Proaktive autonome Navigation für mobile Agenten

Ein Schritt in Richtung mobiler Agentensysteme der nächsten Generation

Dissertation

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

vorgelegt dem Rat der Fakultät für Mathematik und Informatik
der Friedrich-Schiller-Universität Jena
im Juli 2004

von
Dipl.-Inf. Christian Erfurth
geboren am 30. Juni 1971 in Naumburg (Saale)

Gutachter:
1. Prof. Dr. Wilhelm R. Rossak, Friedrich-Schiller-Universität Jena
2. Prof. Dr. Johann Eder, Universität Klagenfurt

Tag der letzten Prüfung des Rigorosums: 26. Oktober 2004

Tag der öffentlichen Verteidigung: 8. November 2004

Kurzfassung

In logischen oder virtuellen Netzwerken, die durch mobile Agentensysteme aufgespannt werden, ist eine hohe Dynamik des Gesamtsystems anzutreffen: Agentenserver und Dienste kommen hinzu oder terminieren, Verbindungsqualitäten ändern sich. Durch die zunehmende Größe der Netzwerke kommt im praktischen Einsatz ein Quantitätsfaktor dazu, der Skalierungsprobleme aufwirft. Diese Aspekte werden beim heutigen Stand der Technik mobiler Agentensysteme wenig beachtet. Es wird einfach vorausgesetzt, dass Auftraggeber und Programmierer des Agenten das Netzwerk vollständig und auf aktuellem Stand kennen. Das führt notwendiger Weise zu Problemen bei der Rundreise des mobilen Agenten, der zur Bearbeitung der übertragenen Aufgabe eine Menge von Agentenservern im Netzwerk nach der durch den Programmierer vorgegebenen Route besuchen muss. Diese Route ist in vielen Fällen entweder unvollständig oder inkorrekt.

In dieser Arbeit wird das generische Rahmenwerk *ProNav* vorgestellt, das diese inhärente Dynamik und Größe zu beherrschen hilft. *ProNav* ist eine Erweiterung für mobile Agentensysteme, die mobilen Agenten ein selbständiges, d. h. autonomes und proaktives Navigieren unter Berücksichtigung der Systemdynamik des realen Netzwerks ermöglicht.

Aus der Sicht eines mobilen Agenten ist *ProNav* ein (Infrastruktur-)Dienst, der auf einem Agentenserver in Anspruch genommen werden kann. Spezielle Funktion von *ProNav* ist die Schaffung der Voraussetzung für die Lokalisierung von im Netzwerk verteilten Informationen und Diensten auf der Applikationsebene. Mit Unterstützung aller Agentenplattformen wird eine Informationsbasis für Agenten aufgebaut, die diese zur Wahrnehmung des Agentennetzwerks befähigt: Das Agentennetzwerk wird kartographiert.

Dem Agenten selbst ermöglicht *ProNav* eine proaktive und autonome Routenplanung in der heterogenen und dynamischen Umgebung des Agentennetzwerks. Der Programmierer oder Besitzer muss sich nicht länger einer Routenplanung seines Agenten widmen. Der Agent agiert und reagiert im Netz selbständig, indem er seine Route mit Hilfe von *ProNav* selbst findet, regelmäßig aktualisiert und die Dynamik der Umgebung für sich nutzt, anstatt mit einer fix vorgegebenen Route an ihr zu scheitern.

Aus der Sicht eines mobilen Agentensystems ist *ProNav* eine Erweiterung der Architektur, die als zusätzliche Schicht zwischen dem eigentlichen Agentensystem und der Applikation eingebracht wird. Diese Schicht besteht aus drei Komponenten: Dem Kartenmodul, dem Routenplaner und dem Migrationsoptimierer. Jede Komponente kann unabhängig angesprochen werden, jedoch sind alle für eine integrierte Zusammenarbeit konzipiert. Die vom Kartenmodul erstellte „Webkarte“ wird durch die Agentenplattform selbst aktuell gehalten, während der Routenplaner, der die Webkarte nutzt, durch den mobilen Agenten explizit aufgerufen wird. Der Migrationsoptimierer dient aus technischer Sicht der Anpassung der Übertragung eines Agenten an die Fähigkeiten des eingesetzten mobilen Agentensystems.

ProNav kann auf alle mobilen Agentensysteme, die eine grundlegende Infrastruktur-Organisation aufweisen, aufgesetzt werden. Das an der Friedrich-Schiller-Universität Jena entwickelte mobile Agentensystem TRACY wurde als Basis für eine prototypische Realisierung und für die Evaluierung genutzt.

Abstract

Logical or virtual networks based on mobile agents exhibit a high level of internal dynamics: Agent servers and services are added, deleted, or modified, connection quality changes over time, etc. In addition, the size of these networks steadily grows, as practical experience shows, and adds a quantitative factor that makes scalable solutions and structures mandatory. These aspects cannot be handled by the current state-of-the-art in mobile agent technology.

Today's systems expect a mobile agent's user and/or programmer to know the current structure of the underlying network and to keep this knowledge up-to-date all the time. The agent is programmed with a fixed itinerary (route) through the network that is based on the current knowledge the programmer has to offer. This leads to problems with the dynamical and quantitative aspects of the network and may cause incomplete or even incorrect routing of the agent on its itinerary.

This thesis presents the generic framework called *ProNav* that offers functions to manage both mentioned network aspects. *ProNav* is designed as an extension of mobile agent systems that provides independent routing on the application level, i. e. offers autonomous and proactive navigation that takes into account the dynamics and size of real world networks.

From agent's perspective, *ProNav* is just another (infrastructure-)service provided on each agent server. Its most important feature is to locate services and information in the network and to offer this type of data to any mobile agent currently planning its itinerary. This is achieved by integrating the data that is locally acquired by each agent server into a map that enables each agent to recognise and analyse its virtual environment.

For a mobile agent the opportunity for a proactive and autonomous planning of its itinerary is offered by *ProNav*, even if the network is highly heterogeneous and dynamic. The agent's programmer and user do not have to do the itinerary planning for the agent. The agent is enabled to fulfill this task itself and independently of its owners. *ProNav* also provides enough information and flexibility to abandon the notion of a fixed route through the network and allows regular updates and changes in the itinerary during its execution. This helps to react immediately and in an autonomous fashion to changes in the environment.

From an architectural perspective, *ProNav* extends any mobile agent system by working as an intermediate layer in-between the actual agent system and the application layer that is formed by specialised mobile agents and their user and application interfaces. *ProNav* is divided into three major components: the map module, the route planner, and the migration optimizer. In principle each component may be used independently by any mobile agent. However, only by integrating their services a mobile agent will achieve full autonomy and pro-activity for the itinerary planning task.

Each map module keeps its data up-to-date by using the local resources of its agent server. These local maps are integrated by means of the route planner module into a more global view that allows for incremental itinerary planning for each specific mobile agent. The migration optimizer can be utilized to improve further the technological implementation of an itinerary, e. g. to optimise time or cost factors on the route. This module interacts directly with the underlying mobile agent system's migration component.

ProNav may be used with any mobile agent system that offers a compatible infrastructure organization. The current prototype, also used during the evaluation phase of the project, relies on the mobile agent system TRACY that was developed at the Friedrich-Schiller-University Jena.

Danksagung

Ich danke allen, die zum Gelingen dieser Arbeit beigetragen haben.

Mein besonderer Dank gilt meinem Betreuer Prof. Dr. Rossak für seine kritisch-konstruktiven Diskussionen, hilfreichen Anregungen und vor allem der beständigen Unterstützung während der gesamten Promotionszeit. In vielen wertvollen Gesprächen wurden die Ideen dieser Arbeit verfeinert und durchdacht.

Ich danke meinen Kollegen, die mit vielen konstruktiven Diskussionen und einer kooperativen Zusammenarbeit in den vergangenen Jahren meine Arbeit vorangebracht haben.

Bedanken möchte ich mich auch bei den Studenten, die durch ihre Studien- und Diplomarbeiten zu dieser Arbeit beigetragen haben.

Dank gilt auch meinen Freunden und Bekannten für ihr Interesse und ihre Unterstützung.

Nicht zuletzt danke ich meiner Familie für Ihren Beistand.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Die Welt der Agenten

„Agenten sind immer und überall“ – Die Aussage deutet auf eine Vielfalt von Agenten und eine Vielfalt ihres Vorkommens hin. Die Vielfalt spiegelt sich in der Charakterisierung des Agentenbegriffs wider. Gemeinsamkeiten sind dennoch zu finden und der Agentenbegriff lässt sich auf einen Ursprung zurückführen.

Auf Basis einer Klassifikation der Agententypen werden die Fähigkeiten der verschiedenen Agenten miteinander verglichen. Durch diesen alternativen Zugang wird der Softwareagent näher charakterisiert und deren Schnittstellen zur Umwelt aufgezeigt.

Die jüngste Generation von Agenten, die mobilen Agenten als Spezialtyp der Softwareagenten, steht im Vordergrund dieser Arbeit. Die Neuartigkeit der Mobilität von Softwareagenten und die darunterliegende Technologie wird einführend erklärt. Eine technische Definition der mobilen Agenten wird gegeben. Der derzeitige Stand der Technologie wird vorgestellt und eine Einordnung der Arbeit in das Gebiet der mobilen Agenten vorgenommen.

1.1 Agenten sind immer und überall

In diesem einführenden Kapitel soll das Auftreten von „Agenten“ in der realen und der virtuellen Welt anhand von Beispielen veranschaulicht werden. Eine erste intuitive Begriffsbildung wird vorgenommen und Parallelen werden herausgearbeitet. Im zweiten Teil steht eine einfache Klassifizierung im Vordergrund, welche Agenten in unterschiedliche Typen einteilt.

1.1.1 Wortursprung und Vorkommen

Der Agent ist sowohl in der realen als auch in der virtuellen Welt, der Computerwelt, ein geläufiger Begriff. Die konkrete Bedeutung des Terminus ist in beiden Welten einerseits unterschiedlich, andererseits sind wiederum Gemeinsamkeiten zu finden. Bereits bei der Betrachtung von Agenten der realen Welt fällt eine Mehrdeutigkeit der Begriffsverwendung auf. Viele verschiedene Begriffsklärungen sind anzutreffen. Dennoch wird sich am Ende des Kapitels zeigen, dass die unterschiedlichen Begriffsklärungen auf einen Ursprung zurückzuführen sind.

Im deutschsprachigen Raum wird heute das Wort Agent schnell mit dem Geheimagent assoziiert, wahrscheinlich stark animiert durch zahlreiche Filme mit heldenhaften Agenten. Neben der Filmbranche ist der Begriff in der Musikindustrie zu finden. Ein Manager oder Interessenvertreter einer Musikgruppe wird ebenso als Agent bezeichnet. Zudem sind in einer Agentur arbeitende Personen, die Jobs, Wohnungen etc. vermitteln, Agenten. Das wohl bekannteste Beispiel ist die „Bundesagentur für Arbeit“. Im englischsprachigen Raum ist der Begriff in ähnlicher Bedeutung anzutreffen. So werden Vertreter einer Firma oder Institution als Agent bezeichnet, z. B. der Travel Agent, ein Angestellter eines Reisebüros.

Langenscheidt gibt im Fremdwörterbuch [Lan04] als Erklärung zu Agent genau die oben aufgezeigten Verwendungsarten wieder:

> **Agent**, *der; -en,-en* **1.** meist auf Provisionsbasis arbeitender beauftragter Vermittler, Interessenvertreter, Vertreter **2.** meist auf Provisionsbasis arbeitender Kaufmann, der Verträge vermittelt u. / oder abschließt **3.** in geheimem Auftrag tätiger Diplomat **4.** Spion

Der Begriff des Agenten ist dennoch kein moderner Terminus. Der Ursprung des Wortes ist im Lateinischen zu finden:

agens → der Handelnde,

das Partizip Präsens Aktiv zum Verb

agere → tun, handeln, treiben, wirken.

Damit wird die Intention eines Agenten – der Agent handelt im Auftrag eines anderen – bereits deutlich.

Zu einem interessanten Ergebnis führt eine Suche bei Wikipedia [Wik04], einer freien Enzyklopädie im Internet:

> Ein Agent (lat. *Handelnder*) handelt im Auftrag eines anderen für dessen Interessen. Bei der Vermittlung von Geschäften spricht man stattdessen von einem Makler.
>
> Schon im Römischen Kaiserreich kannte man den *verdeckten Ermittler* der obersten Verwaltung, den so genannten *agentus in rebus*.
>
> Agenten sind im Allgemeinen natürliche Personen, seit den 90er Jahren gibt es auch Software-Agenten, die das Internet durchforsten.
>
> ...

Diese Suche ergibt ein in zweifacher Hinsicht interessantes Ergebnis:

1. Eine klassische Erklärung des Terminus, die bis zum Römischen Kaiserreich zurückzuführen ist,

2. Eine moderne Erklärung, die bereits einen Verweis auf den Begriff der Softwareagenten liefert.

Die erste Beschreibung bei Wikipedia [Wik04] liefert Ansätze der Bedeutung eines Agenten in der virtuellen Welt (virtueller Agent): Ein Agent handelt im Auftrag eines anderen für dessen Interessen. Durch die Unterstützung der Computerbenutzer bei der Bewältigung ihrer Aufgaben hielten so genannte *Softwareagenten* – Helfer und Interessenvertreter – Einzug in die durch Computer repräsentierte virtuelle Welt. Mit zunehmender Popularität des Internets und der verteilten, webbasierten Anwendungen erweiterte sich das Einsatzgebiet der Agenten enorm. Die Bereiche, in denen Agenten anzutreffen sind, unterscheiden sich, ähnlich wie in der realen Welt, sehr stark voneinander.

Populäres Beispiel ist der Bietmechanismus von eBay [eBa04] (früher als Bietagent bezeichnet). Der Bietagent soll einen Artikel für seinen Auftraggeber, dem Bieter, so günstig wie möglich ersteigern. Das dem Bietagent übergebene Gebot für einen Artikel stellt das Höchstgebot (Limit) dar. Der Bietagent erhöht automatisch das aktuelle Gebot solange, bis er der Höchstbietende oder bis das angegebene Limit erreicht ist. Genau genommen erfolgt dieses iterative Vorgehen zwischen zwei Bietagenten: dem des derzeitig Höchstbietenden und dem des aktuellen Bieters. Beide Agenten handeln bis zur jeweiligen Höchstgrenze, die vom Auftraggeber, dem eigentlichen Bieter, vorgegeben wurde. Dieses Beispiel stellt sogar einen Spezialfall dar: Die Agenten verhandeln untereinander ohne Beteiligung der Auftraggeber. Die Rahmenbedingungen sind dabei klar durch die Bieter und die zugrunde liegenden Geschäftsbedingungen geregelt.

Ein weiteres mögliches Einsatzgebiet für Softwareagenten liegt im eCommerce-Bereich. Ein Einkaufsagent besucht verschiedene Unternehmen oder Marktplätze im Netz und ermittelt für seinen Auftraggeber ein passendes Angebot. Dabei berücksichtigt der Agent sowohl vorgegebene Präferenzen als auch besondere Gegebenheiten vor Ort. Der Auftraggeber kann sich während der Suche seines Agenten anderen Aufgaben widmen oder sogar die Verbindung zum Internet trennen. Bei seiner nächsten Verbindung mit dem Internet bekommt er vielleicht schon eine Liste mit Angeboten. In einem Spezialfall, in dem auf einem elektronischen Marktplatz Käufer und Anbieter „anwesend“ sind, kann ein Agent sogar als Vermittler tätig werden und passende Geschäftspartner zusammenbringen.

Nach diesem kleinen Exkurs in die Computerwelt wird noch einmal die reale Welt betrachtet. Neben den menschlichen Vertretern von Agenten ist eine weitere Art des Agenten zu finden. Bevor das Internet das öffentliche Interesse geweckt hatte, wurde versucht, neue Möglichkeiten der Programmierung einzusetzen, um technischen Geräten ein eigenes Verhalten zu geben und somit zu Helfern, den Robotern oder Embodied Agents, zu machen. Automaten sind schon längere Zeit bekannt. Aber erst durch die Entwicklung von programmierbaren Bauelementen in den 60er/70er Jahren des letzten Jahrhunderts wurde begonnen, Roboter zur Lösung generischer Aufgaben zu nutzen.

Ab wann ein Roboter als Agent bezeichnet werden kann, soll an dieser Stelle nicht diskutiert werden. Der Vorstellung von Menschen sind kaum Grenzen gesetzt, wie in diversen Science-Fiction-Filmen erkennbar ist: Roboter, die im Auftrag der Regierung / Polizei arbeiten; Maschinen, die den Menschen bekämpfen; Menschen, die von Maschinen benutzt werden etc. Fakt ist: Die Technologie ist noch nicht so weit fortgeschritten, dass ein persönlicher Roboter-Butler zum Zeitungskauf an den Kiosk geschickt werden kann.

Zurück zur Begriffsklärung: Erkennbar sind deutliche Unterschiede bei Agenten aus dem menschlichen Umfeld, der Computerwelt und den Robotern. Die Aufgaben wie auch die Möglichkeiten unterscheiden sich stark. Allerdings sind Parallelen erkennbar. Allen Agenten ist eines gemeinsam:

> Agenten handeln im Auftrag und Interesse eines anderen – ihres Auftraggebers oder Besitzers.

Die Namen für Agenten mögen sich unterscheiden, ebenso deren Fähigkeiten und Aufgaben. Aber die ursprüngliche Intention des Handelnden bleibt erhalten und stellt eine erste intuitive Begriffsbildung dar.

1.1.2 Bezeichnungen für Agenten – Ein Klassifikationsversuch

Bisher wurde der Terminus Agent näher betrachtet und Beispiele aus verschiedenen Gebieten, in denen Agenten anzutreffen sind, angeführt. In diesem Kapitel wird versucht,

die Agentenbegriffe aus den unterschiedlichen Bereichen zu erfassen und zu klassifizieren. Die Klassifikation soll nach der Einordnung der Agenten in verschiedene Welten ihres Vorkommens vorgenommen werden. Auf oberster Ebene stehen zwei wesentliche Bereiche: die realen und die virtuellen Agenten. Reale Agenten sind im Gegensatz zu den virtuellen Agenten personifiziert, also mit einem Körper ausgestattet. Neben den Agenten als Person, der *Mensch* als Spion, Vermittler etc., wird unter dem Typus realer, personifizierter Agent der *Roboter* als künstlicher Agent eingeordnet.

Virtuelle Agenten „leben“ innerhalb eines Computers oder eines eingebetteten Systems. Innerhalb des Computers existiert nur Software und virtuelle Agenten werden durch Software integriert. Der virtuelle Agent wird auch als *Softwareagent* bezeichnet. Die feinere Unterscheidung der virtuellen Agenten in stationäre und mobile Agenten wird an der Eigenschaft gemessen, ob der Agent fähig ist, sich über Rechnergrenzen hinweg zu bewegen. Abbildung 1.1 veranschaulicht diese einfache Hierarchie.

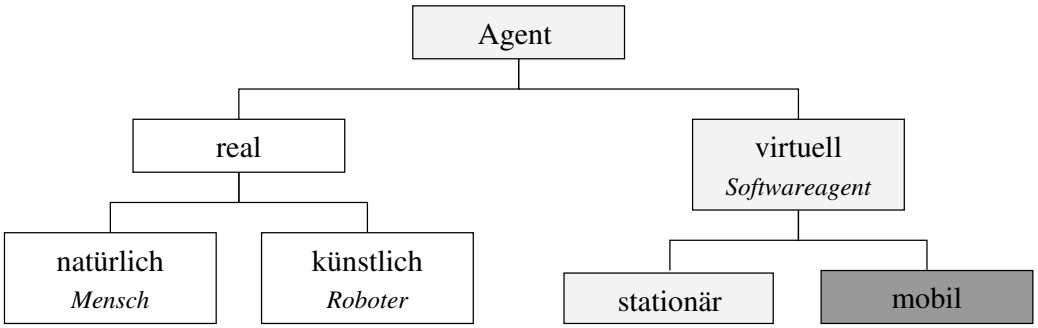


Abbildung 1.1: Klassifikation von Agenten

Die angegebene Einteilung ist einfach gehalten und soll als Übersicht dienen. Franklin und Graesser haben in [FG96] eine ähnliche Klassifikation aufgestellt. Sie haben allerdings die real-künstlichen Agenten (Robotic Agents, Roboteragenten) als eine eigene Spezies in die Taxonomie aufgenommen und erhalten auf erster Hierarchieebene eine Unterteilung in biologische Agenten, Roboteragenten und Berechnungsagenten.

Weitere Unterteilungen sind denkbar und Spezialfälle diskutierbar, beispielsweise können virtuelle Agenten nach ihren Arbeitsgebieten klassifiziert werden:

- Avatare

  Virtuelle Figuren, die eine Kommunikation mit dem Benutzer unterstützen

- Bots (Abk. für Robots)

  Software Roboter, Web-Roboter, die beispielsweise das Internet nach bestimmten Informationen durchsuchen

- Ants (Ameisen)

  Viele gleichartige Agenten, die gemeinsam eine Aufgabe lösen

- Distributed Artificial Intelligence (DAI, verteilte künstliche Intelligenz)

  Verschiedenartige intelligente Agenten, die kooperativ eine gestellte Aufgabe lösen.

Die verschiedenen Bezeichnungen verdeutlichen das breit gefächerte Arbeitsgebiet virtueller Agenten und heben die Fähigkeiten des jeweiligen Agententypus hervor. Neben der Klassifizierung nach Arbeitsgebieten kann eine Klassifikation nach anderen Merkmalen, wie Aufgabenbereiche, Vorgehensweise etc., vorgenommen werden. Zur Einordnung der vorliegenden Arbeit in das Gebiet der Agenten genügt die Klassifikation wie die Abbildung 1.1 auf der vorherigen Seite zeigt.

Im Fokus dieser Arbeit steht der virtuell-mobile Agent (Abb. 1.1 auf der vorherigen Seite: dunkelgrau hinterlegt). Durch die angegebene Hierarchie wird das Randgebiet dieser Arbeit – der virtuelle Agent (hellgrau unterlegt) – definiert. Softwareagenten können während der Arbeit mit virtuell-mobilen Agenten nicht unberücksichtigt bleiben. Nicht Bestandteil dieser Arbeit sind reale Agenten (weiß dargestellt). Zum besseren Verständnis werden bei der folgenden Betrachtung die realen Agenten einbezogen, um einen alternativen Zugang zu den virtuell-mobilen Agenten zu bekommen.

1.2 Die Umwelt des Agenten – Ein alternativer Zugang

Im Kapitel 1.1 wurde eine erste intuitive Begriffsbildung zu einem Agenten, sei es in der realen oder virtuellen Welt, gegeben: Die Intention eines jeden Agenten ist das Handeln im Auftrag eines anderen für dessen Interessen. Für die Erfüllung seines Auftrags, der Aufgabe, durchläuft ein Agent drei Stufen:

1. Die *Wahrnehmung* der Umwelt,

2. Die *Interpretation* des Wahrgenommenen zum Verständnis der Aufgabe,

3. Die *Interaktion* mit der Umwelt zur Übertragung und Bewältigung der Aufgabe.

Die Art und Weise der Wahrnehmung, Interpretation und Interaktion ist bei den Agenten der unterschiedlichen Welten verschiedenen. Die notwendigen Fähigkeiten der realnatürlichen, real-künstlichen und virtuellen Agenten werden unter Berücksichtigung dieser drei Stufen im Folgenden herausgearbeitet. Nach ausführlicher Betrachtung der einzelnen Agententypen erfolgt eine zusammenfassende Darstellung der Gemeinsamkeiten

und Unterschiede (siehe Tabelle 1.1 auf Seite 15). Die virtuell-stationären und virtuell-mobilen Agenten werden gemeinsam betrachtet, da die Gemeinsamkeiten beider Agententypen überwiegen. Auf die Fähigkeiten des virtuell-mobilen Agenten wird gesondert eingegangen.

1.2.1 Real-natürliche Agenten – Der Mensch

Der Mensch als Ausgangspunkt für die Beschreibung der Wahrnehmung, Interpretation und Interaktion steht im Vordergrund dieses Kapitels. Seine Fähigkeiten sind natürlich gewachsen und sein Bewusstsein hat sich im Laufe seines Lebens entwickelt.

Wahrnehmung Die Umwelt eines Agenten in der realen Welt entspricht im Wesentlichen all jenem, was ein Mensch mit seinen natürlichen Sinnen wahrnehmen kann: Sehen, Hören, Riechen, Schmecken und Tasten als Gesamtheit aller Sinne. Stanislav Lem geht noch weiter und kreiert in seinem 1999 erschienenen Buch „Die Megabitbombe – Essays zum Hyperspace“ [Lem03] für die Gesamtheit aller Sinne die Wortschöpfung *Sensorium*. Darunter versteht er:

> Das Sensorium ist die Gesamtheit aller Sinne sowie aller (gewöhnlichen Nerven-) Wege, durch welche die Informationen, die uns über die „Existenz von etwas“ unterrichten, zum zentralen Nervensystem gelangen.

Die Wahrnehmung, „die Unterrichtung der Existenz von etwas“, ist dabei subjektiv, ebenso wie die Verarbeitung der Reize und die Reaktion auf Umweltereignisse. Lem meint dazu weiter:

> Jedes lebendige Geschöpf besitzt ein eigenes Sensorium, das typisch für die Art ist und sich über Millionen von Jahren der natürlichen darwinistischen Evolution entwickelt hat.

Die darwinistische Evolution führte zur Entwicklung eines Sensoriums zur Wahrnehmung. Die Wahrnehmung durch das Sensorium ist auf eine gewisse Reichweite beschränkt. Ein Gegenstand, der ertastet werden möchte, muss direkt erreichbar, greifbar sein. Geräusche können zwar aus der Ferne wahrgenommen werden, eine räumliche Begrenzung ist dennoch gegeben. Zudem ist die menschliche Wahrnehmung auf einen gewissen Zeitpunkt, den Wahrnehmungszeitpunkt, begrenzt. Die räumliche Begrenzung, der Wahrnehmungszeitpunkt, Kausalitäten etc. werden unter dem Begriff des Wahrnehmungskontexts, der durch die Fähigkeiten der Sinnesorgane oder die Umwelt selbst begrenzt sein kann, zusammengefasst.

Die „technische Evolution“ hat in den letzten Jahren eine *Sensorik* zur Wahrnehmung entwickelt, die eine technische Erweiterung des natürlichen Sensoriums darstellt und damit

den Wahrnehmungskontext erweitert. Beispielsweise verbessert eine Brille oder ein Fernglas das Sehvermögen. Mit einem Fernseher kann noch weiter in die Ferne gesehen werden. Eine wesentliche technische Errungenschaft ist die Erweiterung des Sensoriums in anderen Frequenzbereichen. Die Fähigkeit des Sehens kann beispielsweise durch Nachtsichtgeräte erweitert werden.

Der Wahrnehmungskontext kann indirekt durch technische Werkzeuge erweitert werden. Komplexe Systeme, wie das Global Positioning System (GPS), werden nicht nur direkt zur Wahrnehmung einer Position, zur Orientierung, verwendet. GPS kann dazu benutzt werden, um beispielsweise eine Abbildung der realen Welt als Landkarte zu erstellen. Die Landkarte dient als direkte Orientierungshilfe, GPS in dem Fall als indirekte.

Das Wahrnehmungsvermögen eines Menschen bzw. eines Agenten kann in eine natürlich gegebene Wahrnehmung und eine technisch-innovative, erweiterte Wahrnehmung unterteilt werden. Die *natürliche Wahrnehmung* ist offensichtlich ausreichend im Hinblick auf das Überleben der Spezies. Die *Werkzeug-basierte, erweiterte Wahrnehmung* trägt dazu bei, neben dem Erkennen der nicht direkten Umgebung auf eine veränderte Umwelt oder auf Umweltereignisse besser reagieren zu können.

Interpretation Für einen menschlichen Agenten ist es besonders wichtig, die Absichten und Ziele des Auftraggebers richtig zu erkennen. Der Agent muss die Ziele und Absichten verstehen. Dabei wird altes Wissen verwendet und Erfahrung eingebracht. Bestehende Überzeugungen und vorhandenes Wissen können auf Grund neuer Information verändert werden. Erfahrungen können erweitert werden. Die Interpretation ist ein fortwährender *Bildungsprozess*, der in das *Bewusstsein* eines jeden Menschen aufgenommen wird. Jeder Mensch hat im Laufe seines Lebens ein Bewusstsein entwickelt, das ihn befähigt, gestellte Aufgaben zu verstehen und mit der übergebenen Verantwortung umzugehen.

Interaktion Die Interaktion eines Menschen erfolgt einerseits durch das Handeln, das *Agieren* mit seiner Umwelt. Andererseits interagiert er durch verbale und / oder nonverbale Kommunikation. Zur *verbalen Kommunikation* zählt neben der Sprache ebenso die Schrift. Gestik und Mimik, als Mittel der *nonverbalen Kommunikation*, werden zur Verstärkung der Kommunikation genutzt und erweitern die Ausdrucksmöglichkeiten. Der Mensch setzt neben seinen sprachlichen Fähigkeiten (verbale Kommunikation) seinen Körper mit der Fähigkeit der Bewegung (nonverbale Kommunikation) und der Fähigkeit zur Fortbewegung (Agieren) ein.

Fazit: Der real-natürliche Agent, der Mensch, nimmt über das Sensorium die Umgebung in seinem Wahrnehmungskontext auf. Das menschliche Sensorium ist durch eine natürliche Evolution geprägt. Der Mensch hat über Millionen von Jahren eine natürliche

Wahrnehmung entwickelt. Durch technische Sensorik kann der Wahrnehmungskontext erweitert werden (Werkzeug-basierte, erweiterte Wahrnehmung). Das Wahrgenommene wird durch das menschliche Bewusstsein interpretiert. Die Interpretation ist ein Bildungsprozess, der durch altes Wissen und neue Erfahrungen fortgeführt wird. Dem menschlichen Bewusstsein obliegt es je nach Interpretation des Wahrgenommenen entsprechend zu reagieren und demzufolge mit der Umwelt durch Kommunikation und durch Handeln zu interagieren. Die Ausdrucksmöglichkeiten der verbalen Kommunikation werden durch die nonverbale Kommunikation intensiviert, der Interpretationsspielraum eingeschränkt und die Interpretation vereinfacht.

Durch das komplexe Zusammenspiel des Sensoriums kann der menschliche Agent seinen Auftraggeber natürlich wahrnehmen. Der wahrgenommene Auftraggeber interagiert mit dem Agenten, um die Aufgabe, die vom Agent interpretiert wird, zu übermitteln. Die Interpretation wird von der Wahrnehmung und Erfahrung beeinflusst, das zur richtigen Interpretation der Aufgaben, aber auch zu Fehlinterpretationen führen kann.

1.2.2 Real-künstliche Agenten – Der Roboter

Die reale Welt ist äußerst komplex. An die Sensorik eines Roboters werden hohe Anforderungen gestellt. Die Roboter, die im Folgenden betrachtet werden, sind personifizierte und programmgesteuerte Maschinen, deren Arbeitsweise der eines Menschen nachgebildet ist. Personifizierte Roboter sollen manuelle Tätigkeiten eines Menschen ausführen können.

Wahrnehmung Roboter, insbesondere Industrieroboter, haben sich zu unentbehrlichen Helfern von Menschen entwickelt, da sie einerseits für monotone, andererseits für gefährliche Arbeiten eingesetzt werden können. Der Roboter ist zu keiner natürlichen Wahrnehmung der Umwelt fähig. Hier ist der Mensch gefordert, um den Roboter mit einer *Werkzeug-basierten Wahrnehmung* auszustatten.

Die Anforderungen an mögliche Wahrnehmungsfähigkeiten eines Roboters sind so unterschiedlich wie deren Anwendungsgebiete. Die für das jeweilige Einsatzgebiet wichtigen Wahrnehmungsfähigkeiten werden ausschließlich durch *technische Sensorik* nachgebildet. Wird ein Entwurf von Robotern angestrebt, die als Agenten handeln sollen, müssen komplexe Wahrnehmungsfähigkeiten integriert (programmiert) werden, wobei sehr hohe Ansprüche an die Sensorik eines Roboters zu stellen sind.

„Roboter-Agenten“ sollen adaptiv sein. Sie sollen an spezielle Aufgaben angepasst werden oder sich sogar selbst anpassen. Für die selbständige Anpassung benötigt der Roboter eine Werkzeug-basierte Wahrnehmung. Die Informationen von der externen Wahrnehmung müssen über technische Sensorik aufgenommen und verarbeitet werden. Bei der

Sensorik orientiert sich der menschliche Erschaffer meist an seinem eigenen Sensorium. So werden beispielsweise CCD-Kameras für die visuelle Wahrnehmung eingesetzt und Drucksensoren als Tastorgane integriert. Eine Werkzeug-basierte, erweiterte Wahrnehmung des Roboters, die in einem für Menschen nicht wahrnehmbaren Bereich liegt, wird zur Stärke des Roboters.

Interpretation Neben der Wahrnehmung der Umwelt durch den Roboter muss die richtige Interpretation der übergebenen Aufgabe bewältigt werden. Dies muss dem Roboter per Software angelernt werden – *softwaretechnisch-integrierte Interpretation*. Da die Interpretation vom Wahrnehmungskontext abhängt und der Mensch je nach Kontext reagiert, ist es schwer, den Roboter zu programmieren. Dies zeigt die hohe Komplexität der gestellten Aufgabe, Roboter als Agenten zu nutzen. Das Bewusstsein von Menschen in einen „Roboter-Agenten“ zu integrieren, wird vorerst nicht möglich sein. Umso schwieriger wird die Realisierung, wenn die räumliche Distanz zum Roboter enorme Ausmaße annimmt, wie z. B. bei dem Landeroboter „Beagle“ der Mars Express Mission [Eur04]. Erschwerend kommt hinzu, dass die Umweltbedingungen vor Ort nicht denen der Erde entsprechen und teilweise unbekannt sind.

Das Forschungsgebiet der Künstlichen Intelligenz (KI) beschäftigt sich zum Teil mit diesem Bereich. Fundierte mathematische Theorien (z. B. Prädikatenlogik) werden eingesetzt, um Softwarekomponenten ein intelligentes Verhalten zu geben und lernfähig zu machen. Eine Grundidee ist die Nachbildung des menschlichen Gehirns durch neuronale Netze. Diese Netze bestehen aus simplen „Bauteilen“, den Neuronen, die mit anderen Neuronen beliebig vernetzt werden können. Auf Grund einer Kombination von (Eingangs-) Signalen, die über die Vernetzung übertragen werden, liefern die Neuronen ein Ausgangssignal, das auf einer einfachen mathematischen Funktion (z. B. Schwellwertfunktion) basiert. Tatsächlich sind diese Netze lernfähig und liefern bei einer entsprechenden, unbekannten Input-Konstellation eine Menge von Ausgabewerten. Diese Ausgabewerte können zur Einordnung des Wahrgenommenen in ein gelerntes Schema benutzt werden. Die Sensoren bilden dabei die Eingabeneuronen, die für eine Wahrnehmung der Umwelt verantwortlich sind. Ausgabeneuronen steuern eine Reaktion auf Veränderungen der Umwelt (Interaktion). Nachteilig ist die hohe Komplexität dieser Netze, die für einen Menschen schwer erfassbar ist. Dementsprechend kompliziert ist die Gestaltung dieser Netze.

Interaktion Real-künstliche Agenten agieren meist in der Umgebung von Menschen oder sogar für Menschen. Gefordert sind Interaktionsfähigkeiten, die dem Menschen ähneln. Damit ergeben sich zwei wesentliche Interaktionsfähigkeiten, die von einem Roboter verlangt werden:

1. Der Roboter muss mit seiner Umwelt, insbesondere den Menschen interagieren

können,

2. Der Roboter muss sich in der realen Umwelt bewegen und auf Umweltreize reagieren können.

Um mit seiner Umwelt, den Menschen, zu interagieren, ist Kommunikation die intuitive Variante. Die Kommunikation kann einerseits über die Sprache mit Hilfe von Spracherkennungssystemen realisiert werden, andererseits über Schalter am Roboter selbst oder durch schriftliche Kommunikation über Terminal und Monitor (verbale Kommunikation). Eine Nachricht, die an den Softwarekern eines Roboters über derartige Schnittstellen übermittelt wird, kann bereits einen kompletten Algorithmus beinhalten. Die Fähigkeit zur Kommunikation wird im Roboter über Software realisiert – *softwaretechnisch-integrierte Kommunikation*.

Für die Reaktion auf Umweltreize muss der Roboter eine Bewegung zu den Zielen organisieren und umsetzen können. Entsprechend integrierte Softwarekomponenten ermöglichen die Ansteuerung der Hardwarekomponenten – *softwaretechnisch-realisiertes Agieren*. Durch Software können die Signale von der externen Wahrnehmung über die äußere Sensorik verarbeitet, Abhängigkeiten geregelt und Hardwarekomponenten, wie beispielsweise Roboterarme, in Bewegung versetzt werden. Die Realisierung einer nonverbalen Kommunikation ist begrenzt möglich.

Fazit: An real-künstliche Agenten werden nahezu die gleichen Anforderungen gestellt wie an einen menschlichen Agenten. „Roboter-Agenten“ müssen im Vergleich zum Menschen mit einer ähnlichen oder gar besseren Sensorik zur Wahrnehmung ausgestattet werden. Die Wahrnehmung ist keine natürliche, sondern eine Werkzeug-basierte Wahrnehmung.

Neben der Wahrnehmung der Aufgabe ist das Erkennen des gewünschten Ziels (Interpretation) eine weitere Schwierigkeit. Die richtige Deutung der Informationen ist entscheidend. Eine Deutung ohne eigenes Bewusstsein ist schwer möglich. Der Programmierer hat die Aufgabe, ein Bewusstsein in den Agenten zu integrieren. Ein Algorithmus muss programmiert werden, der das Bewusstsein des Menschen nachbildet – softwaretechnisch-integrierte Interpretation.

Ähnliches gilt für die Interaktion mit der Umwelt. Ein Roboter-Agent muss mit Menschen interagieren, kommunizieren können. Die Kommunikationsfähigkeit wird softwaretechnisch integriert. Eine künstliche Motorik zur Bewegung in der realen, natürlichen Welt ist für die Interaktion mit der Umwelt erforderlich. Um diese Bewegung zu steuern, ist Software notwendig – softwaretechnische-realisiertes Agieren.

1.2.3 Virtuelle Agenten – Der Softwareagent

In der Computerwelt übernehmen autonome Programme die vom Menschen gestellten Aufgaben und werden somit zu virtuellen Agenten im „Cyberspace“. Virtuell-stationäre und virtuell-mobile Agenten werden auf Grund der Gemeinsamkeiten bezüglich der Wahrnehmung, Interpretation und Interaktion zusammen unter dem Typus Softwareagent betrachtet.

Wahrnehmung Im Gegensatz zu der Umwelt von Menschen und Robotern, einer realen (Um-) Welt, ist die wahrgenommene Umwelt von Softwareagenten eine rein virtuelle Welt: andere Softwareagenten, Betriebssystemkomponenten, Anwendungen, Dienste in einem Computersystem – „Software sieht Software“. Die Wahrnehmung anderer Softwarekomponenten erfolgt über (standardisierte) Schnittstellen – *Schnittstellen-basierte Wahrnehmung*. Diese Schnittstellen können mit dem menschlichen Sensorium und der technischen Sensorik der realen Welt verglichen werden. Die Schnittstellen sind rein virtuell und werden im Weiteren als *Sensoren* bezeichnet.

Neben der externen Wahrnehmung besitzt der Softwareagent eine *interne Wahrnehmung*. Mit dieser kann der Agent seine eigenen Fähigkeiten (z. B. seine Geschwindigkeit mit der er arbeitet) und Informationen (z. B. aktueller Status) wahrnehmen.

Die Sensoren werden nicht nur zur Wahrnehmung der virtuellen Umwelt verwendet, sondern auch für die Wahrnehmung eines menschlichen Auftraggebers. Genau genommen nimmt der Agent den Menschen nicht direkt war, sondern die Informationen vom Menschen. Diese Informationen erhält der Agent über eine Schnittstelle von einer anderen Softwarekomponente. Der Mensch wird aus Sicht des Agenten zu einem Stück Software.

Interpretation Der virtuelle Agent ist nicht zu einer „natürlichen Interpretation“ fähig. Die Fähigkeit zur Interpretation muss im Softwareagenten „programmiert“ werden, ähnlich wie bei einem Roboter – *softwaretechnisch-integrierte Interpretation*. Gegebenenfalls ist dazu der Einsatz von Techniken der KI erforderlich. Werden diese Techniken in einen Softwareagenten integriert, wird von einem intelligenten Softwareagenten gesprochen. Das Gebiet der intelligenten Softwareagenten stellt einen großen Teil der virtuellen Agenten dar, soll jedoch in dieser Arbeit nicht betrachtet werden. Hierfür wird auf weiterführende Literatur verwiesen (z. B. [MT02]).

Interaktion Die Interaktion von Softwareagenten mit der Umwelt beschränkt sich auf die Kommunikation und wird über Schnittstellen realisiert – *Schnittstellen-basierte Kommunikation*. Dabei wird zwischen der Kommunikation innerhalb des Computersystems (virtuelle Welt) und der Kommunikation mit dem Benutzer nach außen (reale Welt) un-

terschieden. In beiden Fällen interagiert der Agent mit wahrgenommenen Softwarekomponenten. Die Kommunikation wird ausschließlich durch die Übermittlung von Nachrichten (schriftliche Kommunikation) geführt. Eine Nachricht kann dabei aus einfachem Text, strukturierten Daten oder Anweisungen bestehen.

Die Kommunikation ist das einzige Mittel zur Interaktion mit der realen Umwelt. Entsprechend ergonomisch muss diese Kommunikation gestaltet werden. An der Systemgrenze sind die Verwendung von Symbolen oder sogar animierten Figuren (Avataren) Techniken zur Verbesserung der Interaktion mit der realen Welt. Der Softwareagent kann beispielsweise durch einen elektronischen Einkaufsberater „personifiziert“ werden. Die Interaktion mit einem Menschen erscheint natürlicher und beschränkt sich nicht nur auf den Austausch von Textnachrichten. Möglich ist die Arbeit mit Spracherkennungssystemen, bei der ein Auftraggeber seine Aufgabe über die natürliche Sprache dem „Agenten“ mitteilen kann und der „Agent“ sogar sprechen „gelernt“ hat. Gestik und Mimik können von der animierten, „personifizierten“ Figur ebenso benutzt werden.

Fazit: Im Vergleich zur realen Welt sind die Anforderungen an die Wahrnehmung und Interaktion in der virtuellen Welt grundlegend anders. Die im Wesentlichen durch Software gestaltete Umwelt muss wahrgenommen werden. Die Wahrnehmung ist Schnittstellen-basiert. Der Wahrnehmungskontext virtueller Agenten wird durch die verfügbaren Softwareschnittstellen definiert.

Bei der Interpretation der übertragenen Aufgabe muss dem virtuellen Agenten ein Bewusstsein integriert werden. Die softwaretechnisch-integrierte Interpretation ist ähnlich wie bei real-künstlichen Agenten äußerst komplex und mit derzeitigen Technologien nur bedingt erreichbar.

Die Schnittstellen-basierte Interaktion bedingt die Kommunikation über Nachrichten. Zur Interaktion mit einem menschlichen Auftraggeber sollte das Mittel zur Visualisierung von Aktionen und Reaktionen (Gestik, Mimik, Motorik) durch Animation auf externen hardware-basierten Geräten (Monitore) genutzt werden.

1.2.4 Virtuell-mobile Agenten – Der mobile Agent

Der mobile Agent ist ein Spezialtyp der virtuellen Agenten mit der Fähigkeit, über Systemgrenzen hinweg zu migrieren (wandern). In diesem Kapitel werden nur Erweiterungen diskutiert, die sich durch den größeren Wahrnehmungsbereich der mobilen Agenten ergeben. Virtuell-stationäre Agenten können ebenso von den Erweiterungen profitieren.

Wahrnehmung – Erweiterung zu Softwareagenten Die Wahrnehmung bedarf einer Erweiterung: Der Agent soll über die Systemgrenzen hinweg „sehen“ können und andere

Systeme wahrnehmen. Über die Schnittstellen-basierte Wahrnehmung muss der Agent die Fähigkeit erlangen, Informationen zu entfernten Orten korrekt wahrzunehmen. Gemeint sind in diesem Fall: Informationen, die den Ort näher beschreiben sowie die Auskunft über die Erreichbarkeit des Ortes, die Dauer der Migration und über Kosten, die während der Reise entstehen können. Letztendlich benötigt ein mobiler Agent eine „Landkarte“ (Webkarte), die er zu seiner Reiseplanung benutzen kann. Die Webkarte weist im Unterschied zu den uns bekannten Landkarten eine um Größenordnungen höhere Dynamik auf. Eine dem Agenten übergebene Karte ergibt in den wenigsten Fällen „im Rucksack des Agenten“ einen Sinn, da sie bereits nach kürzester Zeit unaktuell sein würde. Eine Webkarte muss in entsprechenden Zeitintervallen aktualisiert werden und Bestandteil der externen Wahrnehmung sein.

Für die Wahrnehmung der erweiterten Umwelt von mobilen Agenten spielt das Werkzeug Karte eine wesentliche Rolle. Als Bestandteil der (lokalen) Umgebung eines Agenten ist die Kartenkomponente ein Orientierungsmittel, das über definierte Schnittstellen vom Agenten benutzt werden kann. Durch eine Karte wird die Reichweite der Wahrnehmung erweitert. Im Gegensatz zu einer explorativen Art und Weise, die virtuelle Welt zu erkennen, hat der mobile Agent nun die Chance, eine Vorstellung seiner / der virtuellen Welt aufzubauen und Gesetzmäßigkeiten zu erkennen. Der stationäre Agent kann die Informationen der Karte ebenfalls als Hilfsmittel zur Orientierung benutzen, beispielsweise zur Lokalisierung entfernter virtueller Agenten.

Eine Besonderheit stellt die Wahrnehmung während einer Reise eines Agenten in der virtuellen Welt dar. Während ein Mensch eine Reise genießen kann, für einen Roboter jedoch eine Herausforderung darstellt, werden mobile Agenten zum Zweck der Übertragung schlafen gelegt, wobei jegliche Wahrnehmungsfähigkeiten eingefroren sind. Die Reise für den Agenten existiert als solche überhaupt nicht und wird nur durch einen zeitlichen Versatz und einen Lokalitätswechsel wahrgenommen.

Interpretation – Erweiterung zu Softwareagenten Die Interpretation eines mobilen Agenten kann über die bekannten Techniken der KI erfolgen, wobei eher Einschränkungen als Erweiterungen vorgenommen werden müssen. Einschränkungen sind notwendig, da die einem mobilen Agenten softwaretechnisch-integrierte Intelligenz einen zusätzlichen Ballast bei der Reise darstellt. Die Intelligenz muss ausgelagert werden. Für einen statischen Teil der Intelligenz, dem Algorithmus zum intelligenten Handeln, ist dies möglich. Ein Agent kann die ausgelagerte Intelligenz über Schnittstellen benutzen, um auf Basis seines Wissens Entscheidungen zu treffen. Das Wissen, der dynamische Teil des Bewusstseins, kann nicht ausgelagert werden. Soweit die notwendige Intelligenz nicht vorhanden ist, ist der Agent in seiner Entscheidungsfähigkeit eingeschränkt.

Dennoch ist eine Erweiterung der Interpretation für mobile Agenten notwendig: Der mobile Agent soll autonom entscheiden können, wann und wohin er migrieren kann. Hierfür

muss er die wahrgenommene Karte interpretieren können.

Interaktion – Erweiterung zu Softwareagenten Die Schnittstellen-basierte Interaktion mobiler Agenten ist komplexer als bei stationären Softwareagenten und nicht nur auf ein Computersystem beschränkt. Mobile Agenten halten sich auf ihrer Reise durch das Netzwerk in unterschiedlichen Computersystemen auf und kommunizieren häufiger mit anderen Agenten, beispielsweise zum Austausch von Informationen, Nutzen von Diensten etc. Die Kommunikation der Agenten untereinander erhöht die Chance, das vom Agenten verfolgte Ziel mit guter Qualität zu erreichen.

Fazit: Die grundsätzlichen Anforderungen an die Wahrnehmung mobiler Agenten ist zu denen der stationären Agenten identisch. Durch die Fähigkeit zur Migration wird die Wahrnehmung der virtuellen Welt jedoch deutlich erweitert. Mobile Agenten müssen fremde Orte und deren Ortsinformationen wahrnehmen. Diese Informationen müssen dem Agenten beispielsweise durch eine Karte der virtuellen Welt zur Verfügung gestellt werden. Der Agent muss zur Interpretation dieser Karte fähig sein, um autonom im Agentennetzwerk zu migrieren. Die Interaktion mobiler Agenten wird komplexer, da sie in unterschiedlichen Systemen agieren und verstärkt mit anderen Agenten kommunizieren.

1.2.5 Zusammenfassung: Mensch, Roboter, Softwareagent

Die folgende Tabelle 1.1 stellt einen Überblick über die Gemeinsamkeiten und Unterschiede der Wahrnehmung, Interpretation und Interaktion der einzelnen Agententypen dar.

<table>
<tr><th>Agent</th><th colspan="2">Wahrnehmung</th><th>Mittel zur Wahrnmg.</th><th>Interpretation</th><th>Interaktion</th></tr>
<tr><td></td><td>allg.</td><td>erw.</td><td></td><td></td><td></td></tr>
<tr><td>Mensch</td><td>natürlich</td><td>Werkzeug-basiert</td><td>Sensorium</td><td>Bildungsprozess, Bewusstsein</td><td>Agieren, verbale Komm., nonverb. Komm.</td></tr>
<tr><td>Roboter</td><td colspan="2">Werkzeug-basiert</td><td>Sensorik</td><td>swt-integriert</td><td>swt-real. Agieren, swt-integ. Komm.</td></tr>
<tr><td rowspan="2">SW-Agent</td><td colspan="2">Schnittst.-basiert</td><td rowspan="2">Sensoren</td><td rowspan="2">swt-integriert</td><td rowspan="2">Schnittst.-basierte Komm.</td></tr>
<tr><td colspan="2">intern</td></tr>
</table>

Tabelle 1.1: Wahrnehmung, Interpretation und Interaktion von Agenten

Abbildung 1.2 auf der nächsten Seite präsentiert den an (mobile) Agenten gestellten

Anspruch bei der Integration in die virtuelle Welt. Die Fähigkeiten zur Wahrnehmung und Interaktion mobiler Agenten müssen ausgebaut werden. Die Intention eines mobilen Agenten bedingt die Wahrnehmung der Umwelt über Systemgrenzen hinweg. Zusätzliche Werkzeuge, die ein Agent über Schnittstellen nutzen kann, müssen zur Verfügung stehen. Gute Interpretationsfähigkeiten verbessern die Wahrnehmung des Agenten und gestatten eine effiziente Interaktion.

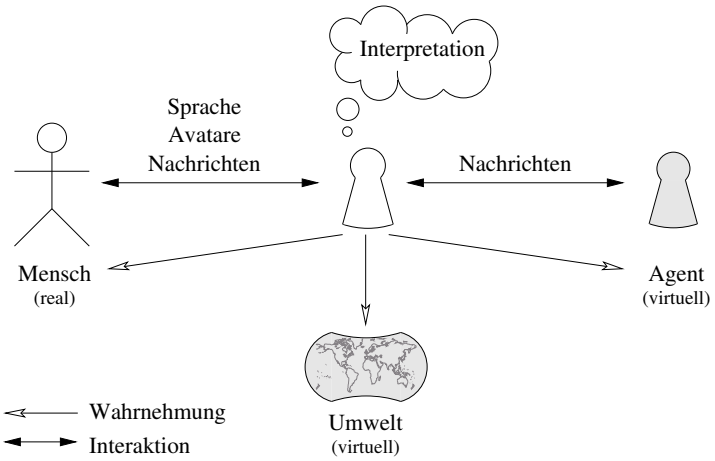


Abbildung 1.2: Integration von Agenten in die virtuelle Welt

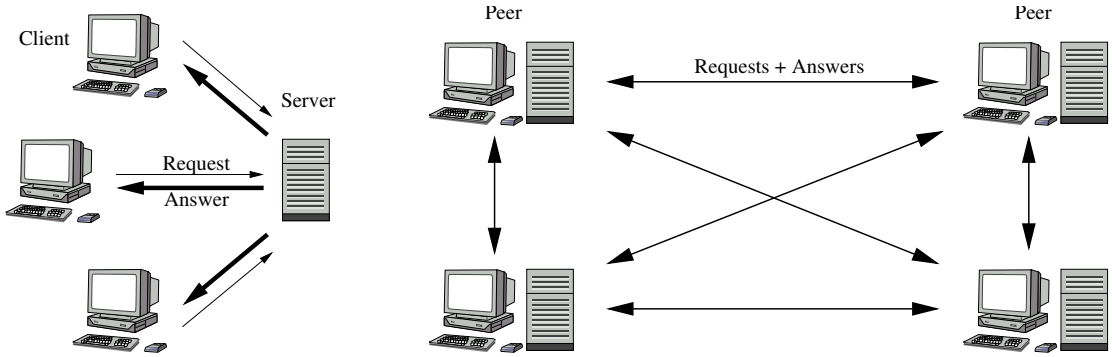
1.3 Mobile Agenten der Softwaretechnik

Zu Beginn dieses Kapitels wird die Neuartigkeit der mobilen Agenten bzw. der Mobilität von Algorithmen den traditionellen Technologien gegenübergestellt. Anschließend erfolgt eine Charakterisierung von mobilen Agenten mit Hilfe von bekannten Definitionen der Softwareagenten und im Speziellen der mobilen Agenten. Zudem wird der Lebenszyklus (mobiler) Agenten dargestellt. Bevor im letzten Teil die Architektur von Agentensystemen, die Ausführungsumgebung für Agenten, betrachtet wird, werden der Stand der Technik und Tendenzen erörtert.

1.3.1 Technologie-Innovation

Mobile Agenten bzw. mobile Algorithmen (Code) können als neues Paradigma der verteilten, objekt-orientierten Programmierung (siehe einführendes Kapitel [BR04]) angesehen werden. Traditionelle Techniken wie der Remote Procedure Call (RPC) [BN84] oder das aus der Java-Welt bekannte Remote Method Invocation (RMI) [WW04] setzen den Client-Server-Gedanken um: Dienstnutzer (Clients / Klienten) nehmen Dienste

von Diensterbringern (Server) wahr (siehe Abb. 1.3). Die Rollen sind klar verteilt und im Normalfall statisch. Ein Ausfall eines Servers führt generell zum Ausfall des angebotenen Dienstes. Klienten haben kaum eine Alternative, da sie meist fest an den Server gebunden sind. In einem Peer-to-Peer-System (P2P-System) werden beide Rollen gleichzeitig von einem so genannten *Peer* übernommen. Die Knoten im Netzwerk sind zugleich Klienten und Server bzw. wechseln bei Bedarf diese Rollen.



(a) Client-Server (b) Peer-to-Peer

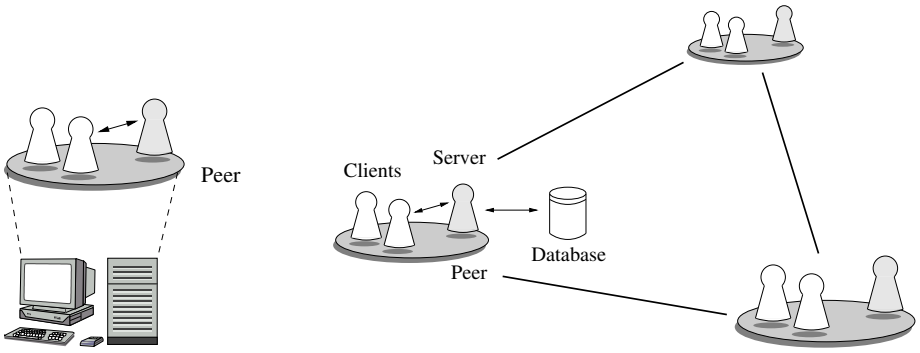
Abbildung 1.3: Der Client-Server-Ansatz vs. Peer-to-Peer

P2P-Systeme sind robuster: Fällt ein als Server fungierender Peer aus, können die Aufgaben durch einen anderen Peer übernommen werden. Diese Netze sind zu einer höheren Dynamik, wie sie in heutigen webbasierten Applikationen notwendig ist, fähig und können mit Problemen, die bei traditionellen Techniken auftreten, besser umgehen.

Dieses System aus mobilen Agenten stellt ein P2P-System dar (siehe Abb. 1.4 auf der nächsten Seite). Auf den Peers befinden sich Agenten, die als Klienten oder Server auftreten können. Typischerweise übernehmen stationäre Agenten die Rolle des Servers (in Abb. 1.4 grau dargestellt). Mobile Agenten (weiß dargestellt) nutzen die auf unterschiedlichen Peers angebotenen Dienste der stationären Agenten.

Das Innovative dieser Technologie ist der Transfer der Verarbeitungslogik, des Algorithmus, zum Server. Die Daten werden dort verarbeitet und die gefilterten, konzentrierten Ergebnisse an den Klienten zurückgesendet. Im Gegensatz dazu wird bei RMI, RPC und ähnlichen Techniken eine Anfrage an den Server gesendet. Die Ergebnisdaten werden ungefiltert an den Klienten zurückgegeben. Eine anschließend notwendige Bearbeitung der Daten wird auf dem Klienten vorgenommen. Unter Umständen sind weitere Anfragen notwendig.

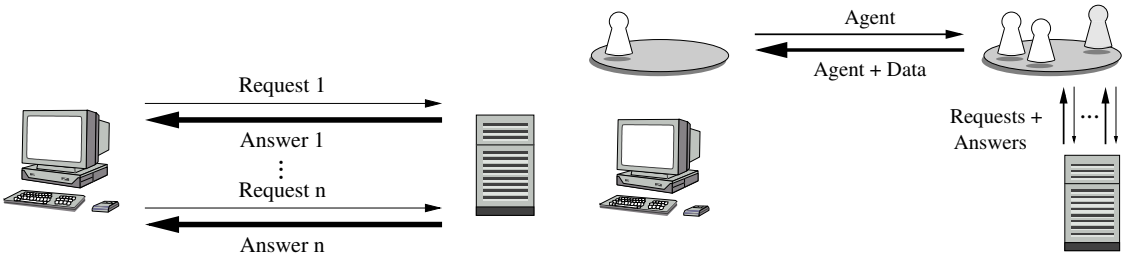
Mobile Agenten wandern zum Server, stellen lokal Anfragen, verarbeiten die Ergebnisdaten an Ort und Stelle und kehren mit der letztendlich gewonnenen Ergebnismenge zurück.



(a) Agenten-Peer (b) Netzwerk mit Agenten

Abbildung 1.4: Ein Netzwerk mit mobilen Agenten als Peer-to-Peer-System

Der Nachteil einer erneuten Datenanfrage des Klienten an den Server wird durch die Idee der mobilen Agenten vermieden. Vorteile bestehen dann, wenn Abfragen Abhängigkeiten aufweisen und / oder die endgültige Ergebnismenge eine hohe Selektivität aufweist. Mobile Agenten verursachen eine geringere Netzwerklast, da die Auswertung vor Ort erfolgt und nicht alle Daten vom Server über das Netzwerk transferiert werden müssen (siehe Abb. 1.5).

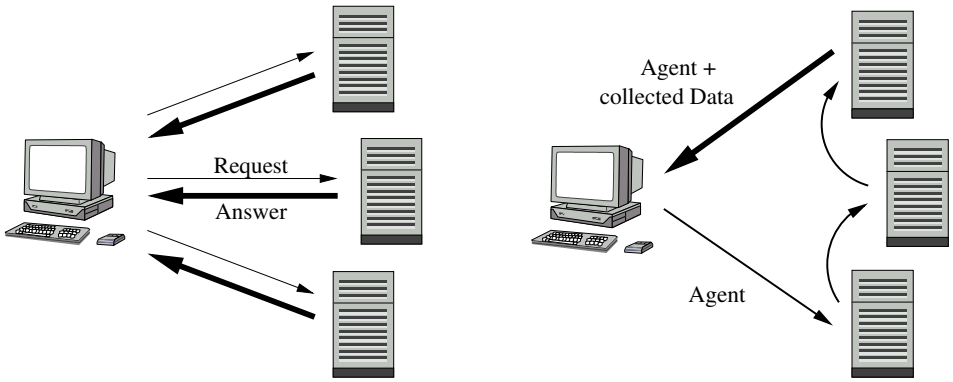


(a) Client-Server (b) Mobile Agenten

Abbildung 1.5: Vorteil mobiler Agenten: Optimierung der Netzlast

Besonders interessant ist der Fall, in dem die Rechenkraft des Klienten eingeschränkt und / oder die Datenübertragungsrate beschränkt bzw. die Verbindung kostenintensiv ist. Mobile Endgeräte, wie PDAs, Smartphones etc., weisen beispielsweise solche Eigenschaften auf. Rechenbelastungen solcher Geräte sowie hoher Datentransfer zu diesen Geräten kann durch das Agentenparadigma vermieden werden. Der im Normalfall „schlanke“ Algorithmus ist schnell zum Server übertragen. Die deutlich besseren Ressourcen des Servers können genutzt werden.

Sollen mehrere Server (Informationsquellen) im Netzwerk abgefragt werden, kommt ein weiterer Unterschied zu den traditionellen Techniken zum Tragen: Der mobile Agent kann diese Quellen in einer Rundreise besuchen, während bei RPC und RMI eine sternförmige Abfrage der Quellen umgesetzt werden muss (siehe Abb. 1.6).



(a) Client-Server (b) Mobile Agenten

Abbildung 1.6: Neue Möglichkeiten: Rundreise Mobiler Agenten

Der mobile Algorithmus verdient auf den ersten Blick nicht unbedingt den Namen Agent, da die Intention des Agenten das Handeln im Auftrag eines anderen ist. Näher betrachtet sind diese mobilen Agenten jedoch nicht nur simple Nachrichtenüberbringer, sondern bewegen sich eigenständig im Netzwerk, um eine übertragene Aufgabe zu lösen. Durch das komplexe Zusammenspiel von Klienten und Server entstehen Applikationen, die im Netzwerk verteilt sind (siehe Abb. 1.7 auf der nächsten Seite).

Agenten-basierte P2P-Systeme zeichnen sich durch die Eigenschaften mobiler Agenten aus, insbesondere die (eigenständige, proaktive) Mobilität von Algorithmen. Solche Systeme besitzen eine hohe Dynamik. Die Komponenten befinden sich in einer Netzwerkumgebung, in der Knoten zum System hinzukommen oder wegfallen und bei der sich Eigenschaften der Verbindungen zwischen den Knoten ändern.

Trotz der Vorteile mobiler Agenten gibt es bis heute keinen Durchbruch der Technologie. Ein Grund dafür sind Sicherheitsbedenken der potentiellen Nutzer. Teilweise werden Parallelen zu Computerviren gezogen. Dieser Vergleich ist falsch und beruht auf Unkenntnis über mobile Agenten. Computerviren sind Programme, die versteckt in ein System eindringen. Mobilen Agenten wird der Zutritt explizit gestattet und zwar in einem geschützten Bereich des Systems, der Java-Sandbox [Dag04]. Würde ein Agent böswillig eindringen wollen, hat er von diesem Bereich aus keinen Zugriff auf das System. Das gleiche Prinzip wird erfolgreich und ohne Bedenken bei Applets [Sun04a], die Teil einer Website sind, eingesetzt. Das eigentliche Sicherheitsproblem besteht für den Agenten und nicht

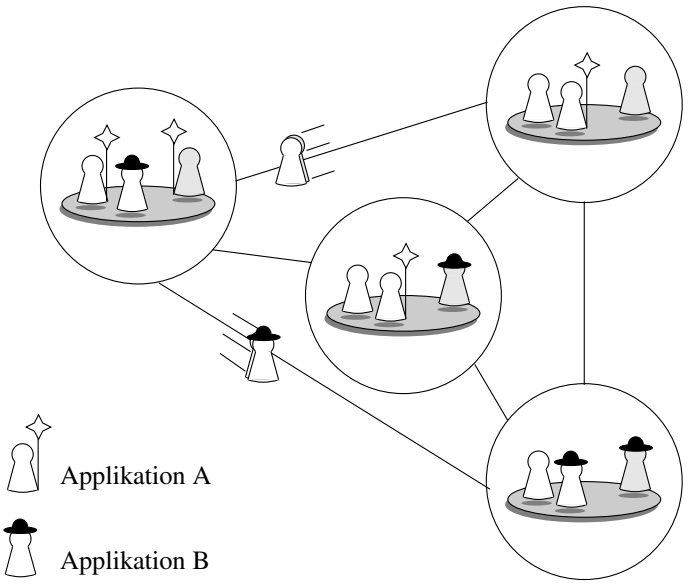


Abbildung 1.7: Verteilte Applikationen im Agenten-basierten P2P-System

durch den Agenten: Der Agent kann durch ein böswilliges System beraubt, modifiziert oder beendet werden.

1.3.2 Charakterisierung mobiler Agenten

Aus Sicht der Softwaretechnik betrachtet basiert die Technologie der mobilen Agenten auf dem Peer-to-Peer-Ansatz und erweitert die Fähigkeiten traditioneller verteilter Ressourcennutzung. Eine erste intuitive Definition könnte lauten:

> Mobile Agenten sind Softwarekomponenten, die zwischen Rechnern in einem heterogenen Netzwerk ausgetauscht werden.

Die Softwarekomponenten sind im einfachsten Fall mehr oder weniger komplexe Algorithmen mit einem Datenanteil. Solche Komponenten können Objekte oder eigene Applikationen sein. Die Bezeichnung mobiler Code oder mobile Komponenten sind daher gebräuchlich, zumindest aus Anwendersicht. Aus Sicht eines Entwicklers betrachtet sind mobiler Code und mobile Agenten nicht gleichzusetzen.

In der oben angegebenen Definition wird ausschließlich die Mobilität betrachtet. Die Intention von Agenten bleibt unberücksichtigt. Mobile Agenten können weitaus mehr als nur mobil sein. Die Charakterisierung mobiler Agenten muss erweitert werden: Agenten handeln im Auftrag und Interesse eines anderen. Agieren Softwareagenten selbständig, werden sie als autonom bezeichnet. Die Object Management Group (OMG) [OMG04],

ein Konsortium zur Standardisierung von Unternehmensanwendungen, beschreibt in der Mobile Agent Facility Specification [OMG00] einen Agenten als autonomes und im Auftrag handelndes Programm:

> An agent is a computer program that acts autonomously on behalf of a person or organization. Currently, most agents are programmed in an interpreted language (for example, Tcl and Java) for portability. Each agent has its own thread of execution so tasks can be performed on its own initiative.

Im Internet ist bei Wikipedia [Wik04] zum Begriff Softwareagent eine Definition mobiler Agenten zu finden, die auf einer Charakterisierung von Softwareagenten basiert:

> Als **Software-Agent** bezeichnet man ein selbständiges bzw. nahezu selbständiges Computerprogramm. Im Allgemeinen muss ein Agent folgende Eigenschaften erfüllen:
>
> - autonom - das Programm arbeitet weitgehend unabhängig
> - proaktiv - das Programm löst Aktionen aufgrund eigener Initiative aus
> - reaktiv - das Programm reagiert auf Änderung der Umgebung
> - sozial - das Programm kommuniziert mit anderen Agenten
>
> Kommt zu den genannten Eigenschaften die Fähigkeit hinzu den Ausführungsort zu wechseln und am neuen Ort die gleiche Aufgabe weiter zu bearbeiten, so spricht man von einem **mobilen Agenten**. Beim Wechsel des Ausführungsortes spricht man von Migration.

Softwareagenten werden meist durch ihre Eigenschaften beschrieben. Franklin und Graesser haben in [FG96] Definitionen zu (autonomen) Softwareagenten zusammengetragen, verglichen und wesentliche Eigenschaften aufgezeigt. Tabelle 1.2 auf der nächsten Seite erläutert diese Eigenschaften.

Einen etwas anderen Schwerpunkt setzt die Foundation of Intelligent Physical Agents (FIPA) [Fou04], eine Organisation zur Standardisierung der Interoperabilität von heterogenen Softwareagenten. Diese Organisation ist um einen Standard im Bereich der Agentenkommunikation bemüht – eine Kommunikation zwischen Agenten unterschiedlicher Systeme soll standardisiert werden. In der FIPA Agent Management Specification [Fou02] ist die Beschreibung eines Agenten dementsprechend auf die Fähigkeit zur Kommunikation orientiert:

> An agent is a computational process that implements the autonomous, communicating functionality of an application ...

Diese Definitionenvielfalt zeigt, dass sich eine allgemein anerkannte Definition des Agentenbegriffs bis heute nicht durchgesetzt hat. Vielleicht weil die (virtuelle) Welt der Agenten so mannigfaltig ist wie die reale Welt. Ein Versuch, die Vielzahl der Definitionen zusammenzufassen, ergibt folgende Beschreibung eines Softwareagenten:

Eigenschaft (Synonyme)	**Erklärung**
reaktiv (Wahrnehmen und Handeln)	Agenten reagieren zeitnah auf Änderungen in der Umwelt
autonom	Agenten üben Kontrolle über ihre eigenen Aktivitäten aus
ziel-orientiert (proaktiv, zielgerichtet)	Agenten reagieren nicht nur auf Umwelteinflüsse, sondern handeln selbständig
zeitlich kontinuierlich	Agenten sind kontinuierlich laufende Prozesse
kommunikativ (sozialfähig)	Agenten kommunizieren mit anderen Agenten oder mit Menschen
lernfähig (adaptiv)	Agenten ändern auf Grund von früheren Ereignissen ihr Verhalten
mobil	Agenten sind fähig, sich von einer Maschine zu einer anderen zu transferieren
Flexibilität	Aktionen sind nicht festgeschrieben
Charakter	glaubhafte „Persönlichkeit“ und emotionale Zustände

Tabelle 1.2: Eigenschaften von Agenten, angelehnt an Franklin und Graesser

> Ein Softwareagent ist im weitestgehenden Sinne eine Softwarekomponente, die eigenständig eine gestellte Aufgabe löst und eine Menge von charakterisierenden Eigenschaften besitzt.

Eine Softwarekomponente ist nur dann als Agent zu bezeichnen, wenn die Aufgabe, die sie verfolgt, im Interesse eines anderen durchgeführt wird. Eine extreme Behauptung wäre: Jede Software sei ein Agent, schließlich werde sie von einem Menschen gestartet oder wenigstens programmiert – eben „beauftragt“, eine Aufgabe zu übernehmen. Da (Standard-) Software im Allgemeinen die oben genannten Eigenschaften nicht besitzt, kann diese auch nicht als Agent bezeichnet werden.

Zur vollständigen Charakterisierung von Agenten gehört des Weiteren die Betrachtung des Lebenszyklus. Der Lebenszyklus eines Agenten unterscheidet sich von dem der (Standard-) Software im Wesentlichen in der Langlebigkeit. Insbesondere der mobile Agent ist auf Grund seiner Fähigkeit zur Migration in der Lage, die Laufzeiten eines Betriebssystems zu übertreffen. In Abb. 1.8 auf der nächsten Seite ist der Lebenszyklus von Agenten, wie er von der FIPA in [Fou02] definiert wurde, dargestellt. Die Besonderheit stellt der Zustand *Migrating* dar, der nur bei mobilen Agenten existiert.

Der grau hinterlegte namenlose Superzustand stellt die Lebenszeit des Agenten dar. Die einzelnen Subzustände *Initiated*, *Active*, *Migrating*, *Suspended* und *Waiting* kennzeichnen die Lebensabschnitte, den momentanen Zustand, eines Agenten. Die Aktivität eines Agenten beginnt mit dem Zustand *Active*. Die Zustandsübergänge können in freiwillige

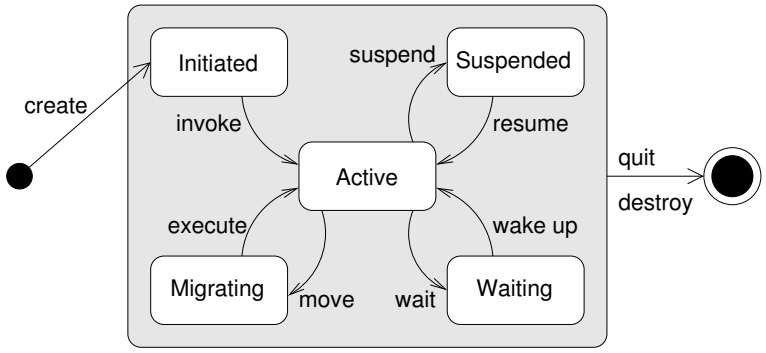


Abbildung 1.8: Lebenszyklus eines (mobilen) Agenten nach FIPA

und erzwungene Übergänge unterschieden werden. Der Agent kann sich beispielsweise freiwillig schlafen legen (Zustand *Waiting*) oder vom System in den Ruhezustand *Suspended* versetzt werden. Die Gründe dafür können verschieden sein, beispielsweise zur Verhinderung von Überlastsituationen.

Die erzwungenen Übergänge sind:

- Vom Systemstart bis zum Zustand *Active* (`create`, `invoke`),
- Die mit den Zustand *Suspended* verknüpften Übergänge (`suspend`, `resume`),
- Der Übergang zum Finalzustand durch den Befehl (`destroy`).

Initiiert durch den Agenten, aber dennoch systemgesteuert sind:

- Der Übergang vom Zustand *Migrating* (`execute`),
- Der Übergang vom Zustand *Waiting* zum Zustand *Active* (`wake up`).

Freiwillige Zustandsübergänge liegen bei den Befehlen `move`, `wait` und `quit` vor.

Zur Übertragung eines mobilen Agenten ist der generelle Aufbau des Agenten von Bedeutung. Ein Agent besteht aus drei Teilen (siehe Abb. 1.9):

- Den Daten,
- Dem aktuellen Ausführungszustand,
- Dem statischen Code, in Java beispielsweise der Bytecode (die Class- oder Jar-Dateien)

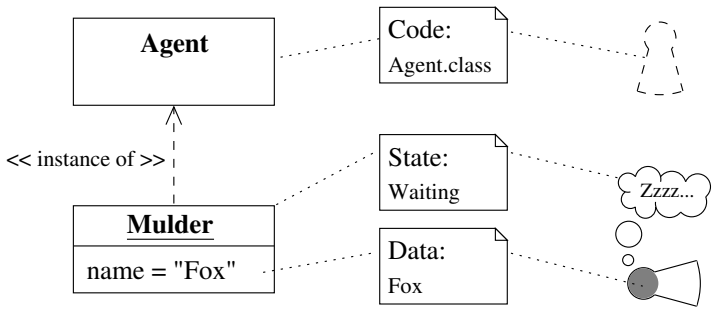


Abbildung 1.9: Genereller Aufbau von Agenten

Die Daten und der Ausführungszustand verkörpern die Agenteninstanz, den konkreten Agenten, und müssen unbedingt bei einer Migration übertragen werden. Der Serialisationsmechanismus von Java erlaubt das Speichern von Datenteil und Zustand des Agenten in einem Bytestream (Folge von Bytes). Dieser Bytestream, auch als *serialisierter Agent* bezeichnet, wird bei einer Migration dem Migrationsziel übergeben. Der statische *Bytecode* des Agenten wird nur dann transferiert, wenn dieser nicht bereits auf dem Zielsystem, z. B. in der Klassenbibliothek, vorhanden ist.

Als Resümee des Kapitels ergeben sich folgende Definitionen:

> Ein *stationärer Agent* ist eine *Softwarekomponente*, die im Interesse eines anderen handelt (*handelnder Stellvertreter*). Die Lebenszeit eines stationären Agenten ist durch die des lokalen Systems beschränkt.

Die Definition des mobilen Agenten ist eine Erweiterung der Definition des stationären Agenten:

> Ein *mobiler Agent* ist eine *Softwarekomponente*, die im Interesse eines anderen handelt (*handelnder Stellvertreter*), sich selbständig über Systemgrenzen hinweg bewegt (*autonome Mobilität*) und dessen Lebenszyklus von einer *Langlebigkeit* geprägt ist, die über jene eines einzelnen Teilsystems hinausgeht.

Generell können stationäre Agenten mit mehr Rechten, wie z. B. Zugriff auf lokale Systemdaten, ausgestattet werden, da sie ähnlich *vertrauenswürdig* sind wie lokale Applikationen.

1.3.3 Architektur der Systeme

Mobile Agenten agieren und bewegen sich in einem Peer-to-Peer-Netzwerk. Damit ein Agent in einem derartigen Netzwerk „lebensfähig“ ist, muss eine Ausführungsumgebung

auf jedem Zielsystem installiert und aktiv sein. Das installierte System, die Software selbst (der Binärcode), die alle zur Ausführung von Agenten notwendigen Teile beinhaltet, wird als *Agentensystem* bezeichnet. Agentensysteme mit mobilen Agenten werden auch als *mobile Agentensysteme* (MAS) bezeichnet.

Wird die Software auf einem Rechner gestartet, stellt diese die Ausführungsumgebung für den Agenten zur Verfügung. Für einen Agenten sind damit grundlegenden Funktionalitäten nutzbar. Die aktive Software wird *Agentenserver* (Agent Server), Agentenplattform oder Agency genannt – eine Instanz des Agentensystems.

Die Bezeichnung Agentenserver ist ungünstig gewählt, da als Server oft auch eine Hardware bezeichnet wird. In dieser Arbeit soll der Begriff Agentenserver oder Agentenplattform die Ausführungsumgebung bezeichnen. Die Hardware wird als Agentenrechner oder vereinfacht *Rechner* bezeichnet.

Innerhalb des Agentenservers befinden sich die Agenten auf so genannten *Plätzen* (place), die zur direkten, lokalen Kommunikation zwischen Agenten genutzt werden. Pro Agentenserver können mehrere Plätze vorhanden sein. Dieses Konzept dient zur Trennung von Kommunikationskanälen.

Das Netzwerk, das durch die Agentenserver aufgespannt wird, wird als *Agentennetzwerk* (Agenten P2P-Netzwerk) bezeichnet und stellt die Gesamtheit der laufenden Instanzen des Agentensystems dar. Die mobilen Agenten bewegen sich innerhalb dieses Netzwerks.

In der Mobile Agent Facility Specification [OMG00] der OMG sind ebenfalls die Komponenten eines Agentensystems definiert. Abbildung 1.10 auf der nächsten Seite stellt diese Komponenten graphisch dar.

In dieser Darstellung ist die Kommunikationsinfrastruktur nicht Bestandteil des Agentensystems, sondern hat den Zweck, interne sowie externe Kommunikation zu unterstützen. Allerdings muss eine entsprechende Schnittstelle innerhalb des Agentensystems existieren.

Die Verbindung zwischen Agentenservern erfolgt über die Kommunikationsinfrastruktur durch das unterliegende Netzwerk mit Hilfe von entsprechenden Betriebssystemfunktionalitäten. Diese Verbindung wird in der Regel zum Austausch von Nachrichten und Agenten benutzt.

Im selben Standard der OMG ist des Weiteren der Begriff der Region definiert: Eine Menge von Agentenservern, welche zu ein- und derselben Verantwortlichkeit gehören, werden als Region bezeichnet. Typischerweise sind dies Agentenserver, die innerhalb eines Administrationsbereichs laufen. Bei den Agentenservern muss es sich nicht um denselben Typ handeln. Sie sollten jedoch untereinander kompatibel sein. Regionen sind unter anderem für Lastverteilung und eine bessere Skalierung bedeutend.

Ein Agentensystem ist als solches für einen Endbenutzer keine eigenständige Anwen-

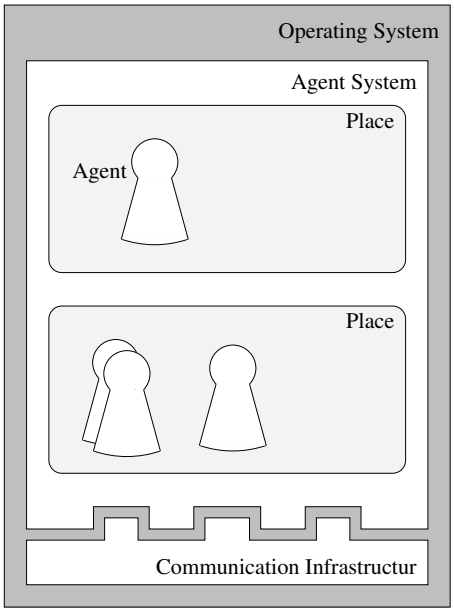


Abbildung 1.10: Komponenten eines Agentensystems nach OMG

dung, sondern dient als so genannte Middleware. Das Agentensystem nutzt und erweitert die Funktionalitäten des Betriebssystems, ohne dessen Bestandteil zu sein. Es ist eine Anwendung, die von anderen Anwendungen genutzt werden kann – sie steht zwischen Betriebssystem und (Endnutzer-) Applikation (siehe auch Abb. 2.1 auf Seite 32 in Kapitel 2.1) und stellt eine Kommunikationsplattform mit erweiterter Funktionalität dar.

1.3.4 Stand der Technik und Tendenzen

Zu einer Charakterisierung des State-of-the-Art werden an dieser Stelle Agentensysteme verschiedener Stufen definiert, welche die Fähigkeiten der Middleware, der mobilen Agentensysteme, verdeutlichen. Die erste Stufe entspricht im Wesentlichen dem Stand der Technik auf dem Gebiet der mobilen Agenten Anfang 2000, dem Beginn dieser Arbeit:

- Ein spezialisierter Agent muss zur Lösung einer Aufgabe programmiert werden

  Der mobile Agent bekommt seine Persönlichkeit aufgeprägt. Inwieweit er dabei „intelligent“ gemacht wird, hängt von der Aufgabe und den programmtechnischen Möglichkeiten ab. Hier gibt es viele Lösungen und Alternativen. Eine Verallgemeinerung ist damit sehr schwierig, wenn nicht gar unmöglich. Im einfachsten Fall soll eine Anfrage an eine Datenbank gemacht werden. In einer komplexeren Aufgabe muss der Agent mit anderen Partnern z. B. über den Kauf von Aktien an der Börse verhandelt.

- Dem Agenten muss die Reiseroute übergeben werden

  Die Reiseroute beinhaltet eine Menge von Agentenplattformen, die vom mobilen Agenten besucht werden sollen und auf denen sein Applikations-Algorithmus angewendet wird. Die Route wird durch den Programmierer vor Reiseantritt festgeschrieben, in den mobilen Agenten hineinprogrammiert. Dies setzt natürlich voraus, dass der Programmierer das Informationsangebot im gesamten Netz kennt und optimal nutzt, da sonst der beste Applikations-Algorithmus ins Leere laufen würde. Die Dynamik des Gesamtsystems wird hierbei nicht berücksichtigt.

Eine Verallgemeinerung des Applikations-Algorithmus zu einem generellen Algorithmus, der eine Vielzahl der Aufgaben eines Agenten löst, ist mit heutigen Technologien nicht möglich. Im Gegensatz dazu ist die Suche nach einer Reiseroute ein allgemeines Problem, das durch ein Agentensystem gelöst werden kann. Dies zeigt den Weg zu einem Agentensystem der zweiten Stufe:

- Der Reiseplan wird vom Agenten zur Laufzeit erstellt – *autonome Migration*

Das Ziel dieser Verbesserung ist, die Aufgabe des Programmierers auf die Erstellung des Applikations-Algorithmus zu reduzieren. Der mobile Agent bestimmt mit Hilfe des Agentensystems die interessanten Anlaufpunkte im Netz und sucht selbständig den günstigsten Weg dorthin. Dies erlaubt dem Programmierer von der Durchführung der Aufgabe zu abstrahieren – der Reise durch das Netzwerk, der Suche nach passenden Plattformen und Diensten, der Optimierung des Ablaufs etc. Eine detaillierte Kenntnis des Informationsangebots im Netz ist für ihn nicht mehr notwendig. Der mobile Agent wird damit zum Dienstleister, der eine Spezifikation der Aufgabe selbständig umsetzt (das *WAS*), anstatt im Detail programmiert werden zu müssen (das *WIE*). Das WAS und das WIE sind entkoppelt. Dies wird in dieser Arbeit als Agentensystem der zweiten Stufe bezeichnet. Die Programmierung der Persönlichkeit (Applikations-Algorithmus) des mobilen Agenten kann dabei soweit reduziert werden, dass diese nur eine Auswahl aus fertigen Schablonen darstellt oder adaptiv aus Erfahrungen lernt (ein erster Schritt zum Agentensystem der dritten Stufe).

Der eigentliche Vorteil liegt in der *autonomen Routenplanung*. Veränderungen in der Netzwerktopologie, in der Qualität oder den Kosten der Verbindungen, in den verfügbaren Ressourcen etc. werden automatisch berücksichtigt. Die Quantität des Abgleichs im Informationsraum wird maschinengestützt, effizient und flexibel bearbeitet, unabhängig vom Kenntnisstand des jeweiligen Programmierers.

Der Stand der Technik im Bereich der mobilen Agentensysteme ist durch Agentensysteme der ersten Stufe gekennzeichnet. Der ursprüngliche Ansatz der mobilen Agenten propagiert keinerlei Bindung oder Organisation der einzelnen Instanzen des Agentensystems oder der Agenten. In den meisten Implementierungen ist ein Agentenserver daher vollständig eigenständig.

Die Abhängigkeit der Dienste von Orten, die fixe Platzierung auf speziellen Servern, wird durch die Mobilität der dienstanbietenden Agenten aufgehoben. Die bisherige statische Struktur klassischer, verteilter Systeme und die relativ enge Kopplung der Komponenten zerfällt mit dem Agentenparadigma. Damit steht die Forschung vor einer neuen Herausforderung: Die Organisation von Systemen mit dynamischen Komponenten und einer variablen, eigenständigen Verteilung muss bewältigt werden. Das Konzept eines Agentensystems der zweiten Stufe kann dieser Herausforderung gerecht werden. Bisher sind in den Agentensystemen bestenfalls Ansätze in diese Richtung zu finden.

1.4 Einordnung der Arbeit in das Gebiet der mobilen Agenten

Diese Forschungsarbeit fokussiert einen Bereich der Informatik, speziell der Softwaretechnik, der sich mit verteilten Systemen [TvS03] befasst. Ein Paradigma zur Unterstützung solcher Systeme, insbesondere der Interaktion zwischen den verteilten Komponenten, sind mobile Agenten, die neuartige Möglichkeiten der Kommunikation erlauben und traditionelle Techniken der Client-Server-Kommunikation ergänzen und erweitern.

Agentensysteme mit mobilen Agenten (mobile Agentensysteme) stehen im Mittelpunkt dieser Arbeit. In diesem Bereich haben sich verschiedene Forschungsrichtungen etabliert. Mitte der 90er konzentrierten sich einige Arbeiten auf geeignete Programmiersprachen für mobile Agenten [CGPV97, Kna95] und Sprachen zur Kommunikation zwischen Agenten [BHR+97]. Weiterhin wurden und werden Sicherheitsaspekte in Agentensystemen untersucht [Vig98, Fra04], Kontrollprobleme bei Agenten (inkl. agent tracking) beleuchtet [Bau97, BR98, RP01] und das Design mobiler Agentensysteme diskutiert [HA98]. Eine weitere Forschungsrichtung ist die Kooperation von Agenten [KOOL03].

Eine andere Richtung der Forschung bilden die intelligenten Agenten, deren Ursprünge aus der KI stammen. Trotz der Autonomie der mobilen Agenten sollen intelligente Agenten in dieser Arbeit nicht betrachtet werden. Die Autonomie des mobilen Agenten bezieht sich auf die autonome Migration, während intelligente Agenten ihre Aufgaben, Abläufe autonom durchführen. Zur besseren Abgrenzung des Autonomie-Begriffs wird die Art der Autonomie als „autonomes Handeln (KI)“ bezeichnet.

Einen für diese Arbeit funktional grundlegenden Bereich bildet die Migration der Agenten in mobilen Agentensystemen. Die intensive Auseinandersetzung mit dem Themengebiet der mobilen Agenten an der Friedrich-Schiller-Universität Jena [Leh04b, Leh04a] zeigt, dass die Dynamik in Peer-to-Peer-Systemen, speziell in mobilen Agentensystemen, eine hohe Autonomie bezüglich der Migrationsentscheidungen der Agenten verlangt. Doch gerade dort liegen die Schwächen von Systemen der ersten Stufe: Agenten haben auf Grund unzureichender Unterstützung bei der Autonomie ein starres Migrationsverhalten.

Diese Dissertation beschäftigt sich mit der Weiterentwicklung der bestehenden Technologie, vor allem mit der Verbesserung des autonomen und proaktiven Verhaltens der Agenten bei der Migration (mobile Agentensysteme zweiter Stufe).

Die Arbeit konzentriert sich somit auf einen Ausschnitt der virtuellen Welt, einem Teil der virtuellen Agenten (wie in Kapitel 1.1.2 klassifiziert). Neben den mobilen Agenten sind stationäre Agenten als Helfer der mobilen Agenten von Bedeutung. Den Schwerpunkt bildet die Routenplanung und Migrationsoptimierung der Agenten. Die Wahrnehmung der Agenten muss erweitert werden. Die Interaktion mit dem Agentensystem bedarf einer Verbesserung (siehe Kapitel 1.2).

Zur Umsetzung der Ideen entstand der Routing Service *ProNav* als Rahmenwerk. Voraussetzung für den Service ist ein mobiles Agentensystem, das im Wesentlichen die Migration von Agenten unterstützen und grundlegende Infrastruktur-Funktionalitäten besitzen muss.

Kapitel 2

Das mobile Agentensystem TRACY

Am Lehrstuhl für Softwaretechnik der Friedrich-Schiller-Universität Jena wurde in den letzten Jahren ein lauffähiges mobiles Agentensystem entwickelt. Das Agentensystem mit dem Namen TRACY wird als Grundlage für diese Arbeit herangezogen. TRACY bietet wichtige Funktionen, die als Basis für die Arbeit genutzt werden.

Zunächst erfolgt eine Einführung in die Eigenschaften des Systems. Die entscheidende Komponente, der TRACY Domain Service, die für diese Arbeit von wesentlicher Bedeutung ist, wird erklärt und die Wahl des Agentensystems über einen Vergleich mit anderen Agentensystemen begründet.

2.1 Überblick über TRACY

Am Lehrstuhl für Softwaretechnik der Friedrich-Schiller-Universität Jena ist seit 1998 die Thematik der mobilen Agenten ein Forschungsschwerpunkt. Unter der Leitung von Prof. Dr. Rossak griff Dr. Braun dieses Thema auf. Seinen Schwerpunkt setzte er auf die Optimierung der Migration mobiler Agenten und begann mit der Entwicklung einer Migrationskomponente [Bra03] als fundamentalem Teil eines mobilen Agentensystems. Das war der erste Schritt zur Entwicklung des Agentensystems TRACY. Durch [Erf99] wuchs TRACY zu einem vollständigen Agentensystem heran und bildete die Basis für weitere Forschungsarbeiten, nicht nur an der Jenaer Universität.

Das mobile Agentensystem (MAS) TRACY ist eine Middleware-Komponente zwischen Betriebssystem und eigentlicher Applikation. Um eine möglichst hohe Portabilität zu erreichen, wurde TRACY in *Java2* [Sun04b] implementiert. Die Eigenschaften dieser Programmiersprache [SM94] und die gute Unterstützung für netzbasierte Anwendungen sind hervorragende Voraussetzungen für die Umsetzung der „mobilen Agenten Idee“. Durch das Konzept der Java virtuellen Maschine (Java VM) ist eine Anwendung auf unterschiedlichsten Betriebssystemen [Sun04c], wie Windows, Linux und Solaris, lauffähig. Notwendig ist jeweils eine Java VM auf dem entsprechenden Zielsystem, die Teil des Java Runtime Environments (JRE) ist. Für die Ausführung von TRACY wird mindestens die JRE-Version 1.4 benötigt.

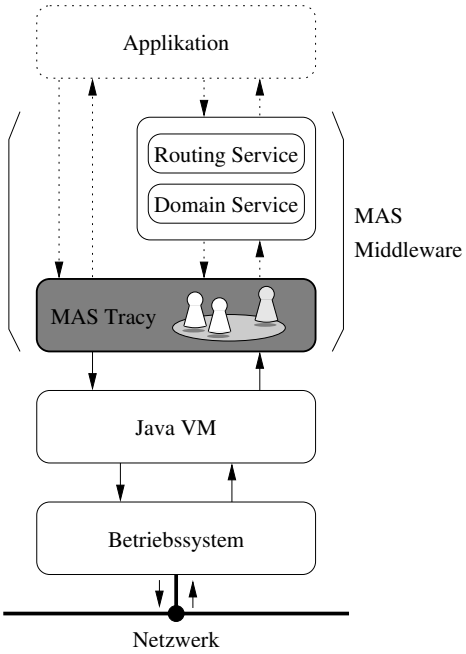


Abbildung 2.1: Schichtenbildung: MAS Middleware, Java VM und Betriebssystem

Die Abbildung 2.1 auf der vorherigen Seite stellt die Komponenten der MAS Middleware-Schicht dar: Neben dem MAS TRACY stehen die Komponenten Domain Service und Routing Service zur Verfügung. Diese beiden Komponenten sind Infrastruktur-Dienste, die typische Querschnittsaufgaben für Applikationen mit mobilen Agenten übernehmen. Dazu zählt beispielsweise das Auffinden von Diensten im Agentennetzwerk. Die Aufgaben dieser Komponenten liegen weder im Bereich eines Agentensystems noch einer Applikation. Die Komponenten sind unabhängig vom speziellen Agentensystem, hier TRACY, und gehören nicht zur Applikationsschicht. Die Querschnittsaufgaben, die von den Komponenten des MAS genutzt werden, müssen in diesem Fall von jeder Applikation neu implementiert werden. Domain Service und Routing Service stellen Dienste zur Verfügung, die von diversen Applikationen genutzt werden können und sind damit funktionelle Bestandteile der MAS Middleware-Schicht.

Der im Bild 2.1 auf der vorherigen Seite dargestellte Schichtenansatz bringt auch Probleme mit sich. Der Zugriff zu Systemfunktionalitäten des Betriebssystems (Operating System, OS) wird z. B. durch die Java VM gekapselt, eine wichtige Eigenschaft für die Portabilität. Allerdings werden dadurch Möglichkeiten, wie der Zugriff auf tiefere Schichten im OS, eingeschränkt. Die sich daraus ergebenden Problematiken werden in Kap. 5.1 näher beschrieben.

TRACY-Agenten TRACY-Agenten sind „gewöhnliche“ *Java Objekte*. Die Implementierung der Algorithmen erfolgt durch Spezialisierung von Klassen, die im Agentensystem vordefiniert sind. In TRACY werden dabei drei Arten von Agentenklassen unterschieden (siehe Abb. 2.2 auf der nächsten Seite), die grundlegende Funktionalitäten und Schnittstellen der jeweiligen Agenten definieren.

Die allgemeine Klasse *Agent* beinhaltet die gemeinsamen Funktionalitäten der Subklassen und definiert den Zugriff auf Systemfunktionalitäten (Kommunikation etc.). Die Agenten erhalten in der entsprechenden Subklasse spezielle Schnittstellen, die den Zugriff auf Systemfunktionalitäten regeln. Der *mobile Agent* besitzt als einzige Agentenklasse die Funktionalitäten zur Migration, ansprechbar durch die Methode `go()`. Dem stationären *Systemagenten* wird mehr Vertrauen geschenkt, weshalb er besondere Rechte beim Zugriff auf Systemressourcen genießt. Der ebenfalls stationäre *Gatewayagent* unterstützt die Anbindung von Fremdsystemen.

Interaktion durch Kommunikation Eine wesentliche Eigenschaft von Agenten ist die Kommunikation untereinander sowie zum Menschen. Die Kommunikation zwischen Agenten unterscheidet sich grundlegend von der Kommunikation objektorientierter Systeme. Zwischen Objekten wird die Kommunikation über direkte Methodenaufrufe geführt. Eine Referenz auf das Empfängerobjekt ist nötig. Bei der Kommunikation zwischen Agenten ist die Kopplung schwächer: Ein Agent hat keine Referenz auf einen anderen

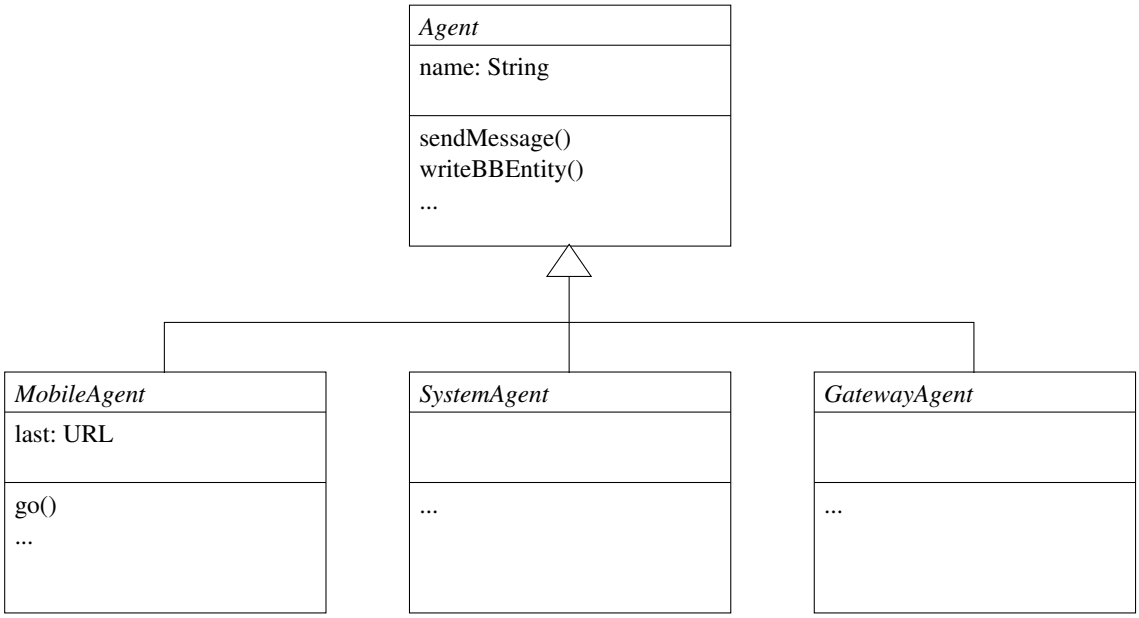


Abbildung 2.2: UML-Darstellung der Agenten in TRACY

Agenten. Eine direkte Kommunikation zwischen Agenten ist *über Referenzen* in TRACY nicht möglich und auch nicht gewünscht. Anderenfalls wird der Agent durch eine offene Referenz beim Verlassen einer Plattform aufgehalten und in seiner Autonomie eingeschränkt. Auch Sicherheitsaspekte sprechen für diese Vorgehensweise.

In TRACY gibt es für Agenten zwei Möglichkeiten zur Kommunikation:

- Direkte Kommunikation von Agent zu Agent über *asynchronen Nachrichtenaustausch*,
- Indirekte Kommunikation über ein so genanntes *Blackboard*.

Die direkte Kommunikation ist nur zwischen Agenten möglich, die auf derselben Plattform residieren. Dieser Ansatz ist mit einer Mail vergleichbar, die Agenten austauschen können. Jeder Agent besitzt ein Postfach im *Inter-Agent Message Handler* (Abb. 2.3 auf der nächsten Seite), in dem andere Agenten Nachrichten hinterlegen können, die ausschließlich für den Eigentümer des Postfaches bestimmt sind. Dazu muss der Sender den Namen des Empfängers kennen. Die Zustellung übernimmt das Agentensystem. Der Eigentümer wird über Eingänge im Postfach informiert, muss sich allerdings selbst um das Abholen der Post kümmern und genießt damit einen hohen Grad an Autonomie. Möchte ein Agent keine Nachrichten empfangen, so kann er das Postfach zeitweise oder ganz schließen, wodurch Nachrichten abgelehnt werden. Der Sender wird darüber informiert.

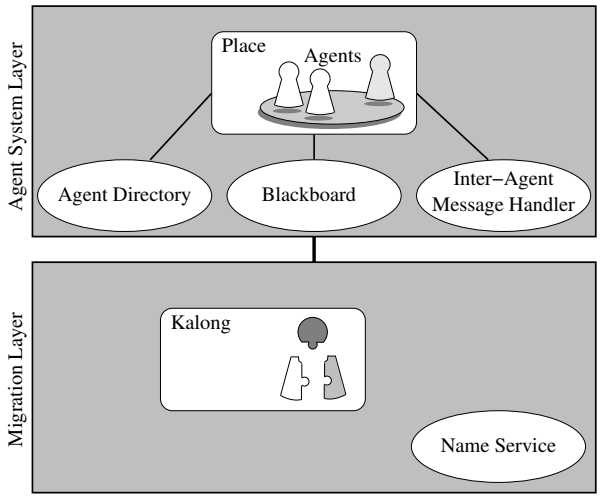


Abbildung 2.3: Die grundlegenden Komponenten von TRACY

Ein Agent besitzt in TRACY nur an seinem momentanen Aufenthaltsort ein Postfach. Dies schränkt die Kommunikation ein. TRACY erlaubt nur lokale Kommunikation. Damit wird das Agentenparadigma streng verfolgt: Möchten zwei Agenten miteinander kommunizieren, so müssen sich diese auf einer gemeinsamen Plattform befinden.

Die zweite Art der Kommunikation im Agentensystem TRACY stellt das Hinterlegen von Nachrichten auf einem *Blackboard*, einer Art Tafel, dar. Agenten können auf diese Tafel in öffentlich lesbare, eingeschränkt lesbare oder private Bereiche schreiben. Entsprechend haben sie die Möglichkeit, die hinterlegten Informationen auszulesen oder sogar zu überwachen. Wird ein Eintrag auf dem Blackboard überwacht, werden die überwachenden Agenten bei einer Änderung informiert. Die geschriebenen Informationen eines mobilen Agenten sind beständig, d. h. sie gehen nicht verloren, wenn der Agent die Plattform verlässt. In praktischen Anwendungen ist das Blackboard ein wichtiges Mittel, um temporäre Informationen, wie Zustände des Systems, zwischenzuspeichern.

Die Kommunikation zwischen Agent und Mensch erfolgt in TRACY über Nachrichten und Textausgaben am Monitor oder mit Hilfe externer Software-Werkzeuge. Anmerkung: Ein Endbenutzer hat keinen direkten Zugriff auf das Agentensystem, da dies eine Middleware darstellt, sondern interagiert direkt mit Agenten oder indirekt über eine Anwendung, die TRACY als Middleware benutzt.

Migration Allgemein wird die Migration von Agenten in starke und schwache Migration (oder auch Mobilität) unterschieden. Die Differenzierung bezieht sich auf die Art und Weise, wie und wann ein Agent seine Ausführung auf einer Startplattform beendet und auf der Zielplattform fortsetzt. Generell entscheidet der Agent durch den Aufruf eines

Migrationsbefehls an beliebiger Stelle über eine Migration – die Ausführung wird beendet und der Agent migriert zu einem von ihm gewählten Ziel. Bei der starken Migration wird die Ausführung des Agenten auf der Zielplattform mit der nächsten, dem Migrationsbefehl folgenden, Anweisung fortgesetzt. Ob diese Realisierung der starken Migration möglich ist, hängt von der verwendeten Programmiersprache ab. In Java bzw. in einer objektorientierten Umgebung ist dieser Mechanismus nicht ohne weiteres möglich.

Die schwache Migration erlaubt ein Fortsetzen der Ausführung auf der Zielplattform nur an fixen Punkten, einer festgelegten Methode des Agenten, die von der Zielplattform nach Ankunft des Agenten aufgerufen wird. TRACY unterstützt eine stärkere Form der schwachen Migration: Die Methode, die nach der Ankunft auf einer Zielplattform aufgerufen werden soll, kann durch den Agenten spezifiziert werden. Dies wird als *schwache Migration mit beliebigem Methodeneinsprung* bezeichnet.

TRACY unterstützt zwei generelle Arten der Übertragung eines Agenten: Die *Pull-Strategie* und die *Push-Strategie*. Die Übertragungsarten unterscheiden sich im Zeitpunkt, zu dem der statische Bytecode eines Agenten übertragen wird. Die Pull-Strategie fordert bei Bedarf den Bytecode nach Ankunft des serialisierten Agenten auf der Zielplattform an (dynamisches Nachladen). Die andere Variante, den Bytecode zusammen mit den serialisierten Agenten zu übertragen, wird als Push-Strategie bezeichnet. Diese Strategien können weiter verfeinert werden (siehe Kapitel 4.4).

Eine Übertragung des Agenten kann in TRACY zusätzlich über verschiedene Protokolle erfolgen. Ein Agent kann beispielsweise über UDP, TCP oder TCP in Kombination mit SSL transferiert werden. Die generische Migrationskomponente *Kalong* [Bra03] (siehe Abb. 2.3) erlaubt die Integration weiterer IP-basierter Protokolle.

Sicherheit In TRACY sind grundlegende, *Java-eigene Schutzmechanismen* integriert, wie der Sandbox-Mechanismus [Dag04] zur Verhinderung des Zugriffs auf lokale Daten und des Öffnens unsicherer Netzwerkverbindungen auf einer Plattform. Erweiterte Sicherheitsmechanismen gegen böswillige Agenten oder böswillige Agentenplattformen sind in TRACY nicht integriert. TRACY-2 hingegen bietet erweiterte Möglichkeiten.

TRACY Name Service Der Name Service in TRACY (siehe Abb. 2.3) ist bei der Kontaktaufnahme zum Austausch von Agenten zwischen Plattformen notwendig. Anders als im MASIF-Standard (Mobile Agent System Interoperability Facility) [OMG00] der OMG beschrieben (siehe auch Kapitel 1.3.3), besitzt TRACY nur einen Platz pro Agentenserver (in TRACY-2 sind mehrere Plätze möglich). Allerdings können auf einem Rechner verschiedene Instanzen des Agentensystems gestartet und somit mehrere Plätze realisiert werden. In TRACY ist ein Namensdienst integriert, der Agentenserver auf einem Rechner verwaltet. Die Namen für einen Agentenserver werden aus einer Kombination von Rech-

nername und lokalem Agentenservername gebildet. Der vollständige Name orientiert sich syntaktisch an einer URL-Form, beispielsweise:

tracy://rechnername.uni-jena.de/scully
"tracy://" + full qualified host name + "/" + local agent server name

Die Agentennamen sind neben einem freien Namensteil mit dem Namen der Heimatplattform, der Plattform auf der sie das erste Mal gestartet wurden, gekoppelt:

dana@rechnername.uni-jena.de/scully
name + "@" + full agent server name

Zusammenfassende Kurzübersicht In der Tabelle 2.1 werden die wesentlichen Eigenschaften des mobilen Agentensystems TRACY dargestellt. Basis bei der Umsetzung dieser Arbeiten unter Verwendung der TRACY-Agenten bildeten neben der Programmiersprache Java2 die Interaktionsmechanismen und die von TRACY gebotenen Migrationsmöglichkeiten.

Eigenschaft	**Umsetzung**
Implementierung	in Java2
Agenten	Java Objekte, basierend auf vordefinierten TRACY-Klassen
Interaktion	direkte Kommunikation über asynchr. Nachrichtenaustausch indirekte Kommunikation über Blackboard entfernte Komm. über Migration eines mobilen Agenten
Migration	schwache Migration mit beliebigem Einsprung Übertragungsstrategien: Push / Pull und Varianten
Sicherheit	über Java-eigene Schutzmechanismen

Tabelle 2.1: Kurzübersicht zu TRACY

Im Laufe der letzten Jahre entwickelte sich TRACY vom Forschungsprojekt der Friedrich-Schiller-Universität Jena immer weiter in Richtung Produkt. Im Rahmen der Exist-seed Förderung [Bun04] wurde the agent factory AG [the04] ausgegründet, um die Technologie zur Marktreife zu bringen. TRACY ist eines der wenigen Agentensysteme, die es zur Produktreife gebracht haben und einen Infrastruktur-Dienst, wie den TRACY Domain Service, anbieten.

Für ausführlichere Darstellungen zur Technologie von TRACY sei auf technische Berichte [BER00, BER01a] verwiesen sowie auf die Dissertation von Braun [Bra03], die insbesondere die Migrationskomponente von TRACY beschreibt. Diese Arbeit stützt sich auf

eine erste Implementierung von TRACY, die derzeit durch eine neue Version, TRACY-2 [BR04], abgelöst wird.

2.2 Der Infrastruktur-Dienst TRACY Domain Service

Ein Agentenserver ist im Agentennetzwerk eine eigenständige, unabhängige Instanz des Agentensystems. Durch das von den Agentenservern aufgebaute logische Netzwerk wird eine vollständig dezentral organisierte Infrastruktur gebildet, in der sich die mobilen Agenten bewegen können. Eine solche Infrastruktur weist im Allgemeinen eine hohe Dynamik auf: Neue Server werden gestartet, Verbindungen getrennt, Server gestoppt etc. Ein einzelner Agentenserver hat, wie der Agent, auf Grund des dezentralen Ansatzes keinerlei Information über das logische Netz. Dies führt zu Problemen bei der Programmierung und der Aktivierung eines Agenten. Bisher wird die Route des Agenten statisch festgelegt. Weder der Programmierer / Benutzer noch der Agent können das Verhalten des Agentennetzwerks voraussehen und die zur Laufzeit des Agenten aktiven Server kennen. Die Kopplung der einzelnen Agentenserver muss daher zwangsläufig erhöht werden. Zentrale Organisationsstrukturen müssen zumindest teilweise etabliert werden. Eine einfache, aber dennoch flexible und effektive Methode, ein solches Agentennetzwerk zu organisieren, wurde mit dem TRACY *Domain Service* [BER01a] entwickelt.

Der TRACY Domain Service ist ein Infrastruktur-Dienst, der auf dem Agentensystem TRACY aufsetzt (siehe Abb. 2.5 auf Seite 41). Durch sein Agenten-basiertes Konzept ist der Dienst auf andere Agentensysteme portierbar. Mit Hilfe von statischen und mobilen Agenten wird die Gesamtheit der Agentenserver in so genannte Domains strukturiert. Als *Domain* wird eine Menge zusammengehöriger Agentenserver bezeichnet, die meist physisch benachbart sind. In der Mobile Agent Facility Specification der OMG [OMG00] wurde der Begriff der Region definiert, der Agentenserver zusammenfasst, die unter derselben Verantwortlichkeit stehen. Die Konzepte der Region sind im Wesentlichen mit denen der Domain identisch. Unterschiede sind in der Organisation der Region und der Domain zu finden.

Komponenten Die Agentenserver einer bestimmten Domain werden als *Domain Nodes* oder *Domain-Knoten* bezeichnet. Jeder Domain-Knoten gehört genau einer / seiner Domain an. Ein Domain-Knoten jeder Domain übernimmt die Verwaltung seiner Domain und wird zum *Domain-Manager*. Jeder hinzukommende Knoten meldet sich bei dem Domain-Manager seiner Domain an. Beim Beenden muss er sich wieder abmelden. Der Domain-Manager besitzt dadurch zu jeder Zeit eine aktuelle Liste aller Domain-Knoten seiner Domain. Jeder lokale Domain-Knoten kann diese Liste von seinem Manager anfordern. Die Agentenserver sind nicht mehr vollständig entkoppelt und dennoch

eigenständige Komponenten in einem Netzwerk aus Agentenservern.

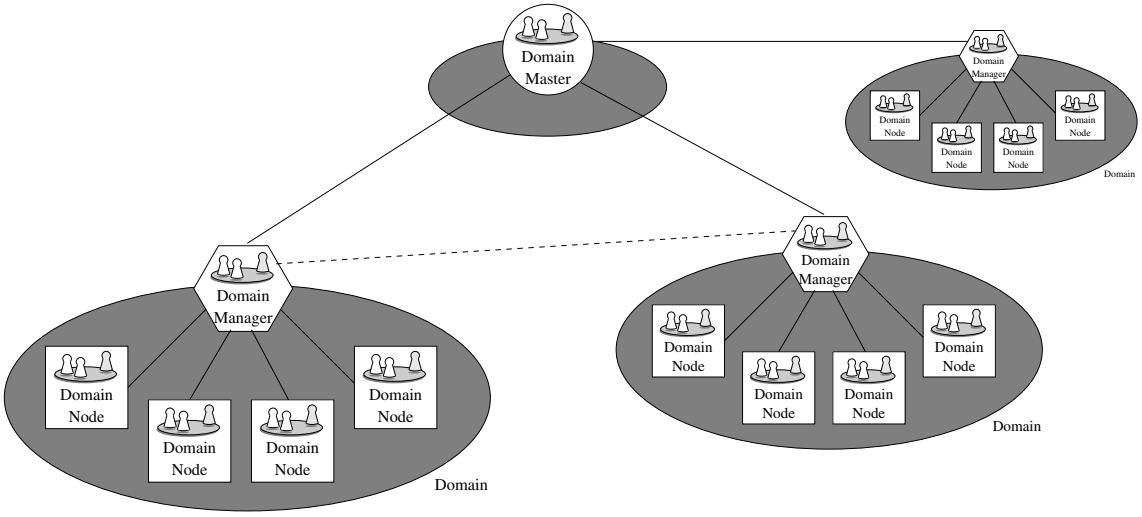


Abbildung 2.4: Komponenten im Domain Ansatz

Abbildung 2.4 zeigt eine schematische Darstellung der Komponenten des Domain Konzepts. Die durchgezogenen Linien stellen eine Managementabhängigkeit dar. Die gestrichelte Linie deutet auf eine Bekanntschaft zwischen Domains hin. Die Rollen werden dem dynamischen Verhalten der einzelnen Komponenten entsprechend verteilt.

Durch den beschriebenen Mechanismus entsteht eine Vielzahl von lokalen Domains, d. h. die Agentenserver werden Cluster-förmig strukturiert. Die Agentenserver einer Domain sind auf logischer Ebene miteinander verbunden. Mit einer weiteren Hierarchieebene werden Domains untereinander logisch verknüpft. Ein so genannter *Domain Master* verwaltet alle Domains, die durch ihren Domain-Manager repräsentiert werden. Die Domain-Manager müssen sich daher beim Master an- und abmelden. Der Master baut sich auf Grund der angemeldeten Manager eine Liste von Domain-Managern bzw. Domains auf. Domains können sich beim Master nach weiteren Domains erkundigen, wodurch die logische Verknüpfung zwischen den Domains entsteht. Ziel ist, nicht *alle* Domains untereinander bekannt zu machen (die Anzahl könnte sehr hoch sein), sondern nur jene, die in einem gewissen Umkreis liegen, also eine bestimmte Entfernung zueinander nicht überschreiten. Der Entfernungsbegriff in Netzwerken ist schwer fassbar und kann einerseits mit Latenzzeiten gemessen, andererseits auf IP-Adressbereiche zurückgeführt werden. Ebenso sind Varianten denkbar, welche die Anzahl der Router (Hops) zählen oder aber mit Hilfe von GPS reale Entfernungsdaten bestimmen. Der Entfernungsbegriff kann weiterhin über logische Metriken, z. B. Art der Services, definiert werden. In der aktuellen Realisierung des Domain Service werden Entfernungen noch nicht berücksichtigt.

Mechanismus Der Domain Master ist ein auserwählter Agentenserver, der seine Rolle zugewiesen bekommt. Als zentraler, beständiger Knoten im sonst dezentralen Agentenansatz ist er fester Kontaktpunkt für alle Domain-Manager und muss bei den Domain-Managern bekannt sein. Die Rollen der Manager werden im Gegensatz dazu dynamisch verteilt. Beim Start eines Agentenservers wird ein UDP-Multicast an alle Rechner des IP-Subnetzes durchgeführt. Die initiale Rolle des Servers ist die eines Domain-Knotens. Bekommt der Agentenserver keine Antwort auf den Multicast, übernimmt dieser die Rolle des Domain-Managers, da er offensichtlich der erste in der Domain ist – eine neue Domain ist gegründet. Falls bereits ein Domain-Manager im Subnetz vorhanden ist, antwortet dieser auf den Multicast mit seiner URL. Anschließend werden Agenten zwischen dem neuen Domain-Knoten und dem Domain-Manager ausgetauscht, um die Kompatibilität bezüglich der Protokolle zu testen und letztendlich den neuen Knoten beim Domain-Manager zu registrieren. Der Agentenserver wird in die Liste des Domain-Managers aufgenommen und die Domain erweitert. Der Prozess läuft automatisch ab und benötigt auf Grund der Nutzung von UDP-Paketen und sehr kleinen Agenten durchschnittlich nur 40 ms in einem 100 Mbit/s Netzwerk [BER01a].

Wird ein Domain-Knoten beendet, sendet dieser einen Agenten zum Manager, um sich abzumelden. Fällt ein Knoten aus oder bricht eine Verbindung zusammen, bemerkt der Manager den Ausfall auf Grund einer regelmäßigen Prüfung der Agentenserver auf seiner Liste von Knoten. Zu diesem Zweck werden Agenten zwischen Manager und Knoten verschickt. Wird der Domain-Manager beendet, wählen die Domain-Knoten einen neuen Manager innerhalb ihrer Domain. Die Festlegung eines Agentenservers, der die Rolle des Managers übernehmen soll, wird mit Hilfe von manuell gesetzten Prioritäten gesteuert: Der Knoten mit der höchsten Priorität wird Domain-Manager. Dadurch kann es wiederum zu einem Wechsel der Rollen beim Starten eines neuen Servers kommen, wenn dieser die höchste Priorität in der Domain besitzt. Leistungsstarke Rechner sollten mit einer hohen Priorität ausgestattet werden.

Realisierung Der Domain Service ist kein Bestandteil des Agentensystems, sondern eine eigene Komponente der MAS Middleware-Schicht (hervorgehoben in Abb. 2.5 auf der nächsten Seite). Die Rollen der Server innerhalb des Agentennetzwerks werden auf Basis von stationären und mobilen Agenten ausgehandelt. Auf jedem Agentensystem wird ein *Domain Information Agent* (DIA), ein stationärer Systemagent, gestartet. Der Start dieses Agenten erfolgt automatisch nach dem Start eines Agentenservers, ähnlich einem Dienst im Betriebssystem.

Die Rolle eines Agentenservers innerhalb einer Domain wird durch die Zustände des lokalen DIA charakterisiert. Die Zustände werden im Zustandsübergangsdiagramm in Abbildung 2.6 auf Seite 42 dargestellt. Diese sind im Lebenszyklus eines stationären Agenten eingebettet (Abb. 1.8 auf Seite 23). Auf die Darstellung der Einbettung wurde im

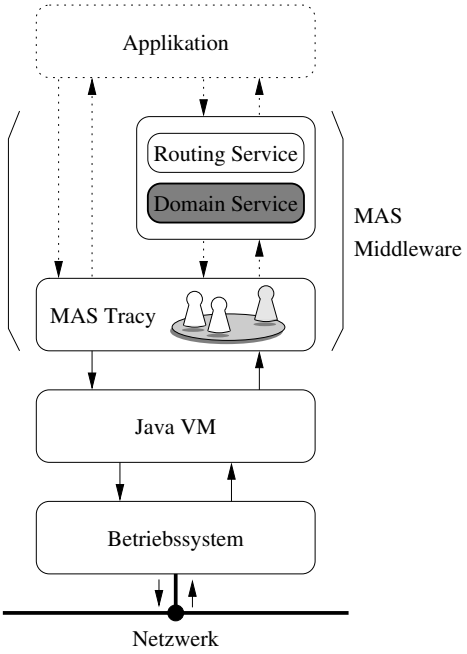


Abbildung 2.5: Der Domain Service im Schichtenmodell

Zustandsübergangsdiagramm zugunsten der Übersichtlichkeit verzichtet.

Nach dem Start des DIA wird im Zustand *Startup* mit Hilfe des UDP-Multicasts auf der Adresse 224.0.0.1 das lokale Subnetz nach einem existierenden Domain-Manager durchsucht. Der DIA des Domain-Managers hat den Zustand *Server Running*, lauscht am Netzwerk und antwortet auf den Multicast mit seiner URL, verpackt in einem UDP-Paket. Bekommt der DIA diese Antwort, wechselt sein Zustand auf *Client Running* und übernimmt die Rolle eines „gewöhnlichen“ Domain-Knotens. Die weiteren Zustände im Diagramm sind informativ und werden nicht weiter betrachtet.

Die DIAs der einzelnen Agentenserver kommunizieren untereinander über mobile Agenten. In der lokalen Domain wird dadurch die Liste der lokalen Knoten ausgetauscht und aktualisiert. Die DIAs der Domain-Manager erfragen wiederum mit Hilfe mobiler Agenten die entfernten Domain-Manager beim DIA des Masters. Der DIA des Masters handelt ebenfalls in der Rolle eines Managers (Zustand: *Server Running*), allerdings auf darüber liegender Hierarchieebene.

Eine Konsequenz aus der Bildung einer Domain auf Basis des Multicast-Ansatzes ist die Beschränkung der Knoten in einer Domain auf eine handhabbare Größe. Die Anzahl der Rechner in einem IP-Subnetz ist limitiert (üblicherweise 254 Rechner im Klasse C Netz). Die Anzahl der Agentenserver kann jedoch größer sein, da auf einem Rechner mehrere Server gestartet werden können. Die Anzahl der Agentenserver einer Domain korrespondiert dennoch mit der Anzahl der Rechner in dieser Domain. Die Anzahl der

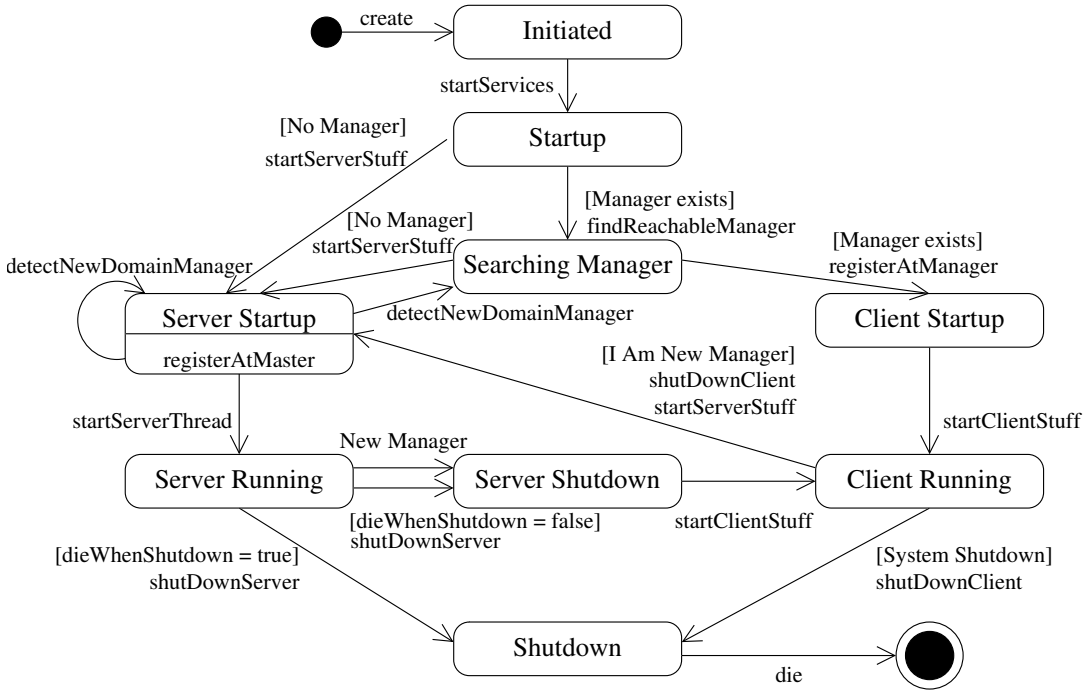


Abbildung 2.6: Zustandsübergangsdiagramm Domain Information Agent

Domain-Knoten liegt in einem Bereich, der problemlos mit Software zu verwalten ist. Dieser Fakt ist für diese Arbeit von entscheidender Bedeutung und wird im Kapitel 4.2 noch einmal aufgegriffen.

Fazit Der TRACY Domain Service ist ein Infrastruktur-Aufsatz für mobile Agentensysteme, der die Gesamtheit aller Agentenserver im Netzwerk organisiert und sich in die Ebene der MAS Middleware einordnet (siehe Abb. 2.5 auf der vorherigen Seite). Dabei werden Mengen von (zusammengehörigen) Agentenservern zu zahlenmäßig beschränkten, gut handhabbaren Domains zusammengefasst. Der ursprünglich vollständig dezentrale Ansatz der mobilen Agenten wird hierdurch aufgeweicht. Jede Domain enthält zur Verwaltung der Domain-Knoten einen zentralen Punkt, den Domain-Manager. Domains wiederum werden über einen Master verwaltet und untereinander bekannt gemacht. Dabei sendet der Master als Antwort auf Domainanfragen eine nach bestimmten Kriterien gefilterte Menge von Domains zurück. Der Filter wird so angesetzt, dass der Zusammenhang des gesamten Agentennetzwerks nicht verloren geht: Ein jeder Agentenserver kennt (transitiv) jeden anderen. Das durch dieses Domain-Konzept aufgebaute logische Netzwerk, das von der darunter liegenden Topologie unabhängig ist, ist hierarchisch strukturiert und kann als Ausgangspunkt für weiterführende Anwendungen benutzt werden. So können (fremde) Agenten den Domain Information Agenten eines Agentenservers befragen, um

Informationen über die Domain zu erlangen. Diese Informationen werden z. B. auf einem Blackboard gehalten.

Der in dieser Arbeit genutzte Domain Service ist auf TRACY aufgesetzt, kann jedoch auf grundsätzlich jedem mobilen Agentensystem aufgesetzt werden.

2.3 Andere mobile Agentensysteme

Vorwiegend im Bereich der verteilten Systeme sind meist auf Basis von Forschungsprojekten mobile Agentensysteme entstanden. Einige dieser Projekte sind im Laufe der Jahre bereits eingeschlafen oder leben als Open Source Projekte im Netz weiter. An dieser Stelle werden einige ausgewählte Agentensystem kurz vorgestellt. Auf eine ausführliche Darstellung und lückenlose Auflistung von Agentensystemen wird verzichtet. Eine Übersicht über Agentensoftware befindet sich auf den Internet-Seiten von Agentlink [Age04], einer Organisation, die europäische Wissenschaftler im Bereich der Agenten vernetzt.

Aglets

Das vielleicht populärste und vermutlich erste in Java programmierte Agentensystem ist Aglets von IBM. Es unterstützt schwache Migration mit einer Anzahl von fixen Methoden, die direkt vor und nach einer Migration aufgerufen werden. Aglets basierte ursprünglich auf der ersten Version von Java. Eine Weiterentwicklung stagnierte lange Zeit. Nachdem Aglets zum Open Source System von IBM (2000) ernannt wurde, erfolgte eine Anpassung des Systems an Java2. Aglets ist eines der wenigen Systeme, die sich bei der Kommunikation am MASIF-Standard [OMG00] der OMG orientieren.

SeMoA

Das mobile Agentensystem SeMoA (Secure Mobile Agents) der Fraunhofer Gesellschaft [Fra04] konzentriert sich auf Sicherheit und versucht, verschiedene Schutzmechanismen in ein Agentensystem zu integrieren. Dabei werden unterschiedliche Szenarien berücksichtigt, die einerseits den Schutz der Agentenplattform und des Systems vor böswilligen Agenten, andererseits den Schutz der Agenten vor böswilligen Angreifern (andere Agenten oder Agentensysteme) betreffen. Des Weiteren bildet die Interoperabilität einen wichtigen Aspekt, um die Schutzmechanismen zu etablieren. So kann das Agentensystem mit Agenten von JADE und Aglets umgehen, aber auch TRACY Agenten können ausgeführt werden. Im Gegensatz zu TRACY ist die Migration in SeMoA jedoch sehr einfach gestaltet und kann durch Agenten nicht adaptiert werden.

Grasshopper

Grasshopper [IKV04a] von der Firma IKV++ Technolgies AG [IKV04b] aus Berlin, mittlerweile unter dem Namen enago Mobile bekannt, ist ein weiteres mobiles Agentensystem. Besonderheit dieses Agentensystems ist die Konformität zu den Standards von MASIF und FIPA [BBCM99]. Dadurch ist die Interaktion mit Agenten anderer Agentensysteme, die standardisierte Kommunikation beherrschen, gesichert. Als Teil des MASIF-Standards [OMG00] wurde eine *Region Registry* integriert, die eine Verwaltung für eine Menge von Agentenservern übernimmt sowie Informationen zum Aufenthalt von Agenten (Agent Tracking) in der jeweiligen Region pflegt. Dieses Konzept ist dem Domain Konzept aus TRACY ähnlich. Der Verwalter einer Region kann allerdings nicht dynamisch bestimmt werden, sondern wird fixiert. Dadurch kann bei einem Ausfall die Region zerfallen. Die Kopplung von Regionen ist möglich, allerdings nicht dynamisch. Die Konsequenz ist, dass Agenten nur Informationen über fremde Regionen erhalten, wenn diese in der Region Registry manuell eingebunden wurden und damit nicht mehr fremd sind.

JADE

Java Agent DEvelopment Framework (JADE) [Tel04b] ist eine Open-Source Plattform für Agenten-basierte Anwendungen und wird von Telecom Italia Lab [Tel04a], dem R&D-Zweig der Telecom Italia Group, vertrieben. Das in Java implementierte Agentensystem hält sich an den durch die FIPA vorgegebenen Standard zur Interoperabilität und fördert die Entwicklung von Agenten auf dieser Basis. Ein Schwerpunkt dieses Agentensystems besteht darin, die größtmögliche Unterstützung bei der Entwicklung von Agenten zu geben. Entsprechende zusätzliche Komponenten sind für das System verfügbar. Auf Grund unzureichender Dokumentation ist die Programmierung der Agenten im Detail schwieriger als erwartet. Die Migration ist sehr einfach gehalten und nicht anpassbar.

Fazit

Im Laufe der Jahre entstand eine Vielzahl von mobilen Agentensystemen. Oft haben die Systeme nur einen prototypischen Charakter. Die wenigsten Systeme erreichten Produktreife. Auf dem Gebiet der MAS haben sich nur wenige Standards etabliert. Entsprechend groß ist die Variation bei der Umsetzung der Idee der mobilen Agenten. Die Mobilität ist meist nur eine nicht stärker betonte Funktionalität. In den Systemen ist die Organisation einer Infrastruktur im Allgemeinen nicht zu finden.

2.4 Einordnung der Arbeit im Agentensystem TRACY

Zur Wahl von TRACY Die Wahl von TRACY als Basis dieser Arbeit berücksichtigte nicht nur die lokale Nähe zur Friedrich-Schiller-Universität, sondern auch die bisherigen Forschungsarbeiten auf dem Gebiet der mobilen Agenten. Der Kenntnisstand über die Systemfunktionalitäten war dementsprechend ausgereift. Den entscheidenden Grund für die Wahl von TRACY bildete dessen gute Unterstützung der Mobilität von Agenten, die grundlegende Eigenschaft mobiler Agenten aus Sicht der verteilten Systeme. TRACY ist modular aufgebaut und begünstigt dadurch Erweiterungen. Der Zugriff auf die aktuellste Version ist gesichert, da die Entwicklung von TRACY vor Ort erfolgt.

Weitere Voraussetzung für diese Arbeit ist ein Infrastruktur-Dienst (Domain Service in Abb. 2.7), der einzelne Agentenserver zu einer Menge lose gekoppelter Knoten in einem Agentennetzwerk verbindet. Informationen über aktive Knoten sind für Agenten in einem derartigen Netzwerk verfügbar. Jeder Knoten ist im Agentennetzwerk, zumindest transitiv über andere Agentenserver bekannt. Der Domain Service, der auf Basis von TRACY implementiert ist, realisiert einen solchen Infrastruktur-Dienst. Dieser wurde als Ausgangspunkt für diese Arbeit benutzt.

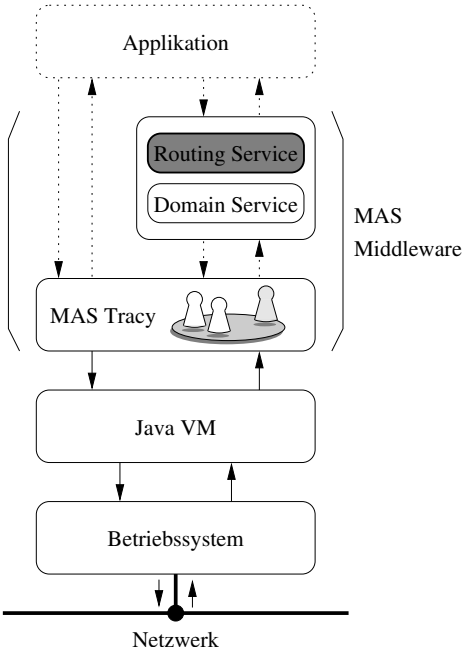


Abbildung 2.7: Der Routing Service im Schichtenmodell

Einordnung Im Mittelpunkt dieser Arbeit steht eine durch mobile Agenten und Applikationen nutzbare Middleware-Anwendung im Agentensystembereich – der Routing

Service *ProNav* (in Abb. 2.7 auf der vorherigen Seite hervorgehoben). Die Umsetzung wurde beispielhaft und prototypisch mit Hilfe des mobilen Agentensystems TRACY, einem Agentensystem der ersten Stufe (siehe Kapitel 1.3.4), vorgenommen. *ProNav* ist jedoch nicht Bestandteil des mobilen Agentensystems, sondern bildet eine Middleware-Komponente, die auf dem Agentensystem und dessen Basisfunktionalitäten (Domain Service) aufsetzt. Dadurch erhält die gesamte MAS Middleware ergänzende Funktionalitäten, die unabhängig von einem konkreten Agentensystem sind.

Aus der Sicht einer Applikation erfüllt die Middleware-Komponente *ProNav* Querschnittsaufgaben, die einen wiederkehrenden Charakter haben. Dadurch wird dem Programmierer ein Teil der Implementierung, die für jede Applikation erneut anstehen würde, abgenommen, als Dienst ausgelagert und zur generellen Nutzung zur Verfügung gestellt. Der Weg zum Agentensystem der zweiten Stufe, bei der sich ein Programmierer auf die Erstellung des reinen Applikations-Algorithmus konzentriert, ist geebnet.

Das in dieser Arbeit vorgestellte Rahmenwerk für mobile Agenten, der Routing Service *ProNav*, versucht, weitere Informationen über das Agentennetzwerk zusammenzutragen. Den Agenten werden die Informationen vom System derart zur Verfügung gestellt, dass ihre Wahrnehmung gestärkt wird. Ihnen werden somit autonome Migrationsentscheidungen und -planungen ermöglicht. Grundlage für die Realisierung von *ProNav* ist die Organisation der Infrastruktur, um das Quantitätsproblem zu lösen, wobei der TRACY Domain Service genutzt wird. Die Beschreibung des Rahmenwerks erfolgt detailliert im Kapitel 4. Die prototypische Implementierung wird mit Hilfe von spezialisierten Agenten, die auf Basis der von TRACY definierten mobilen und stationären Agenten realisiert werden, vorgenommen.

Kapitel 3

Konzepte zur proaktiven autonomen Migration mobiler Agenten

Um die Vorstellungen und Ziele dieser Dissertation zu beschreiben, wird ein Beispielszenario herangezogen. Das Szenario präsentiert einen mobilen Agenten, der sich proaktiv in einem Netzwerk orientiert und seine Reise autonom organisiert. Die Konzepte dieser Arbeit werden durch das Szenario verdeutlicht.

Ein allgemein nutzbares Rahmenwerk, der in dieser Arbeit entwickelte Routing Service *ProNav,* dient zur Realisierung der Konzepte. Die Komponenten dieses Dienstes werden in diesem Kapitel herausgearbeitet. Der Ablauf des Szenarios zeigt das Zusammenspiel der Komponenten sowie die Nutzung dieser von Agenten. Die Analyse des Szenarios führt zur Aufstellung der Thesen dieser Arbeit.

3.1 Anwendungsszenario und Vision

Im ersten Teil des Kapitels werden die Vorstellungen einer autonomen Migration verdeutlicht. Die Analyse des Szenarios zeigt im zweiten Teil die notwendigen Schritte zum Routing Service auf. Als Fazit werden die Ergebnisse zusammengefasst.

3.1.1 Anwendungsszenario autonome Migration

Ein Agent bekommt eine Aufgabe von einem Auftraggeber übermittelt. Hat der Agent den Auftrag bekommen, arbeitet er primär an der Lösung, indem er geeignete Dienste auf Agentenservern in Anspruch nimmt. Zunächst muss der Agent passende Dienste im Agentennetzwerk lokalisieren. In dem Szenario hat ein Agent zu diesem Zweck Zugriff auf eine Informationsbasis, die entsprechende Daten, wie Informationen zu Diensten im Agentennetzwerk, beinhaltet. Diese Informationsbasis, die als Landkarte des Netzwerks betrachtet werden kann, steht auf jeder Agentenplattform zur Verfügung.

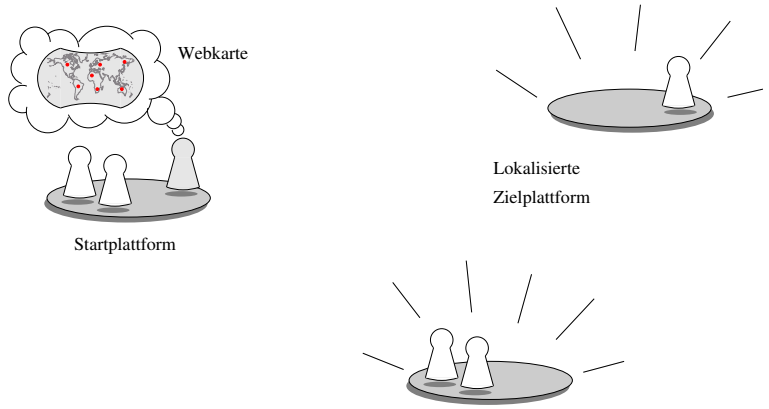


Abbildung 3.1: Nutzung der Webkarte zur Lokalisierung von Diensten im Netz

Auf dieser *Webkarte* sucht der Agent nach Diensten, die zur Lösung der ihm übertragenen Aufgabe beitragen. Das Resultat der Suche ist eine Menge von potentiellen Zielplattformen (siehe Abb. 3.1). Neben den Karteninformationen kann ein Agent eigene gewonnene Erfahrungen einbringen und die Liste ergänzen bzw. ausdünnen. Bevor der Agent die Zielplattformen besuchen kann, muss er die Reihenfolge der Besuche festlegen, d. h. er muss die Planung seiner Reise weiter vervollständigen. Unterstützt wird der Agent wieder durch die lokale Plattform, die ihm einen *Routenplaner* zur Nutzung anbietet. Der Routenplaner benötigt vom Agenten die Stationen seiner geplanten Reise, die Liste der Agentenserver. Er ermittelt einen effizienten Weg durch das heterogene Agentennetzwerk (siehe Abb. 3.2 auf der nächsten Seite). Dazu benötigt der Routenplaner Informationen

über die Qualität der Verbindungen zwischen den einzelnen Stationen. Diese Daten befinden sich ebenfalls auf der Webkarte, die vom Routenplaner zur Berechnung verwendet wird. Das Ergebnis ist ein Reiseplan. Dieser dient als Vorschlag für den mobilen Agenten. Ob der Agent die offerierte Route abläuft, liegt in der Autonomie des Agenten.

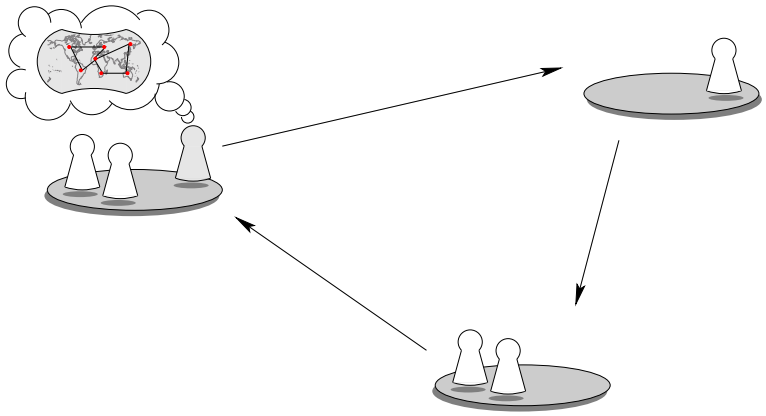


Abbildung 3.2: Planung einer initialen Reiseroute

Die Suche und Planung der Route liefert dem Agenten eine mögliche und optimierte Reiseroute durch das Agentennetzwerk. Diese enthält eine Liste mit den für den Agenten interessanten Plattformen in der zu besuchenden Reihenfolge. Der entstandene Weg durch das Netzwerk kann dabei nach verschiedenen Kriterien, wie Zeit oder Kosten etc., optimiert sein. Allerdings basiert dieses Ergebnis nur auf einem „Schnappschuss“, der eine Situation zum Zeitpunkt der Planung im Netzwerk beschreibt.

Damit ist die initiale Routenplanung des Agenten abgeschlossen. Im nächsten Schritt kann die Reise des Agenten beginnen – der Reiseplan wird ausgeführt. Der Agent besucht in der ermittelten Reihenfolge die gefundenen Zielplattformen, die geeignete Dienste anbieten. Auf einer Plattform angekommen kommuniziert und kooperiert der Agent lokal mit anderen Agenten, um seinem Aufgabenziel näher zu kommen.

Während der Reise des Agenten können sich auf Grund der Dynamik des Agentennetzwerks Informationen der Webkarte ändern. Zudem sind die Informationen vor Ort aktueller und präziser, insbesondere bei Reisen des Agenten zu weiter entfernten Plattformen. Der reisende Agent sollte die Route von Zeit zu Zeit prüfen und gegebenenfalls dynamisch gemäß der Optimierungskriterien anpassen (siehe Abb. 3.3 auf der nächsten Seite). Diese iterative Art der Routenplanung sorgt für eine stets aktuelle Wahrnehmung der Umwelt und ermöglicht die Reaktion auf die dem Agentennetz inhärente Dynamik.

Ist das Ende der Reiseroute erreicht, kehrt der Agent zum Auftraggeber zurück und präsentiert die Ergebnisse seiner Reise. Erfüllt der Agent seine Aufgabe bereits vor dem Ende der Route, kann er die Abarbeitung seines Reiseplans abbrechen und zum Auftraggeber zurückkehren.

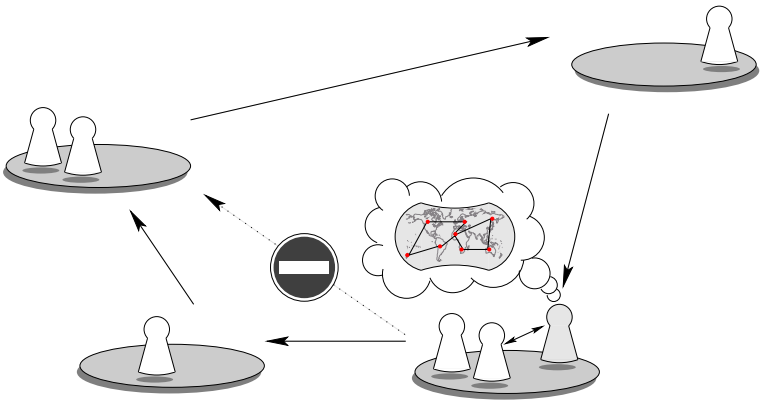


Abbildung 3.3: Dynamische Anpassung einer Reiseroute

3.1.2 Analyse des Szenarios

Zu Beginn des Szenarios wird die Aufgabe dem Agenten übergeben. Die Übergabe erfolgt durch die Interaktion des Agenten mit einem Benutzer, in der Regel dem Eigentümer des Agenten. Die Beschreibung der Aufgabe enthält normalerweise keine Informationen über die Art und Weise, wie der Agent die Aufgabe erfüllen soll. Das *WIE* der Aufgabe ist dem Agenten überlassen. Der Agent muss auf technischer Ebene autonom die Reise durch das Agentennetzwerk organisieren – autonome Routenplanung.

Der Auftraggeber konzentriert sich auf den Kern der Aufgabe und teilt dem Agenten das *WAS*, das Ziel der Aufgabe, mit. Das WAS einer Aufgabe erfordert bei dem Agenten die Fähigkeit der Interpretation. In einem Agenten das Verständnis der Aufgabe zu integrieren, ist schwierig und umfangreich. Ein einfacher Ansatz zur Verbesserung der Interpretation bzw. der Interaktion eines Agenten, der sich mit dem Einsatz virtueller Figuren (Avatare) beschäftigt, wurde untersucht [Gle04]. Der zur Interaktion fähige Avatar, wird benutzt, um die Übermittlung der Aufgabe interaktiv zu gestalten. Ziel der Interaktion ist, den Interpretationsspielraum möglichst stark einzuschränken, d. h. dem Auftraggeber genau die Daten zu entlocken, die der Agent zur Erfüllung seiner Aufgabe benötigt.

Die vorliegende Arbeit konzentriert sich auf die autonome Planung und Durchführung (WIE) der Reise des Agenten. Damit ein Agent Ziele im Agentennetzwerk ausfindig machen kann, muss er über eine geeignete, erweiterte Wahrnehmung verfügen (siehe Kapitel 1.2.4). In Analogie zu einem reisenden Menschen orientiert sich der Agent in dem Szenario auf einer Karte. Im Gegensatz zu einer gewöhnlichen Landkarte unterliegt die Webkarte allerdings einer ständigen Änderung, da die Dynamik in einem Netzwerk tendenziell hoch ist – potentielle Ziele fallen weg, kommen hinzu, Wege ändern sich etc. Die Informationen auf der Webkarte haben keinen spezifischen Anwendungsfokus und sind allgemein gehalten. Dennoch stellt die Webkarte eine Orientierungshilfe für unterschied-

liche Agenten zur zielorientierten Wahrnehmung dar.

Diese Webkarte steht dem mobilen Agenten lokal auf jeder Agentenplattform zur Verfügung. Die jeweilige Plattform bzw. ein lokaler Dienst übernimmt die Aktualisierung der Karte. Die dezentral organisierten Plattformen arbeiten bei der Erstellung und Aktualisierung der Karte zusammen und tauschen kartographierte Teile aus. Dadurch verfügt jede Plattform eine aktuelle Variante der Webkarte. Die Karte einer Plattform fokussiert die lokale Umgebung, die entsprechend detailliert dargestellt ist. Die wichtigsten Informationen für entferntere Gebiete, sofern diese bekannt sind, werden auf der Karte zusammenfassend dargestellt. Für einen Betrachter der lokal verfügbaren Karte entsteht eine Sichtweise auf das Agentennetzwerk, die mit dem Fish-Eye-View aus der Fotografie vergleichbar ist: Das Zentrum ist scharf während die äußeren Bereiche an Schärfe verlieren. Der Fokus liegt auf dem (lokalen) Zentrum und repräsentiert einen hohen Detaillierungsgrad. Je weiter man aus dem Zentrum herausblickt, desto ungenauer ist die Ansicht. Obwohl Details nicht mehr erkennbar sind, erhält man eine grobe Vorstellung von der Umgebung. Der Detaillierungsgrad der Karte wird bewusst verringert, um Probleme einer dezentralen, globalen Karte, nämlich die Skalierbarkeit und Aktualität der Daten, zu vermeiden. Durch die Verringerung des Detaillierungsgrads kann mit der Quantität der Daten, die durch das Agentennetzwerk entstehen, und mit der inhärenten Dynamik adäquat umgegangen werden. Dennoch entsteht durch die vielen lokalen Karten mit ihren unscharfen Schnittstellen eine in ihrer Gesamtheit globale, detaillierte Karte des Agentennetzwerks.

Bevorzugt wird der Ansatz die Karte lokal auf den Agentenserver zu platzieren, da die Mitnahme und Pflege einer Karte durch den Agenten in mehrfacher Hinsicht keinen Sinn ergibt:

Einerseits sollen mobile Agenten kleine Programme sein, die schnell durch ein Netzwerk migrieren können. Eine eigene Karte im Gepäck zu haben, bedeutet allerdings höheren Ballast. Der Agent benötigt bei einem Transfer zwischen Plattformen mehr Zeit und verursacht mehr Netzwerklast.

Andererseits muss sich der Agent neben der eigentlichen Aufgabe zusätzlich um die Administration der Karte kümmern. Das verursacht weiteren Ballast, da die Administration wieder mehr Code verlangt. Des Weiteren ist der Agent während eines Transfers eingefroren und kann die Karte nur beim Aufenthalt auf den Plattformen nutzen.

Ein weiterer Grund, dem Agentenserver die Karte zu überlassen, wird unter Verdeutlichung der Anforderungen an eine solche Karte erkennbar: Die Karte wird *nicht* für aufgabenspezifische Zwecke verwendet, sondern:

1. Zur Suche nach Diensten,
2. Zur Planung einer Reiseroute,
3. Zur Optimierung einer Reiseroute.

Hierbei handelt es sich um Querschnittsaufgaben, die in jeder Applikation wiederzufinden sind. Daher wird keine Karte benötigt, die auf eine Aufgabe spezialisiert ist und für einen oder von einem speziellen Agenten erstellt wird.

3.1.3 Fazit

Das vorgestellte Anwendungsszenario zeigt die Autonomie eines Agenten, der ohne Eingriff von außen selbständig einen Weg durch ein heterogenes Netzwerk findet. Ein Agentensystem, das dieses Szenario umsetzen kann, wird als Agentensystem der zweiten Stufe bezeichnet (siehe Kapitel 1.3.4). Durch die Verwendung der im Szenario beschriebenen Webkarte und eines Routenplaners ist eine autonome Migration der Agenten möglich. Der Sprung zu Agentensystemen der zweiten Stufe ist geschafft.

Das Szenario betont die technischen Aspekte, die mit der Realisierung einer autonomen Routenplanung verbunden sind. Dabei werden weder die Interpretation der Aufgabe, das WAS, noch die konkrete Umsetzung der Aufgabenbewältigung auf Applikationsebene berücksichtigt. Das Ziel ist ein allgemein nutzbares Rahmenwerk, der Routing Service *ProNav*, durch das die Wahrnehmung eines Agenten erweitert und das autonome und proaktive Verhalten gefördert wird.

Drei wesentliche Komponenten von *ProNav* werden bereits anhand des Szenarios deutlich:

- Die Webkarte

  Eine dezentrale, dynamische Informationsbasis, die zur Lokalisierung von Diensten im Agentennetzwerk benötigt wird. Die Qualität der Informationen und der Detaillierungsgrad stehen in Relation zur Entfernung der potentiellen Ziele: Die Webkarte fokussiert die lokale Umgebung einer Plattform. Der Detaillierungsgrad nimmt mit zunehmender Entfernung von der lokalen Plattform ab.

- Der Routenplaner

  Eine Komponente, die auf Basis der Karteninformationen und der vom Agenten präferierten Plattformen einen effizienten Weg durch das heterogene Agentennetzwerk plant. Ergebnis ist eine initiale Reiseroute für den Agenten. Diese Planung kann iterativ auf entfernten Plattformen durch den Agenten wiederholt werden.

- Der Migrationsoptimierer

  Eine Komponente zur Optimierung der im Routenplaner erstellten Reiseroute. Die Optimierungskriterien können beispielsweise Zeit oder Kosten sein. Für eine Optimierung kann ein Agent in seine Teile zerlegt werden. Versendet werden nur die

Teile, die mit hoher Wahrscheinlichkeit auf der Reise benötigt werden. Die Optimierungsmöglichkeiten stehen im engen Zusammenhang zum zu Grunde liegenden Agentensystem. Daher ist dieses Modul Agentensystem-spezifisch.

Der Aufbau und das Zusammenspiel der Komponenten wird in Kapitel 4 vertieft und vervollständigt.

3.2 Thesen der Arbeit

Wahrnehmung

T1 Mit Hilfe einer Webkarte wird ein Agent befähigt, seine Umwelt korrekt wahrzunehmen.

Durch die Webkarte werden Ortsinformationen der virtuellen Welt zur Verfügung gestellt. Die Informationen werden dem Agenten in einer Form präsentiert, die ihm eine Verarbeitung ermöglicht.

Quantität und Qualität der Wahrnehmung

T2 Eine Webkarte für Services ist für große Netzwerke möglich.

Das potentiell zahlenmäßig große Agentennetzwerk lässt sich in Stücke handhabbarer Größe zerteilen und kartographieren.

T2a Die Dynamik moderner Netzwerke kann auf Basis einer Webkarte und eines Routenplaners adäquat gehandhabt werden.

Die Webkarte stellt die momentan erwartete Situation (Vorhersage) im Netzwerk dar und berücksichtigt daher die vorherrschende Dynamik. In die Planung der Route kann die Dynamik einfließen und die aktuelle Route des Agenten wird bei Bedarf modifiziert.

T2b Für einen bestimmten mobilen Agenten ist die Planung einer Route durch das Netzwerk effizient durchführbar.

Die Planung einer Route ist mit Hilfe bekannter Heuristiken für den praktischen Einsatz effizient genug möglich.

Organisation

T3 Agentennetzwerke benötigen eine grundlegende Organisation der Infrastruktur, damit die mobilen Komponenten das gesamte Potential eines verteilten Systems nutzen können.

Eine Kopplung der Agentenplattformen auf logischer Ebene ermöglicht die Integration generell nutzbarer Komponenten im Agentensystem, welche die dem System inhärente Dynamik berücksichtigen können.

Verallgemeinerung

T4 Autonome und proaktive Routenplanung durch mobile Agenten ist als Querschnittsaufgabe in einem mobilen Agentensystem realisierbar.

Die Planung einer Route ist unabhängig von einer speziellen Aufgabe eines mobilen Agenten. Diese wiederkehrende Anforderung ist als Funktionalität in einem Agentensystem und nicht im Agenten selbst zu integrieren.

Kapitel 4

Routing Service *ProNav* – Das Rahmenwerk zur proaktiven Navigation

Der in dieser Arbeit entwickelte autonome Routing Service *ProNav* – Proaktive Navigation – wird vorgestellt. *ProNav* ermöglicht mobilen Agenten autonom eine proaktive Routenplanung für ihre applikationsbedingte Navigation im Netz.

Die Komponenten des Services werden im Einzelnen beschrieben. Mögliche Konzepte, die aus dem Szenario hervorgegangen sind, werden analysiert. Die Funktionsweise und Realisierung der Komponenten wird erklärt.

4.1 Allgemeine Beschreibung

Die Webkarte ist eine Informationsbasis, die Daten über Agentenplattformen und Domains verwaltet und anbietet. Daten sind beispielsweise Informationen zu Diensten auf den Plattformen und in den Domains sowie Verbindungsinformationen zwischen Plattformen und Domains im Agentennetzwerk. Die Erstellung der Karte übernimmt das *Kartenmodul*. Für die Etablierung einer Karte stellen sich verschiedene Fragen:

- Ist eine Webkarte des kompletten Agentennetzwerks möglich?
- Wie hoch darf das dynamische Verhalten des Netzwerks sein?
- Welche Daten können kartographiert werden?
- Wie genau und aktuell sind die Daten?

Auf Grund der Verbindungsdaten zwischen Plattformen und Domains ist die Idee einer Routenplanung nicht weit entfernt. Für eine Menge von Zielpunkten der Karte soll der *Routenplaner* eine effiziente Reiseroute durch das Netzwerk erstellen. Bei genauer Betrachtung des Problems stößt man auf die Grenzen der Informatik: Die Anzahl der möglichen Reiserouten führt schon bei wenigen Zielpunkten zu Berechnungszeiten, die in keinen akzeptablen Grenzen liegen. In Kapitel 4.3 wird gezeigt, wie dennoch dieses *Quantitätsproblem* in akzeptablen Zeitschranken gelöst wird.

Im Gegensatz zum Kartenmodul und zum Routenplaner, die auf jedes Agentensystem mit Domainstruktur aufgesetzt werden können, ist der *Migrationsoptimierer* von *ProNav* spezifisch auf TRACY ausgerichtet. Diese Komponente versucht, eine geeignete Migrationsstrategie für einen Agenten zu finden. Ein Ziel ist, nur die Teile des Agenten zu übertragen, die potentiell auf den entfernten Zielplattformen für die Ausführung des Agenten benötigt werden. Hier stellt sich die Frage nach den für den Migrationsoptimierer notwendigen Informationen, damit derartige Abschätzungen möglich sind (siehe Kap.4.4).

Diese drei Komponenten

1. Kartenmodul,
2. Routenplaner,
3. Migrationsoptimierer

ergeben zusammen das Rahmenwerk *ProNav* zur Unterstützung der proaktiven, autonomen Migration mobiler Agenten. Einzuordnen ist *ProNav* als eigenständige Komponente der MAS Middleware (siehe auch Kap. 2.4). Dieser Dienst muss auf jeder Agentenplattform gestartet werden, damit die im Szenario dargestellten Funktionalitäten lokal

verfügbar sind und das Agentennetzwerk vollständig kartographiert werden kann. Zum besseren Überblick sind die Komponenten von *ProNav* und deren Eingliederung in Abbildung 4.1 schematisch dargestellt.

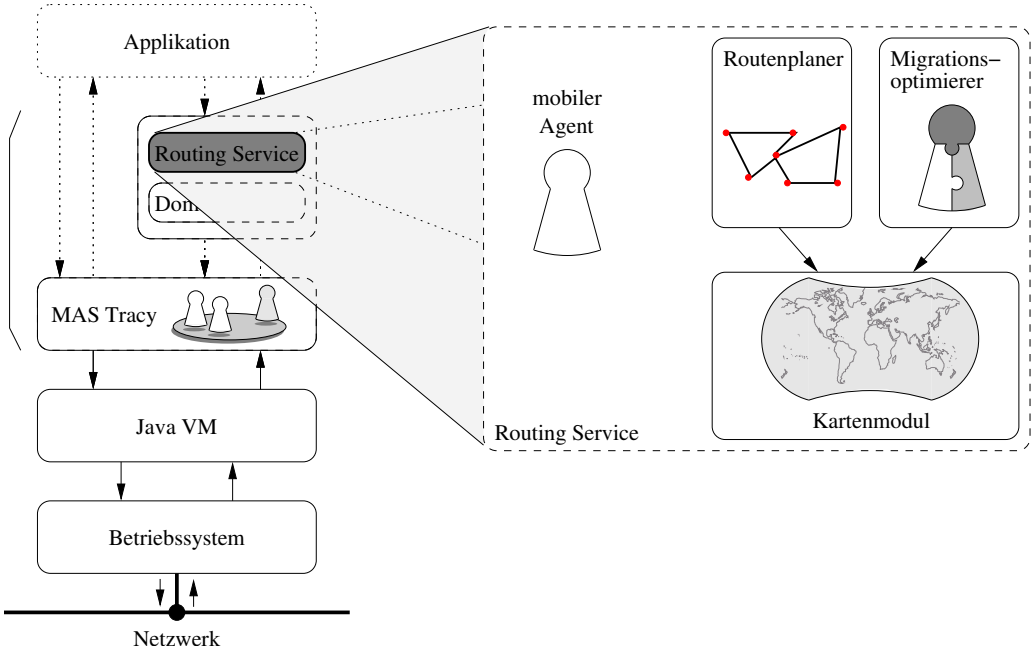


Abbildung 4.1: Komponenten des Routing Service *ProNav*

Durch das vorgestellte Anwendungsszenario wird das Zusammenspiel der Komponenten deutlich. Die Webkarte wird von den anderen beiden Komponenten als Informationsbasis genutzt. Bevor eine Optimierung der Migration auf technischer Ebene vonstatten gehen kann, muss der Routenplaner eine initiale Reiseroute erstellt haben. Diese Abhängigkeiten charakterisieren die Reihenfolge, in der die Module von einem Agenten im Regelfall benutzt werden. Dennoch sind die Module eigenständig nutzbar und können von einem Agenten einzeln genutzt werden. So muss ein Agent nicht die Dienste aller Komponenten in Anspruch nehmen, wenn er beispielsweise nur eine geeignete Migrationsstrategie vom Migrationsoptimierer vorgeschlagen bekommen möchte. Die notwendigen Eingabeparameter müssen dem Agenten dabei bekannt sein. Die Karte muss entsprechende Daten beinhalten.

4.2 Kartenmodul

Das Kartenmodul dient der Erstellung der Webkarte und bildet die Informationsbasis, die eine Grundlage für weitere Komponenten darstellt (siehe Abb. 4.2 auf der nächsten Seite).

Dieses Modul unterstützt in erster Linie einen mobilen Agenten bei der Lokalisierung von Diensten im Agentennetzwerk.

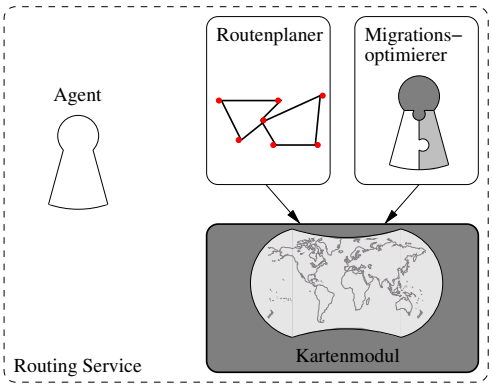


Abbildung 4.2: Kartenmodul im Routing Service *ProNav*

4.2.1 Allgemeine Beschreibung und Analyse

Aufgabe des Kartenmoduls ist das Kartographieren eines logischen Netzwerks der Agentenserver. Doch so einfach wie die Formulierung der Aufgabe ist, so schwierig ist die Umsetzung der Idee. Zunächst gilt zu klären, was auf der Karte verzeichnet werden soll. Darauf wird am Ende dieses Abschnitts eingegangen. Des Weiteren stellt sich die Frage, ob ein zentraler oder verteilter Ansatz gewählt werden soll (Soll die Karte zentral oder dezentral im System verfügbar sein?)

In einem zentralen Lösungsansatz muss eine globale Informationsbasis die kompletten Daten an *einer* zentralen Stelle verwalten. Das grundlegende Problem dieses Ansatzes ist die enorme Menge an Daten in der praktischen Anwendung. Jeder Agentenserver muss bei Änderungen seiner Daten oder der lokalen Umgebung die Informationen einem Manager melden. Neben der aufwendigen Datenverwaltung im Manager müssen alle Agenten zur Planung ihrer Route die zentrale Informationsbasis kontaktieren. Die zentrale Managementeinheit wird damit zum Hot-Spot. Dieser zentrale Ansatz steht des Weiteren im Widerspruch zu den Prinzipien der dezentralen Architektur des Agentennetzwerks.

Ein dezentral-verteilter Lösungsansatz führt zur Verteilung und Verwaltung der Karteninformationen auf den Agentenservern im Netzwerk. Ziel ist eine Abbildung der vollständigen „Welt des Agentennetzwerks“ auf jede dieser Karten, damit auf jedem Agentenserver die Informationen für einen Agenten verfügbar sind. Das führt zu einem dem zentralen Ansatz ähnlichen Problem: Die Informationen müssen nicht nur an einem zentralen Punkt gesammelt werden, sondern an jedem Punkt, an dem eine Karte vorhanden ist. Die entstehende Datenflut im Netzwerk wäre unermesslich. Von der Idee, eine „Weltkarte“ für

das gesamte Agentennetzwerk aufzubauen, muss Abstand genommen werden. Die Quantität der Informationen ist technisch nicht handhabbar, da sie nicht mit der Anzahl der potentiellen Agentenserver skaliert.

Ein anderer Extremfall entsteht, wenn jeder einzelne Agentenserver eine minimale „Nachbarschaftskarte“ aufbaut. Diese Karte beschreibt die Sicht des lokalen Agentenservers zu seinen adjazenten Nachbarn. Die Nachbarschaftsbeziehung soll dabei stark begrenzt sein. Möglichst wenige Agentenserver sollen Nachbarn des lokalen Servers sein. Die Menge der Informationen auf der Karte wird begrenzt. Auf dieser Karte befinden sich nur Informationen über die Nachbarn und keinerlei Informationen über weiter entfernte Agentenserver. Das führt wiederum dazu, dass sich ein Agent von Nachbar zu Nachbar bewegen muss, um Informationen im Netzwerk zu suchen. Durch diesen Ansatz entsteht eine Vielzahl von kleinen lokalen Karten, die in ihrer Gesamtheit eine „Weltkarte“ ergeben. Ein Agent kann die kompletten Informationen nicht erfassen. Passende Dienste werden vom Agenten im Netzwerk nicht gefunden. Die Zerteilung ist zu stark, eine Routenplanung ist nicht möglich.

Eine Annäherung der „Weltkarte“ und der „Nachbarschaftskarte“ an eine „Umgebungskarte“ ist ein möglicher Kompromiss. Eine solche Karte, die dezentral auf jedem Agentenserver erstellt und gewartet wird, beinhaltet eine (beschränkte) Sicht auf das Netzwerk vom jeweiligen Agentenserver aus. Die lokale Umgebung eines Agentenservers ist beispielsweise das Subnetz, die Domain, in dem sich der Agentenserver befindet. Die Informationen der Karte werden vom Agentenserver mit seiner Sicht auf das Netzwerk zusammengetragen, d. h. vollständige Informationen über seine begrenzte Anzahl von Nachbarn. Zudem können auf der Karte eines Agentenservers „entferntere“ Agentenserver, die nicht der Domain angehören, verzeichnet sein. Die „Umgebungskarte“ fokussiert auf die lokale Umgebung einer Plattform und stellt reduzierte Informationen über die Einbettung der Plattform in die „Weltkarte“, dem Kontext der lokalen Umgebung zur Verfügung. Diese Sicht kann mit dem Fish-Eye-View aus der Fotografie verglichen werden. Von einem Agentenserver aus gibt es jedoch keine ganzheitliche Sicht auf die Welt aller Agentenserver.

Mit der „Umgebungskarte“ ist ein Ansatz gefunden, der die Menge der Informationen auf der Karte begrenzt. Neben der Begrenzung der Informationen muss die Karte mit der Dynamik des Agentennetzwerks umgehen können. Im Gegensatz zu einer herkömmlichen Landkarte weisen diese Karteninformationen eine hohe Dynamik auf. In [ER02] wurde das dynamische Verhalten der einzelnen Komponenten detaillierter untersucht und verschiedene Stufen der Dynamik charakterisiert:

- Dynamisches Netzwerk

  Die Dynamik eines Netzwerks lässt sich folgendermaßen charakterisieren:

  - Plattformen, die dynamisch wegfallen und hinzukommen

  - Verbindungen zwischen Plattformen, die eine eigene Dynamik aufweisen (Änderung der Qualität, Verbindungsabbruch, Änderung der Verbindungscharakteristik insbesondere bei Funknetzen etc.).

- Dynamische Dienste

  Dienste können auf Plattformen hinzukommen und wegfallen.

- Dynamische Agenten

  Die mobilen Agenten selbst bewegen sich ebenso im Netzwerk, ihre nächsten Schritte sind von außen nicht vorhersehbar.

Als Resümee kann gesagt werden, dass es nicht möglich ist, auf einer Karte Informationen aufzunehmen, die nur für kurze Momente im Agentensystem Bestand haben. Dies ist auch nicht erwünscht, da Diensterbringer auf der Karte nur erscheinen sollen, wenn sie eine gewisse Beständigkeit im System zeigen – wenigstens für einige Minuten. Dies beantwortet, die zu Beginn gestellt Frage: Wie hoch darf das dynamische Verhalten des Netzwerks sein?

Des Weiteren kann die generelle Frage, welche Daten auf der Karte kartographiert werden sollen, beantwortet werden. Die eigentliche Frage dahinter ist: Welche Aufgaben sollen mit der Karte erfüllt werden? Die Karte dient zur Lokalisierung von Diensten und zur Bereitstellung von Informationen, die für eine Routenplanung und eine Migrationsoptimierung nutzbar sind. Die sich daraus ergebenden Karten-, Dienst- und Netzwerkinformationen sowie migrationsspezifischen Daten werden in folgende Kartenelemente untergliedert:

- Knoteninformationen

  Informationen über Agentenserver, auch Eigenschaften (Properties) genannt, werden in diesem Teil der Karte abgelegt. Beispiele sind: Name (Ort) des Agentenservers, angebotene Dienste, unterstützte Protokolle etc.

  Diese Eigenschaften werden auf der Karte in Form von Property-Listen der Agentenserver abgelegt.

- Verbindungsinformationen

  Informationen zu den Qualitäten der Verbindungen zwischen den Agentenservern, wie aktuelle Durchsatzraten, Latenzzeiten, Referenzmigrationszeiten etc.

  Diese Informationen werden in Form einer Verbindungsmatrix gespeichert.

4.2.2 Funktionsweise

Die Funktionsweise des Kartenmoduls wird aus Sicht einer lokalen Plattform erläutert. Für die Erstellung einer Karte auf der Plattform wird eine Liste von Agentenservern der Umgebung benötigt. Zu diesem Zweck wird TRACY Domain Service (Kapitel 2.2) verwendet. Die Domain bildet die lokale Umgebung der Plattform.

Nachdem die Plattform und der Domain Service gestartet wurde, kann das Kartenmodul seine Arbeit aufnehmen. Ein mobiler Agent erfragt eine Liste der Domain-Knoten beim Domain-Manager. Diese Liste bildet den Startpunkt zur Kartenerstellung. Falls die gestartete Plattform das einzige Mitglied der Domain und damit Domain-Manager ist, wird an dieser Stelle nichts weiter getan. Die Plattform selbst bildet die Karte der Domain. Ist die Plattform nicht der einzige Knoten der Domain und löst auf Grund einer höheren Priorität den bisherigen Domain-Manager ab, dann erhält sie die Liste der Domain-Knoten vom alten Domain-Manager. Die Liste ist nun lokal verfügbar und die Kartographierung der Verbindungs- und Knoteninformationen kann beginnen.

Die Kartographierung der Verbindungsinformationen erfolgt mit Hilfe so genannter Messexperimente, die mit jedem aufgelisteten Domain-Knoten der Reihe nach durchgeführt werden. Der Ablauf einer Messung ist stets gleich. Nach einer Kontaktaufnahme mit einem Domain-Knoten aus der Liste wird eine Messung gestartet, das Experiment durchgeführt, die Messung gestoppt und der Kontakt wieder abgebrochen. Nach einer einstellbaren Wartezeit wird ein neues Messexperiment mit dem nächsten Domain-Knoten gestartet. Die Liste der Domain-Knoten wird dabei zyklisch abgearbeitet. Die Ergebnisse der Messungen werden pro Knoten jeweils in einer Zeitreihe aufgenommen, die Basis für eine Vorhersage des nächsten erwarteten Messwertes ist. Der Vorhersagewert wird auf der Karte verzeichnet. Der gewählte Ansatz der Implementierung des Kartenmoduls [Sch02] lehnt sich an die Konzepte des *Network Weather Service* [WSP97] an.

Die Messungen werden von Sensoren durchgeführt. Sensoren sind Programme, die über Netzwerkverbindungen kommunizieren, um Messungen vorzunehmen. Für jede Art von Messexperiment muss ein Sensor des Kartenmoduls auf der Plattform gestartet werden. Dieser tauscht mit dem jeweiligen Sensor des Experimentpartners Pakete zur Messung aus. Für die Messexperimente sind folgende Sensoren im System integriert:

- Latenzsensor

  Misst die Round Trip Time (RTT) eines minimalen Paketes zu einer Agentenplattform (Implementierung siehe Anhang A.2).

- Transferratensensor

  Misst die aktuell verfügbare Transferrate zu einer Plattform durch Versenden eines Paketes definierter Größe.

- Verfügbarkeitssensor

  Prüft die Verfügbarkeit einer Plattform durch einen Kommunikationsversuch.

- Migrationssensor

  Ermittelt die Zeit, die ein Standardagent zur Migration zu einer Plattform benötigt. Als ein Standardagent wird ein Agent mit einer durchschnittlichen Größe bezeichnet, die fix angegeben oder laufend über eine Statistik aller den Agentenserver besuchenden mobilen Agenten ermittelt wird.

Weitere Sensoren können problemlos in das System integriert werden. Eine vordefinierte Schablone für einen Sensor ist in Anhang A.2 angegeben.

Nachdem die lokale Plattform mit jedem Domain-Knoten der lokalen Domain mindestens ein Messexperiment durchgeführt hat, sind auf ihr die Verbindungsdaten zu jedem anderen Domain-Knoten verfügbar. Der Austausch der Knoteninformationen ist „unbemerkt“ geschehen. Das Transferratenexperiment wird dazu benutzt, Nutzdaten im System zu verbreiten. Die Angaben zu Agentenservern (Knoteninformationen) sowie die bereits ermittelten Verbindungsdaten werden in die zu übertragenden Pakete verpackt. Die lokale Plattform hat dadurch während der Messung der Transferraten ihre eigenen Knoteninformationen und die bisher gemessenen Verbindungsinformationen in der Domain propagiert. Knoten- und Verbindungsinformationen anderer Domain-Knoten wurden der lokalen Plattform ebenfalls über deren Transferratenexperimente zugeschickt. Diese Informationen wurden in die Karte aufgenommen. Ergebnis des Datenaustauschs ist eine vollständige Karte der lokalen Domain mit Verbindungs- und Knoteninformationen, die auf jedem Domain-Knoten lokal verfügbar ist.

Aus der Sicht eines Domain-Knotens ergibt sich folgender Mechanismus zur Kartographierung der lokalen Domain:

- Ermittlung anderer Domain-Knoten der lokalen Domain,
- Sequentielle Messungen der Verbindungseigenschaften zu allen Domain-Knoten,
- Austausch der Knoteninformationen sowie der bereits ermittelten Verbindungsinformationen während des Transferratenexperiments .

Verbindungskanten werden durch diese Vorgehensweise von beiden Seiten aus kartographiert. Das ist bei symmetrischen Verbindungen überflüssig. In dieser Arbeit wird allerdings von der Möglichkeit unsymmetrischer Verbindungen ausgegangen.

Der geschilderte Mechanismus zur Erstellung der Karte wiederholt sich auf der Ebene der Domain-Manager. Der Domain-Manager erfragt bei dem Domain Master eine Liste der benachbarten Domains und führt in größeren Zeitabständen Messungen mit den ihm

bekannten Domain-Managern durch. Dadurch werden Verbindungsdaten zwischen Domains kartographiert und Informationen über die entfernten Domains ausgetauscht. Die Domaininformationen, die vom Domain-Manager erstellt werden, ergeben sich auf Grund der lokalen Domainkarte, deren Inhalte zusammengefasst und auf das Wesentliche komprimiert werden.

In jeder einzelnen Domain entsteht als Resultat der Messungen eine dynamische Karte (*Dynamic Domain Map*). Diese Karte wird durch den oben beschriebenen Verteilungsmechanismus auf jeden Knoten der Domain übertragen. Diese Karte enthält einen lokalen, detaillierten Teil der eigenen Domain sowie einen Teil mit zusammengefassten Informationen über fremde Domains.

Zum Zugriff auf die Karte wurde eine geeignete Schnittstelle definiert (siehe Anhang A.3), die von mobilen Agenten und anderen *ProNav*-Komponenten genutzt werden kann.

4.2.3 Realisierung

Das Kartenmodul wurde mit Hilfe von stationären und mobilen Agenten umgesetzt [Sch02] und greift nicht in die Implementierung des Agentensystems ein. Dadurch ist dieses Modul nicht TRACY-spezifisch, eine Portierung auf andere Agentensysteme ist möglich und beschränkt sich im Wesentlichen auf eine Anpassung der Agenten. Abbildung 4.3 stellt die Struktur des Moduls dar.

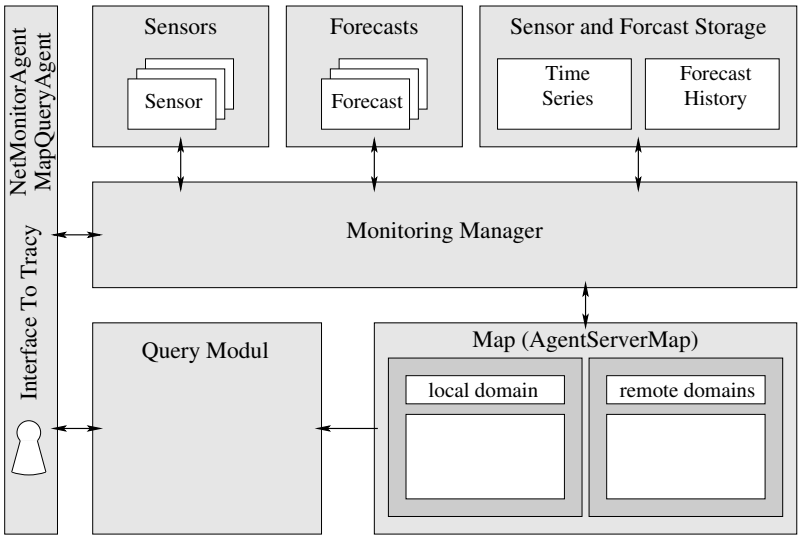


Abbildung 4.3: Struktur des Kartenmoduls

Die zentrale Komponente bildet der *Monitoring Manager*, der Messungen koordiniert und

organisiert. Die wichtigsten Aufgaben dieser Komponente sind:

- Bestimmung des nächsten Experimentziels für einen Sensor,
- Eintragen eines neuen Messwertes auf der lokalen Karte,
- Erzeugung von Paketen mit Dateninhalt zum Austausch von Informationen,
- Eintragen von Informationen aus Paketen fremder Agentenserver auf der Karte,
- Hinzufügen neuer und Entfernen alter Agentenserver von der Karte,
- Überwachen der Komponenten und Logging.

Ein *Sensor* ist ein nebenläufiger Prozess zur Messung von Verbindungseigenschaften. Seinen jeweiligen Messpartner erhält er über den Monitoring Manager. Für eine Messung kommuniziert ein Sensor über einen Socket mit dem Sensor des gleichen Typs auf der Zielplattform. Im Kartenmodul wurde ein generischer Sensor (siehe Anhang A.2) definiert, der zur einfacheren Kommunikation Methoden zum Aufbau einer Verbindung, zum Senden und Empfangen und zum Verbindungsabbau realisiert. Der Programmierer eines speziellen Sensors muss sich damit nicht um Details, wie beispielsweise die Portauflösung, kümmern und kann sich auf die Umsetzung seiner Sensorlogik konzentrieren. Ist eine Messung abgeschlossen, wird das Ergebnis dem Monitoring Manager übergeben, welcher den gemessenen Wert in einer *Time Serie* (Zeitreihe) einträgt. Nun legt sich der Sensor für eine definierbare Zeit schlafen, um anschließend den Messprozess zu wiederholen.

Die *Forecasts* (Vorhersagemodule) ermitteln auf Basis einer vom Manager übergebenen Zeitreihe eine Vorhersage für den nächsten erwarteten Wert. Dabei berechnen alle beim Manager registrierten Vorhersagemodule einen Wert, der jeweils in der *Forecast History* abgespeichert wird. Der Wert des Vorhersagemoduls, das im vorherigen Lauf den besten Wert berechnet hatte (und wahrscheinlich wieder den besten Wert ermittelt hat), wird auf der Karte verzeichnet. Die Ermittlung des besten Vorhersagemoduls erfolgt mit Hilfe des vom Sensor aktuell gemessenen Wertes und den für diese Messung vorhergesagten Werten, die in der Forecast History gespeichert wurden.

Folgende Vorhersagemodule werden aktuell benutzt:

- Last

  Der aktuell gemessene Wert wird als neuer Vorhersagewert genommen – ein einfaches, aber durchaus gutes Verfahren. Starke Veränderungen der Werte werden schnell erkannt.

- Average

  Das klassische Verfahren: Das arithmetische Mittel der letzten gemessenen Werte bildet den neuen Vorhersagewert.

- Median

  Der Median, der numerische Mittelwert der nach Werten sortierten Zeitreihe, bildet den Vorhersagewert. Einzelne Spitzenwerte bleiben bei diesem Verfahren unberücksichtigt und werden ausgeblendet.

Die Datenstruktur der Karte (*AgentServerMap*) teilt sich in einen lokalen Kartenteil, der *local domain* (lokale Domain), und einen Kartenteil mit Informationen über entfernte Domains (*remote domains*). Die Struktur dieser beiden Teile ist jedoch gleich, da auf der Karte (und damit auf beiden Teilen) grundsätzlich nur zwei Kategorien von Daten abgelegt werden: Knoteninformationen und Kanten- bzw. Verbindungsinformationen (siehe Abb. 4.4). Die Knoteninformationen beinhalten Daten zu Agentenplattformen, wie angebotene Dienste, unterstützte Protokolle oder weitere beliebige Informationen über den Knoten. Verbindungsdaten, wie Transferrate und Latenzzeit, werden als Kanteninformationen in der Karte aufgenommen.

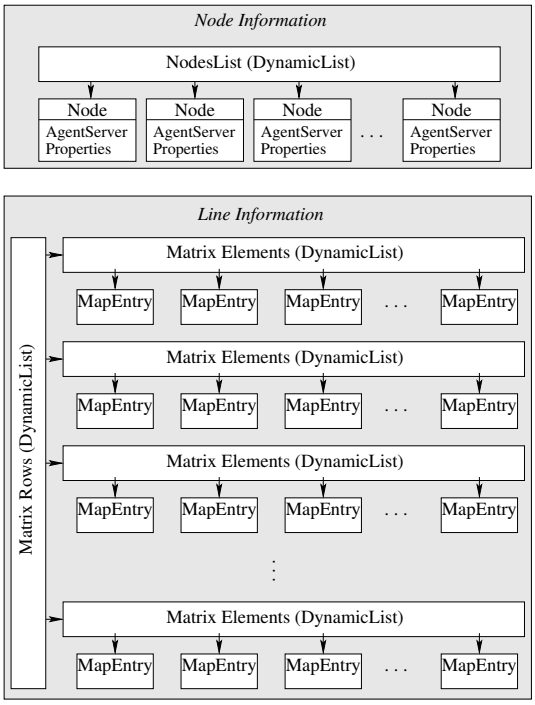


Abbildung 4.4: Struktur der Karte

Die Knoteninformationen werden in der Karte als eine Liste von Agentenservern (*NodesList*) mit einer Menge von Eigenschaftseinträgen (*Properties*), welche die Dienste, Protokolle etc. bezeichnen, aufgenommen (siehe Abb. 4.4 Node Information). Sowohl Agentenserver als auch lokale stationäre oder mobile Agenten können in den jeweiligen Listen Einträge hinzufügen oder entfernen. Gegebenenfalls müssen entsprechende Automatismen im System integriert werden, die für die Bekanntmachung entsprechender Informationen auf der Liste sorgen (z. B. automatische Registrierung neuer Dienste). Für die Informationen auf der Property-Liste sollten Konventionen festgelegt oder Standards verwendet werden, wie beispielsweise für die Beschreibung von Diensten mit WSDL.

Für die Verbindungsinformationen ergibt sich auf Grund der logischen Vernetzung der Agentenserver eine Verbindungsmatrix, dargestellt im unteren Teil der Abbildung 4.4 auf der vorherigen Seite (Line Information). Eine Position in der $n \times n$ Matrix (*MapEntry*) steht für eine bestimmte Richtung einer Verbindung zwischen zwei Agentenservern. An dieser Position werden die entsprechenden Eigenschaften (ermittelte Vorhersagen) dieser Verbindung eingetragen. Pro Sensor entsteht ein Eigenschaftswert eines Eintrags. Die Hauptdiagonale beinhaltet keine Werte, da Messungen eines Agentenservers mit sich selbst zu keiner sinnvollen Aussage führen und deshalb nicht durchgeführt werden. Die Matrix enthält damit $(n-1) * n$ eingetragene, gerichtete Verbindungen. Ein Eintrag ist leer, wenn keine Verbindung zwischen den Agentenservern aufgebaut werden konnte. Die Größe der Matrix wird zur Laufzeit automatisch angepasst, wenn neue Agentenserver hinzukommen oder alte entfallen. Dabei werden nicht mehr wahrgenommene Agentenserver nicht sofort, sondern erst nach einer bestimmten Zeitspanne entfernt.

Im Kartenteil der entfernten Domains (remote domain) steht ein Eintrag in der Liste der Agentenserver (*Node* einer *NodesList*) stellvertretend für eine Domain. Der eingetragene Agentenserver ist der Domain-Manager der repräsentierten Domain. Die Eigenschaftswerte, die für solch eine Domain eingetragen werden, entstehen durch das Zusammenfassen von Kartendaten (lokaler Teil) auf dem entsprechenden Domain-Manager. Dabei werden die Knoteninformationen der lokalen Domain einfach durch eine Mengenvereinigung der Knoteneigenschaften aller Domain-Knoten zusammengefasst. Resultat ist eine von Duplikaten befreite Liste von Eigenschaftswerten, die die Fähigkeiten der gesamten entfernten Domain widerspiegeln. Des Weiteren werden zu dieser Liste Verbindungsdaten der Domain als Eigenschaftswerte hinzugefügt. Zur Ermittlung dieser Werte werden die Einträge der Verbindungsdaten vom Domain-Manager zu seinen Domain-Knoten zu Grunde gelegt. Es werden Eckdaten ermittelt, die repräsentativ für die Verbindungen in der Domain stehen: Minimum, Maximum, Durchschnitt und Median über alle Verbindungen vom Domain-Manager zu seinen Domain-Knoten. Dieser Mechanismus bringt die Unschärfe des Fish-Eye-View in die Karte: Die detaillierte Karte einer lokalen Domain schrumpft auf einen einzigen Karteneintrag einer entfernten Domain. Gleichzeitig entstehen durch diese Karteneinträge die überlappenden Schnittstellen auf dem Puzzleteil der (für einen Agentenserver nicht sichtbaren) Weltkarte.

Die Verbindungsmatrix im Kartenteil der entfernten Domains beinhaltet Verbindungseigenschaften, die zwischen den Domain-Managern gemessen wurden. Die Messungen werden, wie bei der lokalen Domain beschrieben, durchgeführt, allerdings ist das Zeitintervall der Messungen größer. Die Verbindungsmatrix enthält üblicherweise auch mehrere Leereinträge, da sich nicht alle in einer Domain bekannten fremden Domain-Manager untereinander kennen.

Das *Query Modul* ist die Abfrageschnittstelle, die gezielte Informationen, einzelne Kanten- oder Knoteninformationen, von der Karte nach außen geben kann. Möglich ist auch, ganze Matrixzeilen auszulesen sowie die ausgelesenen Informationen mit neuen Suchanfragen logisch (AND, OR, XOR) zu verknüpfen. Als zusätzliche Schnittstelle für Kartenanfragen von Agenten existiert ein stationärer Agent, der *MapQueryAgent*, welcher die Funktionalitäten der Query Module den Agenten zur Verfügung stellt.

Die Schnittstelle zum Agentensystem bildet der stationäre *NetMonitorAgent*. Durch den Start dieses Agenten wird das Kartenmodul gestartet. Der NetMonitorAgent kommuniziert mit dem Domain Information Agent (Domain Service), um den Manager der Domain zu erfragen und damit die Liste der Domain-Knoten vom Manager zu erhalten, die den Startpunkt für die Erstellung der Karte bildet. Simple Anfragen an die Karte können über entsprechende Nachrichten an diesen Agenten gestellt werden.

4.3 Routenplaner

Der Routenplaner ermittelt auf Wunsch eines Agenten zu seiner Liste von Agentenserverzielen einen effizienten Weg durch das heterogene Netz. Dabei benutzt er zur Planung die Daten, die auf der Webkarte zur Verfügung stehen (siehe Abb. 4.5).

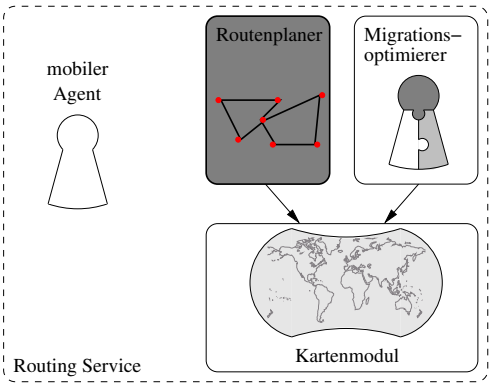


Abbildung 4.5: Routenplaner im Routing Service *ProNav*

4.3.1 Allgemeine Beschreibung und Analyse

Möchte ein mobiler Agent eine Liste von n Agentenservern besuchen, so gibt es $n!$ Möglichkeiten bei der Wahl der Reihenfolge, in welcher der Agent die Server besuchen kann. Dabei unterscheiden sich die aus den Reihenfolgen entstehenden Reiserouten im Wesentlichen durch die Übertragungszeiten zwischen den Agentenservern im Netzwerk. Eine zufällig (schlecht) gewählte Reiseroute kann zu einer deutlich längeren Reise des Agenten führen. Ziel des Routenplaners ist, die Summe der Übertragungszeiten des Agenten zu minimieren, wodurch der Agent seine Reise und somit seine Aufgabe so schnell wie möglich durchführen kann. Um eine optimale Lösung zu erhalten, müssten alle $n!$ Möglichkeiten getestet werden. Das ist für größere n praktisch nicht durchführbar, da die Rechenzeit exponentiell ansteigt. Der Agent steht vor dem NP-vollständigen *Traveling Salesman Problem* (TSP). Auf Grund der Häufigkeit dieses Problems existieren heuristische Ansätze [Lin65, JM97], die zu erstaunlich guten Lösungen des Problems führen. Favorisiert wird eine Lösung, die in kurzer Zeit ein nahezu optimales Ergebnis liefert. Diese Vorstellung ist kontradiktorisch, denn oft gilt: Je besser die Lösung, desto länger die Berechnungszeit bzw. desto höher der Berechnungsaufwand. Um den Routenplaner optimal zu gestalten, galt es, mehrere Algorithmen zu testen, um eine Aussage über die Eignung in diesem Anwendungskontext zu erhalten.

Das Gebiet des TSP, das umfassend in [GP02] dargestellt wird, gliedert sich in das symmetrische TSP (STSP) [JM02] und das asymmetrische TSP (ATSP) [JGM+02]. Für diese Gebiete existieren eine Vielzahl von Algorithmen aus diversen Bereichen, wie lokale Suche, genetische Algorithmen, Simulated Annealing und neuronale Netze. Für die Wahl eines für das Problem der Routenplanung im Agentennetzwerk geeigneten TSP-Algorithmus gilt es zu entscheiden, wie die Struktur der Eingabe für den Algorithmus aussieht, d. h. sind die Wege zwischen den Orten symmetrisch oder asymmetrisch. Übersetzt auf das Anwendungsgebiet bedeutet dies, ob die Übertragungszeit eines Agenten von Agentenserver A nach Agentenserver B dieselbe ist wie in die entgegengesetzte Richtung. Experimentell hat sich gezeigt, dass das meist nicht der Fall ist (z. B. DSL). Daher wurde nach einem passenden Algorithmus für asymmetrische Eingaben gesucht.

Die Qualität einer Routenplanung hängt entscheidend davon ab, wie viel Zeit für die Routenberechnung durch den TSP-Algorithmus verwendet wird. Sie ist Funktion

1. der lokalen Rechenleistung,
2. der Rechnerauslastung,
3. dem Unterschied zwischen optimaler und zufälliger Route

   (im Wesentlichen abhängig von den Unterschieden der einzelnen Übertragungszeiten der Verbindungen).

Damit sich die Routenplanung für einen Agenten lohnt, muss ein optimales Verhältnis zwischen Berechnungszeit und der (ermittelten) Zeit, die ein Agent für die Migration benötigt, gefunden werden. Letztendlich muss die Summe aus Berechnungszeit und Migrationszeit des Agenten minimiert werden. Da dies vom Berechnungszeitpunkt und dem Ort (Rechenleistung des lokalen Rechners) abhängt, kann je nach aktueller Situation ein passender TSP-Algorithmus ausgewählt oder ein Algorithmus entworfen werden, der anpassungsfähig ist.

Durch diese Feststellung konnten einige Ansätze zur Lösung des TSP bei den Untersuchungen ausgeschlossen werden, da sie entweder zu viel Rechenzeit benötigen (genetische Algorithmen, Simulated Annealing) oder in dynamischen Umgebungen schlechte Ergebnisse liefern (neuronale Netze). Im Gegensatz dazu sind lokale Optimierungen vielversprechend.

Die Basis für einen TSP-Algorithmus ist ein (gerichteter) gewichteter Graph. Die Knoten stellen die Orte und die Kanten die Wege zwischen diesen Orten dar. Die Gewichte drücken Entfernungen, die Länge der Wege (beispielsweise die Übertragungszeit eines Agenten oder die verfügbare Datenübertragungsrate) zwischen den Orten aus. Ein derartiger Graph ist in einer Matrix darstellbar, wobei die Einträge die Gewichte der Wege sind. Die Koordinaten geben die Richtung eines Weges an. Eine entsprechende Matrix ist im Kartenmodul zu finden. Allerdings bedarf die Matrix noch einer Überarbeitung, bevor diese zur Berechnung einer günstigen Route mittels eines TSP-Algorithmus herangezogen werden kann. Einerseits muss aus den Einträgen ein Kantengewicht für die jeweilige Verbindung berechnet werden (Berechnung wird in Kap. 4.3.3 diskutiert). Andererseits muss die Matrix in eine Standardform gebracht werden, die der Dreiecksungleichung genügt, d. h. der Pfad zwischen je zwei Knoten ist der kürzeste Weg (geringstes Gesamtgewicht). Damit sind keine „Abkürzungen“ über einen anderen Knoten möglich. Die angepasste Matrix entspricht einem gerichteten Graph, der das Agentennetzwerk darstellt und dessen Kanten Gewichte in Form von Matrixeinträgen besitzen.

4.3.2 Funktionsweise

Eine Liste von Zielplattformen bildet die Eingabe für den Routenplaner. Daraufhin wird das Kartenmodul nach Verbindungsinformationen befragt, um die oben erwähnte Standardmatrix zu berechnen. Ein TSP-Algorithmus berechnet daraufhin einen günstigen Weg durch den generierten Graph. Das Ergebnis wird dem Agenten mitgeteilt. Im Detail ergeben sich somit folgende Schritte:

1. Aufbau der Verbindungsmatrix und Berechnung der Gewichte

   Ergebnis: Gewichteter, gerichteter Graph für alle bekannten Agentenserver.

2. Ermittlung der kürzesten Wege zwischen den Zielservern

Ergebnis: Gewichteter, gerichteter Graph mit kürzesten Wegen für die Zielserver.

3. Anwendung eines TSP-Algorithmus

   Ergebnis: Reiseroute für den Agenten.

4.3.3 Realisierung

Der Routenplaner wird mit Hilfe eines stationären Agenten, dem *RoutenPlanerAgent*, in *ProNav* integriert. Dieser Agent ist Ansprechpartner für mobile Agenten, die den Routenplanerdienst in Anspruch nehmen wollen. Die wesentlichen Komponenten des Moduls, dargestellt in Abbildung 4.6, ergeben sich auf Grund der geschilderten Funktionsweise und den damit verbundenen Aufgaben:

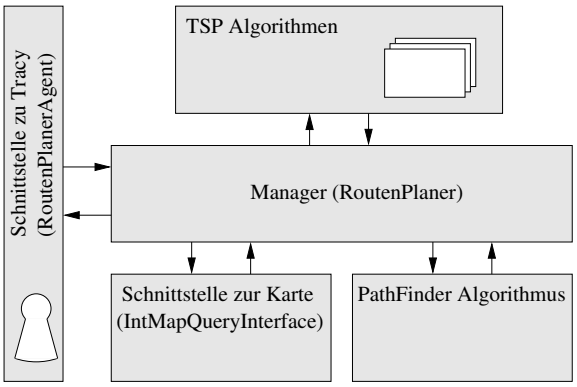


Abbildung 4.6: Struktur des Routenplaners

- Der *RoutenPlanerAgent* als Schnittstelle zu TRACY,

  Nimmt Anfragen zur Routenplanung von anderen Agenten entgegen. Eine Nachricht an diesen Agenten kann eine Liste von Agentenservern und eventuelle spezielle Anforderungen enthalten, wie beispielsweise die Übertragung mittels eines speziellen Kommunikationsprotokolls.

- Der *RoutenPlaner*, der die Abläufe koordiniert,

  Übernimmt die Steuerung der einzelnen Komponenten des Moduls und regelt zwischen diesen den Datenaustausch.

- Das *IntMapQueryInterface* als Schnittstelle zur Karte,

  Baut aus der lokalen Karte eine Verbindungsmatrix auf, welche speziell für den PathFinder geeignet ist. Die erforderlichen Daten werden vom Kartenmodul (MapQueryInterface) abgefragt. Es entsteht der gewichtete, gerichtete Graph.

- Der *PathFinder*, welcher die kürzesten Wege zwischen den einzelnen Zielservern berechnet,

  Der PathFinder berechnet mit Hilfe der Verbindungsmatrix des IntMapQueryInterface die kürzesten Wege zwischen den Agentenservern, die von einem Agenten als Zielserver angegeben wurden. Ergebnis ist wieder eine Verbindungsmatrix, die allerdings nur noch die Zielserver enthält und für die die Dreiecksungleichung gilt. Ein Eintrag in der Matrix enthält nun neben einem Gewicht eventuell auch eine Route, falls eine Verbindung zwischen zwei Servern über dritte Server kürzer / schneller ist als eine direkte Verbindung. Die entstandene Matrix bildet die Eingabe für einen TSP-Algorithmus.

- Verschiedene *TSP-Algorithmen* zur Bestimmung einer möglichst optimalen Reiseroute.

  Diese Komponente ermittelt auf Basis einer Verbindungsmatrix einen günstigen Weg durch das Agentennetzwerk. Dazu kann ein voreingestellter Algorithmus oder ein durch den Agenten ausgewählter TSP-Algorithmus benutzt werden (eventuell ist eine Kombination mehrerer Algorithmen erforderlich).

Die implementierte Lösung [Sta03] des Routenplaners kann jedoch nur einen Weg berechnen, wenn für die Ziele entsprechende Daten auf der Karte vorhanden sind. Er muss des Weiteren bei der Berechnung der Gewichte auf gleichwertige Kartendaten (gleiche Sorte) zurückgreifen können, damit für die Berechnung die gleiche Berechnungsvariante benutzt werden kann.

Es existieren verschiedene Berechnungsvarianten, um die Verbindungsdaten der Karte zu Gewichten im Graphen zu konvertieren. Ein simpler Ansatz ist die Reziprok-Bildung der Transferraten, die in KByte pro Sekunde auf der Karte verzeichnet sind. Als Gewicht ergibt sich damit die Zeit (in Sekunden), welche pro zu übertragenden KByte Daten erforderlich ist. Als Variante kann bei den Transferzeiten die Latenzzeit des Antwortpaketes vor der Berechnung abgezogen werden. Die Genauigkeit der ermittelten Gewichte kann dadurch verfeinert werden.

Alternativ oder für den Fall, dass diese Kartendaten nicht vorhanden sind, können die Daten des Migrationssensors zur Gewichtung benutzt werden. Die Daten des Sensors entsprechen der Zeit, die ein Standardagent zur Migration benötigt (dient als Referenzwert). Eine Konvertierung ist dabei nicht notwendig, da das Resultat einer Migrationsmessung, das auf der Karte in Sekunden geführt wird, bereits als Gewicht interpretiert werden kann.

Zur Auswahl eines geeigneten TSP-Algorithmus sollen an dieser Stelle einige Algorithmen vorgestellt werden. Wie bereits erwähnt, sind lokale Optimierungen vielversprechend. Dabei sind ATSP-Algorithmen teilweise Modifikationen von STSP-Algorithmen. Aus diesem Grund werden auch Algorithmen für STSP betrachtet. Für eine nahezu voll-

ständige und detailliertere Beschreibung von TSP-Algorithmen sei auf [GP02] von Gregory Gutin und Abraham P. Punnen verwiesen.

Für die lokalen Optimierungen ist eine bereits ermittelte Route notwendig. An dieser Stelle sollen kurz die Algorithmen vorgestellt werden, die eine initiale Tour (Rundreise) berechnen. Die wohl bekannteste heuristische Konstruktion einer Tour wird mit dem *Nearest Neighbour* Algorithmus berechnet. Hierbei wird nach einer zufälligen Wahl eines Startpunktes immer zum nächstgelegenen Ort weitergegangen bis alle Orte besucht sind. Die Laufzeit ist $O(n^2)$. Eine andere Möglichkeit mit gleicher Laufzeit bietet der Aufbau eines *minimalen Spannbaumes*. Die Tour ergibt sich durch die Traversierung des Baumes. Es lässt sich sogar beweisen, dass die erzielte Rundreise maximal doppelt so lang wie der optimale Weg ist (vorausgesetzt, die Dreiecksungleichung gilt).

Eine etwas andere Herangehensweise ist das Zuordnen von Knoten zu anderen Knoten mit dem Ziel, möglichst die Summe der Kanten zu minimieren (*Assignmentproblem*). Ergebnis ist das so genannte *minimal Matching*. Dies kann eine optimale Route sein, meistens entstehen jedoch mehrere unverbundene Kreise. Das Assignmentproblem ist in $O(n^3)$ lösbar. Diese Lösung ist eine untere Schranke für die Länge der optimalen Rundreise. Aufbauend auf dem minimal Matching gibt es eine Reihe von Algorithmen, die eine Tour aus dem Ergebnis, den Kreisen, konstruieren. Beispielsweise verbindet der *Patch*-Algorithmus in jedem Schritt je zwei Kreise nach bestimmten Auswahlkriterien miteinander, so dass der neu entstandene Kreis eine minimale Länge besitzt. Sind alle Kreise miteinander verbunden, ist eine „recht gute“ Tour entstanden.

Auf Basis einer initialen Tour, die durch einen der beschriebenen Algorithmen ermittelt wird, können nun lokale Optimierungsverfahren angewendet werden, um die Tour zu verbessern. Ein Beispiel ist der *2-Opt*-Algorithmus, der zwei Kanten im Kreis austauscht, um eine kürzere Tour zu finden (siehe Abb. 4.7). Nach dem gleichen Prinzip arbeitet der

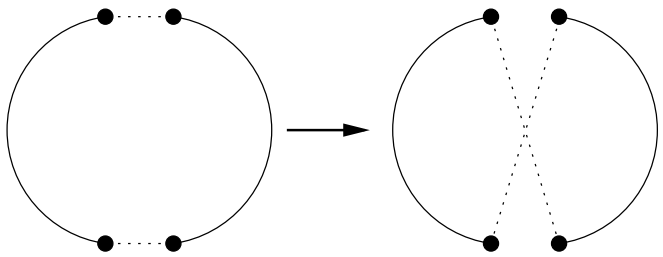


Abbildung 4.7: Arbeitsweise des 2-Opt-Algorithmus

3-Opt-Algorithmus, bei dem der Kreis an drei Stellen zerschnitten wird. Dadurch entstehen mehr Möglichkeiten des Zusammenfügens der Routenteile und damit auch mehr Chancen, einen optimalen Weg zu finden. Beim Zusammensetzen der Routenteile wird die in Bezug auf die Länge optimale Variante gewählt.

Die *Lin-Kernighan-Heuristik* tauscht mehrmals zwei Kanten in einem (Super-) Schritt, wobei ein Tauschen von Kanten zur Verschlechterung führen kann, aber das Ergebnis eines Schrittes eine Verbesserung mit sich bringt. Die Auswahl geeigneter Superschritte bildet dabei die Schwierigkeit. Bei guter Implementierung liefert die Lin-Kernighan-Heuristik sehr gute Ergebnisse in akzeptabler Zeit (für STSP).

Bei ATSP ergeben sich Probleme mit den Schritten der lokalen Optimierung, die Anfangs- und Endpunkt der Route vertauschen (wie z. B. beim 2-Opt). Extra Rechenzeit muss aufgewendet werden, um die Länge der gedrehten Teilstrecke zu berechnen. Meist lohnt ein Drehen einer Teilstrecke nicht (da dieser Weg durch vorangegangene Berechnungen entstanden ist) und führt zur Verschlechterung. Adaptionen der STSP-Algorithmen verwenden daher nur den Teil der möglichen Schritte, die keine Teilstrecken umdrehen. Das führt generell zu einer Verschlechterung der Ergebnisse, falls diese Algorithmen doch auf ein symmetrisches TSP stoßen.

Des Weiteren können lokale Optimierungsverfahren iteriert werden. Aus den berechneten Lösungen wird die Beste ausgewählt. Schon nach wenigen Iterationen ist meist eine bessere Lösung erreicht. Das Schema sieht dabei folgendermaßen aus:

1. Erzeuge eine Startroute S
2. Verbessere S durch ein lokales Optimierungsverfahren
3. Solange nicht abgebrochen wird
   (a) Erzeuge eine zufällige Mutation S' aus S
   (b) Verbessere S' durch ein lokales Optimierungsverfahren
   (c) Ist das S' kleiner als S, dann ersetze S durch S'

Bei der Realisierung wurde im Routenplaner eine Kombination aus Patch-Algorithmus und der iterierten Version des 3-Opt gewählt. Dadurch werden gute Ergebnisse für asymmetrische Verbindungsmatrizen erreicht.

Die Wahl des Domainkonzepts begünstigt den Berechnungsaufwand, da die Anzahl der Knoten auf der Karte durch die Domains reduziert werden. Demgegenüber würde eine Berechnung auf Basis einer Weltkarte eventuell andere Verfahren notwendig machen, da schon $O(n^2)$-Probleme nicht in adäquater Zeit gelöst werden können.

4.4 Migrationsoptimierer

Dieser Agentensystem-spezifische Teil optimiert auf Basis einer fertigen Reiseroute die Details der Migration. Beispielsweise können Zeit oder Kosten gespart werden, da oftmals nur Teile eines Agenten auf entfernten Plattformen zur Ausführung benötigt werden.

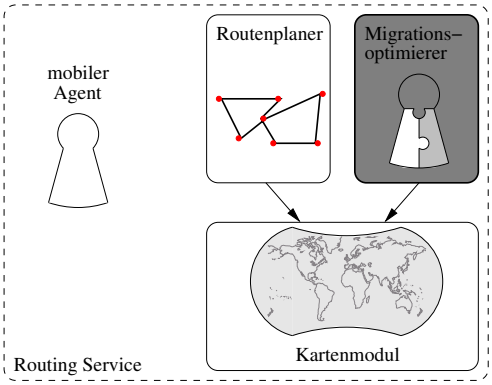


Abbildung 4.8: Migrationsoptimierer im Routing Service *ProNav*

4.4.1 Allgemeine Beschreibung und Analyse

Der Migrationsoptimierer ist eine TRACY-spezifische Komponente von *ProNav*. Er nutzt die Möglichkeit in TRACY, aus verschiedenen Migrationsstrategien zu wählen, um die Übertragung optimal in Bezug auf die Netzwerklast zu gestalten. Migrationsstrategien werden nach dem Zeitpunkt der Übertragung des Bytecodes (siehe Kapitel 2.1) unterschieden: entweder zeitgleich mit dem serialisierten Agenten oder zu einem späteren Zeitpunkt, nachdem der serialisierte Agent übertragen wurde. Verfeinerungen der beiden Strategien ergeben sich durch die Variante, den Bytecode partiell oder vollständig zu übertragen. In Abbildung 4.9 auf der nächsten Seite sind die sich aus der Beschreibung ergebenden Möglichkeiten in einer Matrix dargestellt.

Bei einer *Push*-Strategie wird der Bytecode zusammen mit dem serialisierten Agenten übertragen. Der Bytecode ist somit vor Beginn der Ausführung des Agenten auf der entfernten Plattform verfügbar. Im Gegensatz dazu wird bei einer *Pull*-Strategie der Bytecode während der Ausführung des Agenten auf der entfernten Plattform bei Bedarf nachgeladen. Das Nachladen ist nur möglich, wenn eine Verbindung von der entfernten Plattform zu einem Server aufgebaut werden kann, auf dem der benötigte Bytecode zur Verfügung steht. Ist dies nicht möglich, kann der Agent nicht weiter ausgeführt werden.

Des Weiteren kann eine Einteilung nach der Granularität des Bytecodes getroffen werden. Wird der komplette Bytecode übertragen, ist die Ausführung des Agenten gesichert (Pull-All oder Push-All). Werden nur Teile (z. B. einzelne Klassen) übertragen (Pull-Per-Unit oder Push-Units), kann weiteres Nachladen erforderlich sein. Eine nachhaltige Verbindung zu einem Agentenserver, der die fehlenden Codeteile zur Verfügung stellt, ist erforderlich, um die Ausführung des Agenten nicht zu behindern. Bei der Push-Strategie kann weiterhin unterschieden werden, wohin der Bytecode versendet wird: Nur zur nächsten Plattform (Push-Units-To-Next oder Push-All-To-Next) oder zu allen Zielen der Reiseroute (Push-Units-To-All oder Push-All-To-All). Werden dabei nur Teile des Code ver-

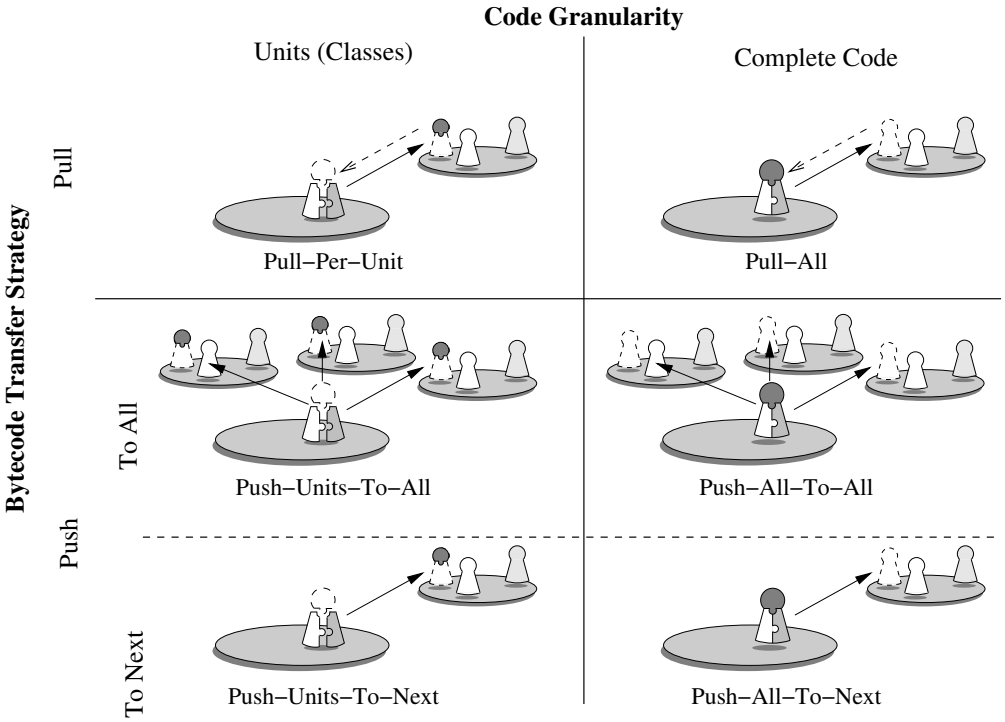


Abbildung 4.9: Übersicht Migrationsstrategien

schickt, kann ein Nachladen erforderlich sein – eine Kombination der Push- und Pull-Strategie ist das Ergebnis.

Die Wahl einer Strategie hat Einfluss auf die Belastung des Netzwerks. Entsprechende Untersuchungen finden sich in [BER01b]. Der Migrationsoptimierer soll für eine gegebene Route und einen Agenten eine geeignete Migrationsstrategie vorschlagen. Bei der Bestimmung dieser Migrationsstrategie werden unterschiedliche Strategien auf Basis der Kartendaten durchgerechnet. Die Berechnungen basieren auf dem in [Bra03] entwickelten mathematischen Netzwerkmodell. Dieses Modell ermöglicht die Berechnung der Netzwerklast und der Übertragungszeit eines Agenten beim Start von der Heimatplattform, bei der Übertragung zwischen den Plattformen der Route und bei der Rückkehr zur Heimatplattform. Das Modell berücksichtigt bei der Berechnung die Größe des Agenten (serialisierter Agent und Bytecode), die auf der Reise gesammelten Daten sowie die Verbindungsqualitäten (aktuelle Transferraten und Latenzzeiten). Des Weiteren berücksichtigt es, mit welcher Wahrscheinlichkeit ein Codeteil des Agenten zur Ausführung auf einem entfernten Agentenserver benötigt wird.

Für die Implementierung des Migrationsoptimierers [Sch03] wurde das Netzwerkmodell noch einmal aufgegriffen, angepasst und erweitert. Das Wachstum der angesammelten Daten während einer Reise, welches bisher als konstant angesehen wurde, ist nun varia-

bel. In der Erweiterung können des Weiteren die gesammelten Daten vom Agent während der Reise zur Heimatplattform gesendet werden und belasten damit den Agenten nicht unnötig auf seiner weiteren Reise. In das Netzwerkmodell wurden neue Berechnungen eingeführt, die so genannte *Code Server* und *Mirror Server* mit einbeziehen. Ein Code Server ist ein Server, der den Bytecode eines Agenten zur Verfügung stellt und auf Nachladeanfragen entfernter Plattformen die angeforderten Codeteile verschickt (dynamisches Nachladen bei der Pull-Strategie). Normalerweise wird dies von der Heimatplattform übernommen. Wenn diese allerdings nicht erreichbar ist, kann ein Code Server Code-Anfragen befriedigen.

Auf einem Mirror Server können Agenten gesammelte Daten ablegen und quasi „zwischenparken“. Wie bereits erwähnt, kann ein Agent die Daten während einer Reise bei Bedarf zum Heimatserver schicken. Wenn der Heimatserver allerdings nicht erreichbar oder die Verbindung zum Heimatserver relativ schlecht ist, kann ein Mirror Server diese Aufgabe übernehmen. Zu diesem Zweck kann der Mirror Server, wie auch der Code Server, von einem Agenten selbst initialisiert werden. Der Aufwand und die Vorteile sind mit dem erweiterten Netzwerkmodell berechenbar.

Auf die Auswahl einer Migrationsstrategie haben verschiedene Parameter Einfluss, je nachdem welche aktuelle Situation gerade vorherrscht. Das Netzwerkmodell hilft die aktuelle Situation einzuschätzen und die richtige Migrationsstrategie für einen Agenten zu ermitteln. Sofern möglich, kann sogar der Bytecode des Agenten für eine optimale Migration aufgeteilt werden. Für die Ermittlung der Strategie sind folgende aufgeführte Informationen notwendig, die zur Berechnung herangezogen werden müssen. Eine Einteilung dieser Informationen kann in folgende Gruppen vorgenommen werden:

- Verbindungseigenschaften

  Grundlage für eine vergleichende Berechnung der Migrationszeiten bei der Nutzung unterschiedlicher Strategien sind die Daten der Karte, insbesondere die Qualitäten der Verbindung (aktuelle Transferrate, Latenzzeit etc.).

- Knoteneigenschaften

  Besonders interessant ist hier die Wahrscheinlichkeit, mit der Code-Teile des Agenten bereits auf den Zielplattformen sind. Entsprechende Code-Teile müssen unter Umständen nicht verschickt werden.

- Agenteneigenschaften

  Informationen über die Größe des Bytecodes des Agenten spielen eine Rolle bei der Wahl einer Strategie. Interessant ist weiterhin, ob der Agent aus mehreren Teilen (Klassen) besteht, wie groß diese sind und mit welcher Wahrscheinlichkeit diese auf einer Zielplattform benötigt werden.

- Aufgabenspezifische Eigenschaften

  Bei der Reise des Agenten fallen für gewöhnlich Daten an, die mittransportiert werden. Wie das Aufkommen an Daten auf der Reise des Agenten ausfällt, beeinflusst die Strategiewahl. Ebenso ist die Reiseroute des Agenten von Bedeutung.

Die strategiebeeinflussenden Parameter sind sehr vielfältig und hier nur andeutungsweise dargestellt. Für eine ausführlichere Diskussion sei auf [Sch03] verwiesen.

4.4.2 Funktionsweise

Die Funktionsweise dieses Moduls unterscheidet sich auf Grund der Spezifika von TRACY deutlich von der Funktionsweise der bereits beschriebenen Module. Der Agent kontaktiert zwar ebenfalls einen stationären Agenten, allerdings nicht zur eigentlichen Nutzung des Optimierers, sondern zur Ermittlung migrationsrelevanter Daten (Kartendaten). Der Agent muss daraufhin einen Optimierer aufrufen, der sich um die Wahl der Migrationsstrategie und das Versenden des serialisierten Agenten und des Bytecodes, sofern notwendig, kümmert. Es ergibt sich also folgender Ablauf:

1. Übergabe der Route an den Agenten des Migrationsoptimierer,
2. Empfang und Speicherung der migrationsrelevanten Kartendaten im Agenten,
3. Aufruf der Optimierungsstrategie.

Die ersten zwei Schritte müssen auf jeden Fall auf der Startplattform wahrgenommen werden. Danach steht es dem Agenten frei, Aktualisierungen seiner routenspezifischen Daten vorzunehmen. Die Daten werden mit dem Agenten versendet und bei Aufruf des Optimierers (dritter Schritt) zur Berechnung verwendet. Der Optimierer muss auf jeder Plattform aufgerufen werden, damit der Agent seine optimale Strategie weiter verfolgen kann.

Es fällt weiterhin auf, dass ein Agent keine Wahlmöglichkeit zwischen den berechneten Migrationsstrategien hat. Die Wahl der Strategie wird vollständig dem Optimierer überlassen. Möchte dies ein Agent nicht, muss er völlig frei entscheiden (ohne Nutzung des Optimierers), welche Migrationsstrategie er benutzt. Die Ursache dafür liegt in den Spezifika des Agentensystems und wird im folgenden Kapitel weiter erläutert.

4.4.3 Realisierung

Ein großes Problem bei der Berechnung einer geeigneten Migrationsstrategie für einen Agenten stellt die Beschaffung der erforderlichen Daten dar. Die Daten für den Optimierungsprozess müssen teilweise aufwendig erarbeitet und berechnet werden. Einige dieser

Daten werden bereits von der Migrationskomponente in einer tieferen Schicht des Agentensystems berechnet. Der Agent selbst kann auf diese Daten nicht zugreifen. Um doppelte Berechnungen zu vermeiden, muss der Migrationsoptimierer in die tiefere Schicht des Agentensystems (Migrationskomponente Kalong in TRACY) integriert werden. Allerdings tritt in dieser Komponente wiederum ein Problem auf: Die Kartendaten können nicht ausgelesen werden. Diese Probleme führten dazu, dass der Migrationsoptimierer in zwei Komponenten zerlegt wurde:

- MapAgent

  Auslesen von Karteninformationen auf Ebene von TRACY

- Migration Optimizer

  Berechnung einer geeigneten Migrationsstrategie auf Ebene von Kalong

Die Struktur und die wichtigsten Komponenten des Migrationsoptimierers sind in der Abbildung 4.10 in die TRACY-Schichtenarchitektur eingezeichnet.

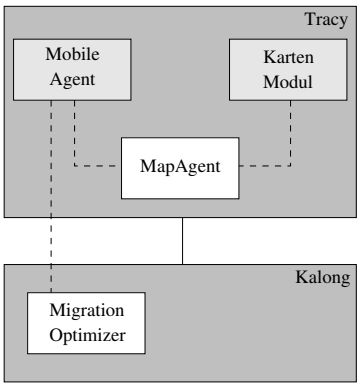


Abbildung 4.10: Struktur des Migrationsoptimierers

Der *MapAgent* übernimmt die Kommunikation mit den mobilen Agenten und übermittelt diesen die Karteninformationen vom Kartenmodul, die vom *Migration Optimizer* später zur Berechnung der geeigneten Migrationsstrategie benötigt werden. Der mobile Agent muss nun bei seiner Migration angeben, dass eine optimale Strategie ermittelt werden soll. Daraufhin wird der Agent vom System auf die Migration vorbereitet und der Optimierer angestoßen. Der Migration Optimizer ist in die Migrationskomponente Kalong eingebettet und nutzt dort implementierte Funktionalitäten.

Nachteilig ist hierbei jedoch, dass der Agent auf Ebene von Kalong bereits eingefroren ist und damit keine Handlungsfreiheit mehr hat, d. h. seine Autonomie bezüglich seiner Entscheidungsfreiheit wird dadurch eingeschränkt. Er kann nur entscheiden, ob er die

Optimierung nutzt oder von vorn herein selbst eine fixe (u. U. nicht optimale) Migrationsstrategie auswählt.

Der Optimierungsprozess im Optimizer gliedert sich in drei Schritte:

1. Initialisierung,
2. Optimierung,
3. Ausführung.

Während der Initialisierung werden alle zur Optimierung notwendigen Informationen gesammelt. Im Einzelnen handelt es sich dabei um folgende Informationen:

- Karteninformationen,
- Momentane Plattform, Heimatplattform und nächstes Migrationsziel (Position in der Route),
- Code-Teile (Units) des Agenten,
- Größe des Agenten und seiner Klassen,
- Daten des Agenten und deren Größe.

Im Optimierungsschritt wird die optimale Strategie ermittelt, die Möglichkeit der Initialisierung eines Code Server geprüft sowie die Option, gesammelte Daten auf dem Heimatserver oder auf einem Mirror Server abzulegen. Für die Berechnung der optimalen Strategie werden Berechnungen alternativer Strategien durchgeführt und miteinander verglichen.

Das Ausführen der Migrationsstrategie hängt davon ab, welche Strategie sich als optimal herausgestellt hat und welche Entscheidung bezüglich der Code und Mirror Server getroffen wurden. Folgende Operationen können durchgeführt werden:

- Versenden des serialisierten Agenten (Ausführungszustand und Daten)

  Der serialisierte Agent ist der eigentliche, reisende Agent. Er wird *immer* zur (nächsten) Zielplattform transferiert.

- Versenden von Bytecode (Code-Granularität: Klassen)

  Ob, welche und wohin Code(-Teile) des Agenten verschickt werden, hängt von der Migrationsstrategie, dem auf entfernten Plattformen vorhandenen Klassen und den voraussichtlich benötigten Klassen ab.

- Versenden von Daten zur Heimatplattform oder zu einem Mirror Server

  Je nachdem wie viele Daten bereits gesammelt wurden, kann über einen Transfer „nachgedacht“ werden. Abhängig von den Verbindungsqualitäten, können die Daten zur Heimatplattform oder zu einem Mirror Server verschickt werden. Eventuell ist das Initialisieren eines Mirror Servers erforderlich.

- Initialisierung eines Code Servers auf der aktuellen Plattform oder auf einem entfernten Server

  Um den Transfer von Code-Teilen (Bytecode) über schlechte Verbindungen zu vermeiden, können Code Server initialisiert werden. Ist nur bei Migrationsstrategien sinnvoll, die Code dynamisch nachladen.

Eine für einen mobilen Agenten geeignete Migrationsstrategie wird durch vergleichende Berechnungen ermittelt. Dazu werden zu den möglichen Migrationsstrategien die erwarteten Migrationszeiten berechnet und verglichen. Für die Vergleichsberechnungen wird das bereits erwähnte mathematische Modell herangezogen. Eine Berechnung der Migrationszeit zwischen zwei Plattformen soll anhand des folgenden Schemas verdeutlicht werden:

Ein Agent möchte von Plattform L_i zur Plattform L_{i+1} migrieren. Die Heimatplattform des Agenten wird mit L_0 bezeichnet. Die Latenzzeit zwischen zwei Plattformen wird durch die Funktion δ dargestellt. Die Funktion τ beschreibt die verfügbare Transferrate zwischen zwei Plattformen. Die Anzahl der zu übertragenden Bytes wird mit B_C (Größe aller Code-Teile bzw. kompletter Bytecode des Agenten) angegeben.

Somit ergibt sich für die Übertragung aller Code-Teile zur nächsten Plattform folgende Berechnungsformel für die Übertragungszeit des Bytecodes T:

Push-All-To-Next $$T = \delta(L_i, L_{i+1}) + \frac{B_C}{\tau(L_i, L_{i+1})}$$

Bei der Berechnung kann die Größe des serialisierten Agenten (und damit der bereits gesammelten Daten) unbeachtet bleiben, da diese bei der Vergleichsberechnung für alle Varianten konstant ist.

Soll der Bytecode auf der nächsten Plattform L_{i+1} von der Heimatplattform L_0 nachgeladen werden, ergibt sich folgende Berechnungsvorschrift:

Pull-All $$T = \delta(L_0, L_{i+1}) + \frac{B_C}{\tau(L_0, L_{i+1})}$$

Für ein Nachladen von m einzelnen Code-Teilen von der Heimatplattform zur nächsten Plattform L_{i+1} mit der Wahrscheinlichkeit $P^k_{L_{i+1}}$, dass das k-te Code-Teil B^k_C nachgeladen wird, ergibt sich folgende Vorschrift:

Pull-Units
$$T = \sum_{k=1...m} P^k_{L_{i+1}} \left(\delta(L_0, L_{i+1}) + \frac{B^k_C}{\tau(L_0, L_{i+1})} \right)$$

Diese beispielhaft vorgestellten Berechnungsvorschriften und entsprechende Vorschriften für weitere Migrationsstrategien müssen im Optimierer implementiert werden. Für weitere Details zum mathematischen Modell und zu den daraus extrahierten Berechnungsvorschriften sei auf [Bra03] und [Sch03] verwiesen.

Der grundlegende Prozess der Optimierung ist mit zwei Schritten beschrieben:

- Berechnung der Vergleichswerte der einzelnen Migrationsstrategien,
- Vergleich der Resultate und Wahl der besten Strategie.

Allerdings kann der Optimierungsbereich (Anzahl der Zielplattformen) unterschiedlich weit gefasst und damit unterschiedliche Optimierungsvarianten definiert werden. Die einfachste Variante ist die Optimierung der Migration zur nächsten Plattform. Das kann beispielsweise sinnvoll sein, wenn die weitere Route dem Agenten noch nicht vollständig bekannt ist, wie sich z. B. aus dem Informationsangebot entsprechend dem Fish-Eye-Prinzip ergeben kann. Eine erweiterte Variante ist die Optimierung mehrerer, aufeinander folgender Sprünge des Agenten im Netzwerk, z. B. innerhalb einer Domain, bestenfalls für die komplette Reiseroute oder, von unterwegs, für die verbleibende Reiseroute des Agenten. Dabei kann einerseits die Strategie für die Route fixiert sein, andererseits die Strategie wechseln. Der Berechnungsaufwand kann sehr hoch werden, wenn die zu verwendende Migrationsstrategie nicht für alle Sprünge fixiert, sondern ein Wechsel der Strategie bei der Optimierung erlaubt wird.

Eine andere Art der Optimierung stellt die Initialisierung eines Code oder Mirror Servers dar. Die Initialisierung eines Code Servers ist empfehlenswert, wenn mehr Zeit für das Nachladen von Code von der Heimatplattform benötigt wird, als von einem „nahen“ Code Server. Voraussetzung dafür ist aber neben der Nutzung einer Pull-Strategie, dass sich der Aufwand der Initialisierung amortisiert. Bei der Optimierung werden entsprechende Vergleichswerte mit und ohne Code Server berechnet. Dabei lassen sich auch unterschiedliche Optimierungsstufen definieren. Eine einfache Optimierung betrachtet nur die Variante, den Code von der Heimatplattform nachzuladen oder den Code Server auf der lokalen Plattform zu initialisieren. Eine mittlere Optimierung wird erreicht, wenn alle bereits vorhandenen Code Server einbezogen werden. Wird zudem noch betrachtet, ob

zukünftige Plattformen Code Server werden können, ist ein hoher Grad der Optimierung erreicht.

Die Initialisierung eines Mirror Server hängt davon ab, ob ein Agent seine gesammelten Daten zu seiner Heimatplattform transferieren möchte. Diese Daten belasten bei der Reise des Agenten immer wieder das Netzwerk. Für den Fall, dass der Agent die Daten auf weiteren Plattformen nicht mehr braucht, ist ein Transfer sinnvoll. Der Optimierer berechnet nun, ob der lokale Server als Mirror Server für weitere Transfers genutzt werden kann, um Transferzeit zu sparen, oder ob der Transfer zur Heimatplattform (oder zu einem vorherigen Mirror Server) weniger Zeit in Anspruch nimmt.

4.5 Zusammenfassung zum Routing Service *ProNav*

Vorgestellt und beschrieben wurden die Komponenten des Routing Service: Kartenmodul, Routenplaner und Migrationsoptimierer. Das Kartenmodul ist für die Erstellung der Webkarte verantwortlich. Diese Karte kann von einem Agenten zur Lokalisierung der Dienste im Netz benutzt werden. Das Kartenmodul kann unabhängig von den anderen Komponenten genutzt werden.

Der Routenplaner berechnet für einen Agenten eine Route durch das Agentennetzwerk, die in Bezug auf die Reisezeit des Agenten möglichst optimal ist. Diese Komponente benötigt zur Routenplanung den Zugriff auf die Webkarte im Kartenmodul. Die Daten bilden die Grundlage für eine Routenberechnung. Ein Agent benutzt die Routenplanung in der Regel erst nach der Lokalisierung von potentiellen Zielpunkten im Netz. Für diese Zielpunkte wird eine Routenplanung für den Agenten vorgenommen.

Der Migrationsoptimierer erarbeitet auf Basis einer ermittelten Reiseroute eines Agenten eine dynamische Migrationsstrategie, die auf technischer Ebene eine Optimierung bezüglich der beim Agententransfer verursachten Netzwerklast durchführt. Dieser Optimierungsschritt ist spezifisch für das Agentensystem TRACY. Dabei ist der Zugriff auf Kartendaten notwendig. Der Agent nutzt diese Komponente normalerweise unmittelbar vor der Migration.

Kapitel 5

Evaluation

Die Komponenten des Routing Service *ProNav* werden in ihrer praktischen Anwendung evaluiert. Die Evaluation zeigt, dass die Realisierung des Beispielszenarios durch *ProNav* möglich ist. Aussagen über die Fähigkeiten und Grenzen von *ProNav* werden getroffen.

Ziel der Evaluation ist ein Nachweis der Tauglichkeit sowie die Charakterisierung eines möglichen Einsatzbereiches. Eine Bestätigung der in dieser Arbeit aufgestellten Thesen wird durch die Evaluation erreicht.

5.1 Kartenmodul

Im entworfenen Rahmenwerk *ProNav* spielt das Kartenmodul, das die Erstellung der Webkarte organisiert, eine zentrale Rolle. Der Fokus der Evaluation liegt auf der Qualität der Kartendaten (Knoten- und Verbindungsinformationen) sowie auf der Tauglichkeit der Karte für ein Agentensystem, insbesondere auf der Aktualität der Karte.

5.1.1 Vorbetrachtungen

Die Qualität der Knoteninformationen, wie der unterstützten Protokolle und der angebotenen Dienste, soll allerdings nicht Gegenstand der Evaluation sein. Diese Informationen werden in den Property-Listen der Agentenserver auf der Karte gehalten. Die Qualität dieser Informationen kann nicht bewertet werden. Für die Richtigkeit der Informationen ist jeder Agentenserver selbst verantwortlich, er pflegt seine Liste von Eigenschaften selbst. In dieser Arbeit wurde der inhaltliche Aspekt der Eigenschaftsbeschreibung nicht näher untersucht. Der Schwerpunkt lag vielmehr auf der Verbreitung der Daten im Netz und auf der Geschwindigkeit der Informationsverbreitung, die generell für die Aktualität der Karteninformationen interessant ist.

Die Messung der Verbindungsqualitäten wird mit Hilfe von Sensoren durchgeführt, die diese in regelmäßigen Abständen wiederholen. Für jede zu messende Eigenschaft muss ein entsprechender Sensor eingesetzt werden. Die im Kartenmodul eingesetzten Sensoren sind in Tabelle 5.1 auf der nächsten Seite mit einer kurzen Beschreibung ihrer Arbeitsweise aufgelistet. Weitere Sensoren können bei Bedarf problemlos in das Kartenmodul integriert werden (siehe Anhang A.2).

Entscheidend für die weitere Verwendung der Verbindungsinformationen ist die Qualität der Sensoren bzw. der gemessenen Werte. Fehlerhafte Daten pflanzen sich bei anschließenden Berechnungen (z. B. beim Routenplaner) fort und liefern verfälschte Ergebnisse. Evaluierungsgegenstand ist somit die Qualität der von Sensoren gelieferten Messwerte und der Verbindungsinformationen, die aus den gemessenen Werten extrahiert werden.

Des Weiteren kann die Qualität des Kartenmoduls an sich evaluiert werden. Beispielsweise wurde der Speicherverbrauch der Webkarte, der in quadratischer Abhängigkeit zu der Anzahl der Agentenplattformen steht, gemessen (siehe [Sch02]). Da es sich bei diesem Modul derzeit allerdings um eine prototypische Implementierung handelt, sei auf eine tiefergehende Evaluation in dieser Arbeit verzichtet und auf einen späteren Zeitpunkt verschoben.

Für die Evaluierung sind Referenzwerte (unabhängige Vergleichswerte) erforderlich, mit denen die Qualität der Informationen ermittelt werden kann. Es gibt verschiedene Varianten derartige Referenzwerte zu erhalten. Eine Möglichkeit ist eine Referenzmessung

Eigenschaft	**Arbeitsweise des Sensors**
Verfügbarkeit [%]	Ein minimales Paket wird zum Messpartner geschickt. Antwortet dieser, ist der Messpartner offensichtlich erreichbar. Der Prozentsatz wird unter Berücksichtigung der letzten gemessenen Ergebnisse berechnet.
Latenzzeit (Round Trip Time) [ms]	Ein minimales Paket wird zum Messpartner gesendet. Dieser antwortet sofort, wenn das Paket angekommen ist. Die dabei verstrichene Zeit wird gemessen.
Transferrate [KByte/s]	Ein Paket mit fixer Größe wird zu einem Messpartner transferiert. Dieser antwortet auf das empfangene Paket. Die benötigte Zeit wird gemessen und die daraus resultierende Transferrate wird berechnet.
Migrationszeit [ms]	Ein Agent fixer Größe wird vom Sensor zur Migration aufgefordert. Angekommen auf der Zielplattform, informiert dieser Agent den dortigen Migrationssensor, der eine Antwort an den Sender schickt. Die benötigte Zeit wird gemessen.

Tabelle 5.1: Übersicht über die implementierten Sensoren

mit einem anderen, bereits etablierten System durchzuführen. Das System soll dabei möglichst hohe Qualität in Bezug auf die Werte liefern. Eine andere Variante stellt die Vorgabe von Referenzwerten dar. Diese Vorgabe stellt den SOLL-Wert dar, an den der Messwert (IST-Wert) möglichst nah herankommen soll. Eine letzte, hier angegebene Variante ist die Nutzung eines mathematischen Modells zur Ermittlung eines Referenzwertes. Mit Hilfe eines möglichst genauen Modells lässt sich ein (theoretischer) Referenzwert berechnen, der experimentell erreicht werden soll. Die Schwierigkeit liegt hierbei ganz klar in der Modellierung.

Die Sensoren zur Messung der Latenzzeit und der Transferrate ermitteln in *ProNav* die wichtigsten Eckdaten für Verbindungsqualitäten und wurden aus diesem Grund zur Evaluierung herangezogen. Die Werte des Verfügbarkeitssensors bedürfen keiner tiefergehenden Qualitätsprüfung, da die direkt gemessenen Werte binär sind (erreichbar / unerreichbar).

Der Migrationssensor, der die Migrationszeit eines Agenten fester Größe misst, ist schwer zu evaluieren. Einerseits können die gemessenen Werte selbst als Referenzwerte interpretiert werden, da sie der benötigten Zeit zur Migration eines (Standard-) Agenten von einem Agentenserver zu einem Zielserver entsprechen. Andererseits ist eine Überprüfung der gemessenen Werte nur anhand eines mathematischen Modells möglich. Einige Aspek-

te lassen sich zwar in Formeln fassen, aber genaue Zahlenwerte sind stark von der aktuellen Situation abhängig, die ein mathematisches Modell schlecht erfassen kann. So wurde in [Bra03] ein mathematisches Modell zur Berechnung der Migrationszeit T_{Mig} eines Agenten a entwickelt (hier vereinfacht und am Beispiel dargestellt):

$$T_{Mig}(a) = \mu_1(B^a_{d,s}) + \delta(L_i, L_j) + \frac{B^a_{d,s} + B^a_C}{\tau(L_i, L_j)} + \mu_2(B^a_{d,s})$$

Ein Agent a wird zwischen den Plattformen L_i und L_j im Modell transferiert. Die Zeit zum Serialisieren des Status und der Daten $B^a_{d,s}$ des Agenten wird durch die Funktion μ_1 modelliert. Die Latenzzeit bei der Übertragung kann durch die Funktion δ angegeben werden. Die Zeit für den eigentlichen Transfer des serialisierten Agenten und des notwendigen Bytecodes B^a_C ist mit der Funktion τ ausgedrückt. Deserialisiert wird der Agent auf der Zielplattform in der Zeit, die durch μ_2 angegeben werden kann. Bei diesem Modell gibt es allerdings einige Unwägbarkeiten. So ist beispielsweise das Serialisieren und Deserialisieren des Agenten von der Rechenpower und der aktuellen Auslastung des Rechners abhängig. Somit wäre ein exakt berechneter Wert mit einer doch starken Ungewissheit verbunden.

In einer praktischen Applikation kann die gemessene Migrationszeit die Rolle eines Referenzwertes übernehmen. Der natürliche Schwankungsbereich der Werte wird durch wiederholte Messungen erkannt. Dadurch ist die Dynamik in Bezug auf die Werte erkennbar und Voraussagen über das erwartete Verhalten bzw. über den nächsten erwarteten Wert können vorgenommen werden. Bei der Wahl des Referenzagenten, der zur Migrationszeitmessung benutzt wird, ist allerdings zu beachten, dass die Größe des Agenten einer typischen Agentengröße entspricht. In praktischen Applikation hat sich jedoch gezeigt, dass diese Größe stark variiert. Die Größe des Referenzagenten könnte zur Evaluation an die Größe eines konkreten Applikationsagenten angepasst werden. Die Evaluation würde damit aber nur für diese Applikation erfolgen. Eine Evaluierung im engeren Sinne ist daher nicht erfolgt.

Damit beschränkt sich die eigentliche Evaluation auf den Latenzsensor und den Transferratensensor. Für die Qualitätsprüfung des Latenzsensors wird ein ermittelter Referenzwert, die *Round Trip Time* (RTT, PING Zeit, 2-fache Latenzzeit), benutzt. Die Referenzwerte wurden durch Messungen der RTTs mit dem `ping`-Befehl des Betriebssystems, in diesem Fall Linux, ermittelt. Die aus mehreren Messungen erhaltenen Durchschnittswerte sind Nanosekunden-Werte und damit genauer als die des Sensors, der Java-bedingt in Millisekunden misst. Bei der Prüfung des Transferratensensors wurden Referenzwerte vorgegeben, d. h. die Nutzdatenrate einer Verbindung wurde auf einen Referenzwert eingestellt und die Messungen sollen zeigen, ob sich die fixierte Nutzdatenrate in den Messergebnissen widerspiegelt.

Für die Evaluation wurde eine isolierte Messumgebung gewählt, die getrennt von anderen

Netzen betrieben wurde. Dadurch wurde unbekannter, störender Netzwerkverkehr vermieden. Dennoch wurde in einigen Experimenten bewusst Netzwerkverkehr mit zusätzlichen Rechnern simuliert, um Ergebnisse unter Einfluss von bekanntem, definiertem Verkehr zu erhalten und unterschiedliche Lastsituationen zu testen. In Abbildung 5.1 ist der Aufbau der Messumgebung, die aus zwei vernetzten Laptops bestand, dargestellt. Auf den Laptops (OS: Linux) wurde je ein Agentenserver gestartet, der jeweils einer eigenen Domain angehörte. Um die beiden Domain Manager bekannt zu machen, wurde auf einem der beiden Laptops ein weiterer Agentenserver, der Domain Master, gestartet. Diese Konstellation ist günstig, da kein zusätzlicher Netzwerkverkehr, der innerhalb einer Domain auftritt, Messergebnisse verfälscht (bei nur einem Knoten in der Domain entsteht kein interner Verkehr). Durch den Master wird ebenso kein Netzwerkverkehr verursacht, da dieser nur zur Kontaktaufnahme zwischen den Domains angesprochen wird. Nachdem die beiden Domain Manager gestartet wurden und sich über den Master gefunden hatten, nahmen diese die Messungen der Verbindungsqualitäten über die entsprechenden, lokal gestarteten Sensoren auf. Um das Verhalten der Sensoren und die Qualität der gemessenen Werte zu untersuchen, wurden unterschiedliche Konfigurationen des Testnetzwerks gewählt.

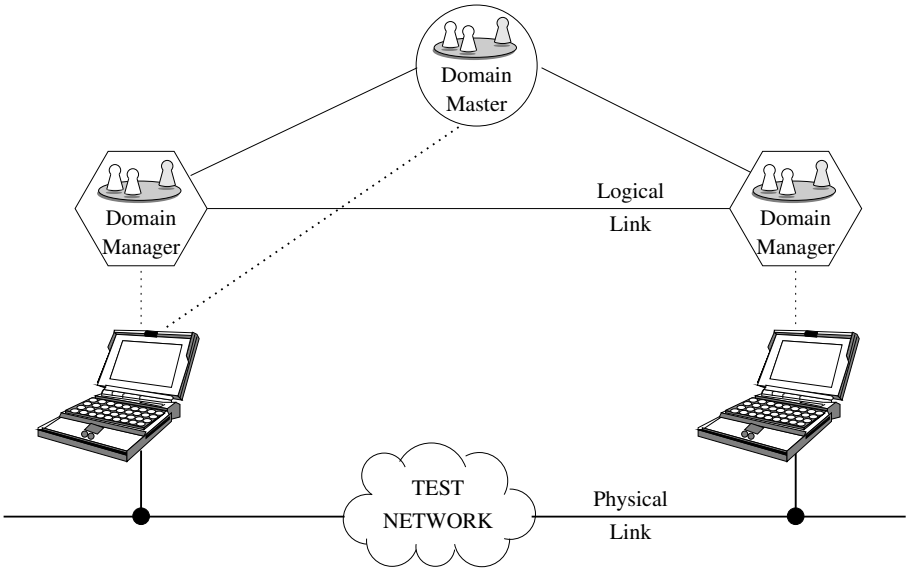


Abbildung 5.1: Messumgebung zur Evaluierung der Netzwerksensoren

5.1.2 Evaluation der Verbindungsdaten

Mit der Evaluation von Latenz- und Transferratensensor soll sich die Qualität der gemessenen Verbindungsinformationen erweisen. Dazu wurden verschiedene Evaluierungsexperimente durchgeführt.

Latenzsensor

Für eine erste Evaluierungsmessung wurde ein Ethernet-Testnetzwerk aufgebaut, in dem die beiden Testrechner über einen 10 Mbit/s Half-Duplex Hub verbunden wurden. Zur Ermittlung der Referenzwerte wurde die RTT mit Hilfe des `ping`-Befehls des Betriebssystems ermittelt. Gemessen wird dabei die RTT eines ICMP-Paketes (Internet Control Message Protocol), die ein guter Indikator für die Latenzzeit, die Distanz (Anzahl Router) und den aktuellen Netzwerkverkehr auf der benutzten Verbindung ist (indirekt durch die Auslastung der Router). Die Arbeitsweise des Packet Internet Groper (PING) spiegelt sich in der Implementierung des Latenzsensors wider. Jedoch agiert der Sensor auf einer höheren Ebene, da Java den Zugriff auf Raw-Sockets (OSI-Schicht 3) nicht erlaubt. Der Sensor benutzt Sockets auf TCP-Ebene (OSI-Schicht 4).

Zur Evaluierungsmessung wurde der Netzwerkverkehr schrittweise erhöht, um Messwerte in unterschiedlichen Lastsituationen zu erfassen. Zu diesem Zweck wurde definierter Verkehr auf dem Netzwerk erzeugt. Es wurden zwei weitere Rechner an das Netzwerk angeschlossen, die einen Datenstrom (TCP) mit einer vorgegebenen Datenrate austauschten. Die dem Datenstrom zur Verfügung stehende Datenrate wurde dabei über das Werkzeug Traffic Control (tc) [Hub04] geregelt und schrittweise erhöht.

Auf die gleiche Art wurden Messungen in einer veränderten Netzwerkkonstellation durchgeführt: Einer der beiden Testrechner wurde per Funk über einen Access-Point angebunden (IEEE 802.11b WLAN). In Abbildung 5.2 auf der nächsten Seite ist das Ergebnis der beschriebenen Messungen zu sehen. Im unteren Teil des Diagramms bei etwa 1 ms RTT befinden sich die Messkurven für das Ethernet. Die oberen Kurven stellen die Messergebnisse im WLAN dar.

Die mit OS-PING gemessenen Zeiten liegen jeweils unterhalb von denen, die mit den Sensoren gemessen wurden. Das hat verschiedene Gründe. Die Sensoren kommunizieren über eine gesicherte End-zu-End Verbindung auf TCP-Ebene. Im Gegensatz dazu arbeitet das OS-PING ungesichert auf IP-Ebene. Die Sensoren befinden sich innerhalb der Java VM. PING allerdings ist Bestandteil des Betriebssystems. In der Summe ergibt sich ein zusätzlicher zeitlicher Aufwand, der sich nachteilig auf die durch die Sensoren gemessenen Werte auswirken kann. Die Unterschiede der Sensorwerte zu den PING-Referenzwerten sind dennoch sehr gering. Meist liegen die Sensorwerte sogar weniger als 1 ms über den Referenzwerten. Dies zeigt die gute Qualität der Daten, die damit für weitere Berechnungen problemlos verwendet werden können. Ein weiterer Aspekt muss noch berücksichtigt werden: Die Zeitmessung in Java (Version 1.4 des JDK) ist nur im Millisekundenbereich möglich. Dadurch kann bei den eingezeichneten Werten, die Durchschnittswerte verkörpern, ein Fehler von $\pm 0{,}5$ ms auftreten[1]. Die nächste Java-

[1] Wird eine RTT vom 0 ms gemessen, gibt der Sensor als Messwert 0,5 ms zurück. Somit beträgt der Fehler $+0{,}5$ ms im Bereich von 0 ms bis 1 ms.

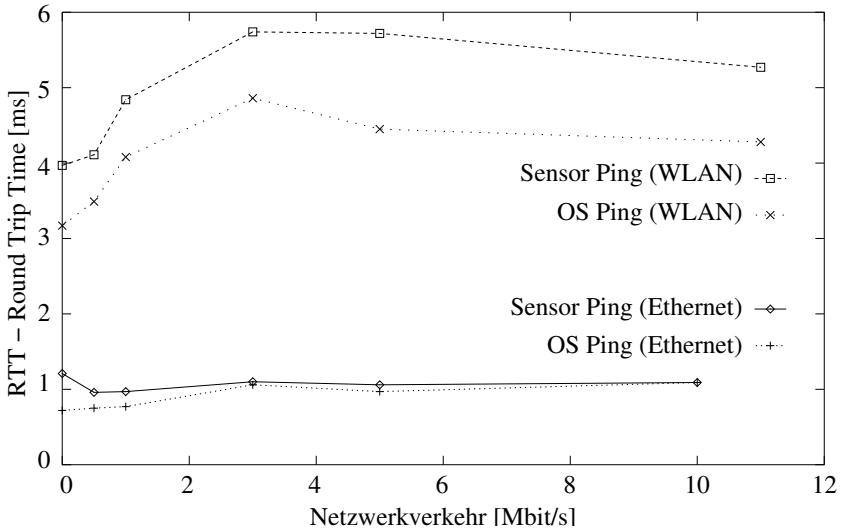


Abbildung 5.2: Vergleich der RTT in einem 10 Mbit/s Ethernet und einem Funknetz

Version verspricht eine Verbesserung: In Zukunft wird eine Zeitmessung in Nanosekunden möglich sein. Eine Wiederholung der Messungen mit angepassten Sensoren ist dann interessant und die Qualitätsprüfung kann genauer erfolgen.

Weiterhin fällt im Diagramm der Abbildung 5.2 auf, dass die RTT im WLAN höher liegt als im Ethernet. Der Grund dafür ist das verwendete Zugriffsverfahren CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance). Im Gegensatz zum Ethernet (Collision Detection) handeln bei diesem Zugriffsverfahren die Partner im WLAN das Senden aus (Grund: Hidden Terminal Problem). Diese Zeiten sind damit typisch für derartige Netze. Zudem sind Funknetze auf Grund des geteilten Mediums anfällig in Bezug auf Störungen. Häufig gehen Pakete verloren und müssen erneut gesendet werden. Dass die vom Sensor gemessenen Werte qualitativ gut sind, bestätigen die jeweiligen OS-PING-Werte. Des Weiteren erlauben die RTTs dieser Evaluierungsmessung einen Rückschluss auf die Art der physikalischen Verbindung zwischen den Rechnern.

Die Netzwerkauslastung beeinflusst die RTT des Ethernet nur gering. Ein leichter Anstieg ist bei den PING-Werten zu erkennen. Bei den Sensorwerten reicht offensichtlich die Messgenauigkeit nicht aus, um diesen Anstieg zu erkennen.

Im Gegensatz zum Ethernet hat eine zusätzliche Netzwerklast im WLAN mehr Auswirkungen auf die RTTs. Bedingt durch das andere Zugriffsverfahren steigen die Zeiten deutlich an, bis etwa eine Last von 3 Mbit/s erreicht ist. Danach fallen die Zeiten etwas. Durch den erhöhten Netzwerkverkehr (im Ethernet-Teil der WLAN-Testumgebung) kommt es vermehrt zu Kollisionen der PING- und Sensorpakete mit den Lastpaketen. Das leichte Fallen der RTTs kann nicht mit der Theorie dieser Netze begründet werden. Eine Vermutung ist, dass bei der Erzeugung von Netzwerklast, die mit dem Werkzeug Traffic Control

beschränkt wurde, Nebeneffekte auftraten.

Geht ein ICMP-Paket (PING) verloren oder dauert die Übertragung zu lange, wird keine RTT gemessen. Die Pakete werden nur als verloren in die Statistik aufgenommen. Auf der gesicherten TCP-Verbindung, auf der die Sensoren arbeiten, wird ein neues Paket angefordert, falls ein Paket verloren gegangen ist. Ebenso kann der Fall eintreten, dass auf Pakete gewartet werden muss. Dadurch können bei den Messwerten „Ausreißer“ nach oben entstehen. Extreme Ausreißer werden bei den Messreihen meist ignoriert, da sie den Durchschnittswert enorm verfälschen.

Im nächsten Evaluierungsexperiment wurde der Prozessor des Rechners mit Fraktalberechnungen belastet, um die Abhängigkeit der RTT von der aktuellen Rechenlast zu prüfen. Netzwerklast wurde nicht verursacht. In Abbildung 5.3 ist ein Diagramm dargestellt, in dem RTT-Messfolgen im WLAN und im Ethernet präsentiert werden.

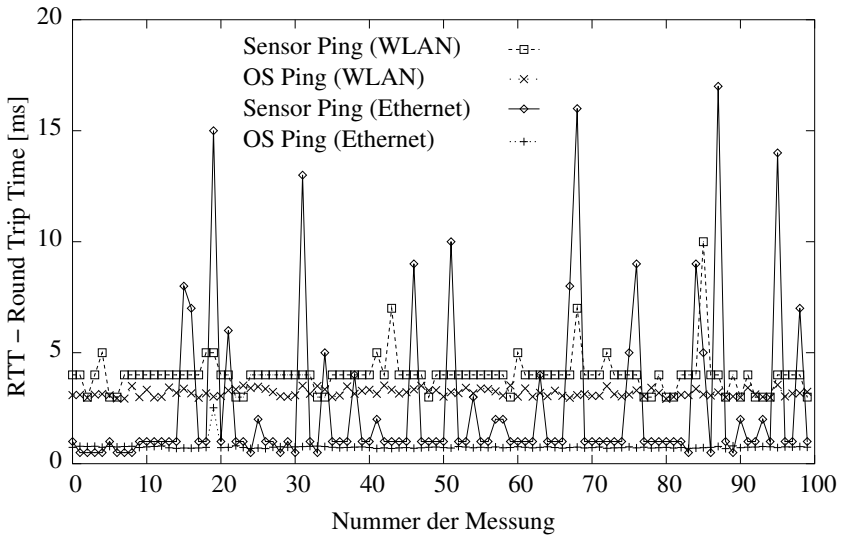


Abbildung 5.3: Messung der RTT auf einem Rechner mit Rechenlast

Jeweils 100 Messungen wurden bei konstanter Belastung des Prozessors hintereinander durchgeführt. Die Referenzwerte sind wieder mit `ping` ermittelt worden. Die relativ schlechte Auflösung der Sensorwerte (Java-bedingt in Millisekunden) ist in diesem Experiment deutlich erkennbar. Die gemessenen Zeiten sind immer ganze Millisekundenwerte (Ausnahme: gemessene 0 ms werden Sensor-intern auf 0,5 ms aufgerundet). Die meisten Sensorwerte liegen für das Ethernet bei genau 1 ms und für das WLAN bei genau 4 ms.

Generell liegen die PING-Werte wieder unterhalb der Sensorwerte. Die Gründe wurden bereits bei der Auswertung des vorherigen Experiments diskutiert. Eine Abhängigkeit der OS-PING-Werte von der reinen Rechenlast ist nicht feststellbar. Vergleichswerte dafür können aus dem vorherigen Evaluierungsexperiment genommen werden (siehe

Abb. 5.2 auf Seite 89, bei 0 Mbit/s Netzwerklast ist die RTT des OS-PING ca. 0,7 ms). Ebenso hat die Prozessorlast die RTT-Werte, die durch den Sensor ermittelt wurden, nicht wesentlich beeinflusst. Durchschnittlich liegen diese Werte wieder bei einer für dieses Netzwerk typischen Größe von etwa 1 ms (siehe vorheriges Messexperiment). Dennoch sind einzelne Spitzen zu erkennen. Diese deuten darauf hin, dass der Ablauf einer Sensormessung kurzzeitig blockiert wurde. Die Paketverarbeitung auf Anwendungsebene wurde verzögert, wodurch diese „Ausreißer“ entstanden. Die RTT-Werte der Sensoren werden zwar durch die Spitzen leicht verfälscht, sind aber dennoch relativ stabil. Umso wichtiger sind Statistiken, die solche Spitzen bei Eintragen der Vorhersagewerte in das Kartenmodul wieder glätten.

Transferratensensor

Für die Evaluationsmessungen des Transferratensensors wurden vier Konstellationen des Testnetzwerks gewählt, d. h. die zwei Laptops wurden unterschiedlich miteinander vernetzt:

S1 Ein Ethernet Subnetz mit einem 10 Mbit/s Half-Duplex Hub.

S2 Zwei 10 Mbit/s Half-Duplex Ethernet Subnetze, separiert durch einen Router (CISCO 2600).

S3 Ein IEEE 802.11b WLAN im Managed Mode (mit Access Point)

S4 Ein IEEE 802.11b WLAN im Ad-hoc-Mode (ohne Access Point)

Zur Messung der Transferraten in den Netzwerkkonstellationen wurde, wie bereits erwähnt, die zur Verfügung stehende Nutzdatenrate (Bitratenlimit) vorgegeben. Dazu wurde wieder das Werkzeug Traffic Control (tc) [Hub04] genutzt. Die Nutzdatenrate wurde dabei schrittweise erhöht. Die Evaluierungsmessungen sollen die eingestellten Nutzdatenraten (Referenzwerte) bestätigen.

Konstellationen S1 und S2 Im Diagramm in Abbildung 5.4 auf der nächsten Seite sind die Messergebnisse für die Netzwerkkonstellationen S1 und S2 dargestellt. Die protokollierten Messwerte sind Median- und Durchschnittswerte. Die senkrechten Balken im Diagramm stellen die Messschwankungen dar. Die fixierte Nutzdatenrate, das Nutzdatenratenlimit, ist im Diagramm logarithmisch skaliert. Die Transferraten (gemessener Durchsatz) wurden mit dem Transferratensensor ermittelt, der zu diesem Zweck ein Paket definierter Größe zu einem Agentenserver schickt, eine Antwort abwartet und die benötigte Zeit misst.

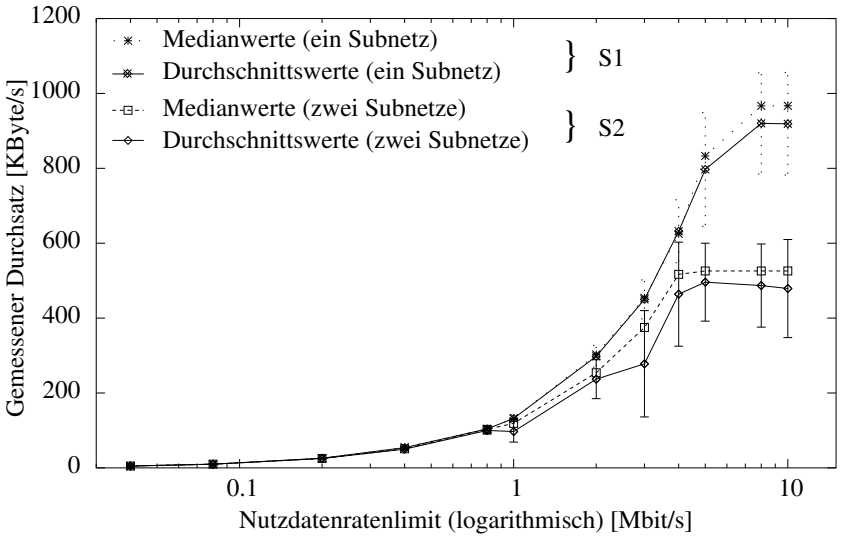


Abbildung 5.4: S1 und S2: Messung der Transferrate im Ethernet

In dem abgebildeten Diagramm fällt sofort auf, dass die Messungen in den unterschiedlichen Konstellationen (S1 und S2) ab 1 Mbit/s Nutzdatenrate auseinander laufen. Dabei liegt die Sättigung in einem Subnetz (S1) höher als bei den gekoppelten Ethernet Netzen (S2). Die Ursache für die Unterschiede ist vermutlich mit der verwendeten Hardware zu begründen. Bei der Konstellation S1 wurde ein 10 Mbit/s Hub verwendet, der Pakete einfach nur weiterleitet – OSI-Schicht 1 (Erst bei einem Switch wird das Routing auf MAC-Adressen-Ebene – OSI-Schicht 2 – realisiert). Dahingegen wird das Routing beim Router auf Basis von IP-Adressen – OSI-Schicht 3 – durchgeführt (Konstellation S2). Das simple Weiterleiten von Paketen im Netz ist die schnellste Variante des Pakettransfers. Demgegenüber ist das auf IP-Adressen basierenden Routing langsamer, da das zu routende Paket erst in einen Puffer geladen (store) und „ausgepackt“ werden muss, um die IP-Adresse und damit den weiteren Weg des Pakets (select route) festzustellen. Danach wird das Paket wieder „verpackt“ und in Richtung Ziel weitergeleitet (forward). Anmerkung: Routing mit Hilfe von MAC-Adressen ist etwas schneller. Jedes Paket wird somit zweimal verschickt, was mindestens zu einer Verdopplung des Latenzanteils führt. In der Summe entsteht ein größerer Aufwand, der sich in den Transferzeiten und damit in der Transferrate bemerkbar macht.

Der gemessene Durchsatz kann von der zur Verfügung stehenden Nutzdatenrate abweichen. TCP im Internet arbeitet mit der TCP Slow Start Option (RFC 2001), die zur Vermeidung von Überlastsituationen benutzt wird. Beim Senden von Paketen darf nur eine bestimmte Zahl von unbestätigten Paketen unterwegs sein. Ist diese Anzahl erreicht, darf der Sender keine weiteren Pakete senden. Die Anzahl der erlaubten Pakete (Fenstermechanismus) wird schrittweise erhöht (oder bei Überlast verringert). Das hat zur Folge,

dass bei einer Übertragung nur einiger, weniger Datenpakete die zur Verfügung stehende maximale Nutzdatenrate nicht ausgenutzt wird.

Bei höheren Nutzdatenraten schwanken die Werte (senkrechte Balken in Abb. 5.4 auf der vorherigen Seite) stark. Ein Grund dafür ist die ungenaue Auflösung der Zeitmessung in Millisekunden, die bereits bei dem Latenzsensor Schwierigkeiten bereitet hat. Bei einer hohen Nutzdatenrate verringert sich die Transferzeit der Messpakete, die in diesem Fall 30 KByte groß waren (Nutzdaten). Das bedeutet, dass aus einem kleinen Unterschied bei der gemessenen Transferzeit ein großer Sprung bei der Transferrate resultiert. Daher steigen die relativen Messfehler mit abnehmender Transferzeit, was sich auch im Diagramm durch eine stärkere Schwankung der Messwerte ausdrückt. Bei niedrigen, zur Verfügung stehenden Datenraten sind daher die Schwankungen wesentlich geringer, da hier offensichtlich die Messgenauigkeit in Millisekunden durch die längeren Paketlaufzeiten ausreicht. Allerdings ist an den Transferzeiten unterhalb von 1 Mbit/s Nutzdatenrate auch zu erkennen, dass die Größe der Messpakete zu hoch gewählt ist. Es wäre wünschenswert, die Paketgröße dynamisch an die verfügbare Nutzdatenrate anzupassen.

Ein Blick auf die Tabelle 5.2 auf der nächsten Seite, die eine detaillierte Darstellung der Messwerte (Medianwerte) und deren Abweichungen enthält, verdeutlicht das Problem der Messungenauigkeit erneut. Je mehr Nutzdatenrate zur Verfügung steht (eingestellte Nutzdatenrate), desto schneller ist das Paket transferiert (gemessene Transferzeit – IST). Bei hoher Nutzdatenrate hat somit der Anteil der Latenzzeit (ca. 1 ms) im Verhältnis gesehen großen Einfluss auf die aus der Transferzeit berechnete Transferrate (gemessene Transferrate). Zieht man die Latenzzeit von 1 ms von der gemessenen Transferzeit 31 ms (S1, Messung mit 8 Mbit/s Nutzdatenrate) ab und berechnet die Transferrate für 30 ms, erhält man genau 1000 KByte/s. Verglichen mit der gemessenen Transferrate (ca. 967,74 KByte/s) ergibt dies eine Differenz von 32,26 KByte/s. Dies verdeutlicht den Einfluss der Latenzzeit bei kurzen Transferraten. Ebenso groß ist der Einfluss von geringen Abweichungen bei den Messwerten (Transferzeiten) auf die Transferraten. Bei kurzen Transferzeiten führen bereits kleine Schwankungen zu großen Fehlern. So bewirkt beispielsweise die Messungenauigkeit in Java von $\pm$0,5 ms einen Fehler von $\pm$16,13 KByte/s in der Transferrate.

Bei einer eingestellten Nutzdatenrate von 10 Mbit/s, dem theoretischen Referenzwert, sind die Abweichungen zur gemessenen Transferrate sehr stark. Jedoch sind die gemessenen absoluten Werte gar nicht so weit von dem realen Maximum eines solchen Ethernets entfernt. Bei der Konstellation S1 (ein Subnetz) tritt eine Sättigung bei einer gemessenen Transferrate von 970 KByte/s ein (bzw. bei etwa 1000 KByte/s, wenn die Latenz berücksichtigt wird). Die theoretische Datenrate für das 10 Mbit/s Netzwerk liegt zwar bei 1280 KByte/s, doch ist dies die ideale, theoretische Menge an Rohdaten (Brutto-Daten), die über eine derartige Verbindung maximal pro Sekunde gesendet werden kann. Abzüglich des Overhead, wie Paket-Header, der vermehrt auftretenden Paketkollisionen

im verwendeten CSMA/CD-Verfahren (Carrier Sense Multiple Access with Collision Detection; IEEE 802.3 Standard) und abhängig von der Protokollkonfiguration erreicht man etwa 80% der maximalen Datenrate, die für reine Nutzdaten zur Verfügung steht. Das ergibt eine effektive Datenrate von etwa 1024 KByte/s (Netto-Daten). Die Abweichungen zu den gemessenen Daten beträgt somit nur noch 56,26 KByte/s (bzw. 24 KByte/s, wenn die Latenz berücksichtigt wird). Somit wird das reale Maximum eines solchen Ethernets durch die Messungen bestätigt.

eingestellte Nutzdatenrate		**gemessene Transferrate**	**Abweichung**		**Transferzeit Paket** [ms]	
[Mbit/s]	[KByte/s]	[KByte/s]	absolut [KByte/s]	relativ	SOLL	IST
10,0	1280,0	967,74193548	-312,25806452	32,27%	23,44	31
8,0	1024,0	967,74193548	-56,25806452	5,81%	29,30	31
5,0	640,0	833,33333333	193,33333333	23,20%	46,88	36
4,0	512,0	625,00000000	113,00000000	18,08%	58,59	48
3,0	384,0	454,54545455	70,54545455	15,52%	78,13	66
2,0	256,0	303,03030303	47,03030303	15,52%	117,19	99
1,0	128,0	132,74336283	4,74336283	3,57%	234,38	226
0,8	102,4	104,52961672	2,12961672	2,04%	292,97	287
0,4	51,2	55,45286506	4,25286506	7,67%	585,94	541
0,2	25,6	26,36203866	0,76203866	2,89%	1171,88	1138
0,08	10,24	10,49685094	0,25685094	2,45%	2929,69	2858
0,04	5,12	5,23743017	0,11743017	2,24%	5859,38	5728
10,0	1280,0	526,31578947	-753,68421053	143,20%	23,44	57
8,0	1024,0	526,31578947	-497,68421053	94,56%	29,30	57
5,0	640,0	526,31578947	-113,68421053	21,60%	46,88	57
4,0	512,0	517,24137931	5,24137931	1,01%	58,59	58
3,0	384,0	375,00000000	-9,0	2,40%	78,13	80
2,0	256,0	254,23728814	-1,76271186	0,69%	117,19	118
1,0	128,0	119,52191235	-8,47808765	7,09%	234,38	251
0,8	102,4	103,80622837	1,40622837	1,35%	292,97	289
0,4	51,2	51,02040816	-0,17959184	0,35%	585,94	588
0,2	25,6	25,46689304	-0,13310696	0,52%	1171,88	1178
0,08	10,24	10,14198783	-0,09801217	0,97%	2929,69	2958
0,04	5,12	5,06842372	-0,05157628	1,02%	5859,38	5919

Tabelle 5.2: Übersicht Messungen S1 (oben) und S2 (unten): Transferraten (Medianwerte), -zeiten und Abweichungen

In der Konstellation S2 kommt durch den Router eine zusätzliche Sende-/Empfangsstation auf OSI-Schicht 3 hinzu. Dadurch werden zwei IP-Subnetze miteinander gekoppelt. Diese Subnetze werden von den Paketen sequentiell durchlaufen. Auf dem Router kommt eine zusätzliche Verzögerung durch die Verarbeitung der Pakete hinzu. Die in einem Subnetz erreichbaren Nettotransferraten können in dieser Konstellation nicht angesetzt werden. Die Sättigung tritt daher bereits bei niedrigeren Datenraten ein, wie die durchgeführten Messungen bestätigen. Die in der Tabelle dargestellte Abweichung oberhalb der Sättigung

ist somit eine „theoretische“ Abweichung.

Des Weiteren verdeutlicht die Tabelle 5.2 auf der vorherigen Seite, dass die Messpaketgröße an die zur Verfügung stehende Nutzdatenrate angepasst werden sollte. Die Paketgröße von 30 KByte ist bei einer niedrigen, zur Verfügung stehenden Datenrate nicht tragbar und belastet das schwache Netzwerk viel zu stark. Es muss ein guter Kompromiss zwischen Paketgröße, Datenrate und Latenzzeit gefunden werden. Als Anhaltspunkt zur Ermittlung einer geeigneten Paketgröße kann die Transferzeit, die eine Funktion dieser Größen ist, benutzt werden.

Einige gemessene Werte, die in der Tabelle aufgelistet sind, liegen oberhalb des vorgegebenen Nutzdatenratenlimits. Dies hätte nicht passieren dürfen, da das Limit hart definiert war. Kleine Abweichungen sind durch die Ungenauigkeit der Zeitmessungen zu erklären. Bei der Konstellation S1 sind allerdings größere Abweichungen (von 15% bis 24%) aufgetreten. Die Ursachen für die Abweichungen liegen nicht im Kartenmodul, denn eine möglicherweise schlechte Programmierung des Sensors würde zu längeren Transferzeiten führen und nicht zu einer Verkürzung. Ein Datenpaket, das zwischen Sensoren ausgetauscht wird, muss erst vollständig beim Messpartner angekommen sein, ehe eine bestätigende Antwort zurückgesendet wird. Es ist also anzunehmen, dass das verwendete Werkzeug Traffic Control die Datenrate nicht exakt auf den eingestellten Wert beschränkt hat. Eine Wiederholung der Messungen mit einem anderen Werkzeug wird in Zukunft angestrebt. Trotz dieser Abweichungen spiegelt sich die eingestellte Nutzdatenrate in den Messwerten gut wider.

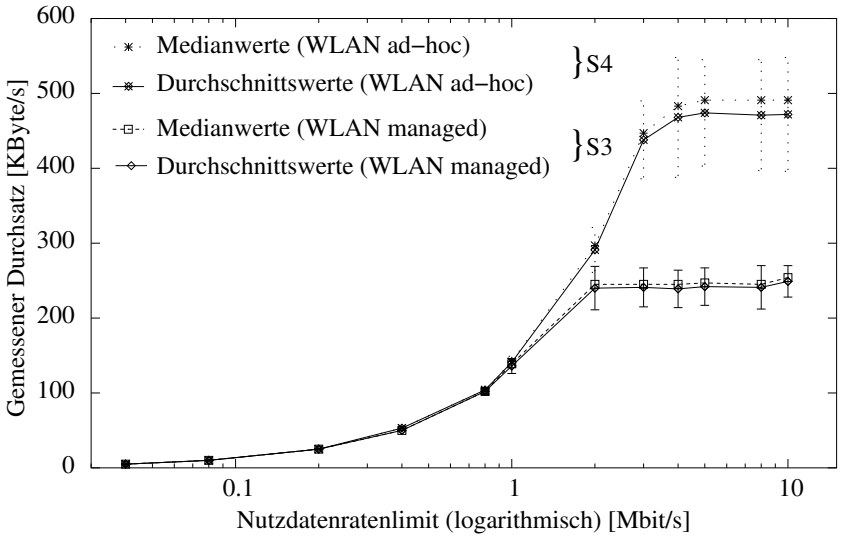


Abbildung 5.5: S3 und S4: Messung der Transferrate im WLAN

Konstellationen S3 und S4 Weitere Evaluierungsmessungen wurden für die Konstellationen S3 (WLAN managed) und S4 (WLAN ad-hoc) durchgeführt. In Abbildung 5.5 sind die Messergebnisse in einem Diagramm zusammengefasst. Auffällig ist, wie bei den vorherigen Konstellationen (S1 und S2), der Unterschied im Sättigungslevel der beiden Netzwerkkonstellationen. Dieser Unterschied entsteht, da im Ad-hoc-Modus die Partner direkt kommunizieren. Im Managed Mode übernimmt der Access Point dagegen das Routing der Pakete. Das Weiterleiten passiert innerhalb des Funknetzes auf OSI-Schicht 2 (MAC-Adressen-Ebene). Der Access Point verwendet zur Kommunikation mit den angemeldeten Kommunikationspartnern einen fixierten Kanal. Auf Grund des geteilten Mediums muss das Empfangen und das Senden der Pakete sequentialisiert werden. Dadurch halbiert sich die verfügbare Datenrate im Managed Modus zu der im Ad-hoc-Modus.

eingestellte Nutzdatenrate		**gemessene Transferrate**	**Abweichung**		**Transferzeit Paket** [ms]	
[Mbit/s]	[KByte/s]	[KByte/s]	absolut [KByte/s]	relativ	SOLL	IST
10,0	1280,0	254,23728813	-1025,76271186	403,47%	23,44	118
8,0	1024,0	245,90163934	-778,09836066	316,43%	29,3	122
5,0	640,0	247,93388430	-392,06611570	158,13%	46,88	121
4,0	512,0	245,90163934	-266,09836066	108,21%	58,59	122
3,0	384,0	245,90163934	-138,09836066	56,16%	78,13	122
2,0	256,0	245,90163934	-10,09836066	4,11%	117,19	122
1,0	128,0	139,53488372	11,53488372	8,27%	234,38	215
0,8	102,4	102,04081633	-0,35918367	0,35%	292,97	294
0,4	51,2	50,50505051	-0,69494949	1,38%	585,94	594
0,2	25,6	25,35925613	-0,24074387	0,95%	1171,88	118
0,08	10,24	10,15228426	-0,08771574	0,86%	2929,69	295
0,04	5,12	5,07185123	-0,04814877	0,95%	5859,38	591
10,0	1280,0	491,80327869	-788,19672131	160,27%	23,44	61
8,0	1024,0	491,80327869	-532,19672131	108,21%	29,3	61
5,0	640,0	491,80327869	-148,19672131	30,13%	46,88	61
4,0	512,0	483,87096774	-28,12903226	5,81%	58,59	62
3,0	384,0	447,76119403	63,76119403	14,24%	78,13	67
2,0	256,0	297,02970297	41,02970297	13,81%	117,19	101
1,0	128,0	142,85714286	14,85714286	10,4 %	234,38	210
0,8	102,4	104,52961672	2,12961672	2,04%	292,97	287
0,4	51,2	53,57142857	2,37142857	4,43%	585,94	560
0,2	25,6	25,48853016	-0,11146984	0,44%	1171,88	117
0,08	10,24	10,14541765	-0,09458235	0,93%	2929,69	295
0,04	5,12	5,07099391	-0,04900609	0,97%	5859,38	591

Tabelle 5.3: Übersicht Messungen S3 (oben) und S4 (unten): Transferraten (Medianwerte), -zeiten und Abweichungen

Weiterhin wird die Sättigung früh, bei knapp 500 KByte/s, erreicht. Das sind etwa 65% weniger als theoretisch möglich (11 Mbit/s). Das ist jedoch kein Indiz für schlechte Messungen, sondern zeigt nur die typische Problematik der Funkkommunikation. Der Over-

head im Funknetz ist deutlich höher als im Ethernet. Das Zugriffsverfahren ist ein Grund für eine niedrigere, erreichbare Datenrate. Zudem entsteht zusätzlicher Kommunikationsaufwand im 802.11 Standard durch Protokoll-Header, Fehlerkontrolle, Flusskontrolle und Quittungsverkehr. Des Weiteren müssen Sendepausen eingehalten werden, damit Geräte den Zugang zum Kommunikationsmedium erhalten können. Der Netto-Durchsatz (Nutzdaten) liegt damit erheblich unter dem Brutto-Durchsatz (Rohdaten). Hinzu kommt ebenso die Abhängigkeit des Funksignals von den Umgebungsbedingungen, wie Dämpfung, Reflexionen oder Störquellen. Bei idealen Bedingungen führt dies zu einem realen, optimalen Durchsatz an Nutzdaten von nur ca. 5,5 Mbit/s (IEEE 802.11b Standard).

Für die Qualität und die Genauigkeit der Messwerte sowie für die Schwankungen in den oberen Bereichen trifft das bereits zu den Konstellationen S1 und S2 Gesagte zu. In Tabelle 5.3 auf der vorherigen Seite ist eine analytische Darstellung der Messwerte (Median) für die Netzwerkkonstellation S3 und S4 zu finden.

Fazit Sensoren

Die Ergebnisse der Latenz- und Transferratensensormessungen sind alles in allem zufrieden stellend. Werden die aufgetretenen Probleme mit der Java-bedingten Ungenauigkeit der Messzeiten berücksichtigt, ist die Qualität der Verbindungsinformationen des Kartenmoduls (der Webkarte) erwartungsgemäß gut. Mit der neuen Version von Java ist auf eine Verbesserung zu hoffen.

In den aufgezeigten Diagrammen spiegeln die Messergebnisse dennoch nicht direkt die vorgegebenen theoretischen Referenzwerte wider. Allerdings liegt das reale Maximum der Nutzdatenrate und des Netto-Durchsatzes (Rohdaten) der einzelnen Netzwerke (Ethernet, WLAN) von Natur aus unter den theoretischen Referenzwerten. Unter Berücksichtigung des realen Maximums des jeweiligen Netzwerks sind die von den Sensoren gemessenen Werte qualitativ gut zu bewerten. Die durch den Latenzsensor und Transferratensensor gemessenen Werte zeigen eine Qualität der Karteninformationen, die für aufbauende Berechnungen, der Routenplanung und Migrationsoptimierung, ausreichend ist.

Weiterhin waren in den durchgeführten Messungen der Latenzzeiten und Transferraten Schwankungen in den Werten aufgefallen. Diese Abweichungen liegen dennoch weit unter den Schwankungen, die im Ethernet und im WLAN technologiebedingt auftreten. Beide Netze sind eher auf Robustheit und nicht auf Qualität ausgelegt. Umso wichtiger ist die Verwendung von statistischen Vorhersagen, die einerseits technologiebedingte Schwankungen in den Netzen glätten und andererseits Messfehler relativieren. Im Kartenmodul sind entsprechende Vorhersagen implementiert.

Die Evaluation des Transferratensensors hat zudem Schwachstellen im Modul aufgedeckt. So ist die Paketgröße, die vom Sensor zur Messung benutzt wird, dynamisch an die zur

Verfügung stehende Datenrate des Netzwerks anzupassen. Als Entscheidungsgröße kann die Transferzeit des Paketes verwendet werden. Unnötige Belastungen schwacher Verbindungen können dadurch vermieden sowie Ungenauigkeiten in schnellen Netzen verringert werden.

Bei der Durchführung der Messexperimente wurde eine mögliche gegenseitige Behinderung der einzelnen Sensoren nicht berücksichtigt. Prinzipiell können sich zwei Sensoren unterschiedlichen Typs, die zur gleichen Zeit Messungen durchführen, gegenseitig behindern. In den Evaluierungsexperimenten wurde zumindest dieser Effekt vermieden, jedoch kann es in einer realen Umgebung zu solchen Konflikten kommen. Diese Konflikte treten im Verhältnis gesehen selten auf. Zudem werden durch statistische Vorhersagen solche Messfehler ausgeglichen.

Eine weitere Schlussfolgerung kann aus den Ergebnissen der Evaluation formuliert werden: Die Sensoren ermöglichen eine sinnvolle Wahrnehmung des Agentennetzwerks, der virtuellen Welt des Agenten. Die generelle Aussage, dass solch eine Wahrnehmung der virtuellen Welt in adäquater Weise durch den Agenten möglich ist, wird dadurch untermauert:

T1 Mit Hilfe einer Webkarte wird ein Agent befähigt, seine Umwelt korrekt wahrzunehmen.

5.1.3 Evaluation der Karte

Bei der Evaluation der Karte werden Antworten auf verschiedene Fragen gesucht, z. B. Wie alt sind die Kartendaten? Wie schnell kann die Karte aktualisiert werden? Sind unscharfe Kartenteile für den Agenten hilfreich? Die Untersuchungen sollen die Tauglichkeit der Karte für ein Agentensystem herausstellen.

Aktualität der Karte

Neben der Qualität der Informationen auf der Karte ist ebenso die Aktualität der Kartendaten von Bedeutung. Die Aktualität der Kartendaten steht in engem Zusammenhang mit den Transferratenmessungen. Das zwischen den Agentenservern ausgetauschte Paket enthält Nutzinformationen, nämlich die Kartendaten des Senders. Sollen die Kartendaten möglichst aktuell sein, so müssen diese Messungen entsprechend oft durchgeführt werden. Das führt allerdings zu einer Erhöhung der Netzwerklast. Daher muss ein Kompromiss zwischen Aktualität der Karte und erwünschter, maximaler Netzwerklast gefunden werden.

Das Prinzip, nach dem jeder Sensor arbeitet, kann durch folgenden Pseudocode dargestellt werden:

for *every known agency s* **do**
    do measurement with s;
    wait duration;
end.

Im dadurch definierten Messintervall (einer Schleifeniteration), bestehend aus einer Messung (measurement) und einer definierbaren Messpause (duration), wird ein Messwert für einen Messpartner ermittelt. Ist die Messpause bei allen Agentenservern gleich gesetzt, hat jeder Agentenserver im Messintervall genau ein Messexperiment durchgeführt. Wird die Länge der Messpause auf 0 gesetzt, werden kontinuierlich Messungen durchgeführt. Nachteilig bei klein gehaltenen Messpausen sind einerseits gegenseitige Behinderungen der Messungen zwischen den Agentenservern und andererseits die stärkere Netzwerkbelastung. Ein Wert für eine geeignete Messpause ist im Wesentlichen von der Anzahl der Agentenserver (mit denen Messexperimente durchgeführt werden) und den Netzwerkeigenschaften abhängig. Die Laufzeit einer Messung (Transferzeit) ist dabei von der Größe des auszutauschenden Datenpakets, der Latenzzeit und der Geschwindigkeit des Netzwerks abhängig.

Für einen Belastungstest, welcher das Verhalten der Messungen mit unterschiedlichen Messpausen in einem Netzwerk zeigen soll, wurden die Messpausen schrittweise von 32 s bis auf nahezu 0 s verringert. Die Messungen wurden jeweils mit 2, 3 und 5 Agentenservern in einem Netzwerk durchgeführt. Das generelle Verhalten war in allen Experimenten gleich und soll in Abbildung 5.6 durch die Messungen mit 5 Agentenservern veranschaulicht werden.

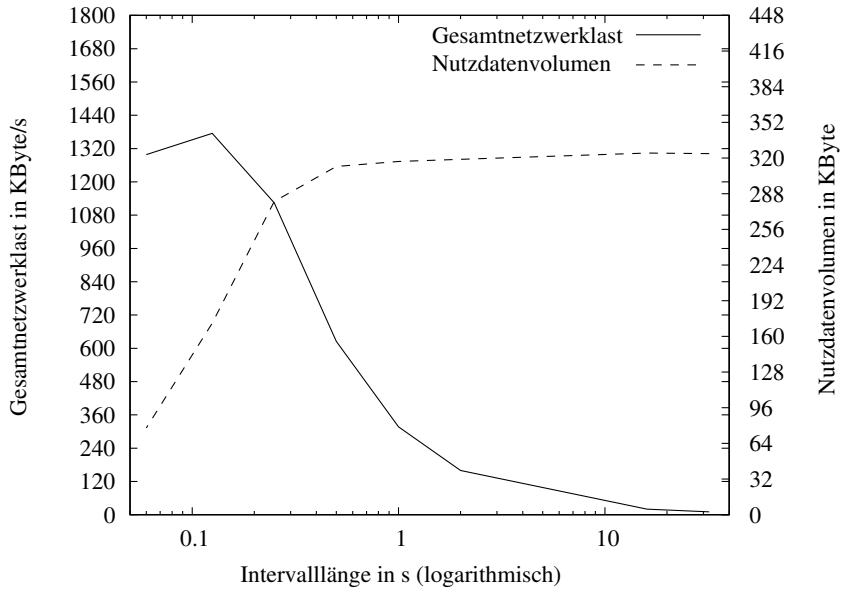


Abbildung 5.6: Belastungstest – Netzwerklast durch Transferratenexperimente

Die Gesamtnetzwerklast berechnet sich dabei folgendermaßen:

$$Gesamtnetzwerklast = \frac{Paketgröße * Zahl\ durchgeführter\ Messungen}{Gesamtdauer\ des\ Messexperiments}$$

Die verwendeten Testrechner waren mit 10 Mbit/s Full Duplex (Ethernet) angebunden. Bei der Messkonstellation wurden Datenpakete von 64 KByte Größe zwischen den fünf Messpartnern ausgetauscht. In jedem Messintervall wurden damit 320 KByte Daten über das Netzwerk versendet (Datenvolumen – von rechts nach links abfallend im Diagramm). Wird nun der Abstand zwischen den Messungen, die Messpause, halbiert, so sollte sich die Anzahl der Messungen bei gleich bleibender Gesamtdauer des Messexperiments verdoppeln, ebenso wie die im gesamten Netz verursachte Netzwerklast (Gesamtnetzwerklast – von rechts nach links aufsteigend im Diagramm). Das Datenvolumen pro Intervall sollte an sich gleich bleiben. Im angegebenen Diagramm ist dieses Verhalten bis zu einer Länge des Intervalls von etwa 0,5 s zu beobachten. Unter 0,5 s bricht das Datenvolumen zusammen und die Netzwerklast verdoppelt sich nicht länger.

Zur besseren Erklärung dieses Verhaltens muss die Intervalllänge betrachtet werden. Die Intervalllänge setzt sich aus der eigentlichen Messung und der Messpause zusammen. Die eigentliche Messung nimmt an sich immer die gleiche Zeit in Anspruch. Wird die Messpause halbiert, halbiert sich daher die Intervalllänge nicht. Bei großer Messpause fällt der Messanteil gering aus und eine Halbierung der Messpause scheint auch zu einer Halbierung der Intervalllänge zu führen. Bei kleiner Messpause ist der Messanteil an der Intervalllänge relativ groß und eine Halbierung der Messpause entspricht nicht annähernd einer Halbierung der Intervalllänge.

Der Einbruch des Datenvolumens von 320 KByte zeigt, dass die zur Übertragung der Datenpakete notwendige Zeit nicht mehr ausreichend, d. h. eine Intervalllänge von ca. 0,2 s zu kurz für die Messung ist. Die Messexperimente beginnen sich gegenseitig zu behindern und nicht alle Messexperimente können in der bisher benötigten Zeit fertig gestellt werden. Bei etwa 0,1 s ist die Behinderung der Messungen untereinander deutlich erkennbar. Die verursachte Netzwerklast erfährt ein Maximum und nimmt mit kürzeren Messabständen wieder ab, da die Kollisionen ständig zunehmen. Das System ist ausgelastet. Für diese Konstellation mit 5 Agentenservern stellt der Abstand zwischen den Messungen von etwa 0,5 s eine Grenze dar, welche die maximal sinnvolle Auslastung des Systems kennzeichnet und somit keinesfalls unterschritten werden sollte.

Anhand der Abbildung 5.6 auf der vorherigen Seite kann keine genaue Aussage über den Messabstand, ab dem sich die Messungen gegenseitig behindern, gemacht werden. Um diesen Punkt genauer festzustellen, werden die von den Agentenservern gemessenen Werte in einem weiteren Diagramm aufgezeigt (Abb. 5.7 auf der nächsten Seite). Durch die von den Agentenservern gemessenen Werte wird zudem die Qualität der Messungen selbst deutlicher. Im Diagramm sind die gemessenen Transferraten zwischen den

Agentenservern im Verhältnis zu der Intervalllänge dargestellt. Eingezeichnet sind die Messergebnisse von einem der fünf Server. Bei bereits ca. 1 s brechen die Transferraten zusammen. Das bedeutet, dass die Experimente sich gegenseitig zu behindern beginnen und dadurch die Messergebnisse verfälschen. Damit sollte bei dieser Konstellation die untere Schranke von etwa 1 s Abstand zwischen den Messungen nicht unterschritten werden.

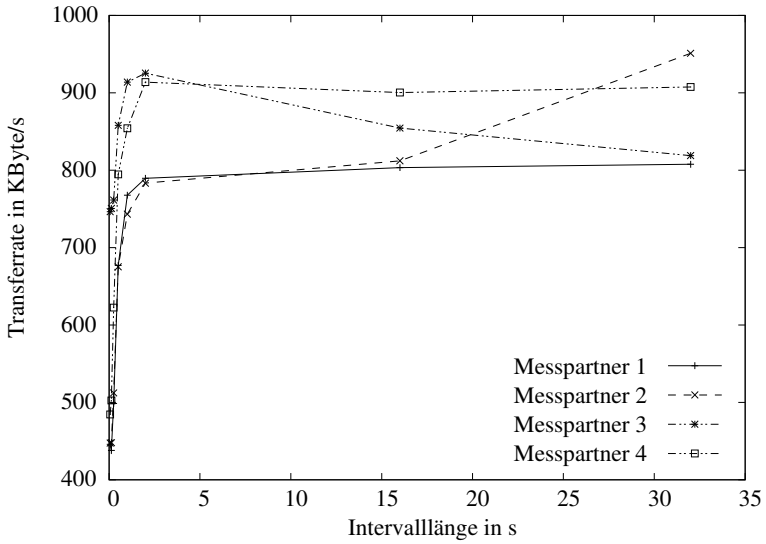


Abbildung 5.7: Belastungstest – Gemessene Transferraten des Sensors

Noch eine Bemerkung zur Gesamtnetzwerklast in diesem Zusammenhang: Normalerweise verteilt sich die von den Agentenservern verursachte Netzwerklast über das gesamte Netzwerk. Im ungünstigsten Fall treffen alle Messungen an einem Agentenserver zusammen, d. h. das gesamte Datenvolumen liegt an einem Agentenserver und ist nicht mehr verteilt. In diesem Fall könnte die Auslastung des Agentenservers die des Netzwerks zusätzlich begrenzen. Dieser Fall ist aber sehr unwahrscheinlich und mit der nächsten Messung ist das Datenvolumen wieder im Netz verteilt.

Neben der experimentellen Bestimmung einer Limit-Intervalllänge kann diese auch mittels theoretischer Berechnungen bestimmt werden. Wird eine Formel zur Berechnung der Gesamtnetzwerklast hergenommen, kann durch einfaches Umstellen der Formel die Intervalllänge bestimmt werden.

$$Gesamtnetzwerklast = \frac{Paketgröße \ * \ \#Server}{Intervalllänge}$$

$$Intervalllänge = \frac{Paketgröße \ * \ \#Server}{Gesamtnetzwerklast}$$

Bedingung für die Berechnung der Intervalllänge ist ein fixer, vorgegebener Wert der Gesamtnetzwerklast.

Die Aktualität der Karte wird bestimmt durch die Intervalllänge. Je kürzer die Intervalllänge gefasst wird, desto schneller aktualisiert sich die Karte. Wie ebenfalls bereits erwähnt, muss daher ein Kompromiss zwischen verursachter Netzwerklast und Aktualität der Karte gefunden werden. Wichtig ist daher, eine angemessene Zeit für einen Zyklus der Messexperimente zu finden. Die Zeit, die ein Agentenserver benötigt, um mit allen anderen Messpartnern ein Messexperiment durchzuführen, wird als Länge eines Aktualisierungszyklus oder als Aktualisierungszeit bezeichnet und berechnet sich aus:

$$Aktualisierungszeit = Intervalllänge \ * \ (\#Server - 1)$$

$$Aktualisierungszeit = \frac{Paketgröße \ * \ \#Server}{Gesamtnetzwerklast} * (\#Server - 1)$$

Während der Aktualisierungszeit werden von einem Agentenserver die Messungen zu allen anderen Agentenservern genau einmal durchgeführt. Dabei werden die Ergebnisse der vorangegangenen Messungen jeweils dem Messpartner übermittelt. Die Aktualisierungszeit hängt somit nicht nur von der Intervalllänge, sondern auch von der Anzahl der Agentenserver ab. Das Alter der Informationen auf der Karte übersteigt bei erfolgreich durchgeführten Messungen somit keine zwei Zyklen. Teilweise sind die Daten sogar nur einen Zyklus alt. Dies gilt für die vom Agentenserver selbst gemessenen Daten.

Am Beispiel der Konstellation mit den 5 Agentenservern ergibt sich somit als untere Schranke für die Aktualisierungszeit und das Alter:

$$Aktualisierungszeit = Intervalllänge * (\#Server - 1) = 1\,s * 4 = \mathbf{4\ s}$$

$$max.\ Alter = 2 \ * \ Aktualisierungszeit = \mathbf{8\ s}$$

In dieser Konstellation wird damit ein maximales Alter der Karteninformationen von 8 s erreicht. Die Netzwerklast, die im Normalfall im gesamten Netzwerk verteilt ist, beträgt

für diesen Fall (64 KByte Datenpakete) 320 KByte/s. Wird die gewünschte Netzwerklast halbiert, verdoppelt sich die Aktualisierungszeit und das Alter der Kartendaten. Wird die Anzahl der Agentenserver im Netzwerk erhöht, muss sich die Intervalllänge ebenfalls vergrößern, da sich sonst die Messungen gegenseitig behindern.

Wird der Fall betrachtet, dass die Anzahl der Agentenserver ansteigt, ist interessant, wie sich das System verhält, d. h. wie das System skaliert. Bei einem zusätzlichen Agentenserver entstehen $2 * n$ zusätzliche Messungen, wobei n der bisherigen Anzahl von Servern entspricht. Jeder Agentenserver führt in einem Messdurchlauf mit jedem anderen Agentenserver im Netz Messungen durch. Damit berechnet sich die Anzahl der Messungen pro Zyklus aus:

$$\#Messungen\ pro\ Zyklus = (\#Server - 1)\ *\ \#Server$$

Wie zu erkennen ist, sind die Messungen quadratisch abhängig von der Anzahl der Server.

Damit kann die Aktualisierungszeit bei bekannter Paketgröße und gewünschter Netzwerklast mit folgender Berechnungsvorschrift ermittelt werden:

$$Aktualisierungszeit = \frac{Paketgröße\ *\ \#Messungen\ pro\ Zyklus}{Gesamtnetzwerklast}$$

Im Diagramm in Abbildung 5.8 auf der nächsten Seite ist die berechnete Aktualisierungszeit für eine unterschiedliche Anzahl von Servern dargestellt. Die Paketgröße ist auf 64 KByte gesetzt. Die Netzwerklast, bzw. die Menge der Daten, die pro Sekunde im Agentennetzwerk unterwegs ist, beträgt 320 KByte/s. Damit wird die Anzahl der Messungen pro Sekunde auf 5 Messungen zwischen Agentenservern begrenzt. Im Diagramm ist deutlich die quadratische Abhängigkeit der Aktualisierungszeit von der Serveranzahl zu erkennen. Die Ursache ist die quadratische Anzahl von logischen Verbindungen ($(\#Server-1)*\#Server$), die zwischen den Agentenservern existieren. Um so wichtiger ist die Reduzierung der Serveranzahl auf eine handhabbare Größe. Aus diesem Grund ist das Domain Konzept eingesetzt worden, um die Menge der Agentenserver auf der Webkarte zu unterteilen und damit auf eine beherrschbare Größe zu reduzieren.

Unschärfe der Karte

Die Unschärfe der Karte schränkt einen Agenten bei der Erfüllung seiner Aufgabe ein. Das der Agent trotz dieser Einschränkung seine Aufgabe erfüllen kann, soll bei der Evaluierung gezeigt werden. Eine experimentelle Evaluierung kann allerdings nicht durch-

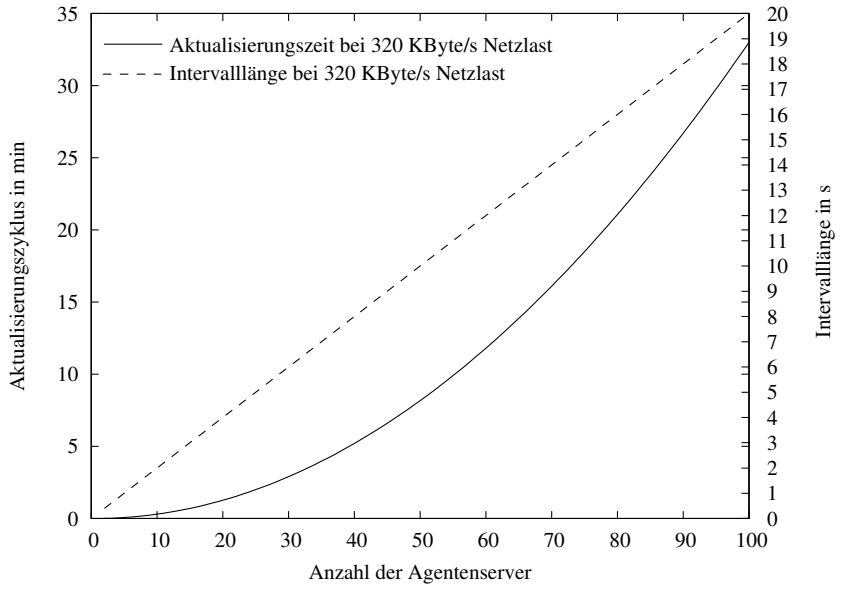


Abbildung 5.8: Aktualisierungszeit der Karte

geführt werden. Dazu müsste ein entsprechend umfangreiches Agentennetzwerk existent sein. Daher wird diese Evaluierung auf eine Diskussion zur Unschärfe der Karte reduziert. Ziel dieser Diskussion ist es, herauszustellen, dass einerseits auf unscharfen Kartenteilen genügend Informationen für den Agenten vorhanden sind und andererseits die Webkarte mit den unscharfen Kartenteilen für weitere Module von *ProNav*, den Routenplaner und den Migrationsoptimierer, verwendet werden kann.

Mit der Anzahl der Verbindungen wächst ebenso die Anzahl der auf einer Karte zu speichernden Verbindungsdaten. In einer Domain mit beschränkt vielen Knoten ist dies noch handhabbar. Würden diese kompletten Verbindungsdaten noch für jeden Agentenserver aus einer oder gar mehreren anderen Domains gespeichert werden, würde die Menge an Kartendaten die Agentenserver enorm belasten. Die Menge an Daten erlangt eine Größe, die nicht mehr handhabbar ist. Daher werden nur die Verbindungsdaten zwischen Domain-Managern sowie zusammengefasste, stark reduzierte Informationen über eine Domain (zusammengefasste Domainkartendaten) auf der Karte gespeichert. Dies wird als Unschärfe der Webkarte bezeichnet.

Es stellt sich die Frage, ob die zusammengefassten Domainkartendaten ausreichend sind, um einen Agenten bei der Lösung seiner Aufgabe behilflich zu sein. Die Reduzierung der Daten bezieht sich im Wesentlichen auf die Verbindungsinformationen innerhalb einer fremden Domain. Umfangreiche und detaillierte Verbindungsdaten über eine fremde Domain sind für einen Agenten noch nicht bedeutend. Der Agent muss auf Grund der Informationen einer fremden Domain, z. B. angebotene Dienste, erkennen können, ob eine

Migration zu dieser Domain für die Lösung seiner Aufgabe hilfreich ist. Diese Informationen sind auf der zusammengefassten Domainkarte verfügbar. Die Verbindungsinformationen innerhalb der fremden Domain werden vorerst nicht benötigt. Der Agent kann anschließend zur fremden Domain migrieren und dort die Verbindungsdaten der Domain abfragen.

Bei der Zusammenfassung der Kartendaten einer Domain kann daher ein Großteil der Daten entfernt werden. Damit wird der quadratische Anteil der Verbindungsdaten einer Domainkarte eliminiert und eine enorme Reduzierung erreicht.

Allerdings dürfen diese Informationen nicht komplett eliminiert werden. Für Berechnungen des Routenplaners oder Migrationsoptimierers, die auf den Kartendaten aufbauen, sind jedoch Verbindungsdaten innerhalb fremder Domains von Bedeutung. Zu diesem Zweck werden bei der Komprimierung der Kartendaten statistische Repräsentanten der Verbindungsdaten ermittelt und zusätzlich auf die zusammengefasste Domainkarte aufgenommen.

Die statistischen Repräsentanten werden aus den Verbindungsdaten des Domain-Managers gebildet. Hierfür werden aus den Daten, die vom Domain-Manager selbst ermittelt wurden, verschiedene statistische Werte berechnet, wie beispielsweise Minimum, Maximum, Median etc.

Die Kartendaten können sogar noch weiter reduziert werden. Für den Agenten ist zur Lösung seiner Aufgabe eine genaue Angabe des Orts (der Name des Agentenservers) in der fremden Domain noch nicht erforderlich, d. h. der Knoten selbst ist nicht interessant, sondern nur die den Knoten qualifizierende Information (vorhandene Dienste, Fähigkeiten). Daher werden bei der Reduzierung ebenso die Namen der Agentenserver entfernt. Zur Lokalisierung von fremden Domains mit interessanten Diensten genügt diese Information. Damit kann der Agent den Dienstort nicht mehr direkt anspringen. Jedoch ist der Domain-Manager, der die Domain repräsentiert, dem Agenten bekannt. Der Manager kennt wiederum den genauen Dienstort in der Domain. Der Agent muss damit einen zusätzlichen Zwischenstopp auf dem Manager der Domain einlegen und den Dienstort erfragen. Im Übrigen entspricht dies dem Verhalten, das in der realen Welt zu beobachten ist: Ein Mensch orientiert sich auf einer Landkarte. Hat er als Grobziel die Stadt erreicht, orientiert er sich beispielsweise auf einem detaillierten Stadtplan.

Zusammenfassend reduziert sich die Karte einer fremden Domain zu einer Liste von Eigenschaften: Knoteninformationen aller Domain-Knoten sowie die statistischen Repräsentanten der Verbindungsinformationen. Diese zusammengefassten Domainkarten werden auf der Karte des Agentenservers in der Liste der Domains (NodesList der remote domains) als Eigenschaften (Properties) gespeichert (siehe Abb. 4.4 auf Seite 65).

Die Grundlage für die Berechnungen des Routenplaners und des Migrationsoptimierers ist über Domaingrenzen hinweg möglich, da trotz der Unschärfe der Karte genügend In-

formationen vorhanden sind. Der Agent kann die übertragene Aufgabe erfüllen und für die weiteren Module von *ProNav* sind alle Informationen vorhanden, um auf den Kartendaten basierende Berechnungen durchzuführen.

Fazit Karte

Die Evaluierung der Karte hat gezeigt, dass die erreichte Aktualisierungszeit in für ein Agentensystem akzeptablen Bereichen liegt und die Unschärfe der Karte einen Agenten bei der Lösung seiner Aufgaben nicht behindert. Des Weiteren kann die Aktualisierung der Kartendaten an die Dynamik des Agentennetzwerks in gewissen Grenzen angepasst werden. Die Grenzen werden dabei durch die zur Verfügung stehende Nutzdatenrate sowie durch die quadratische Abhängigkeit von der Agentenserveranzahl gesetzt. Die Evaluation zeigt damit ebenso, wie wichtig eine Organisation der Infrastruktur ist. Die Menge der Agentenserver muss zwangsläufig partitioniert werden, um sie beherrschbar zu machen. Zu diesem Zweck wurde der Domain Service gewählt.

Durch das Zusammenfassen der Domainkartendaten entstehen unscharfe Kartenteile, die dennoch genügend Informationen für einen Agenten beinhalten. Zusammen mit den Verbindungsdaten zwischen den Domains können diese komprimierten Kartendaten für eine Routenplanung und eine Migrationsoptimierung genutzt werden.

Alles in allem erweist sich die Webkarte als tauglich für ein Agentensystem. Die erwartete Dynamik eines solchen Systems liegt in Bereichen, die das Kartenmodul erfassen kann. Bei einer typischen Domaingröße von ca. 60 Knoten ist eine Aktualisierung der Kartendaten in ca. 10 min (siehe Abb. 5.8 auf Seite 104) möglich. Bis zu etwa dieser Kartengröße skaliert das System sehr gut. Bei einer Größe von 100 Knoten ist die Aktualisierungszeit noch vertretbar. Damit kann die These 2a dieser Arbeit für die Webkarte glaubhaft vertreten werden:

T2a Die Dynamik moderner Netzwerke kann auf Basis einer Webkarte und eines Routenplaners adäquat gehandhabt werden.

5.1.4 Fazit Kartenmodul

Die Evaluation der Sensoren hat gezeigt, dass die Qualität der Verbindungsinformationen auf der Karte generell sehr hoch ist. Die Round Trip Zeiten, die vom Latenzsensor gemessen wurden, sowie die ermittelten Transferraten spiegeln die realen Charakteristika des Netzwerks gut wider.

Die Aktualität der Kartendaten ist eng gekoppelt an die Anzahl der notwendigen Messungen im Netzwerk. So ist innerhalb einer Domain die Aktualität der Daten bei einer typischen Anzahl von Domain-Knoten hoch. Doch mit zunehmender Knotenanzahl fällt bei

den notwendigen Messungen zunehmend die quadratische Abhängigkeit von den Knoten ins Gewicht. Bei genauer Betrachtung ist diese hohe Messanzahl jedoch nicht notwendig, da die logische Vernetzung (jeder ist mit jedem verbunden) die zugrunde liegende Topologie und damit die reale, physische Vernetzung der Rechner nicht widerspiegelt. Die physische Vernetzung steht im linearen Verhältnis zu der Anzahl der Rechner im Netz und kann daher als Anhaltspunkt für die Organisation der Messungen auf logischer Ebene sein. Damit skaliert das System auch bei höherer Knotenanzahl.

In einer Diplomarbeit [Adl03] hat sich zudem die Anwendbarkeit der Webkarte in einem kleinen Beispielszenario gezeigt. Die Anwendung mit dem Namen „Trusted Scientists“ vernetzt Wissenschaftler, die ihre Arbeiten in einem geschlossenen Anwenderkreis publizieren und austauschen wollen. Zur Suche nach Dokumenten werden mobile Agenten benutzt. Diese finden die Informationen, die für eine Recherche im Agentennetzwerk notwendig sind, auf der Webkarte. Hat ein Agent Dokumente gefunden, so können diese nach weiteren passenden Referenzen durchsucht werden. Wurden passende Referenzen gefunden, kann der Agent auch nach diesen referenzierten Dokumenten im Agentennetzwerk suchen.

Zwischen den Domains werden Karteninformationen ausgetauscht, wodurch die beschriebene Unschärfe für entfernte Domains auf der Karte entsteht. Die Daten über die entfernten Domains sind aber dennoch ausreichend für die autonome Routenplanung des Agenten. Die Quantität der Informationen ist somit auf eine Menge reduziert, die eine adäquate Kartographierung ermöglicht. Zudem wird durch die Partitionierung des Agentennetzwerks in Domains die Dynamik handhabbar. Die Webkarte ist somit ein Werkzeug des Agenten zur Wahrnehmung der aktuellen, virtuellen Umwelt. Daher sind mit der These 2 sinnvolle Annahmen formuliert worden. Die Behauptung der These 2a zur Quantität und Dynamik ist durch die Einteilung in Domains bestätigt. Die These 2b zur Qualität wird im folgenden Kapitel gezeigt. Damit sind insgesamt die Thesen zu 2 glaubwürdig vertreten.

5.2 Routenplaner

Die Routenplanung des Agenten soll einerseits in möglichst kurzer Zeit erfolgen und andererseits zu einer nahezu optimalen Reiseroute führen. Die für den Routenplaner gewählten Algorithmen sollen in dieser Evaluation ihre Tauglichkeit unter Beweis stellen.

5.2.1 Vorbetrachtungen

Die Evaluation des Routenplaners ist experimentell schwierig. Eine entsprechende Testumgebung müsste aus einer Vielzahl von Agentenservern und damit von Rechnern be-

stehen. Für diese Testumgebung müsste des Weiteren der optimale Weg bekannt sein. Es sind leider nicht die entsprechend nutzbaren Kapazitäten vorhanden, um eine derartige Testumgebung zu etablieren. Daher sind in der Evaluation nur zwei Messexperimente beschrieben, die mit relativ wenig Agentenservern durchgeführt wurden.

Im zweiten Teil dieser Evaluation wurden weitere Untersuchungen mit Hilfe von Testinstanzen und realen Instanzen durchgeführt. Die Testinstanzen sind generiert und so gewählt, dass die Problemklasse vergleichbar zu der Routenplanung im Agentensystem ist. Eine entsprechend hohe Anzahl an Zielpunkten kann gewählt werden und die Ergebnisse der Berechnungen können mit anderen Algorithmen gut verglichen werden. Die Algorithmen können ebenfalls ihre Fähigkeit an realen Instanzen, auch Realworld-Instanzen genannt, unter Beweis stellen. Diese Instanzen sind Beispiele für das TSP aus der realen Welt.

5.2.2 Experimentelle Untersuchungen

Das Erste, im Folgenden beschriebene Experiment soll zeigen, ob die Routenplanung zu einer Verkürzung der Reisezeit eines Agenten führt. Wie hoch eine Verbesserung ist, hängt dabei sicher von den Unterschieden in den Verbindungsqualitäten ab. Das zur Verfügung stehende Netzwerk von Rechnern weist allerdings wenig Unterschiede in den Verbindungsqualitäten auf. Daher wurde ein inhomogenes Netzwerk künstlich erzeugt.

Im zweiten Experiment wurde der Routenplaner getestet. Zu diesem Zweck wurden Routenberechnungen für ein kleines Netzwerk mit verschiedenen Algorithmen durchgeführt. Die Ergebnisse der Berechnung wurden mit dem Optimum, welches hier ein Minimum darstellt, verglichen.

Forcierter Ring Als Testumgebung wurden drei Rechner gewählt, die über ein Ethernet mit 100 Mbit/s Full Duplex vernetzt sind. Auf jedem Rechner wurde ein Agentenserver gestartet. Die Verbindungen zwischen den drei Agentenservern a_1, a_2 und a_3 wurden mit dem Werkzeug Traffic Control [Hub04] beschränkt. Zwei Laufrichtungen wurden erlaubt, wobei die Startplattform immer a_1 war:

R1 $a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_1$

R2 $a_1 \rightarrow a_3 \rightarrow a_2 \rightarrow a_1$

Die Richtung R2 wurde fix auf eine Transferrate von 500 Kbit/s beschränkt. Bei der Richtung R1 wurde die Beschränkung variabel gestaltet. Das Limit wurde stufenweise von 500 Kbit/s auf 16 Mbit/s erhöht. In den Diagrammen 5.9 und 5.10 wurde das Verhältnis zwischen langsamer und schneller Richtung auf der x-Achse aufgetragen.

Im Experiment wurde ein 100 KByte großer mobiler Agent benutzt, der entweder eine zufällige Route wählte oder die optimale Route lief. Die in den Diagrammen aufgetragenen Werte sind Durchschnittswerte aus den Testläufen.

Im Diagramm 5.9 sind die Zeiten für die zufällige Auswahl der Route (random) und für die optimale Route (Optimal) dargestellt. Ist das Netzwerk homogen bezüglich der Transferraten, ist eine zufällig gewählte Route erwartungsgemäß genauso gut wie eine berechnete, optimale Route. Je größer der Unterschied in den Transferraten der Verbindungen wird, desto sinnvoller wird die Benutzung eines Routenplaners für den Agenten. Bei geringen Unterschieden kann jedoch der zusätzliche Aufwand der Routenberechnung zu groß sein. Bestenfalls erkennt der Routenplaner selbst, wenn sich eine Berechnung nicht lohnt.

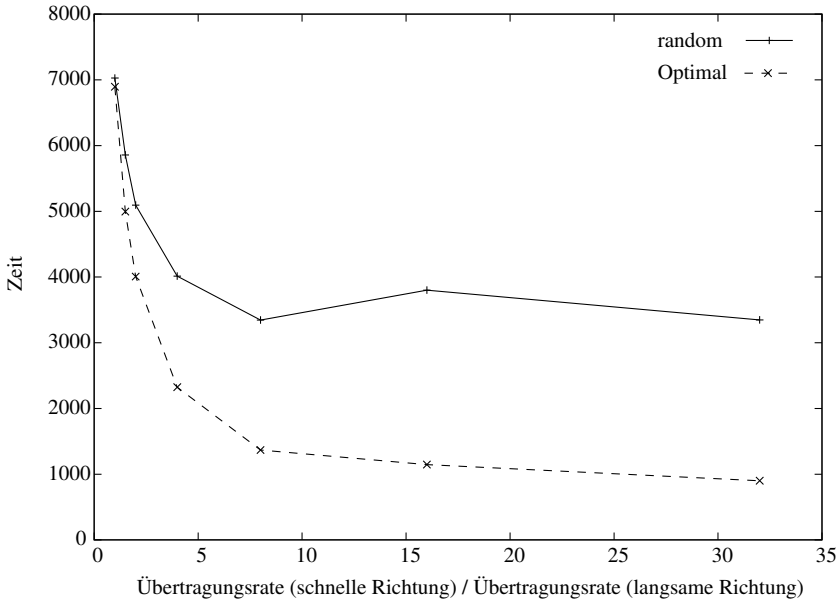


Abbildung 5.9: Zufällige vs. optimierte Route

Im Diagramm 5.10 auf der nächsten Seite sind die Zeiten für die zufällige Auswahl der Route (random) und die optimale Route im Verhältnis zueinander dargestellt. Die Beschleunigung ist deutlich zu erkennen. Bei 5-fachen Unterschied in der Nutzdatenrate ist der Agent mit der optimalen Route doppelt so schnell wie mit einer zufällig gewählten Route.

Test des Routenplaners Für den Test wurde ein Netzwerk aus sechs Rechnern gewählt, auf dem jeweils ein Agentenserver gestartet wurde. Auf Grund des homogenen Netzwerks wurden die Nutzdatenraten der Verbindungen zwischen den Agentenservern zufällig im

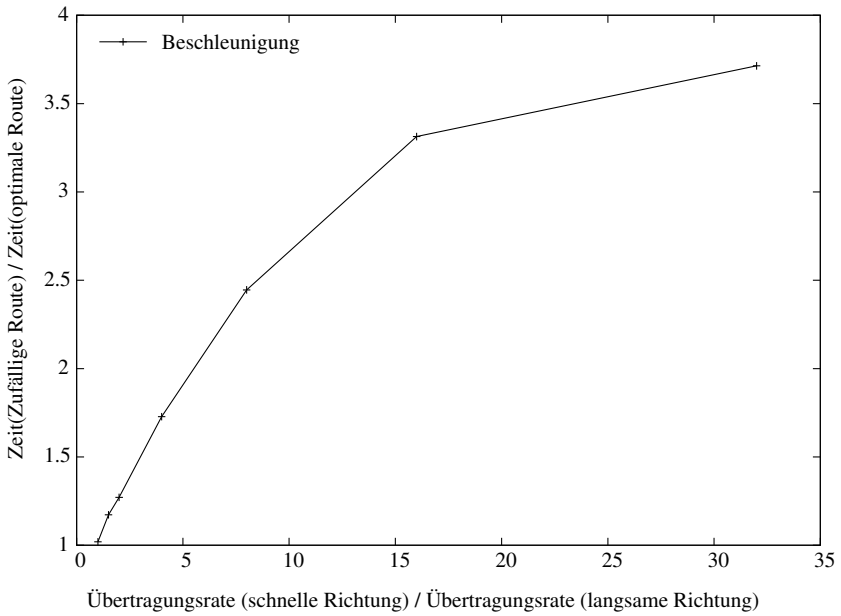


Abbildung 5.10: Beschleunigung bei optimierter Route

Bereich von 500 Kbit/s bis 16 Mbit/s beschränkt. Die von den Agentenservern bzw. von den Sensoren ermittelte Karte diente als Basis für die Routenberechnung. Folgende Algorithmen wurden zur Routenberechnung benutzt und verglichen:

- random

  zufällige Reihenfolge der Server,

- Random

  zufällige Reihenfolge der Server unter Verwendung kürzester Wege zwischen den Servern,

- NearestNeighbour

  Nächster-Nachbar-Algorithmus,

- AIOpt3P

  Iterierter 3-Opt mit Patch-Startroute und $0,5 * n$ Iterationen ($n =$ Anzahl der Server).

Die Ergebnisse sind in Tabelle 5.4 auf der nächsten Seite dargestellt. Für die Algorithmen ist sowohl die prozentuale wie auch die zeitliche Abweichung von der optimalen Route in der Tabelle zu sehen. Diese Werte verdeutlichen die Qualitäten der Algorithmen. Erwartungsgemäß schneidet der AIOpt3P sehr gut ab.

Algorithmus	**% über dem Optimum**	**Zeit über dem Optimum** [ms]
random	64.0%	720
Random	29.6%	335
NearestNeighbour	9.2%	106
AIOpt3P	0.277%	2

Tabelle 5.4: Routenplaner – Qualität der Ergebnisse verschiedener Algorithmen

Die Berechnungszeit für die Reiserouten ist bei dieser geringen Anzahl von Zielen vernachlässigbar klein und liegt im Nanosekundenbereich.

5.2.3 Untersuchungen mit Testinstanzen

Auf Grund einer fehlenden realen Messumgebung wurde zum Test der implementierten Algorithmen auf verschiedene Testinstanzen zurückgegriffen. Mit diesen Instanzen können Aussagen über die Geschwindigkeit der Algorithmen gemacht werden. Im Vordergrund der Evaluation steht der favorisierte AIOpt3P Algorithmus. Die Berechnungen mit den Testinstanzen wurden auf einem Pentium II mit 333 MHz durchgeführt. Als Betriebssystem wurde Linux mit installiertem JRE 1.4 verwendet.

Generierte Testinstanzen Der implementierte ATSP-Algorithmus AIOpt3P, der eine Patch-Startroute ermittelt und anschließend den 3-Opt-Algorithmus iteriert, wird mit den folgenden TSP-Instanzen (siehe auch [JGM+02]) getestet:

- tmat

  zufällige, asymmetrische Matrizen, die nur kürzeste Wege enthalten,

- super

  zu einer gegebenen Zahl von Ketten wird die annähernd kleinste, gemeinsame Superkette gesucht (Problem aus der Gen-Rekonstruktion),

- coin

  In einer Stadt mit einem rechtwinkligen Straßensystem (nach amerikanischen Vorbild) soll das Geld aus gleichmäßig in der Stadt verteilten Zahltelefonen abgeholt werden.

Diese Testinstanzen wurden gewählt, da bei diesen die Dreiecksungleichung zutrifft und ein hoher Symmetrieanteil enthalten ist. Für die generierten Testinstanzen wurde eine

Algorithmus	% über HK			**Zeit** [s]		
	tmat100	**super100**	**coin100**	**tmat100**	**super100**	**coin100**
$0,5 * n$ Iterationen						
AIOpt3P	0,45%	0,84%	9,11%	0,30	0,30	0,18
AIOpt3	3,77%	1,43%	9,51%	0,18	0,18	0,17
$10 * n$ Iterationen						
AIOpt3P	0,21%	0,33%	4,12%	2,75	2,79	2,66
AIOpt3	0,91%	0,53%	4,29%	2,66	2,71	2,66

Tabelle 5.5: Die TSP-Algorithmen AIOpt3P und AIOpt3 im Vergleich

Größe von 100 gewählt. Die Route des Agenten wird in praktischen Anwendungen in dieser Größenordnung liegen. In der Tabelle 5.5 sind die Ergebnisse aufgetragen.

Die Tabelle zeigt für die beiden TSP-Algorithmen die erreichte Qualität und die dafür benötigte Zeit bei den unterschiedlichen Testinstanzen. Da eine optimale Route für die generierten Instanzen nicht bekannt ist, wird die Qualität mit Hilfe der Held-Karp-Lowerbound (HK), einer über lineare Gleichungssysteme ermittelten unteren Schranke, bestimmt. In der Tabelle ist jeweils der Prozentsatz über HK für die Algorithmen angegeben.

Die Anzahl der Iterationen wurde auf $0,5 * n$ und auf $10 * n$ der Problemgröße n festgelegt. Bei einem Vergleich der erzielten Werte wird deutlich, wie Aufwand und Nutzen im Verhältnis stehen.

Beide Algorithmen nutzen den iterierten 3-Opt-Algorithmus. Die Startroute wird bei AIOpt3 mit Hilfe des NearestNeighbour ermittelt und bei AIOpt3P wird der Patch-Algorithmus genutzt. In [JGM+02] sind entsprechende Vergleichswerte des I3Opt für die Instanzen zu finden. Dadurch ist ein direkter Vergleich des implementierten I3Opt möglich. Die implementierte Variante ist um 0,3% bis 1,3% schlechter als die in [JGM+02] präsentierte (Vergleich hier nicht präsentiert). Trotzdem ergibt sich durch die Kombination des I3Opt mit dem Patch-Algorithmus ein so großer Qualitätsgewinn, dass AIOpt3P im Fall von tmat100 besser abschneidet (I3Opt: 0,30% über HK bei $10 * n$ Iterationen). Bei super100 schneidet AIOpt3P fast genauso gut ab wie I3Opt (I3Opt: 0,28% über HK bei $10 * n$ Iterationen).

Weiterhin fällt auf, dass die Qualität der gefundenen Lösungen bei der fast symmetrischen Instanz coin100 sehr schlecht ist. Asymmetrische TSP-optimierte Algorithmen liefern meist schlechtere Ergebnisse für symmetrische Probleme im Vergleich zu den symmetrischen Verwandten. Illustriert werden kann dies durch den Versuch den kürzesten Weg durch alle 127 Biergärten von Augsburg (symmetrisches TSP) zu finden (siehe Abb. 5.11).

Im Diagramm sind für dieses STSP die berechneten Lösungen des iterierten 3-Opt (I3Opt) für STSP mit einer NearestNeighbour-Startroute, des AIOpt3 (ebenfalls NearestNeigh-

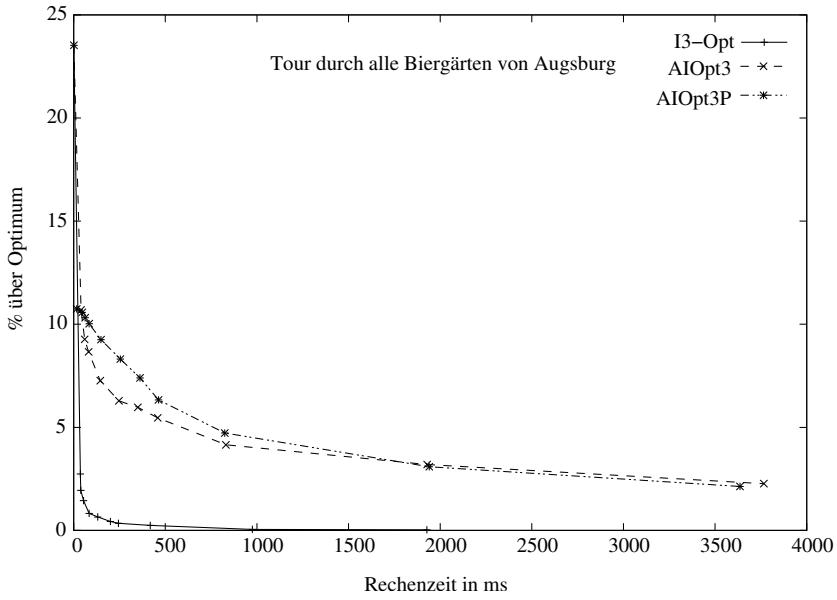


Abbildung 5.11: Vergleich von symmetrischen und asymmetrischen Algorithmen für STSP

bour-Startroute) und des AIOpt3P (Patch-Startroute) dargestellt jeweils für ATSP. Zu erkennen ist in diesem Diagramm, dass die erzielten Ergebnisse der ATSP-Algorithmen die des STSP-Algorithmus nicht erreichen.

Realworld-Instanzen Der AIOpt3P wurde zudem an Realworld-Instanzen getestet. Die Instanzen ftv110 (eine verkleinerte Variante des ftv170-Problems) stammen von einem Lieferproblem für pharmazeutische Produkte in Bologna. Der Vorteil des gewählten Patch-Algorithmus wird durch das Diagramm in der Abb. 5.12 auf der nächsten Seite deutlich. Dieser Algorithmus liefert sehr gute Startrouten für das ATSP. Die gefundene Startroute liegt nur ca. 3% über dem Optimum. Der anschließend arbeitende iterierte 3-Opt nähert sich schnell dem Optimum an. Der typische Berechnungsverlauf ist im Diagramm gut zu erkennen.

5.2.4 Fazit Routenplaner

Die gewählte Kombination aus dem Patch-Startalgorithmus und einem iterierten 3-Opt-Algorithmus führt zu qualitativ guten Ergebnissen bei der Berechnung einer Route. Ist eine stärkere Asymmetrie in den Instanzen vorhanden, liefert der Patch-Algorithmus eine gute Startroute. Tendieren die Instanzen zur Symmetrie, schneidet der asymmetrisch optimierte iterierte 3-Opt-Algorithmus dennoch gut ab.

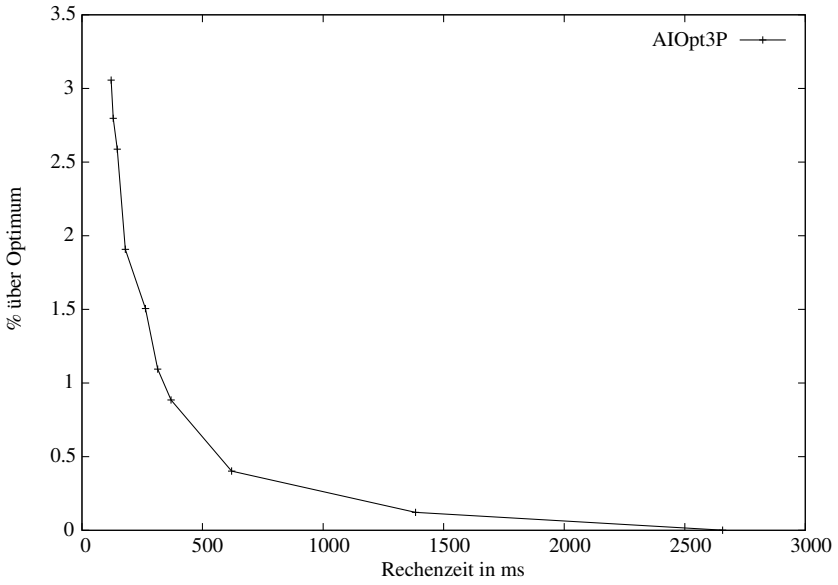


Abbildung 5.12: ATSP-Algorithmus AIOpt3P im Test mit ftv110

Die Möglichkeit, die Anzahl der Iterationen zu steuern, ist besonders wichtig, wenn der Aufwand der Routenplanung und das erzielte Ergebnis miteinander abgeglichen werden sollen. Im Routenplaner ist dazu eine Variante des AIOpt3P-Algorithmus, der AIOpt3PTOpt-Algorithmus, implementiert, der solange optimiert, wie der Routenzeitgewinn in den letzten 20 ms größer als die aufgewendete Rechenzeit ist.

Je nachdem, auf welche Probleminstanz gestoßen wird, übernimmt entweder der Patch-Algorithmus oder der I3Opt-Algorithmus die Hauptarbeit. Ist die Instanz für den Patch-Algorithmus geeignet, wird die Route im ersten Schritt stark optimiert. Die nachfolgenden I3Opt-Schritte bringen keinen großen Gewinn mehr und das Verfahren bricht ab. Ist die Instanz schlecht für den Patch-Algorithmus geeignet, dann wird die Startroute relativ lang sein, sich aber durch nachfolgende I3Opt-Schritte stark optimieren lassen – solange bis die Optimierungen keinen lohnenden Zeitgewinn mehr einbringen. Für den Fall, dass ein nahezu homogenes Netzwerk vorliegt, in dem Routenplanung generell kaum zu Verbesserungen führt, wird dadurch keine unnötige Zeit auf die Berechnung verschwendet.

Die Evaluation des Routenplaners hat gezeigt, dass existierende TSP-Algorithmen zur Routenplanung in einem Agentensystem verwendet werden können. Damit die Route effizient ermittelt werden kann, muss das Quantitätsproblem im Netzwerk gelöst sein. Die Unterteilung des Agentennetzwerks in Domains ist damit zwingend für die Routenplanung und der Qualität der Route notwendig. Unter dieser Voraussetzung ist die Planung einer Route für einen Agenten durch ein Agentennetzwerk effizient durchführbar, d. h. ist in realen Applikationen anwendbar und die zur Planung benötigten Zeiten fallen nicht ins Gewicht. Maximal 100 Agentenserver werden als Ziele im Netz für einen Agenten erwar-

tet. Für derartige Größenordnungen führt die Routenplanung in nur kurzer Zeit zu sehr guten Ergebnissen: In weniger als 0,5 s liegt die bis dahin berechnete Route bereits unter 1% über dem Optimum (110 Ziele – Realworld-Instanzen ftv110, siehe Abb. 5.12 auf der vorherigen Seite). Die These 2b dieser Arbeit ist somit gefestigt:

T2b Für einen bestimmten mobilen Agenten ist die Planung einer Route durch das Netzwerk effizient durchführbar.

5.3 Migrationsoptimierer

Bei der Vorbereitung einer Reise eines Agenten kann neben der Planung der Reiseroute ein weiterer Optimierungsschritt durchgeführt werden. Bei diesem steht der Agent im Vordergrund, für den verschiedene Möglichkeiten für die Art und Weise der Migration existieren (Push-, Pull-Strategien, siehe Kapitel 4.4). Ziel ist die Reduzierung der Netzwerklast und damit ebenso die Beschleunigung der Reise. Unter Umständen wird der Agent nicht vollständig verschickt, sondern nur Teile des Agenten.

5.3.1 Vorbetrachtungen

Der Migrationsoptimierer, der speziell für das Agentensystem TRACY entwickelt wurde, muss auf Basis der Kartendaten sowie einer vorgegebenen Route und unter Einbeziehung des konkreten Agenten die Migration dieses Agenten optimieren. Die zur Optimierung notwendigen Informationen müssen einerseits vollständig vorliegen und andererseits eine möglichst hohe Qualität aufweisen. Eine Optimierung, die sich im Nachhinein als falsch erweist, kann den gegenteiligen Effekt erzielen.

Die Optimierung wird mit Hilfe des mathematischen Netzwerkmodells, das für die Optimierung angeglichen wurde, durchgeführt. Daraus ergeben sich die zur Optimierung notwendigen Informationen:

- Route des Agenten,
- Transferraten und Latenzzeiten zwischen den Zielen der Route,
- Anzahl und Größe der (nicht transienten) Klassen des Agenten (Bytecode),
- Größe der Datenfelder des Agenten,
- Die Wahrscheinlichkeit, mit der Klassen des Agenten auf entfernten Agentenservern benötigt werden,

- Auf entfernten Agentenservern vorhandene Klassen,
- Informationen über unterstützte Protokolle der Agentenserver,
- Vorhandene Code und Mirror Server.

Bereits die Vielfalt der benötigten Daten verdeutlicht die mögliche Komplexität der Optimierung. Klar wird aber auch, dass für den Optimierer die Daten der Karte nicht ausreichend sind. Die fehlenden Daten müssen vom Agenten, von zusätzlichen, fremden Komponenten oder sogar vom Optimierer selbst ermittelt werden. Auf Grund dieser Situation musste der Hauptteil des Optimierers in die Migrationsschicht des Agentensystems integriert werden.

Die folgende Diskussion wird sich vorrangig mit der Ermittlung der notwendigen Informationen befassen und den daraus resultierenden Konsequenzen. Experimentelle Evaluierungsmessungen wurden für dieses Modul noch nicht durchgeführt.

5.3.2 Diskussion

Für eine Evaluierung werden die zur Optimierung notwendigen Informationen in drei Klassen unterteilt: Verbindungs-, Agentenserver- und Agenteninformationen. Bei der Ermittlung einer geeigneten Migrationsstrategie wurden diese Informationen zur Berechnung der Vergleichswerte benutzt. Es wird kurz diskutiert, in welchen konkreten Situationen diese Daten benötigt und wie die Informationen ermittelt werden. Danach wird auf Probleme bei der Realisierung eingegangen.

Verbindungsinformationen Die Verbindungsinformationen, wie Transferraten und Latenzzeiten, sind grundlegende Daten, die zur Ermittlung eines Vergleichswerts herangezogen werden müssen. Bei jeder Berechnung, sei es für eine konkrete Migrationsstrategie, eine Initialisierung von Code oder Mirror Server oder für den automatischen Transfer der ermittelten Daten des Agenten zur Heimatplattform, werden die aktuellen Verbindungsdaten zwischen den Agentenservern benötigt, um Transferzeiten zu ermitteln. Die Verbindungsinformationen werden von der Karte zur Verfügung gestellt und können somit einfach dem Optimierer übermittelt werden. Selbst hat der Optimierer keinen Zugriff auf die Karte, da er in einer tieferen Schicht des Agentensystems integriert ist. Das Auslesen übernimmt ein zusätzlicher Agent, der `MapAgent`, der von einem mobilen Agenten zur Optimierung kontaktiert werden muss. Die ausgelesenen Daten werden als Bestandteil des mobilen Agenten dem Optimierer übergeben.

Das Auslesen der Daten muss nicht unbedingt bei jeder Migration erfolgen. Je nachdem, welche Dynamik bei den Daten vorherrscht und ob Sprünge in eine neue Domain ge-

macht wurden, sollte der mobile Agent selbst entscheiden, wann die Informationen zu aktualisieren sind.

Agentenserverinformationen Ein Teil der Agentenserverdaten ist bereits auf der Karte zu finden. So werden dort Informationen zu den unterstützten Protokollen abgelegt. Diese Informationen sind für den Fall interessant, bei dem ein Agent die zu verwendenden Protokolle einschränkt und dadurch Abhängigkeiten zu Migrationsstrategien entstehen. Diese Abhängigkeiten werden allerdings in der momentanen Implementierung noch nicht berücksichtigt.

Ebenfalls müssen die auf einer Plattform vorhandenen Klassen auf der Karte aufgelistet werden, um auf Grund dieser Daten zum Optimierungszeitpunkt vor der Migration entscheiden zu können, welche Klassen nicht mit übertragen werden müssen. Um diese Informationen zu erhalten, müsste jede Plattform in regelmäßigen Abständen ihren Klassen-Cache auslesen und die Klassen mit der entsprechenden Versionsinformation in die Liste der Servereigenschaften aufnehmen. Potentiell können diese Informationen jedoch sehr umfangreich werden. Daher ist dieser Mechanismus bisher nicht implementiert worden. Das bedeutet aber auch, dass diese Informationen nicht auf diese Weise zur Verfügung stehen.

Informationen über auf einer Zielplattform vorhandene Klassen können dennoch abgefragt werden. Bei der Ausführung einer Migration werden unmittelbar vor dem Transfer der Klassen, als Bestandteil des Migrationsprotokolls, die zu transferierenden Klassen der Zielplattform mitgeteilt. Diese sendet als Antwort eine Liste bereits vorhandener Klassen, welche dann nicht mit transferiert werden. Leider ist zu diesem Zeitpunkt die eigentliche Optimierung bereits abgeschlossen, so dass diese Information nicht in die gesamte Optimierung einfließen kann. Abgesehen davon sind diese Klasseninformationen nur für das nächste Ziel verfügbar. Eine Optimierung über mehrere Zielplattformen der Reiseroute ist damit in TRACY nicht möglich.

Die auf den Zielplattformen vorhandenen Klassen können dem Optimierer dennoch durch den Agenten selbst mitgeteilt werden. Allerdings müsste der Agent diese Daten auf einer früheren Reise gesammelt haben. Damit ist die Optimierung erst bei der zweiten Reise möglich.

Agenteninformationen Die meisten Informationen für eine Optimierung betreffen den Agenten selbst. So sind teilweise Informationen im Agenten zu finden und teilweise Informationen über den Agenten zu berechnen.

Die wichtigste Information ist die Reiseroute. Bei den unterschiedlichen Optimierungsvarianten wird entweder die komplette Route zur Optimierung benutzt oder nur die nächste Plattform. Der Optimierer kann die Route direkt aus dem Agenten auslesen und hat sie

somit immer verfügbar.

Immer ermittelbar ist die Anzahl und Größe der Klassen des Agenten, die auf eine Reise mitgenommen werden müssen. Dazu wird der Bytecode des Agenten analysiert und rekursiv nach benötigten Klassen durchsucht (Java-Systemklassen sowie Agentensystemklassen werden ignoriert). Anschließend wird die Größe der Klassen (Bytecodegröße) ermittelt. Die notwendigen Funktionalitäten existierten bereits in der Migrationskomponente Kalong. Dies war ein Grund zur Integration des Optimierers in dieser Komponente. Die Werte sind bei der Berechnung der Transferzeiten relevant. Bei vielen Klassen können diese Berechnungen jedoch sehr aufwendig werden. Die Klassen müssen sogar zur Bestimmung der Bytecodegröße mit einem Classloader geladen werden. Der Aufwand für eine Optimierung kann dadurch relativ hoch werden.

Neben der Ermittlung der Klassen des Agenten ist für eine Partitionierung des Agenten die Wahrscheinlichkeit interessant, mit der diese Klassen auf entfernten Agentenservern benötigt werden. So sollten Klassen, die auf jeden Fall benötigt werden, in einem gemeinsamen Container verpackt und sofort verschickt werden – eventuell sogar zusammen mit Klassen, die mit sehr hoher Wahrscheinlichkeit benötigt werden. Klassen, welche dieselbe Verwendungswahrscheinlichkeit haben, sollten immer zusammen verpackt und bei Bedarf versendet werden. Das setzt natürlich voraus, dass die Wahrscheinlichkeiten für eine Benutzung bekannt sind. Diese Wahrscheinlichkeiten können sogar von der Zielplattform abhängig sein und werden bei einem Vergleich der Strategien benötigt. Bei geringen Nutzungswahrscheinlichkeiten bietet sich eine Strategie an, die nur Teile des Agenten versendet. Leider sind diese Informationen nur in Ausnahmefällen errechenbar. Vielmehr sollen solche Informationen durch Profiling des Agenten entstehen, das bisher nicht realisiert ist. Zudem wären diese Informationen für eine Optimierung erst nach mehreren Reisen des Agenten verfügbar.

Des Weiteren ist der Datenteil des Agenten interessant. Haben sich einige Daten angehäuft, die auf der weiteren Reise nicht benötigt werden, kann ein Transfer der Daten zum Heimatserver oder zu einem Mirror Server erfolgen. Datenteil und Status des Agenten sind normalerweise gekoppelt und bilden zusammen den serialisierten Agenten. Kalong hat hier Möglichkeiten geschaffen, den Datenteil vom Status zu trennen. Dadurch sind die Voraussetzungen geschaffen, um die Größe des Datenteils zu bestimmen und Entscheidungen über den Transfer zu treffen.

Bei den Vergleichsberechnungen wird oft die Verwendung von Code und Mirror Server mit einbezogen. Die Initialisierung neuer derartiger Server ist mit einem gewissen Aufwand verbunden. Hilfreich sind hier Informationen vom Agenten über bereits vorhandene Code und Mirror Server. Im Agenten müssen entsprechende Bereiche für diese Informationen vorgesehen werden.

Probleme Bei der Ermittlung notwendiger Informationen wurden schnell die Grenzen des bisherigen MAS TRACY erreicht, was auch eine Umsetzung im *ProNav*-System selbst stark eingeschränkt hat. Des Weiteren ist eine Optimierung bezüglich der Auswahl einer Migrationsstrategie erst sinnvoll, wenn der Agent aus mehr als nur einer Klasse besteht und die Agentenklassen eine nicht vernachlässigbare Größe besitzen. Meist ist dies erst der Fall, wenn sich ein Agent zu einem komplexen System entwickelt hat. Dies wiederum ist ein Applikationsproblem, das im Rahmen dieser Arbeit an *ProNav* nicht geklärt werden konnte.

Eine mögliche Optimierung beschränkt sich daher zurzeit auf die zur Verfügung stehende Information. Das sind im Wesentlichen die Transferraten, die Latenzzeiten und die Größen der Klassen.

Diese Daten reichen für eine erfolgreiche Optimierung aber nicht aus. Der folgende Code-Ausschnitt verdeutlicht das Problem:

```
// Calculate transmission times
  // push-all-to-next: Transmit units to next agency
T-patn := delay(Li,Lj)+unitsize/bandwidth(Li,Lj);

  // pull-all: Download units from home at next agency
T-pa   := delay(L0,Lj)+unitsize/bandwidth(L0,Lj);

  // pull-unit: Download each unit from home at next agency
T-pu   := delay(L0,Lj) + SUM(probability(k)*(unit(k)+request))
                       / bandwidth(L0,Lj);

  // Only for the first hop:
  // push-all-to-all: Distribute units from home to all agencies
for s in servers
   T-pata := T-pata + delay(L0,Ls) + unitsize/bandwidth(L0,Ls);

// Select migration stategy
T-min := T-patn;
T-min := T-patn;
MS := ‘‘push-all-to-next’’;
if    ( T-pa < T-min )  then T-min := T-pa;
                            MS := ‘‘pull-all’’;
elsif ( T-pu < T-min )  then T-min := T-pu;
                            MS := ‘‘pull-units’’;
else ( T-pata < T-min ) then T-min := T-pata;
                            MS := ‘‘push-all-to-all’’;
endif;
```

Das Beispiel berechnet die Vergleichswerte unterschiedlicher Migrationsstrategien für die Migration zum nächsten Ziel (niedriges Optimierungslevel). Fehlen die Informationen zu den Nutzungswahrscheinlichkeiten der Codeteile des Agenten, kann kein sinnvoller

Vergleichswert für die Pull-Unit-Strategie berechnet werden. Damit reduziert sich eine Strategieentscheidung meist auf die Auswahl von Push-All-To-Next und Pull-All, wobei Pull-All bei einem relativ homogenen Netzwerk durch den zusätzlichen Request (bei der Berechnung nicht berücksichtigt) meist schlechter abschneidet.

5.3.3 Fazit Migrationsoptimierer

Eine erfolgreiche Optimierung setzt möglichst viele und detaillierte Informationen über den Agenten, die Agentenplattformen und die Verbindungen voraus. Zur Ermittlung dieser Informationen sind teilweise aufwendige Berechnungen notwendig. Einige Daten können nicht ohne Weiteres ermittelt werden, bedingen unter Umständen ein Profiling des Agenten oder eine entsprechend komplexe Anwendung. Die Funktionsweise dieses Moduls konnte daher bisher nicht experimentell evaluiert werden. Die momentane Realisierung muss zur Beschaffung weiterer, notwendiger Informationen, die eine tiefere Optimierung ermöglichen, ergänzt werden. Dies wird mit TRACY-2, das zu dieser Zeit noch nicht fertig gestellt war, und mit zukünftigen Anwendungen möglich sein.

Bei der aktuellen Implementierung mussten einige Optimierungsinformationen, wie beispielsweise die Kartendaten, im Agenten integriert werden. Dies führt zu zusätzlichen Ballast für den Agenten und kann das eigentliche Ziel, weniger Daten zu transferieren, konterkarieren.

Zudem liegt die Vermutung nahe, dass eine Optimierung relativ aufwendig sein kann. Zur Erzielung eines möglichst guten Aufwand-Nutzen-Verhältnisses muss eine erfolgreiche Optimierung eines Agenten mehrfach genutzt werden. Das bedeutet, die einmalige, intensive Optimierung von Standardagenten, die mehrfach gleiche Aufgaben erfüllen, hat einen guten Nutzeffekt. Allerdings kann die Dynamik des Systems eine erneute Optimierung erfordern.

Generell ist eine Optimierung erst dann lohnenswert, wenn der Agent aus mehreren Klassen besteht und ein Größe erreicht hat, die entsprechende Transferzeiten bei den Übertragungen verursacht. Weiterhin fällt auf, das die Optimierung sehr stark vom Agenten selbst abhängt. Bei einer Optimierung wären Heuristiken wünschenswert, die, ähnlich wie beim Traveling Salesman Problem des Routenplaners, relativ schnell und mit angemessenem Aufwand eine Strategieempfehlung ermitteln.

Trotz allem ist eine Optimierung der Migration eines Agenten auf technischer Ebene realisierbar. Die existierende Implementierung hat den Weg zu einer Optimierungskomponente gezeigt.

5.4 Evaluation der Thesen

Die Evaluation der einzelnen Komponenten von *ProNav* hat gezeigt, dass die entwickelten Ideen dieser Arbeit und die aufgestellten Thesen realisierbar sind.

Wahrnehmung Die Webkarte, die ein Abbild des Agentennetzwerks darstellt, kann von einem (mobilen) Agenten zur Orientierung benutzt werden. Die Karteninformationen werden dem Agenten in einer geeigneten Form präsentiert und stellen ein sinnvolles Abbild seiner Umgebung dar. Somit ist der Agent zur Wahrnehmung der Umwelt über das Werkzeug „Webkarte“ befähigt und die grundlegende These 1 ist bestätigt:

T1 Mit Hilfe einer Webkarte wird ein Agent befähigt, seine Umwelt korrekt wahrzunehmen.

Quantität und Qualität der Wahrnehmung Die Umwelt ist für den Agenten durch die Karte effizient wahrnehmbar. Die Betrachtungen zu den Modulen von *ProNav* haben bereits gezeigt, dass die Thesen 2, 2a sowie 2b einer Prüfung standhalten. So ist die Erstellung einer Webkarte, die den Qualitätsansprüchen genügt, durch das Kartenmodul möglich. Diese Karte ist für eine Lokalisierung von Diensten in dynamischen Agentennetzwerken geeignet. Das Quantitätsproblem des Netzwerks ist durch das Domainkonzept lösbar. Der Agent kann somit proaktiv aktuelle Umweltsituationen wahrnehmen und rechtzeitig adaptiv darauf reagieren. Zur Planung können bekannte Heuristiken herangezogen werden, die innerhalb einer Domain und im Gesamtnetzwerk praktisch verwertbare Performance erbringen. Somit ist gezeigt:

T2 Eine Webkarte für Services ist für große Netzwerke möglich.

T2a Die Dynamik moderner Netzwerke kann auf Basis einer Webkarte und eines Routenplaners adäquat gehandhabt werden.

T2b Für einen bestimmten mobilen Agenten ist die Planung einer Route durch das Netzwerk effizient durchführbar.

Organisation Voraussetzung für diese Thesen ist wieder die Reduzierung der Quantität. Somit wird die These 3 zu einer zwingenden Schlussfolgerung der bisherigen Thesen. Es ist eine Organisation in einer Infrastruktur unbedingt notwendig, die auf eine geeignete Art die Menge an Elementen reduziert. Der Domain Service erlaubt eine Organisationsform, die diesen Ansprüchen gerecht wird. Die entstehende Kopplung der Agentenplattformen auf logischer Ebene ist Basis für den Routing Service *ProNav*. Zudem wird die Dynamik des Agentennetzwerks vom Domain Service berücksichtigt.

T3 Agentennetzwerke benötigen eine grundlegende Organisation der Infrastruktur, damit die mobilen Komponenten das gesamte Potential eines verteilten Systems nutzen können.

Verallgemeinerung Die autonome Planung einer Reise durch das Agentennetzwerk stellt sich daher als Aufgabe dar, die generisch und stark strukturiert orientiert ist, d. h. die Aufgabe ist keine für einen Agenten spezifische, sondern eine Querschnittsaufgabe. Eine individuelle Lösung durch jeden einzelnen Agenten scheidet als ineffizient aus, ist sogar teilweise unmöglich (z. B. Beschaffung der Karte) Eine vollkommen zentrale Lösung muss an den anfallenden Quantitäten scheitern. Sinnvoll ist daher nur eine Lösung als Querschnittsaufgabe, die von allen Agentenplattformen gemeinsam realisiert wird:

T4 Autonome und proaktive Routenplanung durch mobile Agenten ist als Querschnittsaufgabe *aller Agentenplattformen* in einem mobilen Agentensystem realisierbar.

Kapitel 6

Zusammenfassung und Ausblick

ProNav und seine Aufgaben

In dieser Arbeit wurde das generische Rahmenwerk *ProNav* vorgestellt. *ProNav* ist ein Aufsatz für mobile Agentensysteme, der das autonome und proaktive Navigieren mobiler Agenten in einem Netzwerk von Agentenplattformen gestattet.

Aus der Sicht der mobilen Agenten ist *ProNav* einer von vielen (Infrastruktur-)Diensten, die auf einer Agentenplattform aufgerufen werden können. Die spezielle Aufgabe von *ProNav* ist es, die Voraussetzung für die Lokalisierung von im Netzwerk verteilten Informationen und Diensten auf der Applikationsebene zu schaffen und diese effizient anzusteuern.

Dem Agenten selbst ermöglicht *ProNav* daher eine proaktive und autonome Routenplanung in der heterogenen und dynamischen Umgebung des Agentennetzwerks. Der Agent wird in der Routenplanung unabhängig von seinem Besitzer und auch seinem Programmierer. Er agiert und reagiert im Netz selbständig, indem er während der Erfüllung seiner benutzerdefinierten Aufgabe seine Route mit Hilfe von *ProNav* selbst findet, ständig anpasst und die Dynamik der Umgebung für sich nutzt, anstatt mit einer fix vorgegebenen Route an ihr zu scheitern.

Aus der Sicht eines mobilen Agentensystems ist *ProNav* eine Erweiterung der Architektur, die als zusätzliche Schicht zwischen dem eigentlichen Agentensystem und der Applikation geschaltet wird. Diese Schicht besteht aus mehreren Komponenten: der Webkarte, dem Routenplaner und dem Migrationsoptimierer. Jede Komponente kann unabhängig angesprochen werden, jedoch sind sie für eine entsprechend integrierte Zusammenarbeit konzipiert. Die Webkarte wird durch die Agentenplattform selbst aktuell gehalten, während der Routenplaner, der die Webkarte nutzt, durch den mobilen Agenten explizit aufgerufen wird. Der Migrationsoptimierer dient aus technischer Sicht der Anpassung der Route an das eingesetzte mobile Agentensystem, sollte dies gewünscht werden.

ProNav kann mit fast allen mobilen Agentensystemen zusammenarbeiten, die eine entsprechende Infrastruktur-Organisation aufweisen. In dieser Arbeit wurde das eigene mobile Agentensystem TRACY als Basis genutzt sowie dessen Fähigkeit, so genannte Domains einzurichten. In dieser Umgebung wurde *ProNav* prototypisch implementiert und evaluiert.

Ausgangsbasis der Arbeit

Bisher blieb die Dynamik, die einem Netzwerk von Agentenservern inhärent ist, in einem mobilen Agentensystem weitestgehend unberücksichtigt. Das führt zu Problemen bei der Rundreise eines mobilen Agenten, der eine Menge von Agentenservern im Netzwerk besucht, um die ihm übertragene Aufgabe zu lösen.

Diese Zielpunkte müssen dem Agenten zurzeit in Form einer statischen Liste vom Programmierer oder vom Benutzer übergeben werden. Die aktuell im Netzwerk vorherrschende Situation beim Start des Agenten kann aber weder vom Programmierer noch vom Benutzer vollständig vorhergesehen werden. Die Vorstellung, die ein Mensch vom Netzwerk hat, kann von der realen Situation abweichen und die Quantität der Informationen kann schnell überfordern. Neue Anbieter tauchen ständig auf, andere verschwinden aus dem Netz.

Auf der Reise des Agenten kann daher eine Anpassung der Route auf Grund der inhaltlichen Dynamik oder technischer Gegebenheiten zwingend notwendig werden. Fixe Vorgaben müssen dann zum Abbruch der Reise oder verfälschten Ergebnissen führen.

Mobile Agentensysteme die vom Nutzer oder Programmierer eine fixe Vorgabe der Route fordern sind heute der Standard. In der Arbeit wurden diese als mobile Agentensysteme der ersten Stufe bezeichnet.

Idee, Ziele und Entscheidungen

Die Idee dieser Arbeit ist die Unterstützung der Agenten bei der autonomen Planung und dynamischen Anpassung ihrer Reiserouten – die proaktive Navigation (*ProNav*). Damit wird der erste (und wichtigste) Schritt von einem mobilen Agentensystem der ersten Stufe zu einem der zweiten Stufe gemacht.

Mit Unterstützung aller Agentenserver wird eine Informationsbasis für Agenten aufgebaut, die diese zur Wahrnehmung des Agentennetzwerks befähigt: Das Agentennetzwerk wird kartographiert. Auf der Webkarte sind Informationen über Agentenserver (Services) und deren Verbindungen verzeichnet. Die Reiseroute wird erst auf Basis dieser Informationen geplant, sobald der Agent aktiviert wird, und kann während der Abarbeitung der Route jederzeit aktualisiert werden. Der Programmierer eines Agenten kann sich auf

die Umsetzung des reinen Applikations-Algorithmus beschränken, der Benutzer auf die Übergabe der konkreten Aufgabenspezifikation. Die Navigation selbst bleibt dem Agenten überlassen, der sich dazu der angebotenen Dienste der Komponenten in *ProNav* bedient.

Ziel dieser Arbeit war die prototypische Umsetzung dieser Idee durch *ProNav*. Ohne in die Implementierung des als Basis dienenden Agentensystems einzugreifen, wurde der Routing Service als Aufsatz auf ein mobiles Agentensystem konzipiert. Die Nutzung der Komponenten von *ProNav* liegt dabei in der Autonomie des einzelnen Agenten, ebenso wie die Verwertung der Ergebnisse.

Bei der Realisierung von *ProNav* musste zuerst das Quantitätsproblem geklärt werden. Unmöglich ist die Erstellung einer „Welt-Webkarte“. Die Anzahl der potentiellen Agentenserver und deren Verbindungen auf dieser Karte übersteigt die Grenzen des beherrschbaren. Diese Menge musste in handhabbare Teile partitioniert, d. h. die Infrastruktur musste organisiert werden.

Dazu wurde der Domain Ansatz von TRACY gewählt, der die Menge der Agentenserver anhand ihrer IP-Subnetze in so genannte Domains einteilt. Die Domains können separat kartographiert werden. Zusammengefasste Domainkarten können zwischen Domains ausgetauscht werden. Ähnlich wie bei einer Straßenkarte bewegt sich ein mobiler Agent damit während seiner Rundreise durch mehrere Kartenblätter, wobei er sich jeweils auf die Informationen konzentriert, die das gerade gültige, lokale Blatt anbietet. Bewegt er sich aus diesem Bereich hinaus, so holt er sich das neue Kartenblatt und plant weiter. Dieser Ansatz ermöglicht es, die Quantität pro Kartenblatt (Domain) im überschaubaren Rahmen zu halten. Mit einer Domaingröße von maximal 60 bis 100 Agentenservern wird gerechnet.

Eine weitere Schwierigkeit war die Dynamik des Systems, das durch die Webkarte erfasst wird. Agentenserver, Verbindungen und Services verändern sich in einem modernen Netzwerk ständig. Die Webkarte, jede einzelne Domain, muss daher ständig neu kartographiert werden. Da die Kartographierung selbst auch Netzlast erzeugt und die Anzahl der Verbindungen mit der Anzahl der Agentenserver pro Domain quadratisch wächst, sind hier aber gewisse Grenzen gesetzt. Hilfreich an dieser Stelle ist wiederum, dass lokale Karten verwendet werden, die dezentral erstellt und gewartet werden.

Die Evaluierung des Kartenmoduls hat gezeigt, dass für eine typische Domaingröße (siehe oben) die Aktualisierungszeiten der Karte in einem für praktische Anwendungen adäquaten Bereich liegen. Diese liegen im Minutenbereich.

Die Routenplanung selbst, auf Basis vollständiger und aktueller Karten, ist ein NP-vollständiges Problem, das Traveling Salesman Problem. Das gilt auch für eine Routenplanung im Agentennetzwerk. Die Suche nach passenden Algorithmen für dieses Problemfeld war erfolgreich. Heuristiken aus der lokalen Optimierung konnten aufgegriffen und

angepasst werden. Durch die Reduzierung der Quantität mit Hilfe des Domain Ansatzes ergeben sich sehr geringe Berechnungszeiten für eine Route, im Bereich von Millisekunden für die bei Agentensystemen erwarteten typischen Routen. Die strukturellen Ergebnisse liegen dabei sehr nahe am Optimum der Route.

Das Ergebnis

Die prototypische Implementierung von *ProNav* hat sich als standhaft erwiesen. Die Agenten können autonom dynamische Agentennetzwerke sinnvoll wahrnehmen, Ziele lokalisieren und eine Routenplanung effizient vornehmen. Innerhalb der typischen Größenordnungen skaliert *ProNav* gut und ist für den Einsatz in einer für Informationssysteme typischen Umgebung sofort geeignet.

Die Organisation von *ProNav* als zusätzliche Schicht in der Infrastruktur hat sich nicht nur bewährt, sondern wurde als die einzig sinnvolle Alternative bestätigt. Jeder Agent kann, aber muss nicht, die Dienste von *ProNav* in Anspruch nehmen. Seine Fähigkeiten werden dadurch, ohne in seine Autonomie einzugreifen, bei Bedarf passend erweitert. Durch die Zusammenarbeit aller Agentenserver innerhalb des von *ProNav* definierten Rahmenwerks wird dabei zugleich ein zentraler Flaschenhals vermieden, das Mengenproblem neutralisiert und die verteilte Leistung im Netz lokal genutzt.

Ein Ansatz, der die Bürde der Kartographierung und Routenplanung auf den einzelnen Agenten abwälzt, wäre am Mengenproblem und der zusätzlichen Belastung der Agenten, und damit der zusätzlichen Netzwerkbelastung, gescheitert. Im Prinzip bleibt, bis auf die Erstellung der Webkarte selbst, ein solcher Ansatz jedoch für den Spezialfall möglich und wird durch *ProNav* nicht ausgeschlossen.

Ausblick

Der vorgelegte Prototyp hat gezeigt, dass einige Konzepte weiter verbessert werden können. So kann die zur Transferratenmessung benutzte Paketgröße automatisch angepasst werden. Die quadratische Abhängigkeit der Verbindungsmessung von der Serveranzahl kann gelöst werden. Java-bedingte Messungenauigkeiten können in Zukunft reduziert werden. Für eine bessere Migrationsoptimierung muss über Heuristiken zur Strategienwahl nachgedacht werden.

Eine Integration von *ProNav* in die neue Version von TRACY ist in Planung, ebenso eine Umsetzung mit einem anderen mobilen Agentensystem. Weitere Testläufe, gerade auch in einer Umgebung mit echter Applikationslast, werden notwendig sein, um die Konzepte weiter zu verfeinern und kritische Schwachstellen aufzudecken. Eine Testumgebung mit starkem Anteil an mobilen Agentenplattformen (auf mobilen Endgeräten) ist als Projekt

in Vorbereitung. Auch der Einsatz von autonomen mobilen Agenten im Bereich der Ad-hoc-Bildung von Clustern in Netzwerken soll weiter vertieft werden.

Literaturverzeichnis

[Adl03] ADLICH, MATTHIAS: *Trusted Science Community – Eine Anwendung zur Dokumentenrecherche mit Hilfe von mobilen Agenten*. Diplomarbeit, Friedrich-Schiller-Universität Jena, Institut für Informatik, Dezember 2003.

[Age04] AGENTLINK: *Agent Software*. URL: http://www.agentlink.org/resources/agent-software.php, 2004.

[Bau97] BAUMANN, JOACHIM: *A Protocol for Orphan Detection and Termination in Mobile Agents Systems*. Technical Report 1997/09, Universität Stuttgart, Fakultät für Informatik, July 1997. 25 p.

[BBCM99] BÄUMER, C., M. BREUGST, S. CHOY, and T. MAGEDANZ: *Grasshopper – A Universal Agent Platform based on OMG MASIF and FIPA Standards*. In KARMOUCH, AHMED and ROGER IMPLEY (editors): *First International Workshop on Mobile Agents for Telecommunication Applications (MATA'99)*, pages 1–18, Ottawa, Canada, October 1999. World Scientific Publishing Ltd.

[BER00] BRAUN, PETER, CHRISTIAN ERFURTH, and WILHELM ROSSAK: *An Introduction to the Tracy Mobile Agent System*. Technical Report Math/Inf/00/24, Friedrich-Schiller-Universität Jena, Institut für Informatik, September 2000.

[BER01a] BRAUN, PETER, JAN EISMANN, and WILHELM ROSSAK: *A Multi-Agent Approach To Manage a Network of Mobile Agent Servers*. Technical Report 12/01, Friedrich-Schiller-Universität Jena, Institut für Informatik, June 2001.

[BER01b] BRAUN, PETER, CHRISTIAN ERFURTH, and WILHELM ROSSAK: *Performance Evaluation of Various Migration Strategies for Mobile Agents*. In *Fachtagung Kommunikation in verteilten Systemen (KiVS 2001)*, Hamburg (Germany), February 2001.

[BHR+97] BAUMANN, JOACHIM, FRITZ HOHL, NIKOLAOS RADOUNIKLIS, KURT ROTHERMEL, and MARKUS STRASSER: *Communication Concepts for Mobile Agent Systems*. In ROTHERMEL, KURT and RADU POPESCU-ZELETIN

(editors): *Proceedings of the First International Workshop on Mobile Agents (MA'97), Berlin, April 1997*, volume 1219 of *Lecture Notes in Computer Science*, pages 123–135, Berlin, 1997. Springer Verlag.

[BN84] BIRRELL, ANDREW and BRUCE J. NELSON: *Implementing Remote Procedure Calls*. ACM Transactions on Computer Systems 2, 1, 2(1):39–59, February 1984.

[BR98] BAUMANN, JOACHIM and KURT ROTHERMEL: *The Shadow Approach: An Orphan Detection Protocol for Mobile Agents*. Technical Report 1998/08, Universität Stuttgart, Fakultät für Informatik, 1998.

[BR04] BRAUN, PETER and WILHELM R. ROSSAK: *Mobile Agents: Foundations Techniques and Programming*. Morgan Kaufmann, October 2004.

[Bra03] BRAUN, PETER: *The Migration Process of Mobile Agents - Implementation, Classification, and Optimization*. PhD thesis, Friedrich-Schiller-Universität Jena, Institut für Informatik, 2003.

[Bun04] BUNDESMINISTERIUM FÜR BILDUNG UND FORSCHUNG: *EXIST - Existenzgründungen aus Hochschulen*. URL: http://www.exist.de/, 2004.

[CGPV97] CUGOLA, GIANPAOLO, CARLO GHEZZI, GIAN PIETRO PICCO, and GIOVANNI VIGNA: *Analyzing Mobile Code Languages*. In VITEK, JAN and CHRISTIAN TSCHUDIN (editors): *Mobile Object Systems: Towards the Programmable Internet (MOS'96), Linz, July 1996*, volume 1222 of *Lecture Notes in Computer Science*, pages 93–110, Berlin, 1997. Springer Verlag.

[Dag04] DAGEFORDE, MARY: *The Java Tutorial – Trail: Security in Java 2 SDK 1.2*. URL: http://java.sun.com/docs/books/tutorial/security1.2/, 2004. Sun Microsystems, Inc.

[eBa04] EBAY INC.: *eBay Deutschland – Der weltweite Online-Marktplatz*. URL: http://www.ebay.de/, 2004.

[ER02] ERFURTH, CHRISTIAN and WILHELM ROSSAK: *Characterization and Management of Dynamical Behaviour in a System With Mobile Agents*. In UNGER, HERWIG, THOMAS BÖHME, and ARMIN MIKLER (editors): *Innovative Internet Computing System - Second International Workshop, IICS 2002, Kühlungsborn (Germany), June 2002*, volume 2346 of *Lecture Notes in Computer Science*, pages 109–119, Kühlungsborn (Germany), June 2002. Springer Verlag.

[Erf99] ERFURTH, CHRISTIAN: *Eine Plattform zur Migration von Komponenten in einem verteilten System*. Diplomarbeit, Friedrich-Schiller-Universität Jena, Institut für Informatik, Dezember 1999.

[Eur04] EUROPEAN SPACE AGENCY: *Mars Express*. URL: http://www.esa.int/SPECIALS/Mars_Express/, 2004.

[FG96] FRANKLIN, STAN and ART GRAESSER: *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*. In *Intelligent Agents III. Agent Theories, Architectures and Languages (ATAL'96)*, volume 1193, Berlin, Germany, 1996. Springer-Verlag.

[Fou02] FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS: *FIPA Agent Management Specification*. URL: http://www.fipa.org/specs/fipa00023/, December 2002.

[Fou04] FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS: *Homepage*. URL: http://www.fipa.org/, 2004.

[Fra04] FRAUNHOFER IGD: *SeMoA – Secure Mobile Agents*. URL: http://www.semoa.org/, 2004.

[Gle04] GLEICHMANN, NILS: *Avatare (virtuelle Figuren) zur Interaktion mit Menschen*. Studienarbeit, 2004.

[GP02] GUTIN, GREGORY and ABRAHAM P. PUNNEN (editors): *The Traveling Salesman Problem and its Variations*. Kluwer Academic Publishers, May 2002.

[HA98] HAMMER, DIETER K. and AD T. M. AERTS: *Mobile Agent Architectures: What are the Design Issues?* In *Proceedings International Conference and Workshop on Engineering of Computer-Based Systems (ECBS'98), Jerusalem, March/April 1998*, pages 272–280. IEEE Computer Society Press, 1998.

[Hub04] HUBERT, BERT: *Linux Advanced Routing & Traffic Control*. URL: http://lartc.org/, 2004.

[IKV04a] IKV++ TECHNOLOGIES AG: *Grasshopper – The Agent Platform*. URL: http://www.grasshopper.de/, 2004.

[IKV04b] IKV++ TECHNOLOGIES AG: *Homepage*. URL: http://www.ikv.de/, 2004.

[JGM+02] JOHNSON, DAVID S., GREGORY GUTIN, LYLE A. MCGEOCH, ANDERS YEO, WEIXIONG ZHANG, and ALEXEI ZVEROVITCH: *Experimental Analysis of Heuristics for the ATSP*. In GUTIN, GREGORY and ABRAHAM P. PUNNEN [GP02], pages 445–487.

[JM97] JOHNSON, DAVID S. and LYLE A. MCGEOCH: *The Traveling Salesman Problem: A Case Study in Local Optimization*. In E.H.L.AARTS and J.K.LENSTRA (editors): *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley and Sons, Ltd., 1997.

[JM02] JOHNSON, DAVID S. and LYLE A. MCGEOCH: *Experimental Analysis of Heuristics for the STSP*. In GUTIN, GREGORY and ABRAHAM P. PUNNEN [GP02], pages 369–444.

[Kna95] KNABE, FREDERICK C.: *Language Support for Mobile Agents*. PhD thesis, Carnegie Mellon University, Paittsburgh, Pa., December 1995.

[KOOL03] KLUSCH, MATTHIAS, SASCHA OSSOWSKI, ANDREA OMICINI, and HEIMO LAAMANEN (editors): *Proceedings of 7th International Workshop, CIA 2003*, volume 2782 of *Lecture Notes in Artificial Intelligence*, Helsinki, Finland, August 2003. Springer Verlag.

[Lan04] LANGENSCHEIDT KG, BERLIN UND MÜNCHEN: *Langenscheidt Fremdwörterbuch Online Edition*. URL: http://www.langenscheidt.de/fremdwb/, 2004.

[Leh04a] LEHRSTUHL FÜR SOFTWARETECHNIK: *Projekt ProNav*. URL: http://swt.informatik.uni-jena.de/pro_nettracy.html, 2004.

[Leh04b] LEHRSTUHL FÜR SOFTWARETECHNIK: *Projekt Tracy*. URL: http://swt.informatik.uni-jena.de/pro_tracy.html, 2004.

[Lem03] LEM, STANISLAW: *Die Megabit-Bombe – Essays zum Hyperspace*. Verlag Heinz Heise, 2003.

[Lin65] LIN, S.: *Computer Solutions of the Traveling Salesman Problem*. Bell System Technical Journal, 44:2245–2269, 1965.

[MT02] MEYER, JOHN-JULES C. and MILIND TAMBE (editors): *Intelligent Agents VIII – 8th International Workshop, ATAL 2001 Seattle, WA, USA*, volume 2333 of *Lecture Notes in Computer Science*. Springer Verlag, 2002.

[OMG00] OBJECT MANAGEMENT GROUP, INC. (OMG): *Mobile Agent Facility Specification*, January 2000.

[OMG04] OMG OBJECT MANAGEMENT GROUP, INC.: *Object Management Group*. URL: http://www.omg.org/, 2004.

[RP01] ROTH, VOLKER and JAN PETERS: *A Scalable and Secure Global Tracking Service for Mobile Agents*. In *Proc. Mobile Agents 2001*, volume 2240 of *Lecture Notes in Computer Science*. Springer Verlag, December 2001.

[Sch02] SCHREIBER, STEFFEN: *Beschreibung und Analyse von dynamischen Netzen für Agentensysteme*. Diplomarbeit, Friedrich-Schiller-Universität Jena, Institut für Informatik, April 2002.

[Sch03] SCHAAF, MARKO: *Entwicklung und Implementierung einer Komponente zur Migrationsoptimierung für Mobile Agenten*. Diplomarbeit, Friedrich-Schiller-Universität Jena, Institut für Informatik, Mai 2003.

[SM94] SUN MICROSYSTEMS, INC.: *The Java Language: An Overview*. Technischer Bericht, Sun Microsystems, 1994. White Paper.

[Sta03] STAHL, MARCO: *Bewertung verschiedener Lösungsanstätze für das Traveling Salesman Problem am WWW und Integration in das Agentensystem Tracy*. Studienarbeit, Juni 2003.

[Sun04a] SUN MICROSYSTEMS, INC.: *Code Samples and Apps – Applets*. URL: http://java.sun.com/applets/, 2004.

[Sun04b] SUN MICROSYSTEMS, INC: *Java Technology*. URL: http://java.sun.com/, 2004.

[Sun04c] SUN MICROSYSTEMS, INC.: *JRE 1.4 Unterstützte Systemkonfigurationen*. URL: http://java.sun.com/j2se/1.4.2/system-configurations.html, 2004.

[Tel04a] TELECOM ITALIA LAB: *Homepage*. URL: http://www.telecomitalialab.com/, 2004.

[Tel04b] TELECOM ITALIA LAB: *JADE – Java Agent DEvelopment Framework*. URL: http://jade.tilab.com/, 2004.

[the04] THE AGENT FACTORY GMBH: *agents for everyone*. URL: http://www.the-agent-factory.de/, 2004.

[TvS03] TANENBAUM, ANDREW S. und MAARTEN VAN STEEN: *Verteilte Systeme: Grundlagen und Paradigmen*. Pearson Studium, April 2003.

[Vig98] VIGNA, GIOVANNI: *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*. Springer Verlag, New York, 1998.

[Wik04] WIKIMEDIA FOUNDATION INC.: *Wikipedia, die freie Enzyklopädie*. URL: http://de.wikipedia.org/, 2004.

[WSP97] WOLSKI, RICH, NEIL SPRING, and CHRIS PETERSON: *Implementing a Performance Forecasting System for Metacomputing: The Network Weather Service*. In *Proceedings of Supercomputing'97 (CD-ROM)*, San Jose, CA, November 1997. ACM SIGARCH and IEEE.

[WW04] WOLLRATH, ANN and JIM WALDO: *The Java Tutorial – Trail: RMI*. URL: http://java.sun.com/docs/books/tutorial/rmi/, 2004. Sun Microsystems, Inc.

Anhang A

Implementierung

A.1 Informationen zum Quellcode

Der Quellcode zu den Komponenten von *ProNav* ist auf dem Server des Lehrstuhls für Softwaretechnik, Fakultät für Mathematik und Informatik der Friedrich-Schiller-Universität Jena, hinterlegt. Die Internetseiten des Lehrstuhls sind unter http://swt.informatik.uni-jena.de zu finden. Informationen zum mobilen Agentensystem TRACY können unter http://tracy.informatik.uni-jena.de nachgeschlagen werden. Eine Beschreibung und weitere Informationen zum Projekt TRACY-*ProNav* sind unter http://swt.informatik.uni-jena.de/pro_nettracy.html aufgeführt.

ProNav besteht aus den drei in Java programmierten Teilen:

1. Kartenmodul
   Paket `netmonitoring`
2. Routenplaner
   Paket `routenplaner`
3. Migrationsoptimierer
   Paket `migplan`

A.2 Generischer Netzwerksensor

Die generische Klasse `netmonitoring.Sensor` enthält alle notwendigen Methoden zur Kommunikation und zur Interaktion mit dem Monitoring Manager. Die Integration neuer Sensoren im Kartenmodul ist einfach, wie der folgende Beispielsensor verdeutlicht:

```
import netmonitoring.*;

public class DummySensor extends Sensor {

  protected void init() {
    lengthOfSeries=20;               // Length of time serie
    delayBetweenExperiments=2000;    // Measurement delay
    unit = ''ms'';
  }

  protected double doExperiment (final AgentServer target) {
    log(2, ''Sensor ''+name+'': Experiment with ''+target.url);
    if (!connectTo(target)) return UNREACHABLE;
    final String answer=receive();
    long time=0;
    if (answer==null) return UNREACHABLE;
    if (answer.equals(name)) {
      // remote sensor is responding, start experiment:
      // first get time for measurement...
      time = System.currentTimeMillis();
      // send a small package of data over the net
      send(''This is short'');
      // wait for the answer
      //answer=receive();
      send(''got: ''+receive());
      send(''got: ''+receive());
      // calculate time
      time = System.currentTimeMillis()-time;
      send(''Quit'');
    }
    disconnect();
    return time;
  }

  protected String processInput(final String input) {
    // first message after a connection was made - send my name:
    if (input == null) return name;
    // we got an input, create an answer:
    // log(2, ''Sensor ''+name+'': received message: ''+input);
    if (input.equalsIgnoreCase(''Quit''))
      return (''Bye'');
    else return ''got: ''+input;
  }

}
```

Die Implementierung des Latenzsensors, der die Round Trip Time eines minimalen Paketes misst, veranschaulicht die Zeitmessung mit einem Sensor. Der Sensor versucht eine Verbindung aufzubauen. War diese erfolgreich, wird die Zeit bis zur Antwort gemessen. Im Quellcode ist die Zeitmessungsperiode durch START und STOP markiert.

```
import netmonitoring.*;

public class SENSOR_Latency extends Sensor {

  protected void init() {
    lengthOfSeries=20;
    delayBetweenExperiments=2000;  // every 2 seconds
```

```
    unit = ''ms'';
  }

  protected double doExperiment (final AgentServer target) {
    log(4, ''Experiment with ''+target.url);
    if (!connectTo(target)) return UNREACHABLE;
    final String answer=receive();
    // disconnect
    long time = System.currentTimeMillis();             // START
    send(''Quit'');                                     //  |
    String s=receive();                                 //  |
    // calculate time                                   //  V
    time = System.currentTimeMillis()-time;             // STOP
    disconnect();
    // if time was 0 ms - this means less than a ms,
    // assume 0.5 ms then.
    final double result = (time != 0) ? time : 0.5;
    log(3, target.url+'': ''+result+unit);

    if (answer==null) return UNREACHABLE;
    if (answer.equals(name)) return result;
    else return UNREACHABLE;
  }

  protected String processInput(final String input) {
    // first message after a connection was made - send my name:
    if (input == null) return name;
    // we got an input, create an answer:
    // log(5, ''received message: ''+input);
    if (input.equalsIgnoreCase(''Quit''))
      return (''Bye'');
    else return ''ACK'';
  }

}
```

Abschließend ist die Methode `connectTo(AgentServer)` des generischen Sensors aufgelistet, die zum Verbindungsaufbau zwischen Sensoren dient.

```
  /**
   * This will open a communication socket to the corresponding sensor
   * on the target agent server. After connecting, methods send and receive
   * may be used to communicate.
   *
   * @param target the target agent server
   * @return true, if the connection was created successfully
   */
  public boolean connectTo(final AgentServer target) {
    if (connected) {
      log (1, ''trying to connect to ''+target.url+
              '', but there is an open connection already!'');
      return false;
    }

    // get the port of the remote sensor
    final Integer p;
    try {
      p = SensorPortClient.getClient().getPort(target.hostName,
                                          name+''@''+target.serverName);
```

```
    } catch (ConnectException e) {
      log(1, ''Couldn't get port of ''+name+''@''+target.url+''!'');
      return false;
    }
    if (p==null) {
      log(3, ''Couldn't get port of ''+name+''@''+target.url+''!'');
      return false;
    }
    int port = p.intValue();
    try {
      conSocket = new Socket(target.hostName, port);
      conOut = new PrintWriter(conSocket.getOutputStream(), true);
      conIn = new BufferedReader(
                    new InputStreamReader(conSocket.getInputStream()));
      connected = true;
    } catch (UnknownHostException e) {
      log(0,''Don't know about host: ''+target.hostName);
    } catch (IOException e) {
      log(3,''Couldn't get I/O for the connection to: ''+target.url);
    }
    return connected;
  }
```

A.3 Die Schnittstelle zur Karte

Mit dieser Schnittstelle ist der Zugriff zu Knoten- und Kanteninformationen der einzelnen Server auf der Karte möglich. Mengenabfragen sowie logische Verknüpfungen sind möglich.

```
package netmonitoring;

import java.util.*;

public interface MapQueryInterface {

  public final int OR = 1;
  public final int AND = 2;
  public final int XOR = 3;

  /**
   * Liefert die Liste der momentan bekannten Agentenserver zurück.
   *
   * @return die Liste der Agentenserver
   */
  public Vector getServers();


  /**
   * Liefert die Liste der Agentenserver zurück, welche alle der gegebenen
   * Knoteneigenschaften haben. Jede Eigenschaft ist ein einzelnes
   * Schlüsselwort.
   *
   * @param properties das Array von Schlüsselwörtern
   * @return die Liste der Agentenserver
   */
  public Vector getServers(String[] properties);
```

```
/**
 * Liefert die Liste der Agentenserver zurück, welche alle der gegebenen
 * Knoteneigenschaften haben, und verknüpft sie mit einer gegebenen Liste.
 * Jede Eigenschaft ist ein einzelnes Schlüsselwort. Die Liste der
 * Agentenserver mit den Eigenschaften wird mit der gegebenen Liste
 * verknüpft und das Ergebnis zurückgegeben. So ist es z.B. möglich,
 * Suchanfragen auf das Ergebnis einer vorhergehenden Anfrage zu beziehen.
 *
 * @param properties das Array von Schlüsselwörtern
 * @param operator der Operator der Verknüpfung, einer aus AND, OR, XOR
 * @param nodes die zweite Liste für die Verknüpfung
 * @return das Ergebnis der Verknüpfung
 */
public Vector getServers(String[] properties,
                         int operator,
                         Vector nodes);


 /**
  * Prüft, ob ein Server in der lokalen Domain ist.
  *
  * @param servName Name des Servers
  * @return  true, falls Server in der lokalen Domain ist
  */
 public boolean isLocalNode ( String servName );

 /**
  * Liefert den Namen des Managers der Domain zurück.
  *
  * @return Name des Domain-Managers
  */
 public String getDomainManagerName();


/**
 * Liefert die Namen aller registrierten Sensoren auf dem Agentenserver.
 *
 * @return die Liste der Namen
 */
public Vector getSensorNames();


/**
 * Liefert die Einheiten aller registrierten Sensoren auf dem Agentenserver.
 * Das Ergebnis ist eine Hashtable. Die Einheit eines Sensors ist als
 * String unter dem Sensornamen abgelegt.
 *
 * @return die Liste der Einheiten, in denen die Sensoren messen.
 */
public Hashtable getSensorUnits();


/**
 * Liefert den aktuellen Wert einer Kanteninformation. Für die Verbindung
 * von Agentenserver a nach b wird der Wert derjenigen Eigenschaft
 * zurückgegeben, welche vom gegebenen Sensor gemessen wird.
 * Solche Eigenschaften sind z.B. Latenz, Durchsatz, Migrationsdauer,
 * Verbindungsqualität
 *
 * @param sensorName der Name des Sensors, welcher die gewünschte Eigenschaft
```

```
     *                   misst
     * @param a der erste Agentenserver
     * @param b der zweite Agentenserver
     * @return der aktuelle Wert der Kanteneigenschaft
     */
    public Double getEdgeInfo(String sensorName,
                              String a,
                              String b);


    /**
     * Liefert die aktuellen Kanteninformationen für die Verbindung
     * von Agentenserver a nach b. Es werden die Werte aller Sensoren
     * zurückgegeben, für die Informationen verfügbar sind. Die Werte
     * liegen in einer Hashtable als Double-Objekte vor und können
     * direkt über den Sensornamen abgefragt werden.
     *
     * @param a der erste Agentenserver
     * @param b der zweite Agentenserver
     * @return die aktuellen Kanteninformationen
     */
    public Hashtable getEdgeInfoAB(String a,
                                   String b);


    /**
     * Liefert die aktuellen Kanteninformationen für die Verbindungen
     * von Agentenserver a zu allen anderen. Es wird eine Hashtable
     * zurückgeliefert. Für jeden Zielserver gibt es einen Eintrag darin,
     * der wiederum eine Hashtable ist. In dieser liegen die Werte
     * als Double-Objekte unter dem Sensornamen vor.
     * <br>
     * Beispiel: Um den Durchsatz von Server ‘‘A’’ nach ‘‘B’’ zu bestimmen,
     * könnte folgende Abfrage verwendet werden:
     * Hashtable info=(Hashtable)getEdgeInfo(‘‘A’’).get(‘‘B’’);
     * Double value=(Double)info.get(‘‘SENSOR_Bandwidth’’);
     *
     * @param a der Agentenserver
     * @return die aktuellen Kanteninformationen
     */
    public Hashtable getEdgeInfo(String a);


    /**
     * Liefert die aktuelle Kanteninformation eines Sensors für die Verbindungen
     * von Agentenserver a zu allen anderen. Es wird in einer Hashtable unter
     * dem Namen jedes Zielservers der Wert des Sensors für diese Verbindung
     * zurückgegeben.
     *
     * @param a der Agentenserver
     * @param sensorName der Sensor
     * @return die aktuellen Kanteninformationen
     */
    public Hashtable getEdgeInfoSensor(String sensorName,
                                       String a);


    /**
     * Liefert die aktuellen Knoteninformationen für einen
     * Agentenserver. Diese werden als Liste von Schlüsselworten ausgegeben.
     *
     * @param a der Agentenserver
```

```
   * @return die aktuellen Knoteninformationen
   */
  public Vector getNodeInfo(String a);

   /**
    * Fügt neue Knoteninformationen (Services) zum lokalen Server hinzu.
    *
    * @param props Knoteninformationen
    */
   public void addNodeInfo( Vector props);

   /**
    * Entfernt Knoteninformationen (Services) über den lokalen Server.
    *
    * @param props Knoteninformationen
    */
   public boolean removeNodeInfo( Vector props);

   /**
    * Liefert eine Liste von Agentenservern, die die angegebenen
    * Knoteninformationen besitzen bzw anbieten. Die Liste der Knoteninfos
    * wird mit dem angegebenen Operator bei der Suche nach
    * passenden Servern verknüpft (AND, OR, XOR).
    *
    * @param props Liste der gewünschten Knoteninformationen
    * @param operator Verknüpfungsoperator @see MapQueryInterface
    * @return Liste der Agentenserver, die diese Knoteninfo abdecken
    */
   public Vector findNodeInfo( Vector props, int operator);

}
```

A.4 Abfrage der Karteninformationen von der TRACY-Konsole

Nach dem Start des Kartenmoduls stehen einem Agenten der `NetMonitorAgent` und der `MapQueryAgent` als Schnittstelle zur Karte zur Verfügung. Die Abfragen werden über asynchronen Nachrichtenaustausch realisiert. TRACY bietet eine administrative Shell, die ebenfalls zum Senden von Nachrichten an Agenten genutzt werden kann. An folgendem Beispiel soll dies demonstriert werden.

Der Befehl `list` führt zur Auflistung aller Agenten der Plattform:

```
MrNice@ipc031.inf.uni-jena.de/james
MigrationTester1087405675702@ipc031.inf.uni-jena.de/james
MapQueryAgent@ipc031.inf.uni-jena.de/james
NetMonitorAgent@ipc031.inf.uni-jena.de/james
DomainAgent@ipc031.inf.uni-jena.de/james
```

Die Liste der Server kann durch die Nachricht `GETSERVERS` erfragt werden. Diese Nachricht muss an den `NetMonitorAgent` durch das Kommando der TRACY-Konsole `send NetMonitorAgent GETSERVERS` gesendet werden.

```
Known Servers:
  0. tracy://ipc031.inf.uni-jena.de/james (node of local domain)
  1. tracy://ipc033.inf.uni-jena.de/james (manager of local domain)
  2. tracy://ipc032.inf.uni-jena.de/james (node of local domain)
```

Die Knoteninformationen werden durch die Nachricht `GETNODEINFO` ausgegeben:

```
Node information:
  0. ipc031.inf.uni-jena.de/james:
     PROP\_NAME=PROP\_PROTO\_satptcp,PROP\_PROTO\_satprmi,PROP\_PROTO\_udp,
     PROP\_PROTO\_Boot,PROP\_SERVICE\_Spass,
  1. ipc033.inf.uni-jena.de/james:
     PROP\_NAME=PROP\_PROTO\_satptcp,PROP\_PROTO\_satprmi,PROP\_PROTO\_udp,
     PROP\_PROTO\_Flugzeug,PROP\_PROTO\_Boot,
     PROP\_SERVICE\_Spass,
  2. ipc032.inf.uni-jena.de/james:
     PROP\_NAME=PROP\_PROTO\_satptcp,PROP\_PROTO\_satprmi,PROP\_PROTO\_udp,
     PROP\_PROTO\_Flugzeug,
```

Die Antwort auf die Nachricht `GETEDGEINFO` sind die Kanteninformationen:

```
Edge information:
  Edges from ipc031.inf.uni-jena.de/james...
    to ipc033.inf.uni-jena.de/james:
        Migration=741.0,Availability=100.0,Bandwidth=265.56016597510376,Latency=1.0,
    to ipc032.inf.uni-jena.de/james:
        Migration=565.6666666666666,Availability=100.0,Bandwidth=2064.516129032258,
    Latency=0.5,
  Edges from ipc033.inf.uni-jena.de/james...
    to ipc031.inf.uni-jena.de/james:
        Migration=546.0,Availability=100.0,Bandwidth=2341.3519813519815,Latency=0.5,
    to ipc032.inf.uni-jena.de/james:
        Migration=630.95,Availability=100.0,Bandwidth=249.0272373540856,Latency=1.0,
  Edges from ipc032.inf.uni-jena.de/james...
    to ipc031.inf.uni-jena.de/james:
        Migration=585.0,Availability=100.0,Bandwidth=275.8620689655172,Latency=0.5,
    to ipc033.inf.uni-jena.de/james:
        Migration=674.0,Availability=100.0,Bandwidth=2560.0,Latency=0.5,
```

A.5 Der RoutenPlanerAgent

Der RoutenPlanerAgent ist Ansprechpartner von mobilen Agenten. Er nimmt die Liste der Zielserver sowie zusätzliche Anforderungen an die Reise entgegen. Der folgende Code-Ausschnitt zeigt den Ablauf nach der Übergabe der Parameter bis zum Senden der Antwort.

```
package routenplaner;

...

public class RoutenPlanerAgent extends SystemAgent {
```

```
...

    /**
     * Empfängt ein Agent eine Nachricht, wird diese Methode aufgerufen.
     * Die Nachricht besteht aus dem Sender, dem Typ und dem Inhalt.
     *
     **/
    public void handleMessage(Message m) throws AgentExecutionException{
        String sender=m.getSender();
        String typ=m.getTyp();
        String content=m.getContent();

...
       // Die empfangene Nachricht ist findRoute
        if (m.getTyp().equals(''findRoute'')){
        // Welcher Algo soll benutzt werden?
            String algoName=null;
            int pos=content.indexOf('': '');
            if (pos>0) {
                algoName=content.substring(0,pos);
                content=content.substring(pos+2);
            }
            // Hat der mobile Agent zusätzliche Bedingungen?
            int propStringStart=content.indexOf(''('');
            int propStringEnd=content.lastIndexOf('')'');
            String propString=null;
            if ((propStringStart>0) & (propStringEnd>propStringStart+1)) {
                propString=content.substring(propStringStart+1,propStringEnd);
                content=content.substring(0,propStringStart);
            }
            // Die Liste der Zielserver des Agenten
            Set targets=getTargets(content);
            targets.add(getServerName());
            List route;
            if (targets!=null) {
                try {
                    // Plane die Route...
                    if (algoName!=null) {
                        route=routenplaner.findRoute(map,
                                                     targets,
                                                     algoManager.getAlgo(algoName),
                                                     propString,
                                                     new Hashtable());
                    } else {
                        route=routenplaner.findRoute(map,
                                                     targets,
                                                     propString,
                                                     new Hashtable());
                    }
                    // die berechnete Reisezeit
                    Object rtime=route.get(0);
                    route.remove(0);
                    // setze den Startserver
                    route=correctList(getServerName(),new LinkedList(route));
                    route.add(0,rtime);
                    // sende das Ergebnis dem mobilen Agenten
                    mySendMessage(sender,typ,route.toString());
...

    }
}
```

A.6 Der TSP-Algorithmus Patch

Das Routenplanermodul besitzt mehrere TSP-Algorithmen. Dazu wurde eine generische Oberklasse `TSPAlgo` der TSP-Algorithmen implementiert:

```
package routenplaner;

import java.util.*;

/**
 * Stellt einen Algorithmus zur Berechnung einer kurzen Route
 * durch alle Orte einer übergebenen Karte bereit.
 * Je nach verwendeter Heuristik kann diese Route auch erheblich länger
 * als die kürztmöglichste sein.
 */
class TSPAlgo {
    /* Name  des Algorithmus*/
    String id;

    /**
     * Ein neues TSPAlgo Objekt wird angelegt.
     */
    TSPAlgo() {
        id = ''Abstrakte Oberklasse'';
    }

    /**
     * Berechnet eine Route durch alle Orte der übergebenen Karte
     * Für die Distanzmatrix der Orte muss die Dreiecksungleichung gelten.
     * Außerdem muss für  (i!=j) gelten:  unendlich>entferung[i][j]>0 .
     *
     * @param map Karte der Orte
     * @param order in dieses Feld wird die Route zurückgegeben
     *                                (order[i]=Nummer des i.Ortes)
     *
     * @return Länge der Route
     */
    public float findWay(MagicMap map, int[] order) {
        return 0;
    }

    /**
     * Gibt den Algorithmusnamen aus.
     *
     * @return Algorithmusname
     */

    public String toString() {
        return ''TSPAlgo'';
    }
}
```

Als Beispiel zur Implementierung eines TSP-Algorithmus wird der Patch-Algorithmus aufgelistet:

```
package routenplaner;
```

```
import java.util.*;

class Patch extends TSPAlgo{

    Patch () {}

    public float findWay(MagicMap map,int order[]) {
        int      n=order.length;
        float    weg[][]=map.weg;
        int      next[];
        int      subtours[]=new int[n]; // i-teSubtour -> Startorte der Subtouren
        float    stLength[]=new float[n];       //Startort -> Länge der Subtouren
        int      stOAnz[]=new int[n];   ///Startort ->Länge der Subtouren in Orten
        int      stAnz=0;                       //Anzahl der Subtouren
        int      stAnzOld=0;                    //Anzahl der Subtouren nach Matching
        boolean  used[]=new boolean[n];
        float    gesamtStrecke,l;
        int      i,i1,i2,aktOrt=0,lastOrt,s,z,anz;
        int      s1,s2,max,maxi=0,maxL,lastTour,aktTour;
        int      min1=-2,min2=-2;
        float    min,d;
        fillDiag(map.weg,Float.MAX_VALUE/100);
        next=APAlgo.solve(map.weg);             /* Matching berechnen*/
        fillDiag(map.weg,0);
        gesamtStrecke=APAlgo.getSum(map.weg,next);
        Arrays.fill(used,false);                /* Sub Touren + ihre Länge bestimmen */
        stAnz=0;
        for(i=0;i<n;i++) {
            if (!used[i]) {
                s=i;
                subtours[stAnz]=s;
                l=0;anz=0;
                aktOrt=s;
                while (next[aktOrt]!=s) {
                    l+=weg[aktOrt][next[aktOrt]];
                    used[aktOrt]=true;
                    aktOrt=next[aktOrt];
                    anz++;
                }
                used[aktOrt]=true;
                anz++;
                l+=weg[aktOrt][s];
                stLength[s]=l;
                stOAnz[s]=anz;
                stAnz++;
            }
        }
        stAnzOld=stAnz;
        lastTour=-1;
        while (stAnz>0) {
            max=0;maxL=0;             /* Subtour mit maximaler Ortanzahl bestimmen */
            for(i=0;i<stAnzOld;i++){
                s=subtours[i];
                if ((s>=0) && (stOAnz[s]>maxL)) {
                    maxL=stOAnz[s];
                    maxi=i;
                    max=s;
                }
            }
            aktTour=max;
            subtours[maxi]=-1;
```

```
            stAnz--;
            if (lastTour>=0) {        /* Verbinde Subtouren  aktTour und lastTour*/
                s1=lastTour;          /* suche minimale Verbindung*/
                min=Float.MAX_VALUE;min1=-1;min2=-1;
                for(i1=0;i1<stOAnz[lastTour];i1++) {
                    s2=aktTour;
                    for(i2=0;i2<stOAnz[aktTour];i2++){
                        d=weg[s1][next[s2]]+weg[s2][next[s1]]
                               -(weg[s1][next[s1]]+weg[s2][next[s2]]);
                        if (d<min) {
                            min=d;
                            min1=s1;
                            min2=s2;
                        }
                        s2=next[s2];
                    }
                    s1=next[s1];
                }
                s1=next[min1];
                next[min1]=next[min2];
                next[min2]=s1;
                stOAnz[aktTour]=stOAnz[aktTour]+stOAnz[lastTour];
            }
            lastTour=aktTour;
        }
        aktOrt=0;i=0;                                   /* Ausgabe zusammenbauen*/
        while(next[aktOrt]!=0){
            order[i++]=aktOrt;
            aktOrt=next[aktOrt];
        }
        order[i++]=aktOrt;
        return map.getLength(order);
    }


    void fillDiag(float matrix[][],float v) {
        int n=matrix.length;
        for(int x=0;x<n;x++) {
            matrix[x][x]=v;
        }
    }

    public String toString(){
        return ''Patch'';
    }
}
```

A.7 Ein Testagent zur Migrationsoptimierung

Die Nutzung der Migrationsoptimierung verdeutlicht der folgende Testagent. Die Route ist zu Testzwecken vorgegeben.

```
import de.unijena.tracy.agent.*;
import de.unijena.tracy.comm.*;
import de.unijena.tracy.util.*;
```

```
import migration.ServerMatrix;

public class TestAgent extends MobileAgent {

    public static final String AGENT_NAME = ‘‘TestAgent’’;
    public static final String MIG_NAME = ‘‘MapAgent’’;
    public static String HOME_SERVER = null;
    private long time = 0;
    private MigrationProperties mp = null;

    private ServerMatrix matrix = null;

    private int count = 0;

    public TestAgent2 test2 = new TestAgent2();

    // Definitionen vor dem Start des Agenten
    public void defineAgent() {}

    // Der Agentenserver wird gestoppt
    public void systemFailure() throws AgentExecutionException { die(); }

    // Diese Methode wird beim Start eines Agenten auf seiner
    // Heimatplattform aufgerufen
    public void startAgent() throws AgentExecutionException {
        // Die Route soll optimiert werden
        mp = new MigrationProperties(‘‘optimize’’);
        // Die Route wird zum Test vorgegeben
        mp.setRoute( new String[] { ‘‘tracy://ipc052.inf.uni-jena.de/james’’
                                   ,‘‘tracy://ipc051.inf.uni-jena.de/james’’
                                   ,‘‘tracy://ipc030.inf.uni-jena.de/james’’
                                   ,‘‘tracy://ipc033.inf.uni-jena.de/james’’} );
        // Setze die Methode, die auf dem Ziel aufgerufen werden soll
        mp.setMethod(‘‘runRemote’’);
        try {
            // Erfragt die zur Optimierung notwendigen Kartendaten
            sendMessage(MIG_NAME+‘‘@’’+getServerName(),‘‘GSP’’,
                        StringArray.stringArrayToString(mp.getRoute()));
        } catch (Exception e) {};
        time = System.currentTimeMillis();
    }

    // Falls die Migration scheitert...
    public void migrationFailed() throws AgentExecutionException {
        super.migrationFailed();
        die();
    }

    // Auf der entfernten Plattform...
    public void runRemote() throws AgentExecutionException {
      System.out.println(‘‘runRemote :’’);
      mp.setMapInfo(matrix.getMatrixString());
      count++;
      if (count <= 2) go(mp);
      else go_home(‘‘runHome’’);
    }

    // Zurück von der Reise...
    public void runHome() throws AgentExecutionException {
        // stop execution
        System.out.println(System.currentTimeMillis()-time+‘‘ ms’’);
```

```
        System.out.println(‘‘Back Home!!!’’);
        die();
    }

    public void handleMessage( Message m ) throws AgentExecutionException {
        // Der Start der Reise
        if (m.getTyp().equals( ‘‘GO’’ )) {
            System.out.println(‘‘...GO...’’);
            go(mp);
        // Übergabe der Kartendaten für den Optimierer in Kalong
        } else if (m.getTyp().equals( ‘‘MAPINFO’’ )) {
            System.out.println(‘‘...MAPINFO erhalten... ’’+m.getContent());
            mp.setMapInfo(m.getContent());
            matrix = new ServerMatrix(m.getContent());
            System.out.println(‘‘...MAPINFO TestAgent ’’);
        }
    }


    public static String dummy60k=  ‘‘asdf.... ’’;

}
```

Anhang B

Glossar der wichtigsten Begriffe

Agency

*Eine andere Bezeichnung für → **Agentenserver**.*

Agent

Ein im Auftrag eines anderen Handelnder.

Agent, mobiler

*Eine Softwarekomponente, die im Interesse eines anderen handelt (handelnder Stellvertreter), sich selbständig über Systemgrenzen hinweg bewegt (→ **autonome Mobilität**) und dessen Lebenszyklus von einer Langlebigkeit geprägt ist, die über jene eines einzelnen Teilsystems hinausgeht. Ein mobiler Agent ist ein → **Softwareagent**.*

Agent, realer

*Ein → **Agent** der realen Welt.*

Agent, real-künstlicher

*Ein → **Roboter**; → **realer Agent**.*

Agent, real-natürlicher

*Ein menschlicher → **realer Agent**.*

Agent, serialisierter

*Daten und Zustand eines Agenten, die zur Übertragung in einen → **Bytestream** geschrieben werden. Dieser Mechanismus wird in Java als Serialisation bezeichnet.*

Agent, stationärer

Eine Softwarekomponente, die im Interesse eines anderen handelt (handelnder Stellvertreter). Die Lebenszeit eines stationären Agenten ist durch die des lokalen Systems beschränkt. Ein stationärer Agent ist ein → ***Softwareagent****.*

Agent, virtueller

Ein → ***Softwareagent*** *in der virtuellen Welt.*

Agent, virtuell-mobiler

Ein → ***mobiler Agent****; →* ***Softwareagent****.*

Agent, virtuell-stationärer

Ein → ***stationärer Agent****; →* ***Softwareagent****.*

Agentennetzwerk

Ein Netzwerk aus → ***Agentenservern****.*

Agentenplattform

Eine andere Bezeichnung für → ***Agentenserver****.*

Agentenserver

Die Ausführungsumgebung für einen Agenten, eine Instanz des → ***Agentensystems****. Basisfunktionalitäten werden dem Agenten zur Nutzung angeboten.*

Agentensystem

Eine Software, die alle nötigen Teile zur Ausführung eines Agenten beinhaltet.

Agentensystem, mobiles

Ein → ***Agentensystem*** *für mobile Agenten.*

Bytecode

Der statische Code des Agenten, die compilierten Klassen. In Java die Class-Dateien.

Bytestream

Ein Folge von Bytes. In Java zur Serialisierung genutzt, um den Zustand von Objekten zu speichern → ***serialisierter Agent****.*

Client

Bezeichnung für die Rolle eines Dienstnutzers im Netzwerk; → ***Klient****; im →* ***Client-Server-System*** *und im →* ***Peer-to-Peer-System****.*

Client-Server-System

Ein Netzwerk, in dem Dienstnutzer (→ ***Klienten****, →* ***Clients****) Dienste von Diensterbringern (→* ***Servern****) in Anspruch nehmen. Die Rollen sind fest vergeben.*

Code Server
*Ein → **Server**, den ein mobiler Agent einrichten und auf dem er bestimmte Klassen seines Codes zwischenspeichern kann.*

Domain
*Eine Menge zusammengehöriger → **Agentenserver**.*

Domain-Knoten
*Ein Mitglied (→ **Agentenserver**) einer → **Domain**.*

Domain-Manager
*Ein → **Domain-Knoten** einer → **Domain**, der mit der Verwaltung der Domain beauftragt ist.*

Domain Master
*Ein → **Domain-Manager**, der → **Domains** verwaltet und untereinander bekannt macht.*

Domain Service
*Ein Konzept zur Strukturierung des → **Agentennetzwerks**. Die Menge der → **Agentenserver** wird in → **Domains** zerteilt.*

Dynamic Domain Map
*Eine → **Webkarte**.*

Informationsbasis
*Bestand an Daten; → **Webkarte**.*

Kalong
Die Migrationskomponente von TRACY.

Kartenmodul
*Eine Komponente von → **ProNav**, die für die Erstellung und Aktualisierung der → **Webkarte** verantwortlich ist.*

Klient
*→ **Client**.*

Migration
*Wanderung eines → **mobilen Agenten** durch das → **Agentennetzwerk**.*

Migration, schwache
*Eine → **Migration**, in der die Ausführung des Agenten auf der Zielplattform mit einer festgelegten Methode fortgesetzt wird.*

Migrationsoptimierer
*Eine Komponente von → **ProNav**, die eine Route eines → **Agenten** auf technischer Ebene optimiert. Berechnung einer geeigneten → **Migrationsstrategie**.*

Migration, starke

*Eine → **Migration**, in der die Ausführung des Agenten auf der Zielplattform mit der nächsten dem Migrationsbefehl folgenden Anweisung fortgesetzt wird.*

Migrationsstrategie

*Die Art der Übertragung des → **Bytecode** eines → **Agenten**. Sie unterscheiden sich im Zeitpunkt der Übertragung, in der benutzten Code-Granularität und der Anzahl der Übertragungsziele. → **Pull-Strategie**; → **Push-Strategie**.*

Mirror Server

*Ein → **Server**, den ein mobiler Agent einrichten und auf dem er seine Daten ablegen oder zwischenspeichern kann.*

Peer

Ein Gleichgestellter, der Client- und Serverrolle in einem erfüllt und daher gleichzeitig Dienste anderer nutzen und selbst anbieten kann.

Peer-to-Peer-System

*Ein logisches Netzwerk, bestehend aus → **Peers**.*

Plattform

*Eine andere Bezeichnung für → **Agentenplattform**; → **Agentenserver**.*

Platz

*Ein logischer Ort auf einer → **Agentenplattform**, dort können mehrere Plätze existieren.*

ProNav

*Proaktive Navigation – ein Routing Service zur proaktiven autonomen Navigation mobiler Agenten, bestehend aus: → **Kartenmodul**, → **Routenplaner**, → **Migrationsoptimierer**.*

Pull-Strategie

*Eine Übertragungsstrategie von Agenten, wobei die Zielplattform bei Bedarf den → **Bytecode** nach Ankunft des → **serialisierten Agenten** anfordert (dynamisches Nachladen).*

Push-Strategie

*Eine Übertragungsstrategie von Agenten, in der der → **Bytecode** des Agenten zusammen mit dem → **serialisierten Agenten** übertragen wird.*

Reiseroute

*Eine Menge von Zielplattformen, die der → **mobile Agent** in einer bestimmten Reihenfolge besuchen will.*

Roboter

Eine personifizierte, programmgesteuerte Maschine, deren Arbeitsweise der eines Menschen nachgebildet ist.

Routenplaner

*Eine Komponente von → **ProNav**, die zu einer Menge von Zielen eine Reiseroute für einen → **mobilen Agenten** erstellt.*

Routenplanung, autonome

*Ein → **mobiler Agent** plant selbständig seine Reiseroute durch ein → **Agentennetzwerk**. Dazu benutzt er die Dienste, die von → **ProNav** angeboten werden.*

Routing Service

*Ein Dienst zur Unterstützung → **mobiler Agenten** bei der autonomen und proaktiven Navigation durch das → **Agentennetzwerk**.*

RTT

Round Trip Time – die Zeit, die benötigt wird, um ein minimales Paket zu einem Rechner im Netz zu versenden und eine Antwort zu erhalten.

Sensor

*Eine technische Wahrnehmungseinheit bei → **Robotern**. Eine Softwarekomponente zur Wahrnehmung bei → **Agenten**.*

Sensorik

*Die Gesamtheit der → **Sensoren** bei → **Robotern**.*

Sensorium

Die Gesamtheit aller Wahrnehmungssinne eines Menschen.

Server

Bezeichnung für die Rolle eines Diensterbringers im Netzwerk.

Softwareagent

Eine Softwarekomponente, die im Interesse eines anderen handelt (handelnder Stellvertreter).

Systemagent

*Ein → **stationärer Agent**.*

TRACY

Das mobile Agentensystem der Friedrich-Schiller-Universität Jena.

Webkarte

*Eine → **Informationsbasis** in → **ProNav** zur Orientierung im → **Agentennetzwerk** für → **Agenten**. Bestandteil des → **Kartenmoduls**. Beinhaltet Knoten- und Kanteninformationen zu → **Domain-Knoten** der lokalen und der entfernten → **Domains**.*

Anhang C

Abkürzungsverzeichnis

ATSP	*Asymmetrisches Traveling Salesman Problem*
Bots	*Robots*
CSMA/CD	*Carrier Sense Multiple Access with Collision Detection*
DAI	*Distributed Artificial Intelligence*
DIA	*Domain Information Agent*
FIPA	*Foundation of Intelligent Physical Agents*
GPS	*Global Positioning System*
HK	*Held-Karp-Lowerbound*
ICMP	*Internet Control Message Protocol*
IEEE	*Institute of Electrical and Electronics Engineers*
IP	*Internet Protocol*
JADE	*Java Agent DEvelopment Framework*
Java VM	*Java Virtual Machine*
JDK	*Java Development Kit*
JRE	*Java Runtime Environment*
KI	*Künstliche Intelligenz*
LAN	*Local Area Network*

MAC	*Media Access Control*
MAS	*Mobiles Agentensystem*
MASIF	*Mobile Agent System Interoperability Facility*
OMG	*Object Management Group*
OS	*Operating System, dt. Betriebssystem*
OSI	*Open Systems Interconnection*
P2P	*Peer-to-Peer*
PING	*Packet Internet Groper*
ProNav	*Proactive Navigation*
RFC	*Request for Comment*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
RTT	*Round Trip Time*
SeMoA	*Secure Mobile Agents*
STSP	*Symmetrische Traveling Salesman Problem*
SW	*Software*
tc	*Traffic Control*
TCP	*Transmission Control Protocol*
Tcl	*Tool Command Language*
TSP	*Traveling Salesman Problem*
UML	*Unified Modeling Language*
URL	*Universal Resource Locator*
WLAN	*Wireless Local Area Network*
WSDL	*Web Service Description Language*

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel und Literatur angefertigt habe.

Jena, den 12. Juli 2004

Lebenslauf

Persönliche Daten

Name	Christian Erfurth
Geburtsdatum	30. Juni 1971
Geburtsort	Naumburg (Saale)

Schulische und berufliche Ausbildung

1978 - 1988	Polytechnische Oberschule Abschluss: 10. Klasse
1988 - 1991	Berufsausbildung zum Schienenfahrzeugschlosser mit Abitur Abschlüsse: Abitur, Facharbeiter
1993 - 1999	Studium der Informatik an der Friedrich-Schiller-Universität in Jena mit Nebenfach Mathematische Ökologie Abschluss: Diplom-Informatiker Während des Studiums Werkstudent bei der IBM Deutschland am Wissenschaftlichen Zentrum der IBM in Heidelberg

Beruflicher Werdegang

1998 - 1999	Freier Mitarbeiter bei der H.A.S.E. GbR in Hünfelden / Kirberg Consulting und Programmierdienstleistungen
seit 2000	Wissenschaftlicher Mitarbeiter am Lehrstuhl für Softwaretechnik der Friedrich-Schiller-Universität in Jena

Jena, 12. Juli 2004