die Komponenten des Lokationsdienstes beim Start des SeMoA-Servers zu initialisieren.

### 4.5.1 Modifikationen an SeMoA

Vorerst seien noch einmal die beiden Klassen von SeMoA erwähnt, die für die Integration des Lokationsdienstes in die Plattform modifiziert werden mussten. Hierbei handelte es sich vor

Allen Dingen darum, eine Schnittstelle zwischen Agentenserver und `LSClient` zu schaffen (siehe auch Anhang C.2.2):

**AgentEvent** Diese Klasse wurde dahin gehend modifiziert, dass sie nicht wie zuvor nur mit einer *AgentCard* zu initialisieren ist, sondern ebenfalls mit einer einem *AgentContext*, und dieser anschließend auch über eine Funktion *getContext()* wieder abrufbar ist. Außerdem wurden zu den bestehenden Status-Zustände *created*, *migrated* und *terminated* hinzugefügt.

**AgentContext** Innerhalb von *AgentContext* war es nötig, der Methode zur Benachrichtigung des *EventReflector* ein *AgentEvent* zu übergeben, das mittels einer Referenz auf den *AgentContext* initialisiert wird. Des weiteren mussten an einigen Stellen in der Klasse neue Ereignisse mit den oben angesprochenen Status-Zuständen generiert werden.

Es ist allerdings zu bemerken, dass diese Modifikationen das Sicherheitsmodell von SeMoA beeinträchtigen. Aus diesem Grund sollte die Schnittstelle zwischen SeMoA und dem Lokationsdienst noch einmal überdacht werden.

### 4.5.2 Genutzte Schnittstellen zu SeMoA

Die im Folgenden beschriebenen Klassen von SeMoA wurden unverändert benutzt, um die Schnittstelle zwischen SeMoA und dem Lokationsdienst herzustellen, und den Lokationsdienst als solchen in SeMoA einzubringen:

**AbstractServer** Diese Klasse wird von den Komponenten `LSServer` und `LSProxy` als Basis verwendet. Sie stellt einen konfigurierbaren Server mit der integrierten Schnittstelle eines SeMoA *service* zur Verfügung.

**AbstractService** Dieses Klasse ermöglicht den Komponenten `LSClient`, `LSClientServiceImpl`, `LSProxyServiceImpl` und `LSServerServiceImpl` die direkte Eingliederung als SeMoA *service* in die Plattform.

**Environment** Über diese Klasse bekommt der `LSClientService` die Informationen, die er für die Implementierung von `localLookup()` und `localLookupAll()` benötigt.

**AgentFilter.In** Durch diese Schnittstelle integriert `LSClient` einen Informationsfilter in die Filter-Pipeline von SeMoA, die ihn über ankommende Agenten informiert (Reaktion: `registerUpdate()`).

**AgentFilter.Out** Durch diese Schnittstelle integriert `LSClient` einen Informationsfilter in die Filter-Pipeline von SeMoA, die ihn über migrierende Agenten informiert (Reaktion: gegebenenfalls `proxyInvalidate()`).

**EventReflector** Über einen `Listener`, den der `LSClient` beim `EventReflector` des SeMoA-Servers registriert, erhält er die Information über neu kreierte und terminierte Agenten (Reaktion: `registerInit()` und `registerDelete()`)

**AgentResource** Dies ist die Schnittstelle die der `CookieManager` benötigt, um auf die Struktur eines Agenten zuzugreifen und dort an bestimmter Stelle das aktuelle (`mutable/magic`) und das temporäre (`mutable/magic.temp`) Cookie zu speichern.

**WhatIs** Durch diese Klasse wird auf komfortable Weise ermöglicht, Variablen zu definieren, die vom Lokationsdienst als Konfigurationparameter verwendet werden.

### 4.5.3 Installation des Lokationsdienstes

Um nun die drei Basiskomponenten `LSClient`, `LSProxy` und `LSServer` in der Ausführungsumgebung von SeMoA zu starten, müssen bereits die in Tabelle 4.4 angegebenen Dienste in dieser Reihenfolge installiert und die dort aufgeführten Variablen in der `WhatIs` Konfigurationsdatei definiert sein.

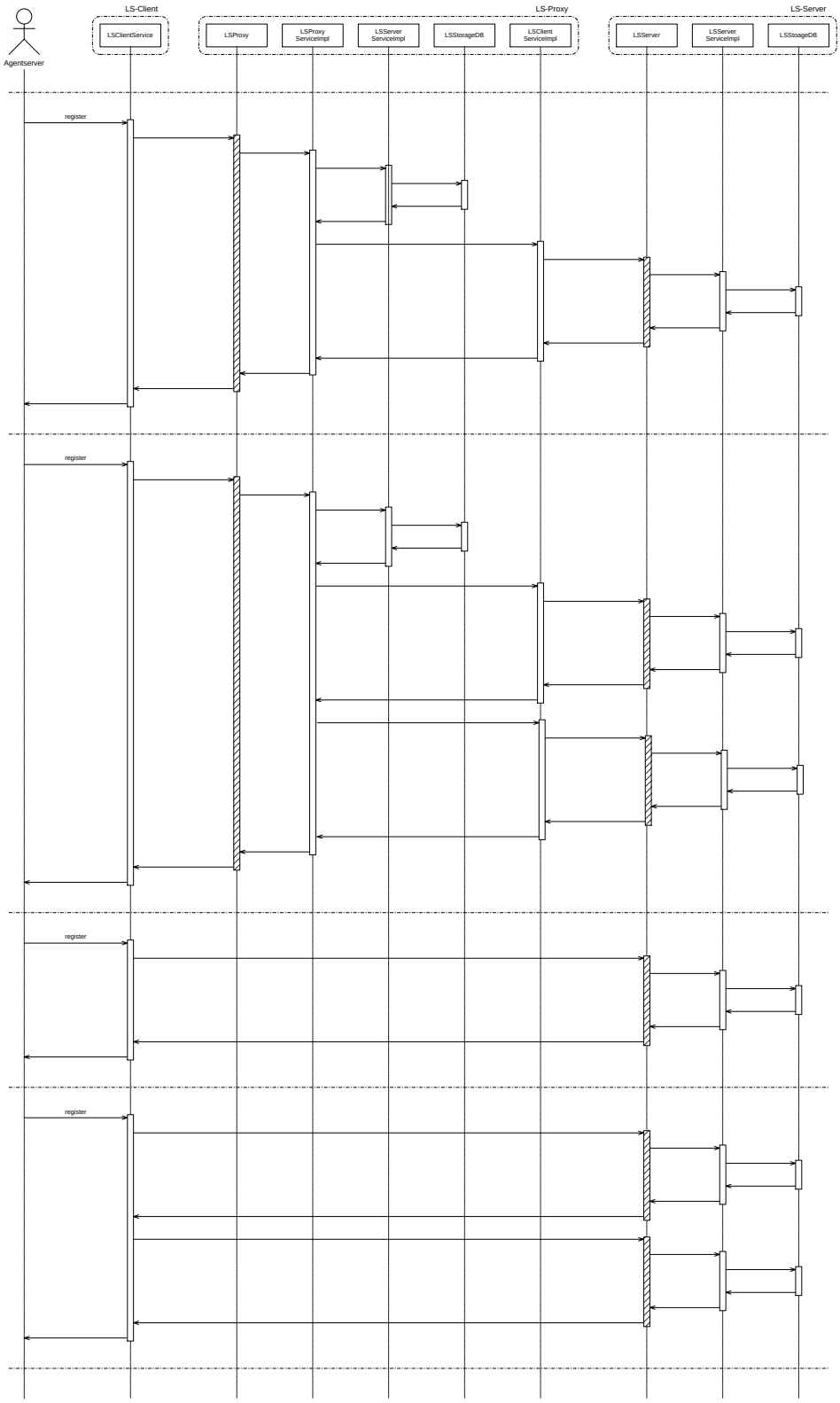


Abbildung 4.13: Sequenzdiagramm für die Bearbeitung von `LSClientService.register()`

| Anfrage über Client | vorher Proxy | vorher Server | ⇒ | nachher Proxy | nachher Server | Reply-State | Error Flags |
|---|---|---|---|---|---|---|---|
| lookup(X) | - | - | | - | - | Not Found | |
| | - | X | | - | X | Acknowledge | |
| | X | - | | X | - | Acknowledge | |
| | X | X | | X | X | Acknowledge | |
| | ? | - | | ? | - | Not Found | [P] |
| | ? | X | | ? | X | Acknowledge | [P] |
| | - | ? | | - | ? | Not Found | [S] |
| | X | ? | | X | ? | Acknowledge | [S] |
| | ? | ? | | ? | ? | IO Error | |
| init(X) | - | - | | X | X | Acknowledge | |
| | - | Z | | - | Z | Cookie Invalid | |
| | Z | - | | Z | - | Cookie Invalid | |
| | Z | Z | | Z | Z | Cookie Invalid | |
| | ? | - | | ? | X | Acknowledge | [P] |
| | ? | Z | | ? | Z | Cookie Invalid | [P] |
| | - | ? | | X | ? | Acknowledge | [S] |
| | Z | ? | | Z | ? | Cookie Invalid | [S] |
| | ? | ? | | ? | ? | IO Error | |
| update(X→Y) | - | - | | Y | Y | Acknowledge | |
| | - | Z | | Y | Z | Acknowledge | |
| | - | X | | Y | Y | Acknowledge | |
| | Z | - | | Z | Y | Cookie Invalid | |
| | Z | Z | | Z | Z | Cookie Invalid | |
| | Z | X | | Z | Y | Cookie Invalid | |
| | X | - | | Y | Y | Acknowledge | |
| | X | Z | | Y | Z | Acknowledge | |
| | X | X | | Y | Y | Acknowledge | |
| | ? | - | | ? | Y | Acknowledge | [P] |
| | ? | Z | | ? | Z | Cookie Invalid | [P] |
| | ? | X | | ? | Y | Acknowledge | [P] |
| | - | ? | | Y | ? | Acknowledge | [S] |
| | Z | ? | | Z | ? | Cookie Invalid | [S] |
| | X | ? | | Y | ? | Acknowledge | [S] |
| | ? | ? | | ? | ? | IO Error | |
| delete(X) | - | - | | - | - | Not Present | |
| | - | Z | | - | Z | Not Present | |
| | - | X | | - | X | Not Present | |
| | Z | - | | Z | - | Cookie Invalid | |
| | Z | Z | | Z | Z | Cookie Invalid | |
| | Z | X | | Z | X | Cookie Invalid | |
| | X | - | | - | - | Acknowledge | |
| | X | Z | | - | Z | Acknowledge | |
| | X | X | | - | - | Acknowledge | |
| | ? | - | | ? | - | Not Present | [P] |
| | ? | Z | | ? | Z | Cookie Invalid | [P] |
| | ? | X | | ? | - | Acknowledge | [P] |
| | - | ? | | - | ? | Not Present | [S] |
| | Z | ? | | Z | ? | Cookie Invalid | [S] |
| | X | ? | | - | ? | Acknowledge | [S] |
| | ? | ? | | ? | ? | IO Error | |

Tabelle 4.3: Implizite Synchronisation von LS-Proxy und LS-Server

| Variablen in `WhatIs.conf` | Beim Start zu installierende Komponenten |
|---|---|
| | `DE.FhG.IGD.util.WhatIs` |
| `KEYMASTER` | `DE.FhG.IGD.security.KeyMasterImpl` |
| `CERTFINDER` | `DE.FhG.IGD.security.CertFinder` |
| `AGENT_EVENTS` | `DE.FhG.IGD.event.EventReflector` |
| `VICINITY` | `DE.FhG.IGD.net.Vicinity` |
| `(LS_SERVER)` | `(DE.FhG.IGD.atlas.core.LSServer)` |
| `(LS_PROXY)` | `DE.FhG.IGD.atlas.core.LSProxy` |
| `(LS_CLIENT)` | `(DE.FhG.IGD.atlas.core.LSClient)` |

Tabelle 4.4: Für den Lokationsdienst in SeMoA zu Installierende Komponenten

# Kapitel 5

# Evaluierung und Diskussion

Nach der erfolgreichen Umsetzung des Lokationsdienstes in das in Kapitel 4 beschriebene Funktionsmodell, wird dieser Prototyp nun evaluiert und damit auf seine Funktionsfähigkeit unter großer Last hin untersucht. Hier geht es zum einen darum, Faktoren aufzudecken, die in Bezug auf die Skalierbarkeit des Dienstes beschränkend wirken könnten, sowie darum, Grenzwerte für Bearbeitungsgeschwindigkeiten anzugeben bzw. abzuschätzen.

Vorerst werden einige Größen genannt, welche die Leistungsfähigkeit des Lokationsdienstes grundlegend beeinflussen und durch die Implementierung des Prototyps bereits festgelegt wurden. Nach der Beschreibung der Testumgebung im zweiten Abschnitt werden im letzten Abschnitt dann die Testergebnisse vorgestellt und diskutiert.

## 5.1 Länge der verschiedenen LSP-Nachrichten

In diesem Abschnitt werden die Längen der Nachrichten abgeschätzt, die durch das *Location Service Protocol* übermittelt werden. Aus diesen Nachrichten baut sich die eigentliche Kommunikation zwischen LS-Client, LS-Proxy und LS-Server im Netzwerk auf, durch sie können Anfragen und die entsprechenden Ergebnisse übermittelt werden. Sie sind deswegen ausschlaggebend für die Bearbeitungsgeschwindigkeit und Fehleranfälligkeit des Dienstes.

Um die unten angegebene Größe dieser Nachrichten nun erklären zu können, müssen vorerst die Datentypen betrachtet werden, die tatsächlich in eine Nachricht eingebettet werden. Diese Datentypen stimmen nicht immer mit den Parametern der in den Abschnitten 3.2.2 und 3.2.3 beschriebenen Schnittstellenfunktionen überein. In Tabelle 5.1 findet sich deswegen eine detaillierte Übersicht:

Die Längen der in Tabelle 5.1 aufgeführten Datentypen lassen sich nun folgendermaßen angeben:

**ContactAddress** Die Kontaktadresse wird durch eine URL dargestellt, die laut Spezifikation auf eine Länge von maximal 255 Bytes beschränkt ist.

**Cookie** Das aktuelle und neue Cookie wurden beim Prototyp in der Länge an die des `ImplicitName` (also 20 Bytes) angepasst.

| Nachrichtentyp | In die Nachricht eingebettete Datentypen |
|---|---|
| `list` | `Timestamp` |
| `listResult` | `EntryList` |
| `lookup` | `ImplicitName` |
| `lookupResult` | `ContactAddress` |
| `invalidate` | `ImplicitName` |
| `refresh` | `ImplicitNameList` |
| `refreshResult` | `ImplicitNameList` |
| `registerEncoded` | `ImplicitName, ContactAddress, EncodedCookie, MAC` |
| `registerEncrypted` | `EnvelopedData` |
| `registerPlain` | `ImplicitName, ContactAddress, Cookie, Cookie` |

Tabelle 5.1: Die Nachrichten des LSP mit Angabe der eingebetteten Datentypen

**EncodedCookie** Dieser Datentyp entsteht durch eine einfache `XOR`-Verknüpfung bei Anwendung von $LSP_{secure}$ (siehe Abschnitt 3.2.3), welche die ursprüngliche Länge des Cookies (also 20 Bytes) nicht verändert.

**EntryList** Repräsentiert eine Liste von Datenbankeinträgen auf dem LS-Proxy bzw. LS-Server, die jeweils `ImplicitName`, `ContactAddress`, `Cookie` und `Timestamp` enthalten. Die Länge dieser Liste ist also mit einem Vielfachen von maximal 279 Bytes (20+255+20+4) anzugeben.

**EnvelopedData** Diese Datenstruktur kapselt neben den verschlüsselten Daten (`ImplicitName`, `ContactAddress`, `Cookie`, `Cookie`) zusätzlich einige Datentypen, die auf der Empfängerseite der Nachricht eine hybride Entschlüsselung ermöglichen (eine konkreter Wert findet sich in der unten dargestellten Tabelle).

**ImplicitName** Seine Größe ist durch den verwendeten *SecureHashAlgorithm* (siehe Abschnitt 4.2.3) zur Berechnung des Hashcodes auf 20 Bytes festgelegt.

**ImplicitNameList** Repräsentiert eine List von `ImplicitName` Angaben und hat somit ein Vielfaches von 20 Bytes zur Länge.

**MAC** Da der *MessageAuthentificationCode* ebenfalls auf dem *SecureHashAlgorithm* basiert, ist er auch 20 Bytes lang.

**Timestamp** Der Zeitstempel gibt die Systemzeit eines Rechners als *long* Wert in Millisekunden an, seine Länge beträgt also genau 8 Bytes.

Bedenkt man noch, dass durch die Einbettung dieser Datenstrukturen in die entsprechenden ASN.1-Struktur einen gewisser *overhead* durch die Kodierung entsteht und diese Nachricht wiederum in der ASN.1-Struktur LSPRequest bzw. LSPReply mit einem weiteren *overhead* gekapselt wird, so ergeben sich folgende Längenangaben. Dabei wird von einer URL bestehend aus 25 Zeichen und einer `EntryList` bzw. `ImplicitNameList` mit jeweils fünf Einträgen ausgegangen:

Die in Tabelle 5.2 mit $^{(*)}$ gekennzeichneten Größen können je nach Anzahl der Listeneinträge um ein Vielfaches anwachsen. Die mit $^{(+)}$ gekennzeichneten Längen sind je nach URL etwas

| Nachrichtentyp | Länge der Nachricht | |
|---|---|---|
| list | 18 | Bytes |
| listResult | 433 (*) | Bytes |
| lookup | 30 | Bytes |
| lookupResult | 39 (+) | Bytes |
| invalidate | 30 | Bytes |
| refresh | 120 (*) | Bytes |
| refreshResult | 124 (*) | Bytes |
| registerEncoded | 103 (+) | Bytes |
| registerEncrypted | 103 (+) | Bytes |
| registerPlain | 421 (+) | Bytes |

Tabelle 5.2: Länge der einzelnen LSP-Nachrichten

kleiner bzw. größer, allerdings durch die maximale Länge einer URL auf jeden Fall nach oben begrenzt. Sieht man von den *refresh* Anfragen ab, so sind die am häufigsten verwendeten Anfragen *lookup* und *registerEncoded* in ihrer Größe nach oben begrenzt und durchaus in annehmbaren Dimensionen.

## 5.2 Beschreibung der Testumgebung

Da sich die realen Bedingungen, in denen der Lokationdienst später einsetzbar sein soll, nicht in einem Labor mit einer beschränkten Anzahl von Rechnern an einem gemeinsame LAN wiederfinden, wurden folgende drei unterschiedliche Tests mit dem Prototyp durchgeführt, die jeweils verschiedene Extremsituationen simulieren sollten.

**(1)** In einem Netzwerk aus drei vergleichbar leistungsstarken Rechnersystemen am gleichen LAN wurde die Anzahl der ständig zyklisch migrierenden Agenten langsam erhöht. Ein Rechner war ausgezeichneter LS-Server mit Zuständigkeit für alle Agenten, ein anderer wurde zum LS-Proxy des LAN gewählt. Durch diesen Test sollte kontrolliert werden, wie aktuell die Einträge bei LS-Proxy und LS-Server sind und wie sich das Gesamtsystem bei steigender Last durch Erhöhung der Agentenzahl verhält. Gemessen wurde die Latenz der Anfragen und die Migrationsgeschwindigkeit der Agenten.

**(2)** Lokal auf einem Rechnersystem wurden durch eine *loop back* Schleife über den TCP/IP-Stack Anfragen an einen LS-Server auf dem `localhost` gesendet, wobei die Anfragefrequenz für die jeweiligen Anfragetypen getrennt langsam erhöht wurde. Hiermit sollte die Bearbeitungsgeschwindigkeit des LS-Proxy bzw. LS-Server mit integriertem `MemoryDB` Datenbankmodul getestet werden, um die maximal zu bewältigende Anfragefrequenz zu bestimmen. Gemessen wurde jeweils die Bearbeitungszeit vom Stellen der Anfrage bis zum Erhalt des Ergebnisses. Die Zeiten wurden gemittelt und ausgegeben.

**(3)** Wiederum ohne Agenten wurden die Anfragen eines LS-Client an eine initialisierte LS-Proxy/LS-Server Konfiguration simuliert. Diesmal wurden die Anfragen allerdings „gleichzeitig" von verschiedenen Rechnersystemen aus und mit unterschiedlichen Anfragetypen gestellt. Getestet werden sollte in diesem Fall zum einen der mögliche Grad der Parallelverarbeitung von Anfragen, zum anderen wiederum die Bearbeitungsgeschwindigkeit. Letztere wurde dann übrigens auch abhängig von der Anzahl der gespeicherten Einträge in der Datenbank des LS-Servers bestimmt.

Das Testszenario bestand in allen Fällen aus gleichwertigen Rechnersystemen mit folgenden Kenndaten:

| | |
|---|---|
| CPU | UIIi 333 MHz |
| RAM | 256 MB |
| Festplatte | 9 GB |
| Betriebssystem | Solaris 5.7 |
| Patch Level | Generic |
| Model | Ultra 10 |

Tabelle 5.3: Kenndaten der Rechnersysteme in der Testumgebung

Die Datenbank des LS-Proxy war dabei mit einer *timeout* Zeitspanne von 30 Sekunden für die enthaltenen Einträge, der LS-server mit einer *timeout* Zeitspanne von 60 Sekunden initialisiert. Durch die Synchronisation der Systemzeit auf allen Rechnersystemen ließen sich vergleichbare Zeitmessungen durchführen. Als Agentenplattform diente SeMoA in der Version 2 mit aktiviertem *verify*, *sign*, *decrypt*, *encrypt* und *policy* Filter. Jeder Agentenserver stellte einen potentiellen LS-Proxy dar, ein besonderer war als LS-Server ausgezeichnet.

## 5.3 Diskussion der Testergebnisse

In Test **(1)** blieb die Latenzzeit der LSP-Anfragen über den LS-Proxy an den LS-Server auch noch mit impliziter Fehlerbehandlung von LS-Client und LS-Proxy (siehe Abschnitt 4.4) unter einer Sekunde. Dabei stieg die Zeit, die ein einzelner Agent brauchte, um über zwei Agentenserver wieder zu seinem Ursprungsserver zu migrieren, schnell vom Sekunden- in den Minutenbereich: Lag die Dauer eines Zyklus' bei einem Agenten im System der drei Agentenserver noch bei knappen 15 Sekunden, so stieg sie bei 10 Agenten auf ca. 75 Sekunden und bei zwanzig dann auf mehr als 2 Minuten an. Dabei ist zu bemerken, dass die jeweilige Zeitspanne für die Migration von einem zum nächsten Agentenserver am Anfang fast ausschließlich vom *verify* Filter in Anspruch genommen wurde. Das diese Zeitspanne bei steigender Agentenzahl so stark anstieg lag nun nicht mehr an dem Filter sondern daran, dass die Agenten in der Migrationswarteschlange standen, also die meiste Zeit inaktiv waren. Grund dafür war die begrenzte Anzahl von Threads, die am `Ingate` des jeweiligen Zielservers Agenten in Empfang nehmen konnten. Erhöht war diese Wartezeit vor Allem beim LS-Proxy und Ls-Server, die parallel auch noch LSP-Anfragen über eigene Threads bearbeiteten, dies allerdings ohne merklich erhöhte Latenzzeit.

In einem abgeschlossenem System aus Agentenservern, migrierenden Agenten und dem zuständigen LS-Proxy ist die Migrationsgeschwindigkeit der Agenten also eindeutig der Flaschenhals. Der Lokationsdienst erfüllt hier mit angemessener Geschwindigkeit wunderbar seine Aufgabe. Erhöht man übrigens die Kapazität des `Ingate` und des LS-Proxy bzw. LS-Server, d.h. die Anzahl der Threads, die Daten aus dem Netzwerk entgegen nehmen können, so erhöht sich die Bearbeitungsgeschwindigkeit bei beiden Instanzen wieder etwas. Die Rechenleistung des jeweiligen Rechnersystems ist dabei irgendwann jedoch wieder begrenzender Faktor.

Betrachtet man einen LS-Server wie in Test **(2)** nun allerdings unabhängig von der Anzahl der mobilen Agenten in einem geschlossenem System als Server mit Zuständigkeit für alle Agenten mit einem bestimmten Hashpräfix im Internet, der weniger oder gar nicht als Agentenserver an der Migration von Agenten beteiligt ist, so sieht das Ergebnis anders aus: In diesem Fall lassen sich tatsächlich die Grenzen des Lokationsdienstes erreichen, die direkt mit den entsprechenden Ressourcen des darunter liegenden Rechnersystems korrelieren. Dabei ist allerdings zu bemerken, dass die mittlere Bearbeitungsgeschwindigkeiten (siehe Tabelle 5.4) der entsprechenden wichtigen Anfragen auch unter steigender Last nur sehr langsam anstiegen:

| | |
|---|---|
| registerEncrypted | 100 ms |
| registerEncoded | 12 ms |
| registerPlain | 10 ms |
| lookup | 9 ms |
| refresh (10 Einträge) | 100 ms |

Tabelle 5.4: Mittlere Bearbeitungszeit der wichtigen LSP-Anfragen

Durch die Organisation der `MemoryDB` Datenbank des LS-Servers mittels eines sortierten, balancierten Baums ließen sich ohne große Geschwindigkeitseinbußen bis zu 250000 Einträge verwalten. Dann wurde eine andere Ressource knapp, nämlich der Hauptspeicher des Rechnersystems: Java meldete entsprechende *exceptions* und beendete dadurch nach und nach die Threads des Lokationsdienstes, der anschließend gar nicht mehr in der Lage war Anfragen entgegenzunehmen. Je nachdem, wie viele Positionseinträge ein LS-Server also speichern soll, müssen seine Ressourcen bzw. die Struktur der Datenbank ausgelegt sein. Übersteigen die Anfragen an den Server dessen Kapazität in dieser Hinsicht, so muss bei diesem entweder das entsprechende Speichermedium aufgerüstet werden, oder der abgedeckte Hashpräfix sollte durch Umstrukturierung auf mehrere LS-Server verteilt werden.

Gerade auch durch Test **(3)** hat sich noch eine andere Überlastungsmöglichkeit eines LS-Servers gezeigt: Übersteigt die Frequenz parallel an einen Server gestellter Anfragen, die durch die Bearbeitungsdauer pro Anfrage (siehe Tabelle 5.4) für den Server maximal mögliche Frequenz, so werden weitere freie Threads in Anspruch genommen, die für die Annahme von Anfragen zur Verfügung stehen. Dieser `ThreadPool` kann kurzzeitig einen Puffer darstellen, bis die Anfragefrequenz wieder etwas nachlässt. Ist das aber nicht der Fall, so findet sich bald kein freier Thread mehr, und weitere Anfragen werden mit einer `IOException` abgelehnt.

Abhängig von der Leistung des unter dem Agentenserver liegenden Rechnersystems kann die

Anzahl der Threads für die Annahme von Anfragen also optimiert werden. Falls der Server trotzdem überlastet wird, sind ähnliche Überlegungen anzustellen wie oben beschrieben.

Sofern ein Agent innerhalb eines LAN mirgriert, so übernimmt der LS-Proxy für dieses Subsystem automatisch die Rolle des LS-Servers, wenn dieser überlastet ist. Wird also eine Anfrage an den LS-Server auf Grund einer `IOException` abgeblockt, so ist das in diesem Fall nicht wirklich tragisch. Allerdings ist dabei zu bemerken, dass die Position der betreffenden Agenten dann nur innerhalb ihrer aktuellen Subsystemen zu ermitteln ist und der nächste Eintrag beim LS-Server erst wieder registriert werden kann, wenn dieser nicht mehr überlastet ist und der Eintrag mit mittlerweile altem Cookie durch ein *timeout* gelöscht wurde.

Man kann abschließend also sagen, dass der entworfene Lokationsdienst durchaus die wichtigsten Anforderungen erfüllt und seine Verfügbarkeit durch die Regulierung von den Ressourcen Speicher, Rechnerleistung einerseits und den Parametern timeout, refresh, capacity des Lokationsdienstes andererseits optimiert werden kann. Darüber hinaus ist es wichtig, sich vor der eigentlichen Installation Gedanken über die gesamte Infrastruktur aus LS-Proxy und LS-Server Komponenten zu machen. Bei der zukünftigen Anwendung dieses Loaktionsdienstes im Internet wird sich zeigen, in wie weit die integrierte Fehlerbehandlung mögliche Netzwerkprobleme tatsächlich ausgleicht und ob die Aktualität von Positionseinträgen in den Datenbanken der LS-Server den größeren Transportverzögerungen von Nachrichten in diesem Medium trotzen kann. Durch die gut durchdachte, fundierte Modellentwicklung und die bereits durchgeführten Tests kann ich dem Ganzen allerdings sehr viel Optimismus entgegen bringen.

# Kapitel 6

# Resümee

## 6.1 Zusammenfassung und Schlussfolgerungen

Im Rahmen dieser Diplomarbeit wurde ein Lokationsdienst für mobile Objekte entwickelt, der es unter anderem dem Besitzer eines mobilen Objekts (eines mobilen Agenten im Speziellen) jederzeit ermöglicht, während dessen Migration durch das Netzwerk dessen aktuelle Position zu ermitteln. Dabei wurde beim Modellentwurf besonderer Wert auf die Analyse, Formulierung und Erfüllung der speziellen Anforderungen an einen solchen Dienst gelegt, auch im Vergleich zu bereits existierenden Namensdiensten und Mobile Agenten Systemen. Es stand die Präsentation eines *sicheren* und *skalierbaren* Modells im Vordergrund, in welchem Attacken „bösartiger" Angreifer erkannt und gegebenenfalls verhindert werden, sowie Fehler, bei der Arbeit im Netzwerk auftretender Fehlerquellen, einkalkuliert und entsprechend behandelt werden, soweit dies möglich ist.

Im ersten Teil der Arbeit wurde ein Überblick über die von dieser Arbeit berührten Bereiche der Informatik gegeben: Agententechnologie, Namens- und Verzeichnisdienste und Skalierbarkeit in verteilten Systemen. Es wurde versucht, die Themen so weit zu besprechen, dass eine fundierte theoretische Grundlage für die Beschreibung der Entwicklungsentscheidungen vorlag.

Der zweite Teil beschreibt die Entwicklung eines Dienstes, der es ermöglicht die Position eines mobilen Objekts über eine Infrastruktur aus Clients, Proxys und Servern auf eine effiziente Art und Weise zu registrieren und anschließend wieder abzufragen. Es wird die Aufgabe der einzelnen Komponenten und ihre Interaktion miteinander beschrieben. Neben der Spezifikation der Schnittstellen zählt hierzu auch der Entwurf eines speziellen Protokolls zur sicheren Kommunikation von Clients und Servern im Netzwerk. Des weiteren wird eine Komponente zur dynamischen Konfiguration und Erweiterung dieses Systems dargestellt. Die Entwicklungsphase abschließend folgt die Analyse konkreter Angriffsszenarien und die Diskussion von Fehlerquellen.

Durch die Implementierung eines Funktionsmodells und dessen Integration in die Mobile Agenten Plattform SeMoA konnte sich die Funktionstüchtigkeit des Modells zeigen und mit diesem Prototyp gleichzeitig die Anpassung des generischen Modells an eine konkrete Arbeitsumgebung erläutert werden.

Die bereits durchgeführten Tests sind vielversprechend. Innerhalb der beschriebenen Testumgebung ließen sich sehr gute Ergebnisse erzielen und durch die Architektur des Lokationsdienstes, die abgesehen von dem Modell des LS-Proxy zur Verbesserung der Leistungsfähigkeit und Fehlertoleranz, einen direkten Kontakt zwischen Client und Server vorsieht, existieren bei der Kommunikation zwischen den Komponenten keine aktiven Zwischenstationen. Im Gegensatz zu vielen existierenden Namens- und Lokationsdiensten werden die benötigten Informationen dadurch also auch in globalen Netzwerken wie dem Internet direkt und im günstigen Fall durch eine einzige Anfrage mit ensprechender Antwortnachricht übermittelt.

## 6.2 Ausblick und Anwendungen

Die zeitliche Begrenzung der Arbeit war Grund dafür, dass der Prototyp vorerst nur die Hauptfunktionalität des beschriebenen Modells implementiert. Was diesen Aspekt betrifft steht die Umsetzung des LS-AdminServers und des LS-RelayAgents noch aus:

Die Komponente LS-AdminServer soll die dynamische Konfiguration der LS-Server Infrastruktur erleichtern. Da die Konfiguration der LS-Server über Initialisierungsdateien aber bereits, in dem abgeschlossenen Modul `ServerInfo` gekapselt, implementiert wurde, ist die zukünftige Einbindung dieser Komponente in das Funktionsmodell über die gegebene Schnittstelle sehr einfach. Das gleiche gilt für den LS-RelayAgent, der leicht als zusätzliches Sicherheitsmodul mit in das Funktionsmodell eingebracht werden kann. Dabei ist übrigens zu bemerken, dass der Lokationsdienst auch ohne diese beiden Komponenten bereits voll funktionsfähig und im Forschungsrahmen der SeMoA-Plattform einsetzbar ist.

Für die erarbeitete Lösung ergibt sich auf Grund ihrer durchdachten und auf Erweiterbarkeit ausgelegten Architektur die Chance, in zwei Richtungen in neue Projekte einfließen zu können. Zunächst sind folgende Ansätze zu nennen, die den eigentlichen Lokationsdienst erweitern:

- Die Implementierung eines Datenbank *backend* im Rahmen der `StorageDB` Schnittstelle.
- Ein standardisierter *logging* Mechanismus könnte das jetzige Modul ersetzen und komfortablen Zugriff bzw. Kontrolle über aufgezeichnete Informationen ermöglichen.
- Verbesserte Sicherheit im Bereich der Anbindung des Lokationsdienstes an den Agentenserver durch ausgiebigere Nutzung des *access controlers* von Java, *policy files* oder der Schnittstellenkapselung durch Proxy-Objekte.
- Die Umsetzung des in Abschnitt 3.1.1 angesprochenen Namensdienstes zur Unterstützung von benutzerfreundlichen Bezeichnern für Agenten bzw. mobile Objekte.
- Unterteilung des LS-AdminServers in eine auf DNS aufbauende Hierarchie von LS-AdminServern verschiedener *sub domains* (.de, .fr, ...)
- Durch eine Erweiterung des LS-Server wird eine direkte *lookup* Anfrage an LS-Server über URLs des folgenden Formats vorstellbar:

  `lsp://<nameserver>:<port>/<implicitName>`

- Ein Proxy-Modell mit mehreren aktiven LS-Proxy in einem LAN, die sich den Netzwerkverkehr zu den LS-Server teilen, aber mit: $\# LSProxy < \# LSServer$
- Aufspaltung des LS-InfoGUI in mehrere Komponenten zur Ermöglichung von Fernwartung: Ein Agent, der zu einem LS-Proxy bzw. LS-Server gesendet wird, lokal eine autorisierte Schnittstelle zur jeweiligen Komponente registriert und die Informationen via SSL zu einem LS-InfoGUI auf einem anderen Rechnersystem transportiert.

Unabhängig von der Weiterentwicklung des Lokationsdienstes an sich, steht aber schon mit dem gegebenen Prototyp die Tür für eine wichtige andere Entwicklung offen:

Der Lokationsdienst stellt in seiner Gesamtheit eine wichtige Komponente einer Mobile Agenten Plattform dar, dessen Vorhandensein bereits als Qualitätsmerkmal der Plattform genannt werden kann. Darüber hinaus kann er aber auch die Basis für einen Nachrichtendienst bilden, der es Agenten und einem Agentenserver ermöglicht, andere Agenten zu kontaktieren, ohne dass diese sich über den aktuellen Aufenthaltsort der Empfänger gewahr sein müssen.

Auf diese Weise wird es möglich, eine Vielzahl an Protokollen umzusetzen, welche die beschriebene Kommunikation zwischen Agenten bzw. Agentenservern und Agenten voraussetzen. Oder es werden wiederum andere Protokolle und Komponenten auf der Basis des Kommunikationsdienstes aufgesetzt: Zu nennen sind in diesem Kontext eine Implementierung von KQML oder ACL als Erweiterung des Kommunikationsdienstes (siehe Anhang B), ein Dienst der über Nachrichten den passiven und/oder aktiven Zugriff auf einen Agentenserver erlaubt oder die Umsetzung von Sicherheitsprotokollen für Agenten (siehe z.B. [74]).

Diese Arbeit bildet also einen weiteren Grundstein für interessante Entwicklungen auf dem Gebiet der mobilen Agenten und Objekte.

# Anhang A

# ASN.1

## A.1 Eine Möglichkeit der Syntaxdeklaration

Durch die ITU 1997 zum Standard erhoben, stellt die *Abstract Syntax Notation One (ASN.1)* eine Sprache dar, die verwendet wird um abstrakte Datentypen zu definieren. Dem Designer stehen dafür eine Reihe von Basisdatentypen zur Verfügung, die sich durch Konstruktoren zu komplexen Datentypen zusammensetzten lassen. Dadurch entsteht eine Hierarchie, die sich in Typen und Werte, Subtypen und Module aufteilt. Dem gleichen Basistyp können, durch Zuordnung verschiedener Bezeichner und anderer Gültigkeitsbereiche, verschiedene Bedeutungen zugewiesen werden, was neben der eigentlichen Syntaxspezifikation des abstrakten Datentyps auch eine Angabe der Semantik erlaubt.

## A.2 Kodierung der Datentypen

Darüber hinaus stellt ASN.1 mittlerweile mehrere standardisierte Kodierungsvarianten zur Verfügung, mit denen ein definierter Datentyp in einen *byte stream* umgesetzt und anschließend auch wieder dekodiert werden kann. Jedem Modul der ASN.1-Deklaration werden dafür implizit oder explizit eindeutige *Object Identifier (OID)* zugewiesen, die nach der Kodierung eines Datentyps dessen eindeutige Dekodierung ermöglichen.

Die *Basic Encoding Rules (BER)* sehen eine Umwandlung eines jeden Datenwertes einer ASN.1-Spezifikation in eine Ganzzahl von Bytes vor, bei der abhängig vom Typ des Wertes und davon, ob dessen Länge bekannt ist, eine von drei alternative Methoden gewählt werden kann. In jeder dieser Methoden besteht die BER Kodierung aus drei oder vier der folgenden Teilen: *identifier*, *length*, *contents*, *end-of-contents*. Bei den *Distinguished Encoding Rules (DER)* und den *Canonical Encoding Rules (CER)*, die eine echte Untermenge von BER darstellen, wird durch restriktivere Regeln die Eindeutigkeit der Kodierung erzwungen. Aufgrund der beklagten Ineffizienz von BER (z.B. der Kodierung eines *boolean* in drei Bytes) wurden zusätzlich aber auch die sogenannten *Packed Encoding Rules (PER)* als neue Kodierungsvariante entwickelt, die den Speicherbedarf für abstrakte Datentypen bei der Kodierung minimieren. Es ist aber durchaus möglich, die Kodierung hingegen in ihrer Einfachheit zu

optimieren oder Sicherheiten gegen Veränderungen der Nachricht auf dem Übertragungsweg einzubringen.

Die genauen Spezifikationen von ASN.1 und die Beschreibung der Standard-Kodierungsvarianten finden sich in X.680-X.683 [67][68][69][70] und X.690-X.691 [71][72].

## A.3 Protokolldefinition mittels ASN.1

Durch die genannten Eigenschaften eignet sich ASN.1 ausgezeichnet für den Entwurf von Kommunikationsprotokollen in heterogenen Netzwerken. Zuerst werden die Typen der zu übermittelnden Nachrichten auf einem hohen Abstraktionsniveau definiert. Dies geschieht unabhängig von später bei der Kommunikation beteiligten Rechnersystemen, Programmiersprachen oder der Repräsentation der Nachricht während des Transports über das Netzwerk.

Erst jetzt wird der Satz an Kodierungsregeln gewählt: Ein ASN.1 Compiler produziert dann aus der ASN.1-Definition die entsprechenden Datenstrukturen einer konkreten Programmiersprache und den Programmcode für Decoder und Encoder.

Durch ASN.1-Spezifikation und dem Wissen über die Kodierungsmethode ist es anschließend also jedem Gegenüber möglich die empfangenen Daten zu dekodieren und zu interpretieren.

Neben diesem kurzen Überblick über ASN.1 finden sich in den Artikeln [37] und [85] jeweils weitere Einführungen in das Thema. Eine sehr gute Referenz mit ausführlicher Beschreibung der Entwurfsmethodik stellen [41] und [12] dar.

Die Protokolle zwischen den Komponenten des in Kapitel 3 entwickelten Lokationsdienstes befinden sich in Anhang C und sind in eben dieser Abstract Syntax Notation One spezifiziert.

## A.4 Überblick über die Datentypen von ASN.1

Die folgenden Tabellen wurden direkt aus [12] übernommen:

| | |
|---|---|
| `BOOLEAN` | Logical values TRUE and FALSE |
| `NULL` | Includes the single value NULL, used for delivery report or some alternatives of the `CHOICE` type (particularly for the recursive types) |
| `INTEGER` | Whole numbers (positive or negative), possibly named |
| `REAL` | Real numbers represented as Floats |
| `ENUMERATED` | Enumeration of identifiers (state of a machine for instance) |
| `BIT STRING` | Bit strings |
| `OCTET STRING` | Byte strings |
| `OBJECT IDENTIFIER, RELATIVE-OID` | Unambiguous identification of an entity registered in a worldwide tree |
| `EXTERNAL, EMBEDDED PDV` | Presentation (6th layer) context switching types |
| `[...]String` | Various types of character strings (see Table A.3) |
| `CHARACTER STRING` | Allows negotiation of a specific alphabet for character strings |
| `UTCTime, GeneralizedTime` | Dates |

Tabelle A.1: Basistypen von ASN.1

| | |
|---|---|
| `CHOICE` | Choice between types |
| `SEQUENCE` | Ordered structure of values of (generally) dierent types |
| `SET` | Non-ordered structure of values of (generally) different types |
| `SEQUENCE OF` | Ordered structure of values of the same type |
| `SET OF` | Non-ordered structure of values of the same type |

Tabelle A.2: Konstruierte Typen von ASN.1

| | |
|---|---|
| `NumericString` | "0" to "9", `space` |
| `PrintableString` | "A" to "Z", "a" to "z", "0" to "9", `space`, "'", "(", ")", "+", ",", "-", ".", "/", ":", "=", "?" |
| `VisibleString`, `ISO646String` | ISOReg [36] entry no. 6; `space` |
| `IA5String` | ISOReg [36] entry no. 1 & 6; `space`, `delete` |
| `TeletexString`, `T61String` | ISOReg [36] entry no. 6, 87, 102, 103, 106, 107, 126, 144, 150, 153, 156, 164, 165, 168; `space`, `delete` |
| `VideotexString` | ISOReg [36] entry no. 1, 13, 72, 73, 87, 89, 102, 108, 126, 128, 129, 144, 150, 153, 164, 165, 168; `space`, `delete` |
| `GraphicString` | all the graphical sets (called 'G') of ISOReg [36]; `space` |
| `GeneralString` | all the graphical sets (called 'G') and all the control characters (called 'C') of ISOReg [36]; `space`, `delete` |
| `UniversalString` | ISO10646-1 [1] |
| `BMPString` | the basic multilingual plane ISO10646-1 [1] (65,536 cells) |
| `UTF8String` | ISO10646-1 [1] |

Tabelle A.3: Das Alphabet der Stringtypen von ASN.1

# Anhang B

# KQML und ACL

## B.1 Kommunikationsprotokolle für Agenten

Nachdem in dieser Arbeit ein Lokationsdienst für mobile Agenten entworfen wurde, ist über einen auf dem Lokationsdienst aufsetzendem Nachrichtendienst der Wissensaustausch zwischen zwei Agenten bzw. zwischen Agent und Softwaresystem vorstellbar. Um diesen Wissensaustausch zu realisieren wird eine *Sprache* benötigt, welche die beteiligten Komponenten verstehen, und die sich wiederum in die drei Ebenen Protokoll, Ontologie und Wissensbeschreibung aufteilen lässt.

Da das Wissen der Agenten innerhalb eines Agentensystems unterschiedlich repräsentiert sein kann, muss festgelegt werden, in welcher Beschreibungssprache das eigentliche Wissen übertragen wird. Dies geschieht in der *Wissensbeschreibungs-Ebene*, wobei sich eine Programmiersprache wie Lisp oder Prolog oder ein Wissensaustauschformat wie KIF [27] eignet.

Die *Ontologie-Ebene* beschreibt auf welche Art welches Wissen repräsentiert wird. Dadurch kann die Komplexität des universellen Wissens auf die bestimmte Repräsentation einzelner Themengebiete eingeschränkt werden.

Kommunikationsprotokolle dienen auf der *Protokoll-Ebene* in Form einer Metasprache nun als Nachrichtencontainer, die eingebettetes Wissen in verschiedenen Repräsentationen transportieren können. Durch sie werden eine eindeutige Nachrichtenidentifikation, Sender und Empfänger der Nachricht angegeben bzw. der Inhalts der Nachricht beschrieben. Diese Daten werden zum Adressieren, beim Transport und beim *routing* der Nachrichten verwendet.

Die beiden bekanntesten Kommunikationsprotokolle sind KQML und ACL.

## B.2 Knowledge Query and Manipulation Language

Die *Knowledge Query and Manipulation Language (KQML)* wurde Anfang der 90er Jahre von dem *DARPA Knowledge Sharing Effort (DARPA KSE)* Konsortium entwickelt. Neben der Beschreibung von KQML als Agenten-Kommunikationssprache in [17] findet sich in [39] auch die genaue Spezifikation des Protokolls.

KQML besteht aus einer Menge von sogenannten *Performativen*, Kommandos in der Ascii-Repräsentation der *Common Lisp Polish-prefix notation*. Durch diese Performativen können verschiedene Kommunikations-Szenarien umgesetzt werden: Clients können durch KQML synchron oder asynchron Anfragen an einen Server stellen und daraufhin ein bzw. mehrere Antwortpakete erhalten.

Das Protokoll sieht zusätzlich das mögliche Vorhandensein eines sogenannten *communication facilitators* vor, der die Aufgabe hat, Dienstnamen zu registrieren, Nachrichten inhaltsbasiert zu routen, Informationen zwischen Anbietern und Klienten zu vermitteln oder in eine andere Repräsentation zu übersetzen. Es kann ein zentraler Communication Facilitator benutzt werden, dessen Adresse als Initialisierungsparameter den Agenten übergeben wird. Andernfalls stellt die KQML-API eine Funktion zur Verfügung, um den Communication Facilitator zu lokalisieren.

Die Voraussetzung von KQML an die Transportebene ist die Verbindung von Agenten durch einen unidirektionalen Kommunikationskanal, in dem zuverlässig einzelne Nachrichten übermitteln werden können. Diese Nachrichten sollen in der gleichen Reihenfolge beim Empfänger eintreffen, in der sie abgesendet worden sind. Bisherige KQML-Implementierungen setzen z.B. auf TCP, SMTP oder CORBA auf. Diese Forderungen könnten aber durchaus leicht von einem Nachrichtendienst erfüllt werden, der auf dem in dieser Arbeit entworfenem Lokationsdienst aufbaut.

## B.3 Agent Communication Language

Die *Agent Communication Language (ACL)* wurde 1997 als Gegenvorschlag zu KQML von der *Foundation for Intelligent Physical Agents (FIPA)* veröffentlicht. Ein Überblick und Vergleich zu KQML findet sich in [8], die Spezifikation wird in [19] [20] gegeben.

ACL orientiert sich weitgehend an dem älteren KQML, allerdings fehlen die Facilitator-Performativen. Dafür erlaubt der zusätzlich definierte Parameter `:envelope` eine genauere Kategorisierung von ACL-Nachrichten. Außerdem wird von der FIPA ein Plattformreferenzmodell definiert, in dem Dienste wie *white and yellow pages*, *message routing*, *exception handling* und *life cycle management* enthalten sind, die von speziellen Agenten verwaltet werden. Damit ist ACL ein fester Bestandteil des FIPA-Entwurfs, der ein Gesamtkonzept für die Kommunikation zwischen Softwareagenten darstellt. Diese Festlegungen erschweren den Einsatz von ACL im Gegensatz zu KQML etwas, wenn es sich um die Einbettung in ein bestehendes Agentensystem handelt.

## B.4 Beispiel-Nachricht in KQML bzw. ACL

Folgende Nachricht, die sowohl in KQML als auch in ACL verfasst werden könnte, veranschaulicht abschließend beispielhaft die allgemeine Struktur, in der Informationen in den beiden Kommunikationssprachen angefragt bzw. weitergegeben werden können.

```
(inform
     :sender        Agent1
     :receiver      Auction-Server
     :in-reply-to   round004
     :reply-with    bid004
     :language      LISP
     :ontology      auction
     :content       (price (bid good03) 150)
 )
```

Abbildung B.1: Eine Beispiel-Nachricht in KQML bzw. ACL

In diesem Fall informiert ein Agent `Agent1` den Auktionsserver `Auction-Server` über sein aktuelles Gebot für die Ware `good03`.

# Anhang C

# Spezifikationen

## C.1 Location Service Protocol (LSP)

### C.1.1 Spezifikation in ASN.1:1997

Das *Location Service Protocol (LSP)* wird hier in der aktuellen Version des ASN.1-Standards von 1997 spezifiziert (siehe auch Anhang A). Dabei wurden die möglichen Status Codes der `LSP_Reply` Datenstruktur, angelehnt an die HTTP Status Codes [64], in Klassen aufgeteilt. Die für `LSP_RegisterEncrypted` verwendete Datenstruktur `EnvelopedData` stammt aus der Spezifikation des PKCS#7-Standards [55].

```
-- Location Service Protokoll (LSP) - Version 1.0
-- Spezification in ASN.1:1997


-- Request Message

LSP_REQUEST ::= CLASS
{
  &requestType   INTEGER UNIQUE,
  &requestBody
}
WITH SYNTAX {&requestType &requestBody}

lookupRequest            ::= LSP_REQUEST { 1 LSP_Lookup }
registerPlainRequest     ::= LSP_REQUEST { 2 LSP_RegisterPlain }
registerEncryptedRequest ::= LSP_REQUEST { 3 LSP_RegisterEncrypted }
registerEncodedRequest   ::= LSP_REQUEST { 4 LSP_RegisterEncoded }
refreshRequest           ::= LSP_REQUEST { 5 LSP_Refresh }
listRequest              ::= LSP_REQUEST { 6 LSP_List }
proxyInvalidateRequest   ::= LSP_REQUEST { 7 LSP_ProxyInvalidate }
```

```
Requests LSP_REQUEST ::=
{
  lookupRequest |
  registerPlainRequest |
  registerEncryptedRequest |
  registerEncodedRequest |
  refreshRequest |
  listRequest |
  proxyInvalidateRequest
}

LSP_Request ::= SEQUENCE
{
  version   INTEGER {lspVer1(1)} (lspVer1),

  type      LSP_REQUEST.&requestType({Requests}),
  body      LSP_REQUEST.&requestBody({Requests}{@type})
}


-- Reply Message

LSP_REPLY ::= CLASS
{
  &replyType   INTEGER UNIQUE,
  &replyBody
}
WITH SYNTAX {&replyType &replyBody}

stateOnlyReply ::= LSP_REPLY { 0 }
lookupReply    ::= LSP_REPLY { 1 LSP_Lookup_Result }
refreshReply   ::= LSP_REPLY { 5 LSP_Refresh_Result }
listReply      ::= LSP_REPLY { 6 LSP_List_Result }

Replies LSP_REPLY ::=
{
  stateOnlyReply |
  lookupReply |
  listReply |
  refreshReply
}

LSP_Reply ::= SEQUENCE
{
```

```
  version   INTEGER {lspVer1(1)} (lspVer1),

  state     INTEGER
  {
    -- Success
    acknowledge('0100'H),
    implicitNameNotFound('0101'H),

    -- Redirection
    someImplicitNamesNotPresent('0200'H),

    -- Client Error
    cookieInvalid('0400'H),
    contactAddressNotExistent('0401'H),
    notAuthorized('0402'H),
    implicitNameNotPresent('0403'H),

    -- Server Error
    wrongVersion('0800'H),
    requestTypeInvalid('0801'H),
    encodingNotSupported('0802'H),
    encryptingNotSupported('0803'H),

    -- IO Error
    ioError('01000'H),
    internalClientError('1001'H),
    requestBodyInvalid('1002'H),
    encodedDataInvalid('1003'H),
    encryptedDataInvalid('1004'H),

    -- Error Flags
    clientErrorFlag('010000'H),
    proxyErrorFlag('020000'H),
    serverErrorFlag('040000'H)
  },

  type     LSP_REPLY.&replyType({Replies}),
  body     LSP_REPLY.&replyBody({Replies}{@type})
}


-- Requests

LSP_Lookup ::= ImplicitName
```

```
LSP_RegisterPlain ::= SEQUENCE
{
  implicitName     ImplicitName,
  contactAddress   ContactAddress,
  newCookie        Cookie,
  currentCookie    Cookie
}

LSP_RegisterEncrypted ::= EnvelopedData     -- from PKCS#7

LSP_RegisterEncrypted_PlainData ::= SEQUENCE
{
  implicitName     ImplicitName,
  contactAddress   ContactAddress,
  newCookie        Cookie,
  currentCookie    Cookie
}

LSP_RegisterEncoded ::= SEQUENCE
{
  implicitName        ImplicitName,
  contactAddress      ContactAddress,
  encodedNewCookie    EncodedData,
  mac                 EncodedData
}

LSP_RegisterEncoded_PlainNewCookie ::= Cookie

LSP_Refresh ::= ImplicitNameList

LSP_ProxyInvalidate ::= ImplicitName

LSP_List ::= Timestamp


-- Replies

LSP_Lookup_Result ::= ContactAddress

LSP_List_Result ::= EntryList

LSP_Refresh_Result ::= ImplicitNameList


-- Basic datatypes
```

```
ContactAddress ::= OCTET STRING
ImplicitName   ::= OCTET STRING
Cookie         ::= OCTET STRING
EncodedData    ::= OCTET STRING
Timestamp      ::= INTEGER

ImplicitNameList ::= SET OF ImplicitName
EntryList        ::= SET OF Entry

Entry ::= SEQUENCE
{
  implicitName     ImplicitName,
  contactAddress   ContactAddress,
  cookie           Cookie,
  timestamp        TimeStamp
}
```

### C.1.2 Spezifikation in ASN.1:1994

Da die Implementierung des Protokolls in Java auf der Umsetzung der älteren ASN.1-Version von 1994 aufbaut, befindet sich die Spezifikation der beiden Datenstrukturen `LSP_Request` und `LSP_Reply` im Folgendem noch einmal in dieser Version. Hier wird die Syntax `ANY DEFINED BY` verwendet, die in dieser Form in der aktuellen Version von ASN.1 nicht mehr existiert, sondern durch den mächtigeren und präziseren Mechanismus der Informationsobjekte ersetzt wurde (siehe oben).

```
-- Location Service Protokoll (LSP) - Version 1.0
-- Spezification in ASN.1:1994


LSP_Request ::= SEQUENCE
{
  version   INTEGER {lspVer1(1)} (lspVer1),

  requestType   INTEGER
  {
    lookupRequest(0),
    registerPlainRequest(1),
    registerEncryptedRequest(2),
    registerEncodedRequest(3),
    refreshRequest(4),
    listRequest(5),
    proxyInvalidateRequest(6),
  },
```

```
  requestBody   ANY DEFINED BY requestType
}

LSP_Reply ::= SEQUENCE
{
  version   INTEGER {lspVer1(1)} (lspVer1),

replyState   INTEGER
  {
    -- Success
    acknowledge('0100'H),
    implicitNameNotFound('0101'H),

    -- Redirection
    someImplicitNamesNotPresent('0200'H),

    -- Client Error
    cookieInvalid('0400'H),
    contactAddressNotExistent('0401'H),
    notAuthorized('0402'H),
    implicitNameNotPresent('0403'H),

    -- Server Error
    wrongVersion('0800'H),
    requestTypeInvalid('0801'H),
    encodingNotSupported('0802'H),
    encryptingNotSupported('0803'H),

    -- IO Error
    ioError('01000'H),
    internalClientError('1001'H),
    requestBodyInvalid('1002'H),
    encodedDataInvalid('1003'H),
    encryptedDataInvalid('1004'H),

    -- Error Flags
    clientErrorFlag('010000'H),
    proxyErrorFlag('020000'H),
    serverErrorFlag('040000'H)
  },

  replyType   INTEGER
  {
    stateOnlyReply(0),
```

```
    lookupReply(1),
    listReply(5),
    refreshReply(6)
  },

  replyBody   ANY DEFINED BY replyType OPTIONAL
}
```

## C.2 Paketdiagramme

Der Lokationsdienst wurde in eine bestehende Paket- und Klassenstruktur eingebunden. Die folgenden drei Abschnitte geben nun einen Überblick über die neu entwickelten Klassen, die Pakete und Klassen, die in unveränderter Form Verwendung fanden, und die Klassen, die angepasst werden mussten.

### C.2.1 Der entwickelte Lokationsdienst

Abbildung C.1 zeigt das Paketdiagramm des entwickelten Lokationsdienstes. Es entspricht den UML-Konventionen, wobei die aufgeführten grau-geschriebenen Klassennamen des `DE.FhG.IGD.util` Pakets Klassen kennzeichnen, die bereits vorhanden waren und ohne Änderung übernommen wurden.

### C.2.2 Verwendete SeMoA-Pakete und Klassen

In Abbildung C.2 sind alle Klassen der SeMoA-Plattform dargestellt, die vom Lokationsdienst verwendet werden. In diesem Fall bezeichnen die mit **(M)** gekennzeichneten Klassennamen des `DE.FhG.IGD.semoa.server` Pakets, im Gegensatz zu allen anderen, solche Klassen, die angepasst werden mussten.

### C.2.3 Verwendete codec-Pakete

Auschließlich von Klassen des `DE.FhG.IGD.atlas.lsp` Pakets des Lokationsdienstes werden die in Abbildung C.3 dargestellten Pakete in unveränderter Form zur Implementierung des in ASN.1 spezifiezierten LSP verwendet. In Anhang A und in [55], [66] und [30] finden sich tiefergehende Informationen zu entsprechenden Standards.

## C.3 Klassenstruktur des Lokationsdienst

Hier sei nur kurz auf den ausführlich dokumentierten Quellcode des entwickelten Funktionsmodells verwiesen der aus platzgründen hier nicht aufgeführt wird. Zudem existiert eine aus dem Quellcode generierte HTML-Dokumentation aller Klassenschnittstellen.

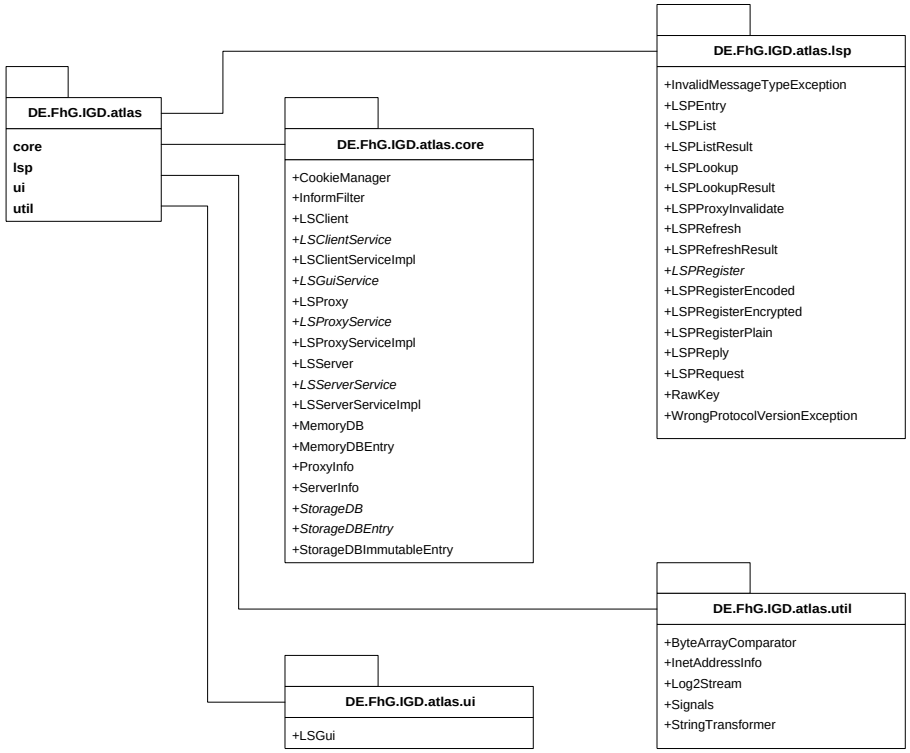


Abbildung C.1: Paketdiagramm des entwickelten Lokationsdienstes

## C.4 Initialisierungsdatei für LSClient und LSProxy

Hier findet sich ein Beispiel für eine Initialisierungsdatei, die beim Start des LS-Client und LS-Proxy eingelesen wird, um im Speicher den *cache* mit den entsprechenden Informationen über installierte LS-Server zu füllen:

```
### LS-Server configuration file


# With PrefixBitLength you set the number of bits,
# which are used to map implicit names to their corresponding LS-Server
#
#   PrefixBitLength = 3, means that the first three bits of an
#                        implicit name (from the left) are used
#                        to map it to its corresponding LS-Server
#
# In this case you need to specify 2^3=8 LS-Server.

PrefixBitLength = 2
```

```
# A new LS-Server is specified with following entries
#
#   x.ServerPrefix = up to 20 bytes long (hex), seperated by spaces
#   x.ServerDN     = the distinguished name of the server
#   x.ServerURL    = the url of the server (the potocol must be "lsp")
#

1.ServerPrefix = 00
1.ServerDN     = CN=LSServer1,OU=IGD-A8,O=FhG,L=Darmstadt,ST=Hessen,C=DE
1.ServerURL    = lsp://LSServer1.igd.fhg.de:50000

2.ServerPrefix = 40
2.ServerDN     = CN=LSServer2,OU=IGD-A8,O=FhG,L=Darmstadt,ST=Hessen,C=DE
2.ServerURL    = lsp://LSServer2.igd.fhg.de:50000

3.ServerPrefix = 80
3.ServerDN     = CN=LSServer3,OU=IGD-A8,O=FhG,L=Darmstadt,ST=Hessen,C=DE
3.ServerURL    = lsp://LSServer3.igd.fhg.de:50000

4.ServerPrefix = c0
4.ServerDN     = CN=LSServer4,OU=IGD-A8,O=FhG,L=Darmstadt,ST=Hessen,C=DE
4.ServerURL    = lsp://LSServer4.igd.fhg.de:50000
```

## C.5 Initialisierungsdateien für SeMoA

Zur Integration des Lokationsdienstes in SeMoA 2 müssen noch einige Ergänzungen zu den bestehenden Initialisierungsdateien hinzu gefügt werden.

### C.5.1 rc

```
#
# Original Version
#
# Boots a SeMoA server based on a given configuration
# script. I refer to this script as rc.conf though it
# can have any name.
#
# Start the server as follows:
#
#   java DE.FhG.IGD.semoa.server.Shell -f rc rc.conf
#
# A number of other scripts will be read and run by
# this one. The 'rc.conf' script allows to switch
# some features of the server on and off without too
```

```
# much thinking. Have a look at the sample provided
# with this script. Please remember that slashes are
# used as file separators even though the current
# platform's path separator is different (e.g. a '\'
# as on a windoze box).
#
# The sample 'rc.init' requires that all rc scripts
# are in ${user.home}/java/cvs/examples/boot. This
# can be changed by setting ${RC_BASE} to a different
# path in your configuration file.
#
# If you change anything herein then replace 'Original'
# at the beginning of this script by something else
# such as 'Custom'.
#
# Author  : Jan Peters
# Version: $Id: whatis.conf,v 1.0 2000/12/15 12:00:00 jpeters Exp $
#

#
# Read in the custom 'rc.conf' script before everything
# else.
#
source ${0}

#
# Read in aliases for commands; makes life a little
# easier.
#
source ${RC_BASE}/rc.aliases

#
# Setup rudimentary server and shell support. Variable
# ${WHATIS_PATH} should be specified in your rc.conf file.
#
echo
echo "Basic services"
echo "--------------"
echo -n "Setup:"
echo -n " WhatIs"
java DE.FhG.IGD.util.WhatIs -init ${WHATIS_PATH}
echo -n " ConsoleFilter"
java DE.FhG.IGD.util.ConsoleFilter
echo "."
```

```
#
# Read in and run the script that configures basic security
# services. Use the ${ENABLE_SSL} variable in rc.conf to
# enable or disable SSL services and gateways.
#
source ${RC_BASE}/rc.security

#
# Run the script that takes care of launching miscellaneous
# services.
#
source ${RC_BASE}/rc.misc

#
# Run the script that brings up all the network and gateway
# services.
#
source ${RC_BASE}/rc.network

#
# Run the script that takes care of launching the location
# services.
#
source ${RC_BASE}/rc.atlas

#
# Clean up the variable switches; makes variable dumps more
# readable after booting.
#
disable source ${RC_BASE}/rc.cleanup

#
# Set the prompt to something decent.
#
prompt "echo -n \${PWD}:\${N}>\\ "

echo
echo "Ready."
```

### C.5.2 rc.alias

```
#
# Original Version
#
# Sets some aliases so that starting classes from the
```

```
# shell is less cumbersome.
#
# Author  : Jan Peters
# Version: $Id: whatis.conf,v 1.0 2000/12/15 12:00:00 jpeters Exp $
#

alias startLSGui   run DE.FhG.IGD.atlas.ui.LSGui &
```

### C.5.3 rc.conf.personal

```
#
# Original Version
#
# Author  : Jan Peters
# Version: $Id: whatis.conf,v 1.0 2000/12/15 12:00:00 jpeters Exp $
#

# ATLAS settings
setenv LS_CONF_FILE     ${RC_BASE}/lsserver.conf

# LS-Client
setenv LS_CLIENT        enable
setenv LS_CLIENT_LEVEL  INFO

# LS-Server
setenv LS_SERVER        enable
setenv LS_SERVER_LEVEL  INFO
setenv LS_SERVER_PORT   50000

# LS-Proxy
setenv LS_PROXY         enable
setenv LS_PROXY_LEVEL   INFO
setenv LS_PROXY_PORT    50010
```

### C.5.4 rc.atlas

```
#
# Original Version
#
# Author  : Jan Peters
# Version: $Id: whatis.conf,v 1.0 2000/12/15 12:00:00 jpeters Exp $
#
echo
echo "Agent Tracking and Location Services (ATLAS)"
echo "--------------------------------------------"
```

```
echo "Launching components:"

${LS_SERVER} echo -n " server"
${LS_SERVER} java DE.FhG.IGD.atlas.core.LSServer
  -port ${LS_SERVER_PORT} -capacity 20 -maxsize 65536
  -loglevel ${LS_SERVER_LEVEL}

${LS_PROXY} echo -n " proxy"
${LS_PROXY} java DE.FhG.IGD.atlas.core.LSProxy
  -port ${LS_PROXY_PORT} -capacity 10 -maxsize 65536
  -serverconf  ${LS_CONF_FILE} -loglevel ${LS_PROXY_LEVEL}

${LS_CLIENT} echo -n " client"
${LS_CLIENT} java DE.FhG.IGD.atlas.core.LSClient
  -serverconf ${LS_CONF_FILE} -loglevel ${LS_CLIENT_LEVEL}
```

### C.5.5 whatis.conf

```
#
# The list of static global definitions in the SeMoA server.
# This list must be loaded into 'WhatIs' using a call such
# as 'java DE.FhG.IGD.semoa.server.WhatIs -init <url>' where
# <url> is a file URL to this file.
#
#
# Author : Jan Peters
# Version: $Id: whatis.conf,v 1.0 2000/12/15 12:00:00 jpeters Exp $
#

# ATLAS related constans.
#
LS_CLIENT = /atlas/client
LS_PROXY = /atlas/proxy
LS_SERVER = /atlas/server
```

## C.6 Zuordnung einer IP-Adresse zu der entsprechenden Netzklasse

Um zu testen, ob sich eine IP-Adresse im gleichen LAN bzw. Subnetz befindet wie eine andere, müssen vorerst die Netzklassen bestimmt werden, in denen die beiden IP-Adressen liegen, um diese dann durch die entsprechende Subnetz-Maske miteindander vergleichen zu können. Diese Funktionalität wird für den LS-Proxy benötigt (siehe Abschnitt 4.3) und unter anderem durch die Klasse `DE.FhG.IGD.util.InetAddressInfo` (siehe Abschnitt C.2.1) zur

Verfügung gestellt. Welcher Netzklasse eine IP-Adresse bei IPv4 angehört entscheiden die ersten 5 Bits in folgender Weise:

**Class A**

```
 Adress-Bereich :  0.0.0.0 - 127.255.255.255
 Subnet-Maske   :  255.0.0.0

+-------+------------------+--------------------+
| 0     |  netid (7 Bits)  |  hostid (24 Bits)  |
+-------+------------------+--------------------+
```

**Class B**

```
 Adress-Bereich :  128.0.0.0 - 191.255.255.255
 Subnet-Maske   :  255.255.0.0

+-------+------------------+--------------------+
| 10    |  netid (14 Bits) |  hostid (16 Bits)  |
+-------+------------------+--------------------+
```

**Class C**

```
 Adress-Bereich :  192.0.0.0 - 223.255.255.255
 Subnet-Maske   :  255.255.255.0

+-------+------------------+--------------------+
| 110   |  netid (21 Bits) |  hostid (8 Bits)   |
+-------+------------------+--------------------+
```

**Class D**

```
 Adress-Bereich :  224.0.0.0 - 239.255.255.255

+-------+----------------------------------------+
| 1110  |     multicast group ID (28 Bits)       |
+-------+----------------------------------------+
```

**Class E**

```
 Adress-Bereich :  240.0.0.0 - 247.255.255.255

+-------+----------------------------------------+
| 11110 |   reserved for future use (27 Bits)    |
+-------+----------------------------------------+
```

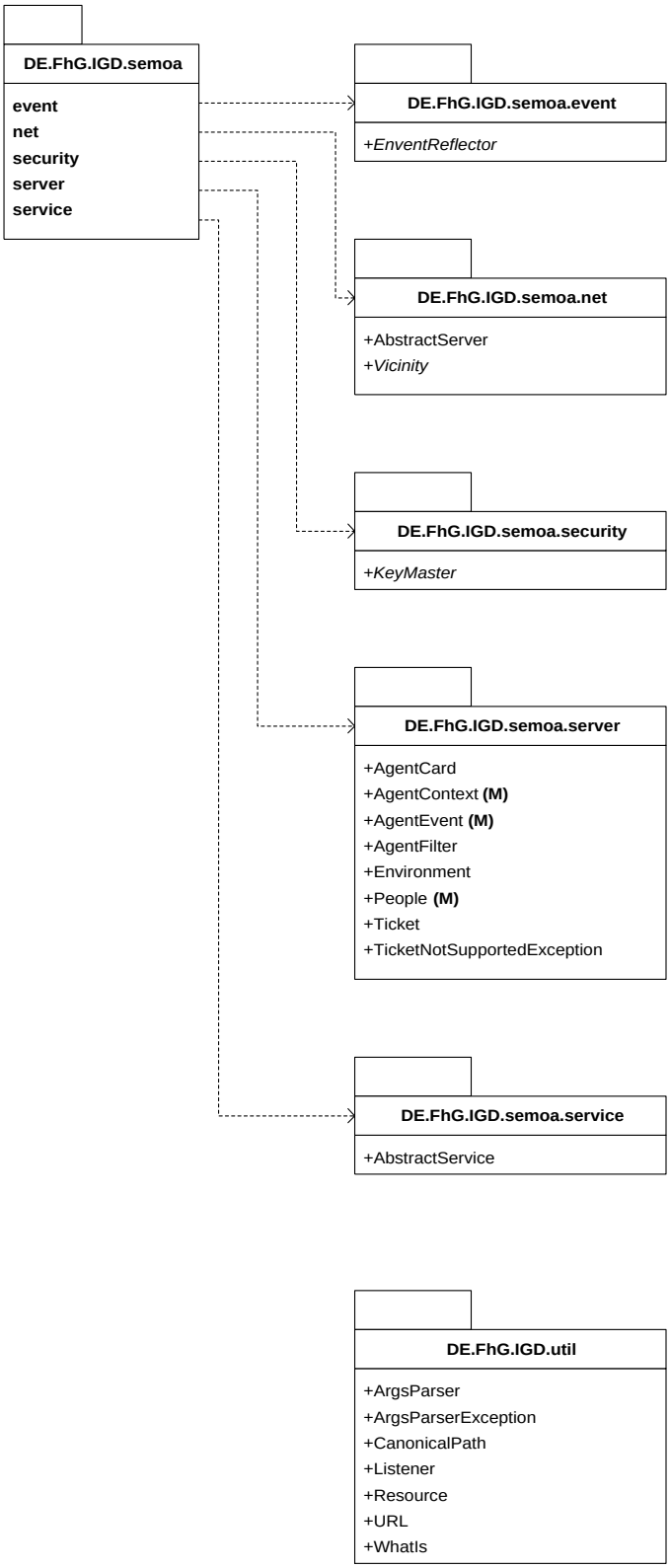


Abbildung C.2: Paketdiagramm der verwendeten SeMoA-Klassen

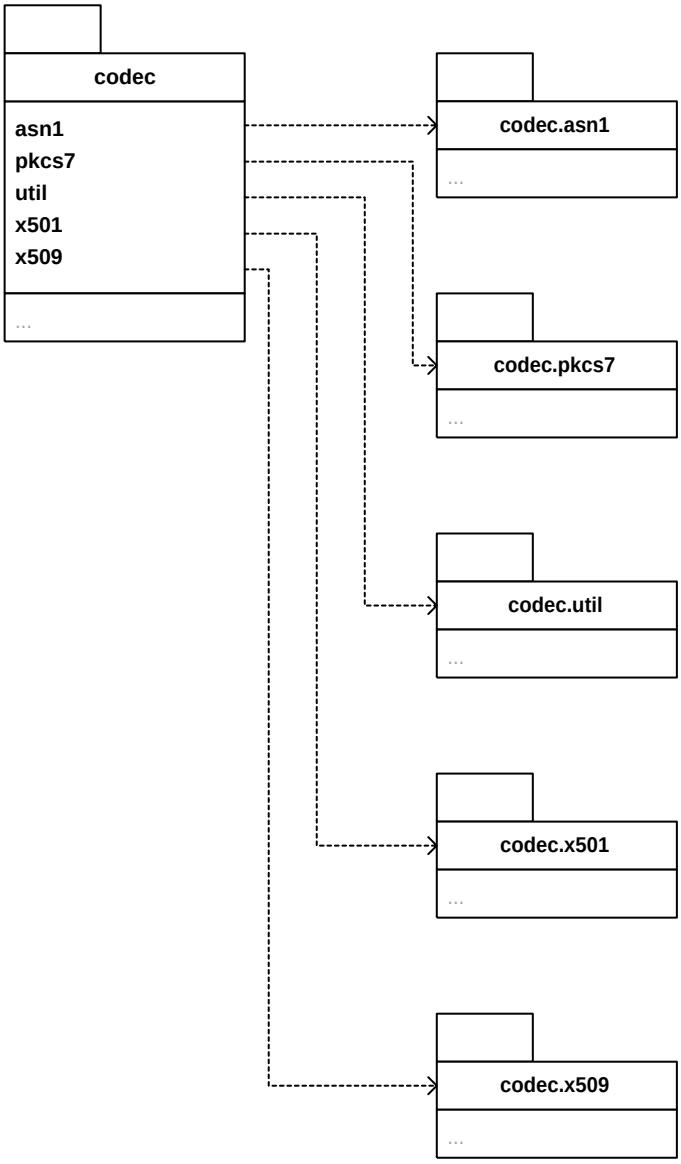


Abbildung C.3: Paketdiagramm der verwendeten codec-Pakete

# Anhang D

# Glossar

Die hier aufgeführten Begriffe können je nach Verwendung durchaus leicht von einander abweichende Bedeutungen haben. Um Missverständnisse zu vermeiden, werden sie nun kurz im Kontext dieser Arbeit beschrieben.

**Agentenplattform** - Die allgemeine Bezeichnung für ein Modell (*framework*) einer Umgebung für Agenten, das neben der eigenen Struktur auch die Schnittstellen definiert, über die Agenten in diese Umgebung eingebettet werden bzw. in ihr agieren können.

**Agentenserver** - Im Gegensatz zu einer Agentenplattform bezeichnet ein Agentenserver die Instanz einer konkreten Ausführungsumgebung für Agenten, die auf einem Rechnersystem installiert wurde.

**Agentensystem** - Ein Agentensystem umfasst eine homogene oder heterogene Struktur aus Agentenservern und Diensten, die über die Grenzen eines Rechnersystems hinweg miteinander in Kontakt stehen bzw. den Austausch von Agenten ermöglichen.

**Ausführungsumgebung** - Die Softwareschicht, die ein Programm (einen Agenten) zur Ausführung bringt, ihm durch Schnittstellen gewisse Dienste zur Verfügung stellt und nach dessen Terminierung gegebenenfalls wieder aus dem System entfernt.

**Ausführungszeit** - Die Zeitspanne zwischen Start und Beenden eines Programms (eines Agenten) in einer bestimmten Ausführungsumgebung, in der es aktiv seinen Kontrollfluss beeinflussen kann.

**Broadcasting** - Das Übermittel einer Nachricht von einem Sender an alle Empfänger in einem Subnetz.

**Cookie** - Ein *bit string* der im Rahmen eines Protokolls vereinbart wird und einen Zustand definiert, der zu einem späteren Zeitpunkt wieder abgefragt werden kann. In dieser Arbeit wird dieser *bit string* zur Autorisierung eines Anfragestellers genutzt.

**Daemon** - Ein Programmteil eines Dienstes, meist ein eigener Thread, der im Hintergrund abläuft und auf gewisse Ereignisse reagiert bzw. gewisse Prozesse zur Ausführung bringt.

**Homeserver** - Bezeichnet den Agentenserver, auf dem ein Agent während seines Lebenszyklus zum ersten Mal zur Ausführung kommt.

**Host** - Ein Rechnersystem, dass durch eine physikalische Adresse bzw. einen sprechenden Bezeichner im Netzwerk sichtbar wird und mit dem dadurch kommuniziert werden kann.

**Hop** - Bei der Übertragung einer Nachricht oder eines Agenten beteiligte Komponente im Netzwerk, die abhängig vom Inhalt der übermittelten Daten Entscheidungen über das weitere Vorgehen trifft und die Übertragung aktiv beeinflussen kann.

**Komponente** - Eine Anzahl von Modulen, die im Zusammenspiel miteinander eine gewisse Aufgabe erfüllen.

**Kontrollfluss** - Die zustandsabhängige Abfolge von Befehlen, die das Verhalten eines Programms bestimmt.

**Lebenszyklus** - Die Zeit von der ersten Ausführung eines Agenten auf dem Homeserver bis zu seiner endgültigen Terminierung.

**Lookup** - Eine Anfrage an einen Server, auf die ein Ergebnis erwartet wird, ohne dass der interne Zustand der Serverdatenbank verändert wird.

**Modul** - Ein Teil eines Programms, das meistens mit einer konkreten Klasse gleichzusetzen ist, die eine gewisse Funktionalität kapselt.

**Private key** - Der private Schlüssel des Schlüsselpaars, das bei asymmetrischen Verschlüsselungsalgorithmen Verwendung findet.

**Public key** - Der öffentliche Schlüssel des Schlüsselpaars, das bei asymmetrischen Verschlüsselungsalgorithmen Verwendung findet.

**Rechnerressource** - Ein Betriebsmittel bzw. eine Komponente eines Rechnersystems die von Programmen (Agenten) in Anspruch genommen werden kann (z.B. Prozessor, Speicher, Drucker).

**Secret key** - Der geheime Schlüssel, der bei symmetrischen Verschlüsselungsalgorithmen Verwendung findet.

**Subnetz** - Durch eine logische Unterteilung erzeugtes Teilnetz. Bei IP-Netzwerken werden Subnetze implizit durch die Vergabe von IP-Adressen erzeugt, die einer bestimmten Netzklasse zugeordnet werden können.

**Update** - Eine Anfrage an einen Server, die im Erfolgsfall den interne Zustand der Serverdatenbank verändert.

**Zeitstempel** - Eine Zeitangabe, die mit Genauigkeit bis in den Millisekundenbereich den Zeitpunkt der letzten Änderung einer assoziierten Datenstrukur festhält.

# Literaturverzeichnis


[1] International Standard ISO/IEC 10646-1:1993. Information Technology - Universal Multiple-Octet Coded Character Set (UCS): Architecture and Basic Multilingual Plane. Available at URL: `http://www.unicode.org/`.

[2] M. van Steen A. Baggio, G. Ballintijn. Supporting Effective Caching in a Wide-Area Location Service. In *Proc. 6th Annual ASCI Conference*, pages 358–363, June 2000. Available at URL: `http://www.cs.vu.nl/pub/papers/globe/asci-ab.00.pdf`.

[3] L. Lo Bello A. Di Stefano and C. Santoro. Naming and locating mobile agents in an internet environment. In *Third International Enterprise Distributed Object Computing Conference (EDOC 99) - Proceedings*, pages 153–161, 1999. Available at URL: `http://alpha2.iit.unict.it/ARCA/publications.html`.

[4] Shakil Ahmed. Notes on LDAP evaluation. Technical report, College of Engineering, San Jose State University, March 1999. Available at URL: `http://www.engr.sjsu.edu/fatoohi/eosdis/ldap.html`.

[5] Marc Majka Alan M. Marcum. NetInfo Binding and Connecting, 1993. `http://www.next.de/NeXTAnswers/1274.html`.

[6] Hans-Peter Bischof. NetInfo - network administration information, 1997. `http://www.cs.rit.edu/~hpb/Man/_Man_NeXT_html/html5/netinfo.5.html`.

[7] A. Black and Y. Artsy. Implementing Location Independent Invokation. In *Transactions on Parallel and Distributed Systems*, volume 1 (1), pages 107–119. IEEE Computer Society, 1990.

[8] Felix Buebl and Dirk Lüdtke. KQML und ACL als Quasi-Standard für Agenten. Seminararbeit bei Dr.-Ing. Robert Tolksdorf. Available at URL: `http://user.cs.tu-berlin.de/~buebl/kqml-acl/kqml-acl.html`, 1998.

[9] C. Kaufman D. Eastlake. Domain Names System Security Extensions. PROPOSED STANDARD RFC2065, Internet Requests For Comments, January 1997. Updates RFCs 1034, 1035. Available at URL: `http://www.faqs.org/rfcs/rfc2065.html`.

[10] M. Demmer and M.P. Herlihy. The Arrow Directory Protocol. In *12th International Symposium on Distributed Computing*, Greece, September 1998. Available at URL: `http://www.cs.brown.edu/people/mph/`.

[11] R. Droms. Dynamic Host Configuration Protocol. DRAFT STANDARD RFC2131, Internet Requests For Comments, March 1997. Available at URL: `http://www.faqs.org/rfcs/rfc2131.html`.

[12] Olivier Dubuisson. *ASN.1 - Communication between Heterogeneous Systems.* OSS - Nokalva, June 2000. Available at URL: `http://asn1.elibel.tm.fr/en/book/`.

[13] D. Eastlake. Secure Domain Names System Dynamic Update. PROPOSED STANDARD RFC2137, Internet Requests For Comments, April 1997. Updates RFC1035. Available at URL: `http://www.faqs.org/rfcs/rfc2137.html`.

[14] D. Eastlake. Storing Certificates in the Domain Name System (DSN). PROPOSED STANDARD RFC2538, Internet Requests For Comments, March 1999. Available at URL: `http://www.faqs.org/rfcs/rfc2538.html`.

[15] GMD FOKUS et al. Mobile Agent System Interoperability Facilities Specification. In *OMG TC Document orbos/97-10-05*, 1997. Available at URL: `http://www.fokus.gmd.de/research/cc/ecco/masif/doc/97-10-05.pdf`.

[16] CORBA FAQ. `http://www.omg.org/gettingstarted/corbafaq.html`.

[17] Tim Finin, Don McKay Richard Fritzson, and Robin McEntire. KQML as an Agent Communication Language. In *Third International Conference on Information and Knowledge Management (CIKM'94)*, November 1994. Available at URL: `http://www.csee.umbc.edu/kqml/papers/kqmlacl.pdf`.

[18] Foundation for Intelligent Physical Agents. `http://www.fipa.org/`.

[19] Foundation for Intelligent Physical Agents. Agent Communication Language. Technical Report Specification Part 2, Version 0.2, FIPA 99, March 2000. Available at URL: `http://www.fipa.org`.

[20] Foundation for Intelligent Physical Agents. FIPA Content Language Library. Technical Report Specification Part 18, Version 0.2, FIPA 99, March 2000. Available at URL: `http://www.fipa.org`.

[21] AIX Support for the X/Open UNIX95 Specification. System Management Guide: Communications and Networks. Technical report, The University of Arizona, October 1998. Available at URL: `http://anguilla.u.arizona.edu/doc_link/en_US/a_doc_lib/aixbman/commadmn/toc.htm`.

[22] AIX Support for the X/Open UNIX95 Specification. Network Information Services (NIS and NIS+) Guide. Technical report, The University of Arizona, October 1999. Available at URL: `http://anguilla.u.arizona.edu/doc_link/en_US/a_doc_lib/aixbman/nisplus/toc.htm`.

[23] A.S. Tanenbaum G. Ballintijn, M. van Steen. Exploiting Location Awareness for Scalable Location-Independent Object IDs. In *Proc. Fifth Annual ASCI Conf.*, pages 321–328, Heijen, The Netherlands, June 1999. Available at URL: `http://www.cs.vu.nl/pub/papers/globe/IR-459.99.pdf`.

[24] A.S. Tanenbaum G. Ballintijn, M. van Steen. Simple Crash Recovery in a Wide-area Location Service. In *Proc. 12th Int'l. Conf. on Parallel and Distributed Computing Systems*, pages 87–93, Fort Lauderdale, Florida, August 1999. Available at URL: `http://www.cs.vu.nl/pub/papers/globe/pdcs.99.pdf`.

[25] A.S. Tanenbaum G. Ballintijn, M. van Steen. Scalable Naming in Global Middleware. In *Proc. 13th Int'l Conf. on Parallel and Distributed Computing Systems (PDCS-2000)*, August 2000. Available at URL: `http://www.cs.vu.nl/pub/papers/globe/pdcs.00.pdf`.

[26] M. van Steen G. Ballintijn and A.S. Tanenbaum. A Scalable Implementation for Human-Friendly URIs. Technical Report Revised version of internal Report IR-466, Vrije Universiteit, Department of Mathematics and Computer Science, November 1999. Available at URL: `http://www.cs.vu.nl/pub/papers/globe/IR-466.99.pdf`.

[27] Michael R. Genesereth. Knowledge Interchange Format. Technical report, dpANS, April 1998. Available at URL: `http://meta2.stanford.edu/kif/dpans.html`.

[28] Li Gong. Java$^{TM}$ 2 Platform Security Architecture, 1998. Available within JKD1.3: [jdk13]/docs/guide/security/spec/security-spec.doc.html.

[29] G. Good. The LDAP Data Interchange Format (LDIF) - Technical Specification. PROPOSED STANDARD RFC2849, Internet Requests For Comments, June 2000. Available at URL: `http://www.faqs.org/rfcs/rfc2849.html`.

[30] Peter Gutmann. X.509 Style Guide. Technical report, University of Auckland, August 2000. Available at URL: `http://www.cs.auckland.ac.nz/~pgut001/pubs/x509guide.txt`.

[31] M Herlihy. The Aleph Toolkit: Support for Scalable Distributed Shared Objects. In *Workshop on Communication, Architecture and Applications for Network-based Parallel Computing*, Orlando, FL, January 1999. Available at URL: `http://www.cs.brown.edu/people/mph/`.

[32] T. Howes and M. Smith. The LDAP URL Format. PROPOSED STANDARD RFC2255, Internet Requests For Comments, December 1997. Obsoletes RFC1959. Available at URL: `http://www.faqs.org/rfcs/rfc2255.html`.

[33] Innosoft International Inc. LDAP FAQ. Technical report, Innosoft International Inc., June 1997. Available at URL: `http://www.critical-angle.com/ldapworld/ldapfaq.html`.

[34] Sun Microsystems Inc. Java$^{TM}$ Software Developer Kit (SDK) - Version 1.3, 1999. Available at URL: `http://java.sun.com`.

[35] Tek-Tools Inc. Kawa$^{TM}$ Integrated Development Environment - Version 4.1a, 2000. Available at URL: `http://www.tek-tools.com`.

[36] Document ISO/IEC. International Register of Coded Character Sets to be used with Escape Sequences. Available at URL: `http://www.itscj.ipsj.or.jp/ISO-IR`.

[37] Burton S. Kaliski Jr. A Layman's Guide to a Subset of ASN.1, BER and DER. Technical report, RSA Laboratories, November 1993. Available at URL: `http://ftp.rsa.com/pub/pkcs/ascii/layman.asc`.

[38] Martin Kuppinger. *Microsoft Windows 98 im Netzwerk*. Microsoft Press, 1999. ISBN: 3-86063-473-9.

[39] Yannis Labrou and Tim Finin. A Proposal for a new KQML Specification. Technical report, University of Maryland Baltimore County, Februrary 1997. Available at URL: `http://www.cs.umbc.edu/~jklabrou/publications/tr9703.ps`.

[40] B. Lampson. Designing a global name service. In *Proc. 4th ACM Symposium on Principles of Distributed Computing*, pages 1–10, Minaki, Ontario, 1986. Available at URL: `http://www.research.microsoft.com/lampson/36-GlobalNames/Abstract.html`.

[41] John Larmouth. *ASN.1 - Complete*. Open Systems Solutions, 1999. Available at URL: `http://www.larmouth.demon.co.uk/books`.

[42] AppleCare Tech Info Library. Mac OS X Server: What is NetInfo?, 1999. `http://til.info.apple.com/techinfo.nsf/artnum/n60038`.

[43] The Mobile Agent List. `http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/mal/mal.html`.

[44] P. Homburg M. van Steen and A.S. Tanenbaum. Globe: A Wide-Area Distributed System. In *IEEE Concurrency*, pages 70–78, January 1999. Available at URL: `http://www.cs.vu.nl/pub/papers/globe/ieeeconc.99.org.pdf`.

[45] S. Kille M. Wahl, T. Howes. Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names. PROPOSED STANDARD RFC2253, Internet Requests For Comments, December 1997. Obsoletes RFC1779. Available at URL: `http://www.faqs.org/rfcs/rfc2253.html`.

[46] T. Howes M. Wahl and S. Kille. Lightweight Directory Access Protocol (v3). PROPOSED STANDARD RFC2251, Internet Requests For Comments, December 1997. Available at URL: `http://www.faqs.org/rfcs/rfc2251.html`.

[47] N.R. Jennings M. Woolridge. Intelligent Agents: Theory and Practice. In *Knowledge Engineering Review*, volume 11 (2), 1995. Available at URL: `http://www.cs.umbc.edu/agents/introduction/ker.ps`.

[48] Microsoft. DCOM Technical Overview. Technical report, Microsoft Corporation, November 1996. Available at URL: `http://msdn.microsoft.com/library/backgrnd/html/msdn_dcomtec.htm`.

[49] Microsoft. DCOM Architecture. Technical report, Microsoft Corporation, July 1997. Available at URL: `http://msdn.microsoft.com/library/backgrnd/html/msdn_dcomarch.htm`.

[50] P. Mockapetris. Domain Names - Concepts and Facilities. STANDARD RFC1034, Internet Requests For Comments, November 1987. Obsoletes RFCs 882, 883, 973. Available at URL: `http://www.faqs.org/rfcs/rfc1034.html`.

[51] P. Mockapetris. Domain Names - Implementation and Specification. STANDARD RFC1035, Internet Requests For Comments, November 1987. Obsoletes RFCs 882, 883, 973. Available at URL: `http://www.faqs.org/rfcs/rfc1035.html`.

[52] Luc Moreau. Distributed Directory Service and Message Router for Mobile Agents. Technical Report ECSTR M99/3, University of Southampton, 1999. Available at URL: `http://www.ecs.soton.ac.uk/~lavm/papers/mob.ps.gz`.

[53] Amy L. Murphy and Gian Pietro Picco. Reliable Communication for Highly Mobile Agents. In D.S. Milojicic, editor, *Proceedings of the 1st International Symposium on Agent Systems and Applications held jointly with the 3rd International Symposium on Mobile Agents (ASA/MA'99)*, pages 141–150, Palm Springs (CA, USA), Oktober 1999. IEEE Computer Society. Available at URL: `http://www.elet.polimi.it/Users/DEI/Sections/Compeng/GianPietro.Picco/listpub.html`.

[54] B. Neuman. Scale in Distributed Systems. In *Readings in Distributed Computer Systems*, pages 463–489, Los Alamitos, California, 1994. IEEE Computer Society Press. Available at URL: `http://www.isi.edu/people/bcn/papers/pdf/94--\_scale-dist-sys-neuman-readings-dcs.pdf`.

[55] Technical Note. PKCS #7: Cryptographic Message Syntax Standaerd. Technical report, RSA Laboratories, 1993. Available at URL: `http://www.rsalabs.com/pkcs/pkcs-7/`.

[56] Technical Note. PKCS #8: Private-Key Information Syntax Standard. Technical report, RSA Laboratories, 1993. Available at URL: `http://www.rsalabs.com/pkcs/pkcs-8/`.

[57] Hyacinth S. Nwana. Software Agents: An Overview. In *Knowledge Engineering Review*, volume 11 (3), pages 1–40, September 1996. Available at URL: `http://www.cs.umbc.edu/agents/introduction/ao.ps`.

[58] Bernd Oestereich. Unified Modeling Language (UML 1.3). Available at URL: `http://www.oose.de/uml`.

[59] OMG. CORBAservices: Common Object Services Specification. Technical Report 98-12-09, Object Management Group, December 1998. Available at URL: `ftp://ftp.omg.org/pub/docs/formal/98-12-09.pdf`.

[60] OMG. The Common Object Request Broker: Architecture and Specification, revision 2.2. Technical Report 98-07-01, Object Management Group, February 1998. Available at URL: `ftp://ftp.omg.org/pub/docs/formal/98-07-01.pdf`.

[61] C. Perkins. IP Mobility Support. PROPOSED STANDARD RFC2002, Internet Requests For Comments, October 1996. Available at URL: `http://www.faqs.org/rfcs/rfc2002.html`.

[62] Christian Queinnec and Luc Moreau. Graceful Disconnection. In Takayasu Ito and Taiichi Yuasa, editors, *Parallel and Distributed Computing for Symbolic and Irregular Applications, PDCSIA'99*, pages 242–252, Sendai, Japan, July 1999. World Scientific Publishing. Available at URL: `http://www.ecs.soton.ac.uk/~lavm/papers/gradisc.ps.gz`.

[63] S. Alexander R. Droms. DHCP Options and BOOTP Vendor Extensions. DRAFT STANDARD RFC2132, Internet Requests For Comments, March 1997. Available at URL: `http://www.faqs.org/rfcs/rfc2132.html`.

[64] J. Mogul et al. R. Fielding, J. Gettys. Hypertext Transfer Protocol – HTTP/1.1. DRAFT STANDARD RFC2616, Internet Requests For Comments, June 1999. Obsoletes RFC2068. Available at URL: `http://www.faqs.org/rfcs/rfc2616.html`.

[65] ITU-T Recommendation. X.500 - The Directory: Overview of concepts, models and services. Technical report, International Telecommunication Union (ITU), 1997. Available at URL: `http://www.itu.int/itudoc/itu-t/rec/x/x500up/x500.html`.

[66] ITU-T Recommendation. X.509 - The Directory: Authentication framework. Technical report, International Telecommunication Union (ITU), 1997. Available at URL: `http://www.itu.int/itudoc/itu-t/rec/x/x500up/x509.html`.

[67] ITU-T Recommendation. X.680 - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation. Technical report, International Telecommunication Union (ITU), 1997. Available at URL: `http://www.itu.int/itudoc/itu-t/rec/x/x500up/x680.html`.

[68] ITU-T Recommendation. X.681 - Abstract Syntax Notation One (ASN.1): Information Object Specification. Technical report, International Telecommunication Union (ITU), 1997. Available at URL: `http://www.itu.int/itudoc/itu-t/rec/x/x500up/x681.html`.

[69] ITU-T Recommendation. X.682 - Abstract Syntax Notation One (ASN.1): Constraint Specification. Technical report, International Telecommunication Union (ITU), 1997. Available at URL: `http://www.itu.int/itudoc/itu-t/rec/x/x500up/x682.html`.

[70] ITU-T Recommendation. X.683 - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 Specifications. Technical report, International Telecommunication Union (ITU), 1997. Available at URL: `http://www.itu.int/itudoc/itu-t/rec/x/x500up/x683.html`.

[71] ITU-T Recommendation. X.690 - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distingushed Encoding Rules (DER). Technical report, International Telecommunication Union (ITU), 1997. Available at URL: `http://www.itu.int/itudoc/itu-t/rec/x/x500up/x690.html`.

[72] ITU-T Recommendation. X.691 - ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER). Technical report, International Telecommunication Union (ITU), 1997. Available at URL: `http://www.itu.int/itudoc/itu-t/rec/x/x500up/x691.html`.

[73] Rational Rose. UML Poster. Available at URL: `http://www.rational.com/uml/resources/quick/index.jsp`.

[74] Volker Roth. Mutual Protection of Co-operating Agents. In Jan Vitek and Christian Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed*

*Objects*, volume 1603 of *Lecture Notes in Computer Science*, pages 275–285. Springer-Verlag Inc., New York, NY, USA, 1999.

[75] Volker Roth. Scalable and Secure Global Name Services for Mobile Agents. 6th ECOOP Workshop on Mobile Object Systems: Operating System Support, Security and Programming Languages (Cannes, France, June 2000), June 2000.

[76] Volker Roth. *The $SeMoA^{TM}$ Code Conventions*. Fraunhofer IGD, Darmstadt, Germany, November 2000. Version 0.1.

[77] Volker Roth and Vania Conan. Encrypting Java Archives and its Application to Mobile Agent Security. In Frank Dignum and Carles Sierra, editors, *Agent Mediated Electronic Commerce: A European Perspective*, volume 1991 of *Lecture Notes in Computer Science*, pages 232–244. Springer Verlag, Berlin, 2000.

[78] Volker Roth and Mehrdad Jalali. Access Control and Key Management for Mobile Agents. *Computers & Graphics, Special Issue on Data Security in Image Communication and Networks*, 22(4):457–461, 1998.

[79] 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., March 2001. IEEE Computer Society Press. Accepted for publication.

[80] I. Weerakoon S. Lazar and D. Sidhu. A scalable location tracking and message delivery scheme for mobile agents. In *Seventh IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 98) - Proceedings*, pages 243–248, 1998. Available at URL: `http://www.computer.org/proceedings/wetice/8751/8751toc.htm`.

[81] Andrs Salamon. DNS TCP/IP Name Resolution. Technical report, DNS Resources Directory, 2000. Available at URL: `http://www.dns.net/dnsrd/`.

[82] Bruce Schneier. *Angewandte Kryptographie. Protokolle, Algorithmen und Sourcecode in C*. Addison Wesley, 1996. ISBN: 3-89319-854-7.

[83] Marc Shapiro, Peter Dickman, and David Plainfoss. SSP Chains: Robust, Distributed References Supporting Acyclic Garbage Collection. Rapport de Recherche 1799, Institut National de Recherche en Informatique et en Automatique, November 1992. Available at URL: `http://www-sor.inria.fr/publi/SSPC_rr1799.html`.

[84] J. Solomon. Applicability Statement for IP Mobility Support. PROPOSED STANDARD RFC2005, Internet Requests For Comments, October 1996. Available at URL: `http://www.faqs.org/rfcs/rfc2005.html`.

[85] Doug Steedman. Extracts from Abstract Syntax Notation One (ASN.1) - The Tutorial and Reference. Technical report, Technology Appraisals Limited, 1990. Available at URL: `http://www.techapps.co.uk/asn1gloss.html`.

[86] Reiner Singer Stefan Middendorf. *Java Programmierhandbuch und Referenzen. Für die Java-2-Plattform*. dpunkt.verlag, 1999. ISBN 3-920993-82-9.

[87] TogetherSoft. Together$^{TM}$ Control Center - Version 4.1, 2000. Available at URL: `http://www.togethersoft.com`.

[88] M. van Steen, P. Homburg F.J. Hauck, and A.S. Tanenbaum. Locating Objects in Wide-Area Systems. In *IEEE Communications Magazine*, volume 36 (1), pages 104–109, January 1998. Available at URL: `http://www4.informatik.uni-erlangen.de/~fzhauck/Pub/`.

[89] M. van Steen, F.J. Hauck, and A.S. Tanenbaum G. Ballintijn. Algorithmic Design of the Globe Wide-Area Location Service. In *The Computer Journal*, volume 41 (5), pages 297–310, 1998. Available at URL: `http://www4.informatik.uni-erlangen.de/~fzhauck/Pub/`.

[90] P. Vixie. Dynamic Updates in the Domain Name System (DNS UPDATE). PROPOSED STANDARD RFC2136, Internet Requests For Comments, April 1997. Updates RFC1035. Available at URL: `http://www.faqs.org/rfcs/rfc2136.html`.

[91] Gnter Wahl. Scale in Distributed Systems. In *OBJEKTspektrum*, 1998. Available at URL: `http://www.sigs.de/publications/docs/obsp/umlkompt/umlkompt.htm`.

[92] M. Wahl and S. Kille A. Coulbeck, T. Howes. Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions. PROPOSED STANDARD RFC2252, Internet Requests For Comments, December 1997. Available at URL: `http://www.faqs.org/rfcs/rfc2252.html`.

[93] M.P. Warres and M.P. Herlihy. A Tale of Two directories: Implementing Distributed Shared Objects in Java. In *ACM Java Grande Conference*, Palo ALto, CA, June 1999. Available at URL: `http://www.cs.brown.edu/people/mph/`.

[94] Paweł T. Wojciechowski and Peter Sewell. Location-independent communication for mobile agents: a two-level architecture. Technical Report Technical Report 462, Computer Laboratory, University of Cambridge, 1999. Available at URL: `http://www.cl.cam.ac.uk/users/pes20/`.

[95] Paweł T. Wojciechowski and Peter Sewell. Nomadic Pict: Language and Infrastructure Design for Mobile Agents. *IEEE Concurrency. The Computer Society's Systems Magazine*, 8(2):42–52, April-June 2000. Available at URL: `http://www.cl.cam.ac.uk/~ptw20/`.