

# Entwicklungs- und Simulationsunterstützung für mobile Anwendungen in multihop Ad-Hoc-Netzen

Dissertation

zur Erlangung des akademischen Grades  
des Doktors der Naturwissenschaften  
am Fachbereich IV der Universität Trier

vorgelegt von

Diplom-Informatiker  
Johannes K. Lehnert

Juli 2004



Gutachter: Prof. Dr. Peter Sturm                      Prof. Dr. Stefan Fischer  
                  Systemsoftware und Verteilte Systeme    Institut für Telematik  
                  Universität Trier                                Universität zu Lübeck

Disputation: 17. Dezember 2004



*Meinen Eltern*



# Zusammenfassung

---

Diese Arbeit stellt eine einheitliche Workbench zur Entwicklung von mobilen Anwendungen in multihop Ad-Hoc-Netzwerken vor. Die einheitliche Workbench besteht aus drei Bausteinen: einem Simulator für diese Netzwerke, einer hybriden Emulationsumgebung für mobile multihop Ad-Hoc-Netzwerke und einer Ausführungsplattform für Anwendungen auf mobilen Geräten. Anwendungen können bereits im Simulator vollständig implementiert und evaluiert werden. Der entstandene Code kann unverändert sowohl im Emulationsteil als auch auf der Ausführungsplattform für mobile Geräte eingesetzt werden. Im passenden dreistufigen Entwicklungsprozeß wird die Anwendung im Simulator implementiert und getestet, bevor sie – zusammen mit einer graphischen Oberfläche – in der hybriden Emulationsumgebung weiter evaluiert und optimiert wird. Zuletzt erfolgt die Erprobung auf mobilen Geräten im Feldversuch.

Mehrere tausend bis zehntausend mobile Geräte können in der Workbench durch die Beschränkung auf die topologischen Aspekte des Ad-Hoc-Netzwerks und eine besondere Bewegungsmodellabstraktion, die die besonders effiziente Berechnung der Bewegungs- und Konnektivitätsdaten der Geräte ermöglicht, effizient simuliert werden. Die Vorausberechnung und Wiederverwendung dieser Daten ist möglich. Simulationen können in Echtzeit detailliert visualisiert werden, wobei die Art der Visualisierung und das Ausgabeformat vom Benutzer definiert werden können.

Die Workbench und der Softwareentwicklungsprozeß für Anwendungen in mobilen multihop Ad-Hoc-Netzwerken werden anhand einer Fallstudie erprobt. Dabei werden die Erfahrungen bei der Implementierung einer Middleware für Ad-Hoc-Anwendungen sowie bei der Entwicklung einer selbstorganisierenden Auktionsanwendung aufgezeigt.





# Danksagung

---

An erster Stelle möchte ich meinem Doktorvater Herrn Prof. Dr. Peter Sturm danken. Er hatte stets ein offenes Ohr für Probleme und Fragen und hat mir durch seine besondere Fähigkeit, Visionen griffig zu formulieren, immer wieder geholfen, das Gesamtbild auch bei einer Vielzahl von Problemen nicht aus den Augen zu verlieren. Mit der ihm eigenen Großzügigkeit ermöglichte er mir vielfältige Dienstreisen und beschaffte jederzeit die benötigte Ausstattung.

Herrn Prof. Dr. Stefan Fischer von der TU Braunschweig danke ich für seine Bereitschaft, als Gutachter für diese Arbeit zur Verfügung zu stehen.

Die DZ BANK International S.A. und hier insbesondere Herr Ulli Storck sind mir über den gesamten Zeitraum meiner Dissertation mit flexiblen Arbeitszeitregelungen entgegengekommen und haben mein Dissertationsvorhaben stets wohlwollend unterstützt und mich gleichzeitig bestmöglich gefördert.

Meinen Kollegen in der Arbeitsgruppe „Systemsoftware und Verteilte Systeme“, Steffen Rothkugel, Hannes Frey und Daniel Görgen, möchte ich für ihre Unterstützung und Hilfsbereitschaft danken. Sie sorgten für ein längst nicht selbstverständliches offenes und angenehmes Arbeitsklima, so daß es eine besondere Freude war, mit ihnen an der Verwirklichung unseres Gesamtprojekts zu arbeiten. In zahlreichen Diskussionen haben sie meine Ideen kritisch hinterfragt und mir viele Anregungen und Hinweise gegeben.

Frau Weitzel danke ich für ihre Hilfe bei allen „Administrivialitäten“; so konnte ich mich ohne zeitraubende Beschäftigung mit Formalitäten ganz auf meine Arbeit konzentrieren.

Meinen Eltern und meinen beiden Brüdern möchte ich für ihre vielfältige Unterstützung und Ermutigung während der gesamten Zeit danken. Sie haben mich immer wieder darin bestärkt, daß ich auf dem richtigen Weg bin.

Nicht zuletzt möchte ich meiner Freundin Sandra Overwin für ihre Liebe und Unterstützung danken.

Jens Doppelhamer, Hannes Frey, Daniel Görden, Sandra Overwin und Thomas Perst haben meine Arbeit korrekturgelesen und vielfältige Hinweise auf Verbesserungsmöglichkeiten gegeben. Dafür nochmals vielen Dank.

# Inhaltsverzeichnis

---

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Ubiquitous Computing . . . . .	1
1.2	Ad-Hoc-Netzwerke . . . . .	2
1.3	Anwendungsentwicklung in Ad-Hoc-Netzen . . . . .	3
1.4	Beitrag dieser Arbeit . . . . .	5
	Einheitliche Workbench . . . . .	5
	Softwareentwicklungsprozeß . . . . .	5
	Effiziente Simulation . . . . .	6
	Praktische Erprobung . . . . .	6
1.5	Gliederung der Dissertation . . . . .	6
<b>2</b>	<b>Stand der Forschung und verwandte Themen</b>	<b>7</b>
2.1	Mobile Ad-Hoc-Netzwerke . . . . .	7
	Routing in Ad-Hoc-Netzwerken . . . . .	8
	Funktechnologien . . . . .	10
	Transportprotokolle in Ad-Hoc-Netzwerken . . . . .	13
	Beschränkungen der Rechenleistung und Energie . . . . .	14
	Sicherheitsaspekte . . . . .	14

2.2	Simulation . . . . .	15
	Kontinuierliche Simulationen . . . . .	16
	Diskrete Simulationen . . . . .	16
2.3	Netzwerksimulatoren für Ad-Hoc-Netze . . . . .	17
2.4	„Quasistandard“ ns-2 . . . . .	20
2.5	Bewegungsmodelle . . . . .	23
	Random Walk Mobility Model . . . . .	23
	Random Waypoint Mobility Model . . . . .	24
	Gauss-Markov Mobility Model . . . . .	25
	Smooth Random Mobility Model . . . . .	25
	Graph-Based Mobility Model . . . . .	26
	Manhattan Grid Mobility Model . . . . .	26
	Obstacle Mobility Model . . . . .	27
	Reference Point Group Mobility Model . . . . .	27
2.6	Emulation von Ad-Hoc-Netzwerken . . . . .	28
<b>3</b>	<b>Workbench: Simulation</b>	<b>31</b>
3.1	Grundlagen . . . . .	32
3.2	Dynamische Aspekte . . . . .	34
	Mobilität . . . . .	36
	Konnektivitätsberechnung . . . . .	36
3.3	Netzwerkmodell . . . . .	40
	Einfaches Netzwerkmodell . . . . .	40
	Realistischere Netzwerkmodelle . . . . .	41
	Nachrichten und Protokolle . . . . .	42
3.4	Hardwareabstraktion . . . . .	43

---

3.5	Benutzerverhalten und Applikationen . . . . .	43
3.6	Visualisierung . . . . .	44
3.7	Statistik . . . . .	47
3.8	Entwicklung einer Beispielapplikation . . . . .	48
	Implementierung der Ping-Anwendung . . . . .	50
	Implementierung des Benutzerverhaltens . . . . .	56
	Konfiguration der Workbench . . . . .	58
	Ausführung der Simulation . . . . .	63
3.9	Performancemessungen . . . . .	63
<b>4</b>	<b>Workbench: Hybride Emulation</b>	<b>69</b>
4.1	Architektur . . . . .	70
4.2	Synchronisation . . . . .	70
4.3	Graphische Oberfläche . . . . .	71
4.4	Hybride Emulation der Beispielanwendung . . . . .	72
	Externes Benutzerverhalten . . . . .	72
	Graphische Benutzeroberfläche . . . . .	77
	Ausführung der Emulation . . . . .	78
<b>5</b>	<b>Workbench: Übertragung auf mobile Geräte</b>	<b>79</b>
5.1	Operating System . . . . .	80
5.2	Netzwerkimplementierung . . . . .	80
5.3	Nachbarschaftserkennung . . . . .	81
5.4	Positionsbestimmung . . . . .	81
5.5	Graphische Oberfläche . . . . .	82
5.6	Ausführung einer Beispielanwendung . . . . .	82

<b>6 Fallstudie: UbiBay</b>	<b>87</b>
6.1 SELMA – Middleware für Ad-Hoc-Netze . . . . .	87
6.2 UbiBay . . . . .	90
6.3 Erfahrungen . . . . .	91
<b>7 Fazit</b>	<b>95</b>
<b>Literaturverzeichnis</b>	<b>98</b>

# 1

## Einleitung

---

### 1.1 Ubiquitous Computing

---

Mark Weiser hat in seinem Aufsatz „The Computer of the 21st Century“ [91] den Begriff des *Ubiquitous Computing*, des allgegenwärtigen Computers, geprägt. Er hatte die Vision von hunderten Computern pro Raum, von Computern, die „verschwinden“ und alltäglicher Bestandteil des Lebens werden – so wie die Schriftsprache und die Fähigkeit zu Lesen fester Bestandteil unseres Lebens sind.

In den sechziger Jahren des vorigen Jahrhunderts nutzten viele Menschen gemeinsam einen Großrechner. Mit dem Aufkommen des Personal Computers verschob sich dieses Verhältnis zu einer 1:1-Beziehung: jeder Nutzer hatte seinen persönlichen Rechner, den ausschließlich er benutzte. Ubiquitous Computing verschiebt dieses Verhältnis weiter zugunsten der Nutzer: nun hat ein Nutzer nicht mehr nur einen persönlichen Rechner, sondern eine Vielzahl von Rechnern zu seiner Verfügung. Als Beispiel für diese Entwicklung sei auf die inzwischen eingetretene Allgegenwart von Mobiltelefonen hingewiesen.

Mit zunehmender Miniaturisierung wird das Verschwinden der Computer immer einfacher, da auch sehr kleine Geräte entsprechend viel Rechenleistung bieten oder diese durch den Zusammenschluß einer Vielzahl weniger leistungsfähiger Geräte

bereitstellen können. Extrem billig herzustellende, intelligente Geräte wie Funketiketten (RFIDs [25, 74]) machen es möglich, daß beispielsweise die Produkte im Supermarktregal Auskunft über ihren Zustand bzw. ihre Eigenschaften geben können. Netzwerke von Sensoren organisieren sich selbständig und geben so Auskunft über ihre Umwelt. Als Beispiel kann die Überwachung von mit Sandsäcken verstärkten Deichen dienen [7]. In jedem Sandsack steckt ein intelligenter Sensor, der mit den Sensoren in den anderen Sandsäcken ein Netzwerk bildet und eindringendes Wasser mit Hilfe dieses Netzwerkes an die entsprechenden Hilfskräfte meldet.

Bis zum vollständigen Eintreten der Vision Weisers ist es sicher noch ein langer Weg, allerdings stellen schon die ersten Schritte auf diesem Weg die Informatik vor neue Herausforderungen. Schon heute gibt es viele Geräte, die mittels entsprechender Funktechnologien wie Wireless LAN [47] oder Bluetooth [64] über kurze Distanzen miteinander kommunizieren können. Auch Szenarien wie das oben beschriebene Deichüberwachungs-Sensornetzwerk sind heute technisch machbar. Viele dieser Kleingeräte sind mobil, sie werden herumgetragen oder sind in Fahrzeuge eingebaut. Sie bilden an ihren jeweiligen „Aufenthaltsorten“ spontan Netzwerke, sogenannte (mobile) Ad-Hoc-Netzwerke.

Diese Ad-Hoc-Netzwerke sind in gewisser Weise das Gegenteil traditioneller Netzwerke: die Zahl der Netzwerkknoten ist im Voraus nicht bekannt und kann sich jederzeit ändern. Netzwerkknoten können zu beliebigen Zeiten zum Netzwerk hinzukommen und es wieder verlassen. Die Netzwerktopologie des Ad-Hoc-Netzwerkes ändert sich aufgrund der hinzukommenden und wegfallenden Knoten sowie der Mobilität ständig, und ohne weitere Vorkehrungen ist eine Kommunikation zwischen Knoten nur innerhalb der Sendereichweite der Knoten möglich. Eine zentrale Netzwerkadministration fehlt.

## 1.2 Ad-Hoc-Netzwerke

---

Man kann zwei Arten von Ad-Hoc-Netzwerken unterscheiden: *singlehop* und *multihop* Ad-Hoc-Netzwerke. In *singlehop* Ad-Hoc-Netzwerken kommunizieren die einzelnen Geräte stets nur mit Geräten in ihrem direkten Sendebereich, über genau einen „Hop“. Diese Art von Ad-Hoc-Netzwerken ist heute Standard und entsprechend weit verbreitet. Als Beispiele können die Kopplung von Bluetooth-Headsets



mit Bluetooth-Mobiltelefonen oder Wireless LANs im Infrastrukturmodus, die den Zugang zur herkömmlichen Netzwerkinfrastruktur über Accesspoints zur Verfügung stellen, dienen.

Mobile multihop Ad-Hoc-Netzwerke erlauben die Kommunikation auch über mehrere Hops hinweg, d.h. es werden entsprechende Routingverfahren oder andere Techniken eingesetzt, um auch außerhalb des Sendebereichs befindliche Geräte erreichen zu können. Aufgrund der hohen Fluktuation und Mobilität in diesen Ad-Hoc-Netzwerken ist die Entwicklung geeigneter Kommunikationsmuster und Anwendungen Gegenstand aktueller wissenschaftlicher Forschung.

Die hohe Änderungsrate der Netzwerktopologie und das Fehlen einer zentralen Infrastruktur bedingen, daß mobile multihop Ad-Hoc-Netzwerke selbstorganisierend sein müssen, d.h. die Entscheidungen eines Gerätes primär auf lokal verfügbarer Information basieren müssen. Sowohl das Netzwerk als auch die Anwendungen müssen neu hinzukommende Geräte jederzeit integrieren können und das Ausscheiden integrierter Geräte aus dem Netzwerk verkraften.

Aus der Mobilität der Geräte ergeben sich weitere Limitierungen. Die Geräte sind klein und haben nur begrenzte Rechenleistung und Speicherkapazität, die im Regelfall mehrere Größenordnungen hinter der aktueller Desktoprechner zurückbleibt. Aufgrund des Batteriebetriebs steht Energie nur sehr begrenzt zur Verfügung, was auch das mögliche Maß an drahtloser Kommunikation einschränkt. Anwendungen müssen diese Restriktionen berücksichtigen.

## 1.3 Anwendungsentwicklung in Ad-Hoc-Netzen

---

Eine Vielzahl von Middlewareplattformen wie CORBA [68] oder DCOM [10] erleichtert die Entwicklung von verteilten Anwendungen in herkömmlichen stationären Netzwerken. Dabei existieren allgemein anerkannte Standards für die Entwicklung verschiedener Anwendungstypen (Client-Server, Peer-to-Peer etc.), die dem Entwickler als Richtschnur dienen.

Im Gegensatz dazu ist bisher unklar, wie Anwendungen für Ad-Hoc-Netzwerke am besten zu implementieren sind und welche Kommunikationsmuster sich am besten für den jeweiligen Anwendungszweck eignen. Hinzu kommen die Auswirkungen der Selbstorganisation, die die Vorhersagbarkeit des Verhaltens der Anwendung in einem großen Ad-Hoc-Netzwerk zusätzlich erschweren.

Aufgrund dieser Unsicherheiten in der Anwendungsentwicklung kann man nicht sicher sein, daß eine fertig entwickelte Anwendung wirklich funktioniert. Die Erprobung einer Anwendung in Feldversuchen ist zwar möglich, aber problematisch. Feldversuche in mobilen multihop Ad-Hoc-Netzwerken sind aufwendig und teuer. Es werden eine Vielzahl von mobilen Geräten (mehrere hundert) sowie eine entsprechende Zahl von Versuchspersonen benötigt, die die Geräte und die Anwendung bedienen und sich auf dem Versuchsgelände bewegen. Auf allen Geräten muß die entsprechende Software zur Verfügung stehen. Versuche mit einer geringen Anzahl von Geräten sind nicht sinnvoll, da das Ad-Hoc-Netzwerk in diesem Fall in viele kleine Partitionen (oft aus nur einem Gerät bestehend) zerfallen kann, zwischen denen nur selten Kommunikation möglich ist. Zudem lassen sich z.B. Hochlasteffekte, wie sie bei Gerätehäufungen auftreten können, nicht ohne eine entsprechend hohe Gerätezahl nachvollziehen.

Die hohen Kosten für Feldversuche stehen dem Test einer nur eventuell funktionierenden Anwendung entgegen. Aus gleichem Grund will der Entwickler sicherstellen, daß seine Anwendung so weit wie möglich fehlerfrei ist, bevor er sie in einem Feldversuch erprobt. Die übliche Vorgehensweise ist daher, die zentralen Algorithmen der Anwendung mittels eines Simulators für Ad-Hoc-Netzwerke zu testen und dann die Anwendung, die auf diesen Algorithmen basiert, auf den jeweiligen mobilen Geräten zu implementieren. Teilweise kommen auch Testbeds zum Einsatz, die die Emulation eines drahtlosen Netzwerks über ein herkömmliches stationäres Netzwerk erlauben und so einen Test der Anwendung in diesem Umfeld ermöglichen.

Problematisch an dieser Vorgehensweise ist, daß die Simulatoren üblicherweise ein völlig anderes Programmiermodell vorsehen als die reale Hardwareplattform. So können zwar die Algorithmen simuliert werden, sie müssen aber später zusammen mit den anderen Teilen der Anwendung neu implementiert werden. Dabei können Fehler entstehen, die erst im Feldversuch (und damit zu spät) auffallen und somit zu dessen Scheitern führen können. Zudem ist die Skalierbarkeit der Simulatoren oft beschränkt und erlaubt nur die Simulation von maximal wenigen hundert Geräten, so daß Hochlastsituationen oder große Ad-Hoc-Netzwerke nicht effizient simuliert werden können.

## 1.4 Beitrag dieser Arbeit

---

### Einheitliche Workbench

---

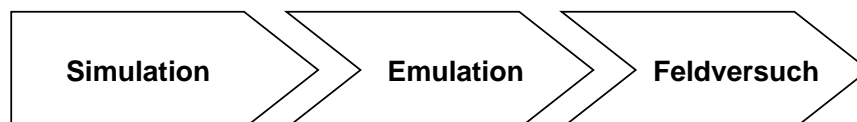
Diese Arbeit stellt eine einheitliche Workbench [32, 57] zur Entwicklung von Anwendungen in mobilen multihop Ad-Hoc-Netzwerken vor. Die einheitliche Workbench besteht aus drei Bausteinen: einem Simulator für diese Netzwerke, einer hybriden Emulationsumgebung für mobile multihop Ad-Hoc-Netzwerke und einer Ausführungsplattform für Anwendungen auf mobilen Geräten.

Der Simulator erlaubt, Anwendungen bereits in der Simulation vollständig zu implementieren und evaluieren. Der dabei entstandene Code kann dann *unverändert* sowohl im Emulationsteil der Workbench als auch auf der Ausführungsplattform für mobile Geräte eingesetzt werden. Der Entwickler einer Anwendung muß den Anwendungscode nur einmal implementieren und kann ihn in allen drei Teilen der Workbench testen. Insbesondere kann der Code unverändert auf den mobilen Geräten ausgeführt werden. Dies wird durch ein einheitliches Programmiermodell und API (Application Programming Interface) in allen drei Umgebungen ermöglicht. Auf diese Weise wird nur tatsächlich getesteter Code in Feldversuchen erprobt.

### Softwareentwicklungsprozeß

---

Passend zu dieser Dreiteilung ergibt sich ein dreistufiger Entwicklungsprozeß (siehe Abbildung 1.1).



**Abbildung 1.1:** Dreistufiger Softwareentwicklungsprozeß bei Verwendung der Workbench.

Der Entwickler implementiert seine Anwendung im Simulator und testet sie dort. Der Simulator unterstützt ihn dabei durch ein hohes Abstraktionsniveau, ausgefeilte Visualisierungsmöglichkeiten und hohe Skalierbarkeit, d.h. große Szenarien lassen sich schnell simulieren. In einem zweiten Schritt kann der Entwickler die so entstandene Anwendung zusammen mit einer graphischen Oberfläche in der hybriden Emulationsumgebung weiter testen und optimieren, bevor er sie (unverändert)

auf die Ausführungsplattform für mobile Geräte überträgt und in Feldversuchen erprobt.

## **Effiziente Simulation**

---

Die Simulationsumgebung der Workbench erlaubt die effiziente Simulation von Anwendungen in mobilen multihop Ad-Hoc-Netzwerken. Mehrere tausend bis zehntausend mobile Geräte können effizient simuliert und visualisiert werden. Die Skalierbarkeit der Simulationsumgebung wird durch eine Bewegungsmodellabstraktion ermöglicht, die eine besonders effiziente Berechnung der Bewegungen der Geräte sowie der Konnektivität zwischen den mobilen Geräten erlaubt. Zusätzlich ist auf der Basis der gewählten Abstraktionen eine Vorausberechnung der Konnektivitäts- und Bewegungsdaten und damit eine weitere, erhebliche Verbesserung der Ausführungsgeschwindigkeit möglich. Die benutzerdefinierte und -erweiterbare Visualisierungskomponente erlaubt die effiziente Darstellung auch komplexer Simulationsszenarien. Die Visualisierungskomponente ist ausgabeformatunabhängig, so daß verschiedene Renderbackends möglich sind.

## **Praktische Erprobung**

---

Die einheitliche Workbench und der Softwareentwicklungsprozeß für Anwendungen in mobilen multihop Ad-Hoc-Netzwerken werden anhand einer Fallstudie erprobt. Dabei werden die Erfahrungen bei der Implementierung einer Middleware für Ad-Hoc-Anwendungen sowie bei der Entwicklung einer selbstorganisierenden Auktionsanwendung (UbiBay) aufgezeigt.

## **1.5 Gliederung der Dissertation**

---

In Kapitel 2 werden der derzeitige Stand der Forschung und verwandte Arbeiten in einschlägigen Themengebieten vorgestellt. Kapitel 3 beschreibt den ersten Teil der Workbench, den Simulator für mobile Ad-Hoc-Netzwerke. Die Kapitel 4 und 5 behandeln die weiteren Teile der Workbench, den hybriden Emulationsteil und die Ausführungsplattform für mobile Geräte. Praktische Erfahrungen aus der Arbeit mit der Workbench werden anhand einer Fallstudie in Kapitel 6 geschildert. Die Arbeit schließt mit einem zusammenfassenden Kapitel.

# 2

## Stand der Forschung und verwandte Themen

---

Dieses Kapitel liefert einen Überblick über den Stand der Forschung und verwandte Arbeiten in den Bereichen mobile Ad-Hoc-Netzwerke, Simulation, Netzwerksimulation, Netzwerkemulation und Bewegungsmodelle zur Simulation von Ad-Hoc-Netzwerken.

### 2.1 Mobile Ad-Hoc-Netzwerke

---

*Mobile Ad-Hoc-Netzwerke* bestehen aus Knoten, die mittels einer drahtlosen Kommunikationstechnologie miteinander kommunizieren und sich frei bewegen. Das Netzwerk besitzt keine feste Infrastruktur, sondern die Knoten organisieren sich mittels kooperativem und altruistischem Verhalten selbst. Direkte Kommunikation zweier Knoten ist nur möglich, wenn sie sich innerhalb der gegenseitigen Sende-reichweite befinden, andernfalls muß das Ad-Hoc-Netzwerk mit Hilfe von Routing-Algorithmien die indirekte Kommunikation ermöglichen. Die Netzwerktopologie ändert sich aufgrund der Knotenmobilität ständig, zusätzlich können Knoten das

Ad-Hoc-Netzwerk jederzeit verlassen bzw. sich dem Netzwerk anschließen. Aufgrund ihrer Eigenschaften sind mobile Ad-Hoc-Netzwerke aus technischer Sicht sehr anspruchsvoll. Einige der technischen Herausforderungen in mobilen Ad-Hoc-Netzwerken werden im folgenden kurz vorgestellt.

## Routing in Ad-Hoc-Netzwerken

---

Im Gegensatz zu stationären Netzen müssen Routing-Algorithmen in Ad-Hoc-Netzwerken mit einer sich ständig ändernden Topologie zurechtkommen, wobei das Fehlen zentraler Einrichtungen noch erschwerend hinzukommt. Routingprotokolle für Ad-Hoc-Netzwerke lassen sich in zwei Gruppen einteilen: *positionsbasierte* [63] und *topologiebasierte* Protokolle [75], die sich nochmals in *proaktive* und *reaktive* Protokolle unterteilen.

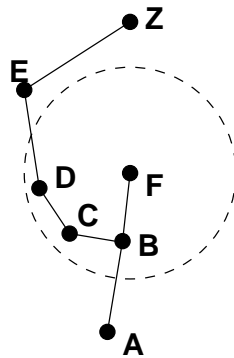
Proaktive topologiebasierte Routingprotokolle reagieren auf Netzwerktopologieänderungen und aktualisieren die Routinginformationen ständig. Die Verwendung proaktiver Protokolle in Ad-Hoc-Netzwerken kann problematisch sein, da die dauernde Aktualisierung der Routinginformationen bei hoher Knotenmobilität kostbare Kommunikationsbandbreite und Energie verbraucht. Beispiele für proaktive Protokolle sind *Destination Sequenced Distance-Vector Routing (DSDV)* [72] und *Wireless Routing Protocol (WRP)* [65].

Reaktive topologiebasierte Protokolle versuchen nicht, dauerhaft aktuelle Routinginformationen vorzuhalten, sondern ermitteln diese erst bei Bedarf. Oft wird dabei die Route durch eine (begrenzte) Form des Netzwerkflutens ermittelt, was zu hohem Nachrichtenaufkommen führen kann. Das Ermitteln einer geeigneten Route kann längere Zeit in Anspruch nehmen, so daß es zu Verzögerungen kommen kann. Vorteilhaft ist, daß reaktive Protokolle nur dann zu zusätzlichem Nachrichtenaufkommen führen, wenn ein mobiler Knoten kommunizieren will. Dies wirkt sich besonders stark bei geringer Kommunikationshäufigkeit aus. Gängige reaktive Protokolle sind *Ad Hoc On-Demand Distance Vector Routing (AODV)* [73], *Dynamic Source Routing (DSR)* [50] und *Temporally Ordered Routing Algorithm (TORA)* [71].

Die ersten heuristischen positionsbasierten Routingprotokolle sind Mitte der achtziger Jahre entstanden [84] und setzen voraus, daß die eigene Position und die Positionen der direkten Nachbarn bekannt sind. Die Position der direkten Nachbarn wird

normalerweise über periodische Broadcasts aktualisiert, während die Position der Zielknoten über einen selbstorganisierenden *Location service* im Ad-Hoc-Netzwerk ermittelt wird. Statt Routen zu berechnen, entscheiden die einzelnen Knoten bei diesen Verfahren allein anhand der Zielposition und der Positionen der direkten Nachbarn über die Route zur Weiterleitung von Paketen. Mögliche Strategien zur Weiterleitung sind z.B. *MFR* (*most forward within radius*), bei der das Paket zu dem Nachbarn weitergeleitet wird, der innerhalb des Senderadius am nächsten in Richtung des Zielknotens liegt, oder *NFP* (*nearest with forward progress*), bei der das Paket zum am nächsten in Richtung des Zielknotens liegenden Nachbarn weitergeleitet wird.

Problematisch an diesen heuristischen Verfahren ist, daß die Auslieferung eines Pakets an sogenannten konkaven Netzwerkknoten zum Erliegen kommen kann, wenn die Weiterleitungsstrategie keinen geeigneten Knoten findet. In einem solchen Fall spricht man von einem *Greedy-Routing-Fehler*, der beispielhaft in Abbildung 2.1 dargestellt ist.



**Abbildung 2.1:** Beispiel für einen Greedy-Routing-Fehler. Es wird eine Route von Knoten A zu Knoten Z gesucht. Knoten F hat innerhalb seines Sendebereichs (Kreis) keinen Nachbarn, der näher in Richtung Z liegt. Trotzdem existiert eine Route zu Knoten Z: über B, C, D und E.

Zur Behebung dieser Greedy-Routing-Fehler kommen Recoverystrategien zum Einsatz, wenn ein Paket an einem konkaven Knoten nicht weiter in Richtung Ziel geroutet werden kann. *Greedy Face Greedy (GFG)* [8] und *Greedy Perimeter Stateless Routing (GPSR)* [52] setzen planares Graphenrouting ein, um das lokale Maximum des konkaven Knotens zu überwinden. Dazu wird ein planarer Graph über die Knoten des Ad-Hoc-Netzwerks gebildet, der dann genutzt wird, um Wege zum Zielknoten zu finden. Eine Alternative ist beschränktes, gerichtetes Fluten, das in

*Distance routing effect algorithm for mobility (DREAM)* [5] eingesetzt wird. Im Fall eines lokalen Maximums wird die erwartete Zielregion (ein Kreis um die Position des Zielknotens) berechnet und das Paket an alle direkten Nachbarn, die sich im Korridor zu dieser Zielregion befinden, weitergeleitet.

In jüngerer Zeit ist mit *Beacon-less Position-based Routing* [26] noch eine Variante des positionsbasierten Routings hinzugekommen, die keine periodischen Broadcasts zur Nachbarschaftsbestimmung benötigt. Da die Knoten ihre Nachbarknoten nicht kennen, leiten sie ankommende Pakete einfach an alle direkten Nachbarn weiter. Jeder empfangende Nachbarknoten berechnet aufgrund seiner Position zum Zielknoten eine Wartezeit, die er wartet, bevor er das Paket weiterleitet. Der Knoten, der die beste Position zum Zielknoten hat, berechnet die kürzeste Wartezeit und leitet das Paket daher zuerst weiter. Alle anderen Knoten verwerfen das Paket. Damit dieses Verfahren funktioniert, nehmen nur diejenigen Nachbarknoten innerhalb eines Weiterleitungsbereichs an dem Verfahren teil, für die gilt, daß sie jeweils die Nachrichten aller anderen Knoten innerhalb des Weiterleitungsbereichs empfangen können.

## Funktechnologien

---

In mobilen Ad-Hoc-Netzwerken werden heute primär zwei Funktechnologien eingesetzt: *Bluetooth* [64] und *Wireless LAN* (IEEE802.11a/b/g) [47, 34]. Dies resultiert aus der breiten Verfügbarkeit entsprechender Hardware zu relativ moderaten Preisen. Hinzu kommen noch spezielle Funktechnologien wie z.B. IEEE802.15.4 [46], optional erweitert um das ZigBee-Protokoll [98], für Sensornetzwerke, die für die besonders limitierten Ressourcen der Sensoren in Bezug auf Energie, Rechenleistung und Speicherkapazität optimiert sind.

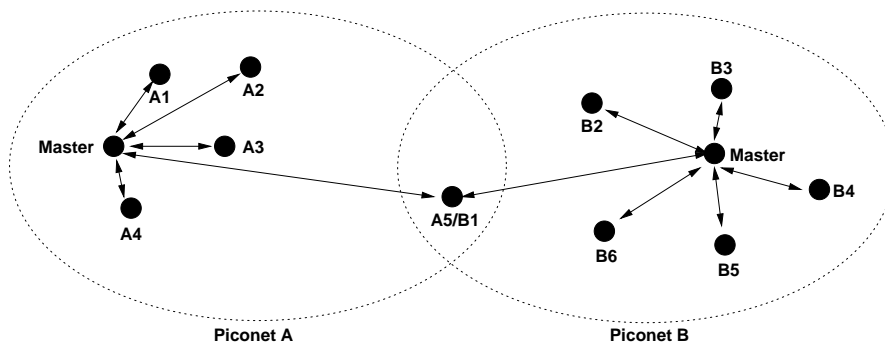
Bluetooth ist ein offener Standard der Bluetooth Special Interest Group, der mit dem Ziel entwickelt wurde, die Kabel z.B. zwischen Desktop-Computern und zugehöriger Peripherie oder zwischen Mobiltelefonen und Headsets durch drahtlose Kommunikation zu ersetzen. Die übliche Sendereichweite ist mit  $10m$  (es gibt auch Bluetoothgeräte mit  $100m$  Reichweite) relativ gering, genau wie die maximale Bandbreite von  $768kBit/s$ . Bei der Entwicklung von Bluetooth wurde besonderer Wert auf einen niedrigen Energieverbrauch gelegt. Bluetooth ist auf der ganzen Welt lizenzfrei einsetzbar und ist sowohl für Sprach- als auch für Datenübertragungen geeignet.



Ein großes Spektrum an vordefinierten Profilen erlaubt, daß Geräte verschiedener Hersteller miteinander kommunizieren und gleichzeitig auch für höhere Protokolle auf wohldefinierte Schnittstellen zugreifen können. So existieren beispielsweise Profile für die Übertragung von Dateien, zur Synchronisation von Daten, zur seriellen Kommunikation, zur Modemeinwahl in Netzwerke, zum Betrieb von Headsets etc. Bluetooth-Geräte müssen nicht alle Profile unterstützen und Hersteller können zudem eigene, nicht standardisierte Profile definieren.

Bluetooth bietet ein eigenes Device Discovery Protokoll, mit dessen Hilfe andere Geräte im Sendebereich gesucht bzw. entdeckt werden können. Zudem erlaubt das Device Discovery Protokoll auch, weitere Informationen über andere Geräte, wie z.B. unterstützte Profile, abzufragen.

Zwei Bluetooth-Geräte, die miteinander kommunizieren, bilden automatisch ein *Piconet*. Dabei übernimmt eines der Geräte die logische Rolle des Masters, während das andere als Slave fungiert. Der Master eines Piconets legt fest, nach welchem Schema die 79 Funkkanäle immer wieder gewechselt werden (*Frequency Hopping*), und definiert, welches Gerät wann senden darf. Maximal sieben Geräte können gleichzeitig in einem Piconet aktiv sein, zusätzlich können bis zu 255 „geparkte“ Geräte Teil eines Piconets sein. Piconets, die einander überlappen, können ein *Scatternet*<sup>1</sup> bilden, d.h. einzelne Geräte sind Teil mehrerer Piconets (siehe Abbildung 2.2).



**Abbildung 2.2:** Beispiel für ein Bluetooth Scatternet. Gerät A5/B1 ist sowohl Bestandteil von Piconet A als auch von Piconet B.

Wireless LAN existiert in mehreren Versionen: IEEE802.11b und IEEE802.11g sind zueinander kompatibel, 802.11b erlaubt eine Übertragung mit maximal 11Mbit/s,

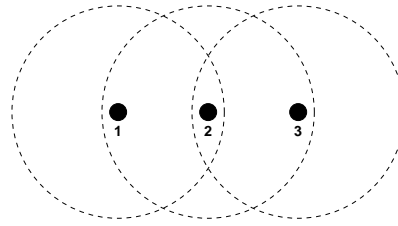
<sup>1</sup>Leider fehlt derzeitigen Bluetooth-Chips oft noch die Scatternetunterstützung, so daß diese Netze in der Realität selten genutzt werden können.

während 802.11g 54 MBit/s ermöglicht. IEEE802.11a erlaubt ebenfalls maximal 54 MBit/s, ist aber nicht kompatibel zu 802.11b. Die Sendereichweite liegt normalerweise bei ca. 100m, wobei dieser Wert innerhalb von Gebäuden erheblich geringer (ca. 30m) ist. Mittels spezieller Antennen und höherer Sendeleistungen ist es möglich, im Freien auch größere Entfernungen zu überbrücken.

IEEE802.11 kennt zwei verschiedene Betriebsmodi: ein Netzwerk kann entweder im Infrastrukturmodus oder im Ad-Hoc-Modus betrieben werden. Im Infrastrukturmodus gibt es einen zentralen Accesspoint, der den Zugang zum herkömmlichen stationären Netzwerk bereitstellt. Alle mobilen Geräte müssen sich innerhalb der Reichweite des Accesspoints befinden und kommunizieren nur direkt mit dem Accesspoint. Zur Abdeckung größerer Areale ist es möglich, mehrere Accesspoints über ein entsprechendes Backbone-Netzwerk zu verbinden. Wireless LAN unterstützt in diesem Fall die Übergabe (Handover) der mobilen Geräte zwischen den einzelnen Accesspoints. Im Ad-Hoc-Modus gibt es keinen dedizierten Accesspoint, stattdessen kommunizieren die Geräte direkt miteinander. Anwendungen des Ad-Hoc-Modus reichen von der spontanen Bildung kleiner Netzwerke z.B. in Meetings bis zur Bildung größerer mobiler multihop Ad-Hoc-Netzwerke.

Ein Problem in Funknetzwerken ist das *Hidden Terminal Problem* [88] (siehe Abbildung 2.3). Falls drei mobile Geräte so plaziert sind, daß zwar Gerät 1 und 2 sowie Gerät 2 und 3 in gegenseitiger Funkreichweite liegen, Station 1 und 3 aber nicht direkt miteinander kommunizieren können, so würden ohne besondere Vorkehrungen die Stationen 1 und 3 gleichzeitig senden. Gerät 2 könnte in diesem Fall nichts empfangen, da Kollisionen entstehen, während die beiden gleichzeitig sendenden Geräte dies nicht merken. In stationären Netzen besteht dieses Problem bei der Zugriffskontrolle auf das Medium nicht, da alle angeschlossenen Stationen sich gegenseitig „hören“ können.

IEEE802.11 bietet als Lösung für das Hidden Terminal Problem das RTS/CTS-Verfahren (Request-to-Send/Clear-To-Send) an. Dabei sendet ein sendewilliges Gerät zuerst ein RTS an den Empfänger. Diese spezielle Nachricht hat den Effekt, daß jedes andere Gerät, das sie empfängt, selbst nicht sendet. Das empfangende Gerät antwortet mit einem CTS, das der sendewilligen Station anzeigt, daß sie senden kann. Zusätzlich sorgt auch ein CTS dafür, daß alle Geräte, die es empfangen, selbst nicht mehr senden. Auf diese Weise werden auch Geräte, die das sendende Gerät nicht „sehen“ kann, am gleichzeitigen Senden gehindert. Der Nachteil am



**Abbildung 2.3:** Das Hidden Terminal Problem. Abgebildet sind drei mobile Geräte mit ihren Senderadien. Da Gerät 1 und 3 nicht direkt miteinander kommunizieren können, kann es passieren, daß beide gleichzeitig senden und Gerät 2 aufgrund von Kollisionen nichts empfangen kann.

RTS/CTS-Verfahren ist, daß es zum einen aufwendig ist und zum anderen im Fall hoher Mobilität der Geräte Kollisionen auch nicht vollständig verhindern kann.

## Transportprotokolle in Ad-Hoc-Netzwerken

---

Herkömmliche Transportprotokolle wie TCP eignen sich nicht für die von großer Fluktuation und Veränderung geprägte Topologie eines mobilen Ad-Hoc-Netzwerks. Im Gegensatz zu stationären Netzen, in denen Nachrichtenverluste primär aufgrund von Verstopfungen auftreten, bedingt die Unzuverlässigkeit der drahtlosen Übertragung (höhere Bitfehlerrate, Verbindungsabbrüche durch Mobilität) in Ad-Hoc-Netzwerken eine viel höhere Nachrichtenverlustrate. In einem solchen Netzwerk führt die Kommunikation über TCP zu einer signifikanten Durchsatzverschlechterung und hohen interaktiven Verzögerungen, da TCP Fehler immer als eine Folge von Verstopfungen interpretiert und entsprechend reagiert (*Slow start*, *Window-Verkleinerung*, *Exponential backoff*). In [4] werden daher verschiedene Techniken zur Anpassung von TCP an Ad-Hoc-Netzwerke aufgezeigt. Dabei werden drei Arten der Anpassung unterschieden: Ende-zu-Ende-Anpassungen (Anpassungen sowohl bei Sender als auch beim Empfänger), geteilte Verbindungen (TCP nur bis zur Basisstation, ab da ein spezielles Protokoll) und Anpassungen auf dem Link Layer.

Bei Ende-zu-Ende-Anpassungen beherrscht der TCP-Sender Verluste, indem er selektive Acknowledgements erlaubt und mittels ELN (Explicit Loss Notification) zwischen Verlusten aufgrund von Verstopfungen und anders bedingter Verluste unterscheidet. Geteilte Verbindungen nutzen TCP nur bis zur Basisstation und setzen ein eigenes, verlässliches Protokoll auf der Funkstrecke ein. Anpassungen auf dem

Link Layer versuchen, Nachrichtenverluste durch lokale Neuübertragungen vor TCP zu verbergen.

Weiterhin gibt es verschiedene Neuentwicklungen von Transportprotokollen speziell für Ad-Hoc-Netzwerke. Als Beispiele können hier *ATP* (Ad-hoc transport protocol) [81] oder *TPA* (transport protocol for ad-hoc networks) [1] dienen.

## Beschränkungen der Rechenleistung und Energie

---

Mobile Ad-Hoc-Netzwerke basieren oft auf mobilen Kleingeräten (dies muß nicht notwendigerweise so sein: mobile Ad-Hoc-Netzwerke können z.B. auch von Fahrzeugen [19] oder Flugzeugen gebildet werden), die mit einer geeigneten Funktechnologie ausgerüstet sind. Diese Geräte sind derzeit in Bezug auf Rechenzeit und Speicherkapazität mehrere Größenordnungen von Desktoprechnern entfernt. Anwendungen für Ad-Hoc-Netzwerke müssen dies berücksichtigen und können z.B. keine großen Datenmengen zwischenspeichern. Weiterhin beruht die Energieversorgung der Kleingeräte auf Batterien oder Akkus und reicht deshalb nur für sehr begrenzte Zeit, meist wenige Stunden. Die Kommunikation per Funk benötigt zusätzliche Energie und verringert so die Laufzeit weiter, wobei der Energiebedarf zur Kommunikation signifikant höher ist als der zur Berechnung bzw. Speicherung von Daten. Daher müssen Anwendungen in Ad-Hoc-Netzwerken die Kommunikation mit anderen möglichst minimal halten, um entsprechend Energie zu sparen.

## Sicherheitsaspekte

---

Aufgrund der drahtlosen Kommunikation, der Selbstorganisation, der fehlenden zentralen Infrastruktur und der aufgrund der Mobilität ständig wechselnden Topologien sind mobile Ad-Hoc-Netzwerke einer Vielzahl von Sicherheitsproblemen ausgesetzt. Bereits die grundlegenden Mechanismen von mobilen Ad-Hoc-Netzen wie z.B. Routing sind für Attacken anfällig [44]. So findet die gesamte Kommunikation drahtlos statt und kann daher sowohl abgehört als auch aktiv gestört werden. Die mobilen Geräte sind im Gegensatz zu herkömmlichen Netzwerkknoten leicht zu entwenden und können nach dem Diebstahl verändert und kompromittiert werden. Die Algorithmen, die in den Basismechanismen verwendet werden, gehen von kooperativem und altruistischem Verhalten der mobilen Knoten aus. Dies gilt sowohl für den MAC-Layer (korrektes Verhalten z.B. zur Kollisionsvermeidung)

als auch für Routingprotokolle, Nachrichtenverteilverfahren oder Nachbarschaftsbestimmung, in denen böswillige Knoten Nachrichten unterdrücken, verändern oder fälschen können und damit die entsprechenden Verfahren unbrauchbar machen oder beeinträchtigen.

Die drahtlose Kommunikation zwischen den mobilen Knoten kann mittels Verschlüsselung abgesichert werden. Routingverfahren können um entsprechende Sicherungsmechanismen erweitert werden (z.B. *Secure Ad hoc On-Demand Distance Vector Routing (SAODV)* [96]). Mit Hilfe eines virtuellen Bezahlsystems [12] kann die Kooperation der mobilen Knoten erzwungen werden, indem Knoten für die Bereitstellung von Diensten bezahlt werden und gleichzeitig selbst für die Nutzung des Netzes bezahlen müssen. Auf diese Weise läßt sich auch eine Überlastung des Netzwerks durch eine Vielzahl sinnloser Anfragen verhindern.

Ohne zentrale Administration können sich Knoten jederzeit dem Netzwerk anschließen oder es verlassen. Die Authentifizierung dieser Knoten ist problematisch, insbesondere, da es keine zentrale Zertifizierungsinstanz im Ad-Hoc-Netzwerk gibt. Mittels *Threshold secret sharing* wird versucht, selbstorganisierende, verteilte Public-Key-Infrastrukturen (PKI) für Ad-Hoc-Netzwerke zu schaffen [54]. Problematisch an diesen Verfahren ist, daß sie auf einer Initialisierung der Geräte durch eine zentrale Zertifizierungsinstanz beruhen. In [14] wird ein alternativer Ansatz vorgestellt, der die Mobilität der Geräte nutzt, um bidirektionale Sicherheitsbeziehungen durch direkte Begegnungen der Geräte bzw. einstufige Freundschaftsbeziehungen aufzubauen. Ein zweiter Kommunikationskanal, der nur auf sehr kurze Entfernungen funktioniert, wird zum Austausch der Sicherheitsinformationen genutzt.

## 2.2 Simulation

---

Die Untersuchung eines Modells anstelle des echten Systems ist eine übliche Vorgehensweise. Sie kommt zur Anwendung, wenn eine Untersuchung des Originalsystems nicht möglich ist, da das zu untersuchende System noch nicht existiert, es bei der Untersuchung zerstört werden würde, oder aber die Untersuchung zu aufwendig ist bzw. zu lange dauern würde.

Das gewählte Modell sollte die Eigenschaften des Systems möglichst gut wiedergeben. Das Ziel ist, Rückschlüsse über das Verhalten des realen Systems zu erhalten. Diese Untersuchung kann analytisch oder in Simulationen erfolgen.

Die analytische Untersuchung eines Modells besitzt den Vorteil, daß die Ergebnisse – unter den getroffenen Annahmen – nachweisbar korrekt sind. Allerdings führt die Komplexität der Modelle oft dazu, daß stark vereinfachende Annahmen getroffen werden müssen, um die analytische Untersuchung beherrschbar zu machen. Dies wirkt sich wiederum negativ auf die Qualität der Untersuchungsergebnisse aus.

Die Simulation eines Modells hat mehrere Vorteile: es lassen sich auch sehr komplexe Modelle untersuchen und die Annahmen können realistischer gehalten werden. Weiterhin lassen sich die zugrundeliegenden Wahrscheinlichkeitsverteilungen und Eingabeparameter jederzeit austauschen, was bei der analytischen Untersuchung meist nicht möglich ist. Zusätzlich lassen sich Simulationen mit entsprechender Visualisierung auch gut zur Veranschaulichung einsetzen. Nachteilig bei Simulationen ist der fehlende Nachweis der Korrektheit und die möglicherweise hohen Anforderungen an Speicherplatz und Rechenzeit bei der Ausführung der Simulation.

Simulationen dynamischer Systeme können prinzipiell in zwei Gruppen unterteilt werden: *kontinuierliche* und *diskrete* Simulationen [55].

## Kontinuierliche Simulationen

Kontinuierliche Simulationen beschreiben die kontinuierliche Änderung der Zustandsgrößen des Modells in der Zeit. Der Zusammenhang zwischen den einzelnen Zustandsgrößen und ihre Änderungen werden meist über Gleichungssysteme und Differentialgleichungen beschrieben. Simulation bedeutet in diesem Zusammenhang die Lösung dieser Gleichungen mittels numerischer Verfahren.

## Diskrete Simulationen

Bei diskreten Simulationen springt die Simulationszeit von einem Zeitpunkt zum nächsten. *Zeitgesteuerte* diskrete Simulationen nutzen dabei ein Zeitinkrement fester Größe und führen jeweils alle Simulationsereignisse aus, die innerhalb dieses Inkrements stattfinden. Nachteilig an diesem Ansatz ist, daß auch die Zeiten simuliert werden müssen, zu denen keine Simulationsereignisse stattfinden, und daß es oft schwierig ist, die geeignete Zeitgranularität zu finden. *Ereignisgesteuerte* diskrete Simulationen vermeiden dieses Problem, indem die Simulationszeit stets bis zum Zeitpunkt des nächsten Ereignisses springt.

Simulationen sind heute in der Informatik wie auch in vielen anderen Bereichen weit verbreitet. Für diese Arbeit sind insbesondere Netzwerksimulationen, speziell von mobilen (multihop) Ad-Hoc-Netzen, interessant.

## 2.3 Netzwerksimulatoren für Ad-Hoc-Netze

---

Netzwerksimulatoren für mobile Ad-Hoc-Netzwerke lassen sich anhand verschiedener charakteristischer Eigenschaften, die ihre Eignung für bestimmte Anwendungsgebiete beeinflussen, unterscheiden.

Eine wichtige Eigenschaft ist der *Detaillierungsgrad* der Simulation, d.h., werden die Netzwerkprotokolle und das drahtlose Netzwerk möglichst genau bis auf die physikalische Ebene simuliert oder stehen mehr die topologischen Eigenschaften des Ad-Hoc-Netzwerkes im Vordergrund [38]. Mit steigendem Detaillierungsgrad wächst der Berechnungsaufwand stark. Soll in einer Simulation z.B. das Verhalten einer Netzwerktechnologie untersucht werden, so ist ein möglichst genaues Modell notwendig, um sinnvolle Ergebnisse zu erhalten. Daher wird man in einem solchen Fall einen Simulator mit hohem Detaillierungsgrad im physikalischen Netzwerkmodell verwenden. Will man aber z.B. einen neuen Informationsverteilungsalgorithmus in einem mobilen Ad-Hoc-Netzwerk mit einer großen Anzahl von Geräten testen, so sind oft nur die topologischen Eigenschaften des Netzes entscheidend und man kann daher den durch den geringeren Detaillierungsgrad eingesparten Berechnungsaufwand für die Simulation der höheren Gerätezahl verwenden.

Generell ist selbst bei Simulatoren mit gleichem Detaillierungsgrad oft die Vergleichbarkeit der Simulationsergebnisse nicht gegeben. In [15] wurde ein Floodingalgorithmus für ein einfaches Szenario – 50 mobile Geräte mit IEEE802.11b auf  $1km \times 1km$  Fläche gleichverteilt, bewegt nach dem *Random Waypoint Mobility Model* [9] – mit den Simulatoren ns-2, OPNET Modeler und GloMoSim simuliert. Dabei kam es teils zu erheblichen, nicht nur quantitativen, sondern auch qualitativen Abweichungen bei den Ergebnissen. Die Autoren merken an, daß Simulationsergebnisse erheblich vom verwendeten Simulator abhängen und daher eine Konzentration der Forschungsgruppen auf ein einziges Simulationspaket kontraproduktiv sein kann.

Die *Entstehungsgeschichte* eines Netzwerksimulators beeinflusst ebenfalls seine Eigenschaften. Falls der Simulator aus einem bestehenden Netzwerksimulator für stationäre Netze entstanden ist, ist die Unterstützung für mobile Netzwerkknoten nicht

immer optimal gelöst, da das ursprüngliche Design dies erschwert oder unmöglich macht. Vorteilhaft ist in diesem Fall allerdings, daß auf erprobte Netzwerkmodelle und -protokolle zurückgegriffen werden kann. Wird ein Netzwerksimulator dagegen speziell für mobile Ad-Hoc-Netze entwickelt, so ist eine optimale Unterstützung mobiler Knoten möglich, und es kann besser auf die spezifischen Probleme mobiler Netzwerke eingegangen werden.

Setzt man zur Entwicklung von Anwendungen für mobile Ad-Hoc-Netzwerke einen Simulator ein, so ist das *Abstraktionsniveau* eine weitere charakterisierende Eigenschaft. Bietet eine Simulationsumgebung Abstraktionen z.B. für die Device Discovery, verschiedene Routingverfahren, Positionsbestimmung etc., so wird die Anwendungsentwicklung dadurch erleichtert und beschleunigt.

Es existiert eine Vielzahl von Netzwerksimulatoren für mobile Ad-Hoc-Netzwerke, die die oben aufgeführten Eigenschaften in verschiedenen Kombinationen und Stärken besitzen. Im Folgenden seien einige gängige Simulationspakete kurz vorgestellt.

Einer der bekanntesten Netzwerksimulatoren ist der Network Simulator in der Version 2 [24], kurz *ns-2* genannt. Er ist ein typisches Beispiel für einen traditionellen Netzwerksimulator, der später für mobile Ad-Hoc-Netze erweitert wurde. *ns-2* wird seit 1990 entwickelt, beginnend mit einem Network Simulator des Lawrence Berkeley National Laboratory (LBL) für stationäre Netze, der auf dem REAL [53] Netzwerksimulator aufbaute. Daraus entstand 1995 *ns-1*, die Version 1 des Network Simulator, und schließlich 1997 *ns-2*. Ab 1998 wurden die Erweiterungen für mobile Ad-Hoc-Netzwerke aus dem Monarch-Projekt der Carnegie Mellon University in *ns-2* integriert [49]. Im Laufe der langen Entwicklungszeit wurde eine Vielzahl von Netzwerkprotokollen und -technologien in verschiedenen Forschergruppen implementiert, in *ns-2* integriert und validiert. Im Bereich mobiler Ad-Hoc-Netzwerke stehen so z.B. eine vollständige Implementierung von IEEE 802.11b [47], Bluetooth [64, 56] sowie alle gängigen Ad-Hoc-Routingalgorithmen zur Verfügung. Dies hat dazu geführt, daß *ns-2* weite Verbreitung auch bei der Simulation mobiler Ad-Hoc-Netzwerke gefunden hat. Aufgrund seiner Bedeutung wird *ns-2* im nächsten Abschnitt noch genauer vorgestellt.

*GloMoSim* [97] (Global Mobile Information Systems Simulation Library) ist ein auf Skalierbarkeit ausgelegter Simulator für mobile Ad-Hoc-Netze, der am Parallel Computing Laboratory der University of California, Los Angeles (UCLA) entwickelt wird. Durch die Parallelisierung der Simulation wird die Simulation auch



sehr großer Ad-Hoc-Netze mit zehntausenden von Geräten angestrebt. Technisch basiert die Implementierung auf *Parsec* [3], einer C-artigen Programmiersprache für parallele Simulationen. Eine am OSI-Modell orientierte Schichtenarchitektur mit standardisierten APIs erlaubt die leichte Austauschbarkeit der Modelle für die einzelnen Schichten und deren unabhängige Entwicklung durch verschiedene Entwicklergruppen. So können verschiedene Protokolle problemlos miteinander verglichen werden. Direkt mitgeliefert werden verschiedene Mobilitätsmodelle, Radiowellenausbreitungsmodelle sowie MAC-Layer (*CSMA*, *IEEE 802.11*). Hinzu kommen noch die Transportprotokolle TCP und UDP, sowie verschiedene Ad-Hoc-Routingprotokolle. *CBR* (Constant Bit Rate), *FTP*, *HTTP* und *Telnet* stehen als Applikationsprotokolle zur Verfügung. Simulationsläufe können sowohl in Echtzeit als auch nachträglich visualisiert werden. GloMoSim wird für die Nutzung an Universitäten kostenlos bereitgestellt.

*QualNet* [76] ist eine stark weiterentwickelte, kommerziell vertriebene Version von GloMoSim der Firma Scalable Network Technologies. Diese Version enthält einerseits eine erheblich größere Zahl von Netzwerkmodellen und -protokollen (Satellitenkommunikation, Quality of Service, Mobilfunknetze etc.) und andererseits komfortable graphische Tools, die das Entwickeln und Aufsetzen bzw. Ausführen von Simulationen erleichtern. QualNet wird sowohl in einer rein sequentiellen als auch in einer parallelen Version angeboten.

*OPNET Modeler* [69] der Firma OPNET Technologies, Inc. ist ein Simulator für stationäre Netze, der insbesondere für Firmen gedacht ist, die ihre Netzwerke planen, untersuchen oder reorganisieren wollen. OPNET zeichnet sich vor allem durch eine große Zahl detailgenau modellierter Netzwerkkomponenten aus; so läßt sich z.B. problemlos ein simulierter Cisco Catalyst 6500 Router (!) in eine Simulation einbauen. Zusätzlich bietet OPNET auch Unterstützung für drahtlose Netze. Mobile Knoten können frei im dreidimensionalen Raum bewegt werden, Landschaftshöhenmodelle können in den Radiowellenausbreitungsberechnungen berücksichtigt werden und es können eigene Ausbreitungsmodelle eingebunden werden. Ein IEEE802.11-MAC-Layer samt darüberliegender Schichten wird zur Verfügung gestellt. Zusätzlich zu den vom Hersteller mitgelieferten Modellen gibt es auch noch Modelle, die von OPNET-Benutzern entwickelt wurden (u.a. ein Bluetoothmodell [42]). Die Parallelisierung von Simulationen durch die Nutzung mehrerer Prozessoren ist mit OPNET möglich. Simulationsszenarien werden über eine graphische Oberfläche definiert: die Knotentypen werden definiert, die Topologie wird festge-

legt und die auszuwertenden statistischen Daten werden definiert. Im Falle eigener Algorithmen und Knotentypen muß man ein NodeModel definieren, das die Eigenschaften der einzelnen Komponenten des Knotens definiert und die Datenflüsse steuert. Mittels eines ProcessModel werden dann die internen Zustände der Komponenten als Zustandsautomat definiert. Beide Modelle werden in C programmiert.

Das *Darhmouth Scalable Simulation Framework* (DaSSF) [59] bietet einen parallelen Simulator für stationäre Netzwerke. Bei der Implementierung, die dem Scalable Simulation Framework (SSF) API [79, 18] folgt, wurde sowohl auf ein möglichst einfaches Programmiermodell als auch auf möglichst hohe Skalierbarkeit geachtet. DaSSF soll Netzwerksimulationen mit mehreren zehntausend Knoten ermöglichen. Dabei kann DaSSF sowohl auf Rechnern mit Shared Memory als auch mit Distributed Memory ausgeführt werden. Aufbauend auf DaSSF wurde *SWAN* (Simulator for Wireless Ad Hoc Networks) [20] entwickelt. Die erste frei verfügbare Version (1.0 Alpha) verwendet portierte Versionen des IEEE 802.11b-Modells aus GloMoSim und unterstützt einfache Mobilitätsmodelle, die Routingprotokolle AODV und DSR, sowie IP und ARP.

Horst Hellbrück von der International University in Bruchsal entwickelt mit *AN-Sim* (Ad hoc Network Simulation) [40, 39] einen Simulator für Ad-Hoc-Netzwerke, der bewußt – wie auch die Simulationsumgebung der in dieser Arbeit vorgestellten Workbench – auf die genaue Simulation der MAC-Layer und Übertragungsprotokolle verzichtet. Die durch diese Vereinfachung gewonnene, zusätzliche Rechenzeit wird genutzt, um möglichst große Szenarien schnell simulieren zu können. Eine graphische Ausgabe des Ad-Hoc-Netzwerkes wird während der Simulation bereitgestellt. Die verschiedenen Eingangsparameter können direkt über eine graphische Benutzeroberfläche verändert werden. Die Ausgabeparameter werden simultan zum Ablauf der Simulation angezeigt. In den neueren Versionen bietet ANSim Schnittstellen zu ns-2 und GloMoSim, so daß ANSim als Szenariogenerator (Mobilität der Knoten) für diese Simulatoren genutzt werden kann.

## 2.4 „Quasistandard“ ns-2

---

Dieser Abschnitt beschreibt die Simulation von mobilen Ad-Hoc-Netzwerken mit ns-2. Der Simulator ns-2 basiert auf einer zweigeteilten Architektur: die Basis bildet eine in C++ implementierte Simulationsbibliothek. Darauf aufbauend werden

die Simulationsszenarien und -parameter in *OTcl* (Object-oriented Tcl) [70], einer interpretierten, objektorientierten Skriptsprache definiert. Die meist in C++ definierten Module für die einzelnen Komponenten des Netzwerkprotokollstacks, wie z.B. MAC-Layer, Ad-Hoc-Routingprotokolle oder TCP, sind dabei direkt aus OTcl zugreif- und konfigurierbar, so daß sich die Module leicht kombinieren lassen.

Ein Simulationslauf mit ns-2 sieht prinzipiell wie folgt aus: der Benutzer definiert ein OTcl-Skript für sein Szenario und führt es im Simulator aus. Die Simulationsergebnisse sind dann am Ende des Simulationslaufes in einem Tracefile verfügbar. Dieses Tracefile kann entweder mit für ns-2 entwickelten Analysetools (z.B. *tracegraph* [61]) oder eigenen Skripten analysiert werden oder aber – falls auch ein Trace für das Visualisierungstool *nam* (Network Animator) [66] erzeugt wurde – mit *nam* visualisiert werden.

Mobile Knoten in ns-2 entsprechen normalen Netzwerkknoten, erweitert um die Fähigkeit, über einen drahtlosen Kanal zu kommunizieren und sich im Raum zu bewegen. Neben den Eigenschaften des drahtlosen Kommunikationskanals (Ausbreitungsmodell, Antenneneigenschaften) können auch der MAC-Layer, die Interface-Queue und der Link-Layer eines mobilen Knotens definiert werden. Aufbauend auf diesen Schichten können dann das Ad-Hoc-Routingprotokoll sowie die Kommunikationsverbindungen (TCP, UDP) und Anwendungen, die diese Verbindungen nutzen (z.B. FTP), konfiguriert werden. Ein *God-Objekt* (General Operations Director) ermöglicht den Zugriff auf globale Daten der Simulation, die ein allwissender Beobachter des Szenarios zur Verfügung hätte. Beispiele für solche Daten sind die Gesamtanzahl der mobilen Knoten oder Informationen über bestehende Verbindungen zwischen den mobilen Knoten.

Alle Knoten bewegen sich auf einer rechteckigen Simulationsfläche. Eine dritte Dimension ist vorgesehen, wird aber nicht genutzt. Die Knoten werden initial an einer bestimmten Position plaziert und dann mittels Bewegungsdirektiven der Art „Knoten  $n_1$  bewegt sich zur Simulationszeit 100 mit  $3.5 \frac{m}{s}$  zum Punkt (300, 200)“bewegt. Diese Bewegungsdirektiven werden prinzipiell auch im OTcl-Simulationsskript definiert, allerdings wird dieser Teil in der Praxis normalerweise mit einem zu ns-2 gehörenden Szenariogenerator erzeugt, da die manuelle Erstellung der Bewegungsdirektiven für mehrere hundert Geräte bei einer mehrere Stunden dauernden Simulation nicht praktikabel wäre. Der Szenariogenerator kann Bewegungsdirektiven für das Random Waypoint Mobility Model erzeugen; für weitere Bewegungsmodelle

muß man entweder eigene Szenariogeneratoren entwickeln oder aber auf bestehende Generatoren wie z.B. *BonnMotion* [89] zurückgreifen.

Analog zur Vorgehensweise hinsichtlich der Bewegungsdirektiven existieren auch Trafficgeneratoren, die zufällige Verbindungen z.B. mit konstanter Bitrate zwischen den mobilen Knoten definieren und die entsprechenden Direktiven für das OTcl-Skript erzeugen.

Das Standardvisualisierungstool *nam* bietet nur rudimentäre Möglichkeiten für die Visualisierung mobiler Ad-Hoc-Netzwerke [17]. Alternativ kann *ad-hockey* [85] genutzt werden, ein in Perl implementiertes Visualisierungstool für ns-2, das zusätzlich auch die interaktive Erstellung von Bewegungsszenarien ermöglicht.

Die Implementierung neuer Protokolle in ns-2 erfolgt über die Definition eines neuen Pakettyps und eines sogenannten *Agent*, der das Verschicken und Empfangen dieses Pakettyps ermöglicht.<sup>2</sup> Die Implementierung beider Komponenten erfolgt in C++, wobei die ns-2-Bibliotheken zum Zugriff auf die Simulationskomponenten und die Speicherverwaltung zur Verfügung stehen. Über spezielle C++-Klassen werden die Agentenfunktionen in OTcl zugreifbar gemacht. Die Einbindung des neuen Protokolls in ns-2 erfolgt durch direkte Änderungen im Sourcecode von ns-2. Pakettyp und Agent werden in die entsprechenden Datenstrukturen des Simulators eingetragen und der Simulator wird neu kompiliert. Danach ist das neue Protokoll in Simulationsskripten nutzbar.

Man geht normalerweise davon aus, daß mehrere Monate Einarbeitungszeit vor einem sinnvollen Einsatz von ns-2 notwendig sind [15, 21, 80]<sup>3</sup>. Die Laufzeiten von Simulationen mit ns-2 steigen mit wachsender Gerätezahl stark an, so daß Simulationen mit mehreren hundert oder gar tausend Geräten nicht praktikabel sind. Insbesondere der Hauptspeicherbedarf ist sehr groß. Dieser Beobachtung entspricht, daß trotz der langen Entwicklungszeit von ns-2 die Optimierung des Simulationskerns nicht abgeschlossen zu sein scheint. So wird in [90] von einer ns-2 Version mit *staged simulation* berichtet, die bis zu 20-mal (!) schneller als die Standardversion von ns-2 ist. Dabei entsteht der größte Teil der Beschleunigung durch die Unterteilung der Topographie in Gitterfelder, um die Suche nach potentiellen Empfängern

---

<sup>2</sup>Natürlich ist es auch möglich, einen bestehenden Pakettyp zu nutzen und nur einen neuen Agenten zu implementieren.

<sup>3</sup>Ivan Stojmenovic schreibt in [80] auf Seite 467: „Although it is desirable to have some kind of benchmark testing facility, the problem with these simulators is a painful learning curve. Several researchers who have used it confirmed that it takes about one month of full-time work to learn how to use these simulators.“

für ein zu versendendes Paket zu beschleunigen. Statt alle Geräte zu testen, werden nur diejenigen in Betracht gezogen, die in Gitterfeldern innerhalb des Senderadius des sendenden Gerätes liegen.

## 2.5 Bewegungsmodelle

---

Eine möglichst realitätsnahe Simulation und Evaluation von Algorithmen und Anwendungen in mobilen Ad-Hoc-Netzwerken hängt neben der Genauigkeit der Simulation der Netzwerkimplementierung auch stark von den Bewegungen der mobilen Knoten ab. Diese Bewegungen können aus zwei Quellen herrühren. Zum einen können „echte“ Szenarien mit mobilen Benutzern beobachtet und deren Bewegungsdaten in der Simulation verwendet werden. Diese Vorgehensweise ist sehr aufwendig, da die Bewegungen von Menschen in einem quadratkilometergroßen Gebiet über einen längeren Zeitraum genau erfaßt werden müssen. Diese Probleme führen dazu, daß die meisten Simulationen von mobilen Ad-Hoc-Netzwerken probabilistische *Bewegungsmodelle* nutzen.

Mit der Realitätsnähe dieser Bewegungsmodelle steht und fällt die Aussagekraft der Analysen, da z.B. unrealistische Konnektivitätsverhältnisse zu falschen Annahmen und Bewertungen führen können, die im nachfolgenden Einsatz der Anwendungen in Feldversuchen nicht verifiziert werden können oder gar zum Nichtfunktionieren der Anwendungen führen. In den letzten Jahren haben sich einige „Standardbewegungsmodelle“ herausgebildet, die von vielen Forschergruppen verwendet werden. Der Aufsatz von Tracy Camp et al. [13] bietet einen guten Überblick.

Prinzipiell lassen sich die Bewegungsmodelle in zwei Gruppen unterteilen: solche, in denen sich die Knoten unabhängig voneinander bewegen (*Entity Mobility Models*) und Bewegungsmodelle, in denen die Knotenbewegungen voneinander abhängen (*Group Mobility Models*). Zu den häufig verwendeten Modellen mit unabhängigen Bewegungen der Knoten gehören das *Random Walk Mobility Model* und das *Random Waypoint Mobility Model*.

### Random Walk Mobility Model

---

Das Random Walk Mobility Model wird manchmal auch als Braun'sche Bewegung bezeichnet und basiert auf einer ganz einfachen Berechnungsvorschrift. Ein Knoten

wählt – ausgehend von seiner aktuellen Position – zufällig eine Bewegungsrichtung (aus dem Intervall  $[0, 2\pi]$ ) und eine Bewegungsgeschwindigkeit aus einem vorher definierten Geschwindigkeitsintervall. Der Knoten bewegt sich nun entweder für eine bestimmte Zeit oder, bis er eine bestimmte Distanz zurückgelegt hat, mit der gewählten Geschwindigkeit in die gewählte Richtung. Falls ein Knoten während einer Bewegung an eine der Grenzen der Simulationsfläche stößt, so prallt er von ihr ab und setzt seine Bewegung in der Abprallrichtung fort<sup>4</sup>. Problematisch am Random Walk Mobility Model sind die häufigen und abrupten Bewegungs- und Geschwindigkeitsänderungen sowie das Abprallen der mobilen Knoten vom Rand der Simulationsfläche.

## Random Waypoint Mobility Model

---

Das Random Waypoint Mobility Model [50] basiert auf einem ähnlichen Prinzip wie das Random Walk Mobility Model. Ein mobiler Knoten wählt zufällig einen Zielort auf der Simulationsfläche und eine Geschwindigkeit aus einem Geschwindigkeitsintervall aus. Der Knoten bewegt sich dann mit der gewählten Geschwindigkeit zum Zielort. Nach der Ankunft am Zielort bleibt der Knoten für eine zufällig gewählte Zeit am Zielort stehen, bevor er wieder einen neuen Zielort wählt. Das Random Waypoint Mobility Model besitzt eine ausgeprägte „Einschwingphase“, in der die Knotendichte in der Mitte der Simulationsfläche hoch ist, da viele Knoten die Mitte der Simulationsfläche durchqueren. Erst mit der Zeit verteilen sie sich langsam wieder über die Simulationsfläche. Falls diese Anfangsphase in die Simulation einfließt, kann dies insbesondere bei kurzen Simulationszeiten zu Ergebnisverzerrungen führen. Darüber hinaus ist die Knotendichte in der Mitte der Simulationsfläche stets höher als in den Außenbereichen, da die Knoten mit einer hohen Wahrscheinlichkeit die Mitte durchqueren [6]. In [94] wird zusätzlich nachgewiesen, daß die durchschnittliche Knotengeschwindigkeit im Random Waypoint Mobility Model über die Zeit immer weiter abnimmt, falls das Geschwindigkeitsintervall mit 0 beginnt. Dies liegt daran, daß sehr langsame Knotengeschwindigkeiten ausgewählt werden können und damit der Weg zum nächsten Zielort länger als die Simulationszeit dauert. Beim Einsatz dieses Mobilitätsmodells müssen diese Probleme daher unbedingt berücksichtigt werden, damit die Simulationsergebnisse nicht

---

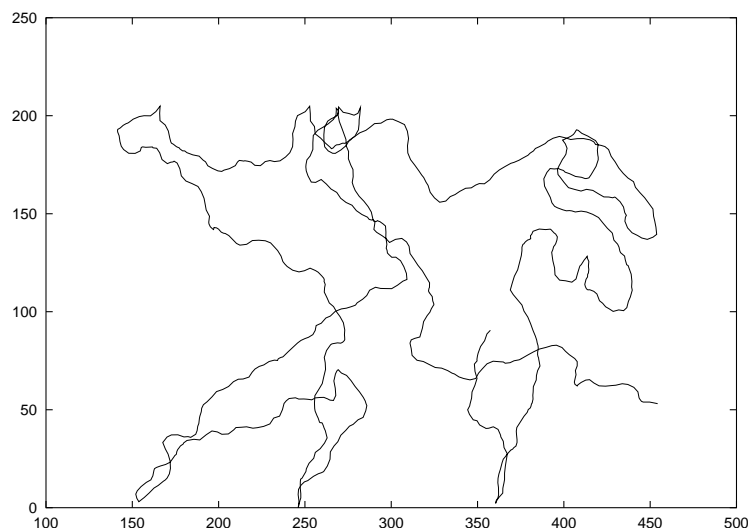
<sup>4</sup>Es gibt auch eine Variante des Random Walk Mobility Model, die die Simulationsfläche als Torus auffaßt und Knoten, die auf die Grenze der Simulationsfläche treffen, an der gegenüberliegenden Seite wieder eintreten läßt.

negativ beeinflusst werden. Darauf aufbauend wird in [95] nachgewiesen, daß dieses Problem bei allen „Random Mobility“-Modellen besteht, die Zielort und Geschwindigkeit unabhängig voneinander wählen und es wird eine allgemeine Konstruktion eingeführt, um das Problem bei dieser Art von Mobilitätsmodellen zu beheben.

## Gauss-Markov Mobility Model

---

Abrupte Richtungsänderungen und Pausen werden im *Gauss-Markov Mobility Model* [58] vermieden, indem die Richtung und Geschwindigkeit aller Knoten in festen Zeitabständen geändert werden (siehe Abbildung 2.4). Dabei basieren die neuen Werte auf den vorherigen Werten, variiert um die Werte zweier normalverteilter Zufallsvariablen. Dies führt zu graduellen, natürlicheren Bewegungen. Um sicherzustellen, daß Knoten nicht für lange Zeit in einer Ecke der Simulationsfläche bleiben, werden sie automatisch von der Ecke wegbewegt, wenn sie sich ihr nähern. Dazu wird die Bewegungsrichtung zur Mitte der Simulationsfläche verändert.



**Abbildung 2.4:** Bewegungsmuster eines mobilen Knotens, der sich nach dem Gauss-Markov Mobility Model bewegt.

## Smooth Random Mobility Model

---

Einen ähnlichen Ansatz verfolgt ebenfalls das *Smooth Random Mobility Model* [6]. Auch bei diesem Modell basieren neu gewählte Richtung und Geschwindigkeit auf

den vorherigen Werten. Zusätzlich wird mit Zielgeschwindigkeiten gearbeitet, d.h. ein mobiler Knoten hat für jede Teilstrecke eine Zielgeschwindigkeit. Vom Beginn der Teilstrecke an versucht er, diese Zielgeschwindigkeit zu erreichen, wobei er langsam beschleunigt. Über eine geeignete Wahl der Zielgeschwindigkeiten lassen sich die Pausen- und Bewegungsphasen steuern. Richtungsänderungen geschehen nicht mit der erreichten Geschwindigkeit, vielmehr bremsen die Knoten vor einer Richtungsänderung ab und beschleunigen danach wieder. Dies wird zur Simulation von Fahrzeugen eingesetzt.

## Graph-Based Mobility Model

---

Das *Graph-Based Mobility Model* [87] basiert auf der Erkenntnis, daß das zufällige Auswählen und direkte Ansteuern von Zielpunkten in den üblichen Mobilitätsmodellen nicht den Verhältnissen in der Realität entspricht, da in der realen Welt Hindernisse wie Häuser existieren, die nicht auf direktem Weg überwunden werden können und sich die Menschen auf Wegen und Straßen bewegen. Daher wird in diesem Modell ein Graph zugrundegelegt, dessen Knoten zulässige Ziele und dessen Kanten zulässige Wege zwischen diesen Zielen definieren. Die mobilen Knoten starten auf einem zufällig ausgewählten Knoten des Graphen und wählen einen Zielknoten zufällig aus. Sie bewegen sich dann nur entlang der Kanten zum Zielknoten, verharren dort für eine zufällig gewählte Zeit, bevor sie den nächsten Zielknoten auswählen. Die Geschwindigkeit für jede Strecke wird zufällig aus einem Geschwindigkeitsintervall gewählt.

Als Variante ist auch eine Steuerung der Ziele und Zeiten über einen Stundenplan denkbar. Jedes Gerät steuert so nacheinander die durch die im Stundenplan angegebenen Ziele an und bleibt dort für die angegebene Zeit. Auf diese Weise kann z.B. das Verhalten von Studenten auf einem Universitätscampus, die im Laufe eines Tages verschiedene Vorlesungen besuchen, simuliert werden.

## Manhattan Grid Mobility Model

---

Die Bewegungen von Knoten in einer Großstadt simuliert das *Manhattan Grid Mobility Model* [23]. Die Knoten bewegen sich auf schachbrettartig angelegten Straßen, die durch quadratische Häuserblöcke getrennt sind. Die mobilen Knoten können sich nur auf den Straßen bewegen, die Häuserblöcke sind unüberwindbar. An jeder



Kreuzung biegen die Knoten entweder mit einer bestimmten Wahrscheinlichkeit nach links bzw. rechts ab oder sie setzen ihren Weg über die Kreuzung hinweg fort. Zusätzlich können die Knoten in festen Zeitabständen in Abhängigkeit von einer Zufallsvariablen ihre Geschwindigkeit ändern.

## Obstacle Mobility Model

---

Das *Obstacle Mobility Model* [48] verfolgt das Ziel, die beliebigen Bewegungen anderer Bewegungsmodelle durch realitätsnähere Bewegungsmuster zu ersetzen. Dazu können in diesem Modell beliebige Polygone als Hindernisse (z.B. Gebäude) auf der Simulationsfläche plziert werden. Zwischen diesen Gebäuden werden Pfade berechnet, auf denen sich die Geräte bewegen. Die Geräte verteilen sich zu Beginn der Simulation zufällig auf die Pfade und wählen jeweils ein Zielgebäude aus, zu dem sie sich dann auf dem kürzesten Pfad bewegen. Am Ziel angekommen, pausiert das Gerät für einen zufälligen Zeitraum, bevor es ein neues Zielgebäude wählt.

Die Pfade zwischen den Gebäuden werden mit Hilfe des Voronoi-Diagramms der Gebäudeecken berechnet. Auf diese Weise wird die intuitive Idee, daß Wege zwischen Gebäuden ungefähr in der Mitte zwischen zwei Gebäuden verlaufen, recht gut getroffen. Zusätzlich berücksichtigt das Obstacle Mobility Model auch noch die Blockierung von Funkstrahlen durch Gebäude.

## Reference Point Group Mobility Model

---

Im Gegensatz zu den bisher beschriebenen Mobilitätsmodellen sind die Knotenbewegungen des *Reference Point Group Mobility Model* [43] voneinander abhängig. Dieses Modell definiert Bewegungen von Knotengruppen, die sich entlang vordefinierter Mobilitätspfade bewegen. Diese Pfade bestehen aus einer Reihe von Checkpoints, die zu bestimmten Zeiten erreicht werden müssen. Auf diese Weise läßt sich ein breites Spektrum von Szenarien modellieren. Jede Knotengruppe hat ein logisches Zentrum, das sich auf dem Mobilitätspfad der Gruppe bewegt. Die einzelnen Knoten haben jeweils einen Referenzpunkt, der sich auf dem Mobilitätspfad der Gruppe bewegt, und werden in jedem Schritt zufällig in der Nähe des Referenzpunktes plziert. Auf diese Weise kann neben der pfadbasierten Bewegung der Gruppe ein zufälliges Verhalten der Gruppenmitglieder erreicht werden.

Neben dieser Auswahl häufig benutzter Mobilitätsmodelle existieren natürlich eine Vielzahl weiterer Mobilitätsmodelle für spezielle Zwecke. Insgesamt geht die Suche nach möglichst realitätsnahen Bewegungsmodellen weiter. Vielleicht führt eine Kombination der erfolgreichen Elemente aus verschiedenen Mobilitätsmodellen letztlich zum Erfolg.

## 2.6 Emulation von Ad-Hoc-Netzwerken

---

Zur Entwicklung von Protokollen und Applikationen für mobile Ad-Hoc-Netzwerke reicht die Implementierung in einer Simulationsumgebung oft nicht aus. Nicht zuletzt die Interaktion mit „echten“ Benutzern läßt sich in der Simulation nicht gut genug simulieren. Gleichzeitig ist aber die Erprobung einer Applikation bzw. eines Protokolls in einem Feldversuch sehr aufwendig, insbesondere wenn zahlreiche Kombinationen von Routingverfahren und Tuningparametern erprobt werden müssen und die Geräteanzahl hinreichend groß sein soll.

Eine mögliche Lösung besteht in der Emulation des Ad-Hoc-Netzwerkes. Die Applikationen können dann im Idealfall direkt für die Zielplattform entwickelt werden und es wird nur das Netzwerk emuliert. Auf diese Weise ist der Test der Applikation unter realen Bedingungen – sprich realistisches Benutzerverhalten – möglich, ohne daß der organisatorische Aufwand eines Feldversuchs betrieben werden müßte.

Die verschiedenen Emulationsansätze unterscheiden sich darin, auf welcher Ebene – Radio-Layer, Netzwerk, Gerät bzw. Applikation – die Emulation ansetzt. *JEmu* [27] emuliert nur den Radio-Layer des Protokollstacks, während die anderen Ebenen unverändert genutzt werden können und der Aufwand einer Implementierung in der Emulation entfällt. Der Emulator für den Radio-Layer leitet die zu versendenden Pakete an einen zentralen Emulationsserver weiter, der die Verbreitung entsprechend der aktuellen Erreichbarkeit vornimmt. Die Emulation des Radio-Layers muß in Echtzeit erfolgen.

Wird hingegen das gesamte Netzwerk emuliert, so stehen zwei Möglichkeiten zur Verfügung. Zum einen kann ein Simulator eingesetzt werden, der entsprechend das gesamte Netzwerk simuliert. Diesen Ansatz verfolgt z.B. die für ns-2 entwickelte Emulationsschicht [62]. Bei diesem Ansatz muß der Simulationskern in der Lage sein, das Netzwerk in Echtzeit zu simulieren, was bei größeren Gerätezahlen problematisch werden kann. Zum anderen kann auch ein stationäres Netzwerk genutzt

werden, wobei die Sichtbarkeit und Bandbreite der einzelnen Knoten mittels spezieller Filter wie IPTABLES [86] oder Kernelmodulen dynamisch verändert wird. So entfällt der Aufwand zur Implementierung des gesamten Protokollstacks. Dieser Ansatz wird häufig verfolgt, als Beispiele können der *NRL Mobile Network Emulator* [16] oder der *NETShaper* [41] dienen.

Die hybride Emulation der Workbench (siehe Kapitel 4) simuliert sowohl das Netzwerk als auch die Ausführung der Applikation auf den mobilen Geräten, wobei die Interaktion mit der Applikation über eine herkömmlich implementierte graphische Benutzeroberfläche erfolgt.

Durch die Emulation entstehen zwangsläufig Verzögerungen, die in realen Implementierungen nicht existieren. Diese Verzögerungen können – insbesondere bei der Emulation einzelner Schichten des Protokollstacks – zu Problemen führen, wenn sich beispielsweise die Timingeigenschaften des emulierten Protokolls durch die Verzögerungen verändern.



# 3

## Workbench: Simulation

---

Die Workbench dient dazu, die simulative Entwicklung und Evaluation von Anwendungen für mobile multihop Ad-Hoc-Netzwerke gegenüber der Verwendung traditioneller Simulatoren zu erleichtern und zu beschleunigen. Der Entwickler soll sich ganz auf seine Anwendung konzentrieren können und nicht durch technische Details oder gar Unzulänglichkeiten eines Simulators aufgehalten werden.

Der Simulator sollte dem Entwickler größtmöglichen Komfort hinsichtlich der Entwicklung von Anwendungen und einen hohen Abstraktionsgrad bieten, dabei aber gut mit steigenden Gerätezahlen skalieren. Die detaillierte Visualisierung von Simulationen mit kleineren Gerätezahlen sollte in Echtzeit oder schneller möglich sein, um so eine möglichst einfache Analyse des Anwendungsverhaltens zu ermöglichen. Zusätzlich sollten im Simulator entwickelte Anwendungen unverändert bzw. mit geringen Änderungen auf einer entsprechenden Ausführungsplattform auf mobilen Geräten ausführbar sein.

Diese Ziele stehen teilweise konträr zueinander, da sich z.B. ein hoher Abstraktions- und Komfortgrad eher negativ auf die Skalierbarkeit auswirkt. Gleiches gilt für die Visualisierung. Daher beschränkt sich der Simulationsteil der Workbench bewußt auf die Simulation der topologischen Eigenschaften des Ad-Hoc-Netzwerks und verzichtet auf eine möglichst exakte Simulation der Funktechnologie und der unteren

Protokollschichten. Die durch diesen Verzicht freiwerdenden Berechnungsressourcen werden entsprechend zur Erreichung der oben genannten Ziele eingesetzt. Diese Einschränkung des Detaillierungsgrads der Simulation kann natürlich zu Ungenauigkeiten und abweichenden Simulationsergebnissen im Vergleich zu traditionellen Simulatoren führen; Untersuchungen wie [15] zeigen allerdings, daß auch traditionelle Simulatoren mit hohem Aufwand nur zu stark abweichenden Ergebnissen kommen. Hinzu kommt, daß für das prinzipielle Funktionieren eines Algorithmus oder einer Anwendung in multihop Ad-Hoc-Netzwerken nicht unbedingt Protokolldetails der verwendeten Netzwerktechnologie entscheidend sind, sondern diese meist einzig und allein von den topologischen Eigenschaften des Ad-Hoc-Netzwerks abhängen.

In den folgenden Abschnitten werden die einzelnen Komponenten der Simulationsumgebung genauer vorgestellt und hinsichtlich der oben formulierten Ziele untersucht.

## 3.1 Grundlagen

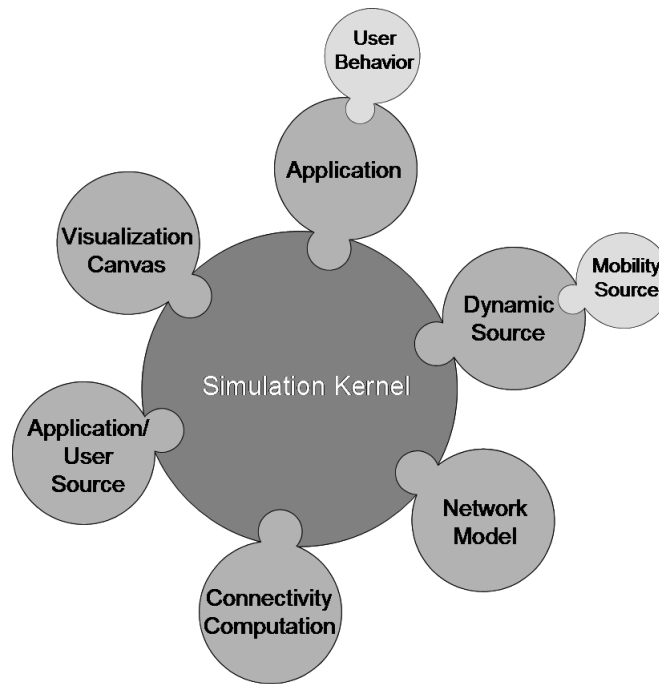
---

Der Simulationsteil der Workbench besteht aus einer ereignisgesteuerten Simulationsumgebung, die in Java [2] implementiert ist. Der eigentliche Simulationskern, bestehend aus dem *Future Event Set (FES)*<sup>1</sup> mit entsprechenden Ereignisklassen, der Verwaltung der Geräte, des Netzwerk- und Bewegungsmodells, wird vollständig vor dem Entwickler, der Applikationen in der Simulationsumgebung entwickelt, verborgen. Stattdessen greift der Anwendungsentwickler nur auf entsprechende Abstraktionen zu. Abbildung 3.1 zeigt die einzelnen Komponenten der Simulationsumgebung.

Die einzelnen Komponenten der Simulationsumgebung wie z.B. Netzwerkmodelle, Bewegungsmodelle, Visualisierungskomponenten etc. sind alle austauschbar. Der Simulationskern verwaltet nur das Zusammenspiel der Komponenten, während die konkrete Implementierung der einzelnen Komponenten durch die Definition von Interfaces veränderbar gehalten wird.

---

<sup>1</sup>Das FES ist die zentrale Datenstruktur einer ereignisgesteuerten Simulationsumgebung. Es ist eine prioritätsbasierte Warteschlange, in der die zukünftigen Ereignisse, nach der Zeit ihres Auftretens sortiert, eingetragen werden. Die zentrale Schleife der Simulationsumgebung entnimmt immer das nächste Ereignis aus dem FES und führt es aus. Der Simulationslauf endet, wenn eine Abbruchbedingung erfüllt ist oder das FES leer ist.



**Abbildung 3.1:** Architektur der Simulationsumgebung: ein durch austauschbare Komponenten erweiterbarer Simulationskern.

Auf diese Weise kann ein Entwickler zum einen die mitgelieferten Implementierungen dieser Komponenten nutzen und so schnell zu Resultaten kommen. Zum anderen kann er die Simulationsumgebung aber auch sehr stark an seine Bedürfnisse anpassen und z.B. ein ganz bestimmtes Bewegungsmodell implementieren. Die Austauschbarkeit von Komponenten beschränkt sich dabei nicht nur auf die „oberen“ Schichten, d.h. die verschiedenen Modelle. Selbst Teile des Simulationskerns wie z.B. die Implementierung des FES können vom Entwickler durch andere Implementierungen ersetzt werden.

Mittels einer vom Entwickler definierten Initialisierungskomponente wird die konkrete Komponentenkonfiguration für einen Simulationslauf festgelegt und der Simulationskern entsprechend konfiguriert. Auf diese Weise ist ein schneller Wechsel verschiedener Komponentenimplementierungen möglich.

Neben den nachfolgend detailliert besprochenen Komponenten stellt die Simulationsumgebung weitere Basisfunktionalitäten wie z.B. Zufallszahlengenerierung zur Verfügung. Über eine spezialisierte Form von Anwendungen ist es möglich, bestimmte Geräte zu Netzwerkverkehrsquellen mit durch den Entwickler definierten Eigenschaften zu machen. Dies erleichtert beispielsweise den Test verschiedener

Routingprotokolle im Rahmen der Simulationsumgebung.

Zur besseren Abschätzung der Güte von Anwendungen bzw. Algorithmen stellt die Simulationsumgebung eine globale Sicht auf die Netzwerktopologie mit Gerätekonnektivität, Senderadien und Verbindungszuverlässigkeiten zur Verfügung. Diese Sicht kann beispielsweise genutzt werden, um Routingverfahren mit optimalen, auf globalem Wissen basierenden Verfahren vergleichen zu können. Diese globale Sicht darf niemals direkt in Applikationen genutzt werden, da sie in einem realen Ad-Hoc-Netzwerk natürlich nicht zur Verfügung steht. Aus diesem Grund ist sie auch im Programmiermodell strikt von den anderen Komponenten getrennt.

## 3.2 Dynamische Aspekte

---

Die Simulationsumgebung muß verschiedene dynamische Aspekte der mobilen Geräte modellieren. Zum einen muß die Mobilität der Geräte simuliert werden, zum anderen die Eigenschaften der Funktechnologie: Geräte können uni- und bidirektionale Verbindungen auf- und abbauen, ihren Senderadius verändern und die Zuverlässigkeit einer bestehenden Verbindung zwischen zwei Geräten kann sich im Laufe der Zeit ändern.

Alle diese dynamischen Aspekte werden für den Simulationskern in Form einer *Dynamic Source* verfügbar gemacht. Eine Dynamic Source liefert auf Anfrage des Simulationskerns die nächste Aktion eines mobilen Gerätes, d.h. eine Bewegungsaktion oder eine die Funktechnologie betreffende Aktion.

Die Implementierung einer Dynamic Source ist nicht vorab festgelegt, so daß die Aktionen aus verschiedenen Quellen stammen können: sie können bei Bedarf berechnet werden, vorab berechnet sein und aus einer Datei ausgelesen werden oder aber interaktiv durch Benutzereingaben entstehen.

Die Möglichkeit, dynamische Aspekte im Voraus zu berechnen, ist insbesondere für die Berechnung der Verbindungen zwischen den Geräten wichtig. Auf diese Weise ist der hohe Berechnungsaufwand pro Szenario und Gerätezahl nur einmal notwendig und die gespeicherten Aktionen können für nachfolgende Simulationsläufe genutzt werden. Das folgende Beispiel zeigt die Konnektivitäts- und Bewegungsdaten einer Simulation von vier mobilen Geräten, die in einer XML-Datei abgelegt sind. Diese XML-Datei kann dann als Eingabe für eine Dynamic Source dienen. Sie



enthält Informationen über den Eintritt der Geräte in die Simulation (<ENTER>), Bewegungen der Geräte zu bestimmten Punkten (<ARRIVAL>), den potentiellen Verbindungsaufbau (<ATTACH>) sowie den Verbindungsabbau (<DETACH>) zwischen Geräten und den Austritt der Geräte aus der Simulation (<EXIT>).

---

```

1<?xml version="1.0" encoding="UTF-8"?>
2<!DOCTYPE COMMANDLIST SYSTEM "CommandList.dtd">
3<COMMANDLIST BOTTOMLEFTX="0.0" BOTTOMLEFTY="0.0"
4      TOPRIGHTX="1000.0" TOPRIGHTY="1000.0">
5  <ENTER TIME="8.661265310954672" ADDRESS="3" RADIUS="27.066872142711762"
6      ENDTIME="162.57886832826165" STARTX="502.7192115287612"
7      STARTY="295.7937129187398" ENDX="640.4228680063618"
8      ENDY="294.38995422828054" />
9  <ENTER TIME="23.772821192895663" ADDRESS="2" RADIUS="82.74393928109343"
10     ENDTIME="174.58870624583935" STARTX="599.5234313442194"
11     STARTY="801.4108515673732" ENDX="718.1121053408186"
12     ENDY="766.8365543841654" />
13 <ENTER TIME="38.63905822788856" ADDRESS="0" RADIUS="75.3892696145113"
14     ENDTIME="77.79433832274282" STARTX="678.8996294035338"
15     STARTY="738.954793127653" ENDX="683.2711226353268"
16     ENDY="776.9625213365392" />
17 <ATTACH TIME="44.125834979997705" SENDER="2" RECEIVER="0" />
18 <ATTACH TIME="49.903475330001164" SENDER="0" RECEIVER="2" />
19 <ENTER TIME="52.08871227727858" ADDRESS="1" RADIUS="13.285073386497604"
20     ENDTIME="181.27233257039018" STARTX="1.2567837451626929"
21     STARTY="647.467729213403" ENDX="114.81190929191214"
22     ENDY="675.5675191587327" />
23 <ARRIVAL TIME="77.79433832274282" ADDRESS="0" ENDTIME="133.36884546163301"
24     STARTX="683.2711226353268" STARTY="776.9625213365392"
25     ENDX="683.2711226353268" ENDY="776.9625213365392" />
26 <ARRIVAL TIME="133.36884546163301" ADDRESS="0" ENDTIME="171.88018065121116"
27     STARTX="683.2711226353268" STARTY="776.9625213365392"
28     ENDX="646.2695408963268" ENDY="773.3181017203669" />
29 <EXIT TIME="162.57886832826165" ADDRESS="3" />
30 <DETACH TIME="171.88018065121116" SENDER="0" RECEIVER="2" />
31 <DETACH TIME="171.88018065121116" SENDER="2" RECEIVER="0" />
32 <EXIT TIME="171.88018065121116" ADDRESS="0" />
33 <EXIT TIME="174.58870624583935" ADDRESS="2" />
34 <EXIT TIME="181.27233257039018" ADDRESS="1" />
35</COMMANDLIST>

```

---

Die Aktionen sind unabhängig von der zu simulierenden Anwendung und dem verwendeten Benutzerverhalten, so daß sich ein hoher Wiederverwendungsgrad ergibt. Insbesondere bei hohen Gerätezahlen und -dichten führt die Vorausberechnung zu einer erheblichen Verringerung der Simulationszeit (siehe auch Abschnitt 3.9).

In den folgenden Abschnitten werden die Berechnung der Mobilitätsmodelle sowie der Konnektivität, aus denen sich die Aktionen der Dynamic Source ergeben, erläutert.

## Mobilität

---

Gerätemobilität in der Simulationsumgebung basiert auf einer einfachen Abstraktion für Bewegungsmodelle. Geräte treten zu einem bestimmten Zeitpunkt in die Simulationsszenerie ein und bewegen sich danach stets für eine bestimmte Zeit mit gleichbleibender Geschwindigkeit auf einer geraden Linie. Diese Geradenstücke können beliebig kurz sein, so daß bei identischem Start- und Endpunkt auch Pausen modelliert werden können. Liegen für ein Gerät keine weiteren Bewegungsdaten vor, so verläßt es die Simulation. Dabei werden die Bewegungsdaten über zweidimensionale kartesische Koordinaten definiert, die Geräte bewegen sich also auf einer rechteckigen Fläche.

Diese Bewegungsmodellabstraktion erlaubt, beliebige Bewegungsmodelle abzubilden oder zumindest zu approximieren. Gleichzeitig erlaubt sie – wie später gezeigt wird – eine erhebliche Optimierung bei der Berechnung der Gerätekonnektivität.

In der Simulationsumgebung werden Bewegungsdaten über *Mobility Sources* bereitgestellt. Diese *Mobility Sources* werden in die entsprechende *Dynamic Source* integriert und liefern so dem Simulationskern die notwendigen Bewegungsaktionen. Die konkrete Implementierung einer *Mobility Source* ist nicht festgelegt. Bewegungsdaten können direkt bei Bedarf berechnet oder aber aus einer Datei eingelesen werden. Diese Flexibilität ermöglicht z.B. eine einfache Anbindung von externen Bewegungsmodellgeneratoren wie BonnMotion [89].

Der Benutzer der Simulationsumgebung kann beliebige eigene *Mobility Sources* definieren und in seine Simulationen einbinden. Der Simulator stellt einige gängige Mobilitätsmodelle bereit, so z.B. Variationen des Random Waypoint Mobility Model (siehe Abschnitt 2.5) oder ein Mobilitätsmodell, in dem das Verhalten der einzelnen Knoten „gekriptet“ werden kann. Über eine Containerimplementierung können Mengen von verschiedenen Bewegungsmodellimplementierungen zu einer *Mobility Source* zusammengefaßt werden.

## Konnektivitätsberechnung

---

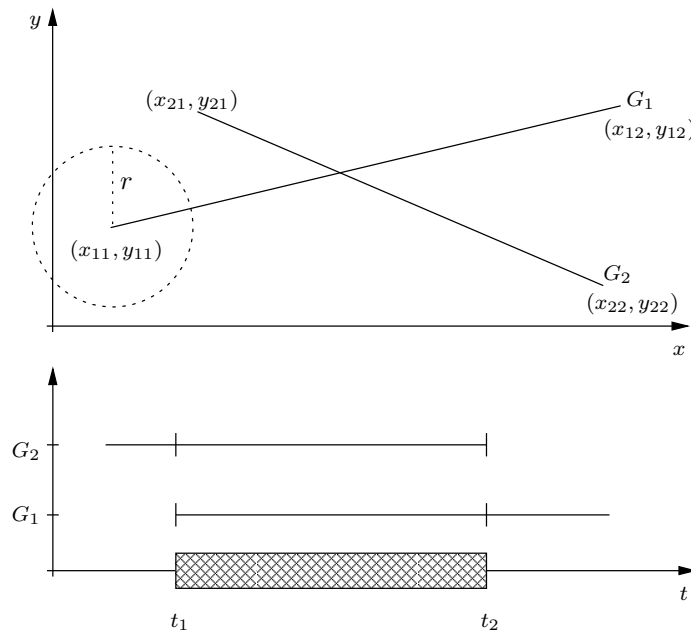
Die Berechnung der Konnektivität zwischen den Geräten ist eine der aufwendigsten Aufgaben in der Simulation von mobilen Ad-Hoc-Netzen, da prinzipiell alle Geräte paarweise miteinander verglichen werden müssen, so daß sich ein quadratischer Aufwand ergibt. Aufgrund der Mobilität der Knoten muß die Konnektivität

immer wieder neu berechnet werden. Es existieren zwei Ansätze zur Konnektivitätsberechnung. Entweder berechnet man die Konnektivität erst bei Bedarf, d.h., wenn ein Gerät ein Paket verschicken möchte, wird ermittelt, welche Geräte dieses Paket empfangen können. Oder aber die Konnektivität wird proaktiv berechnet und steht somit bereits zur Verfügung, wenn ein Gerät senden möchte. Die Berechnung bei Bedarf ist ohne spezielle Verbesserungen nur dann effizienter als die proaktive Berechnung, wenn sich die Verbindungen zwischen den Geräten mit hoher Frequenz ändern und gleichzeitig sehr wenig Kommunikation zwischen den Geräten stattfindet. Der in [90] für ns-2 angegebene Speed-up von 20 durch Aufteilung der Simulationsfläche mittels eines Gitters, um die Zahl der potentiell in Funkreichweite liegenden Geräte und damit die Zahl der notwendigen Vergleiche zu minimieren, zeigt, daß Optimierungen der Konnektivitätsberechnung erhebliche Auswirkungen auf die Simulationsgeschwindigkeit und Skalierbarkeit haben.

Für den Simulationsteil der Workbench kommt aus zwei Gründen nur die proaktive Berechnung in Frage: zum einen soll – um das Ziel des hohen Abstraktionsniveaus und Entwicklungskomforts zu erreichen – die Anwendung jederzeit über hinzukommende oder wegfallende benachbarte Geräte informiert werden. Zum anderen ist ab einer gewissen Zugriffshäufigkeit der mobilen Geräte auf das drahtlose Netzwerk die erforderliche Skalierbarkeit und damit Simulationsgeschwindigkeit ohne proaktive Berechnung der Konnektivität nicht realisierbar.

Der *Link Calculator*, die Komponente, die die Konnektivität der mobilen Geräte berechnet, nutzt dabei die Eigenschaften des Mobilitätsmodells der Simulationsumgebung aus, um eine möglichst effiziente Konnektivitätsberechnung zu ermöglichen. Da es sich bei den Bewegungen stets um Geradenabschnitte handelt, die mit konstanter Geschwindigkeit passiert werden, kann für ein Gerät, das gerade einen neuen Bewegungsabschnitt beginnt, sofort die Menge aller Konnektivitätsaktionen berechnet werden.

Dazu wird für alle zu diesem Zeitpunkt bekannten Geradenabschnitte aller anderen Geräte berechnet, ob und wann der Abstand zwischen den beiden Geräten eine Verbindung zuläßt bzw. wann diese wieder abreißt. Abbildung 3.2 zeigt die Geradenabschnitte für zwei Geräte  $G_1$  und  $G_2$  sowie den Senderadius  $r$  von Gerät  $G_1$ . Zu beachten ist, daß die Geräte ihre jeweiligen Geradenabschnitte nicht zeitgleich und nicht mit gleicher Geschwindigkeit zurücklegen müssen. Das Timingdiagramm in Abbildung 3.2 zeigt die zeitliche Abfolge, in der die beiden Geräte die Geraden-



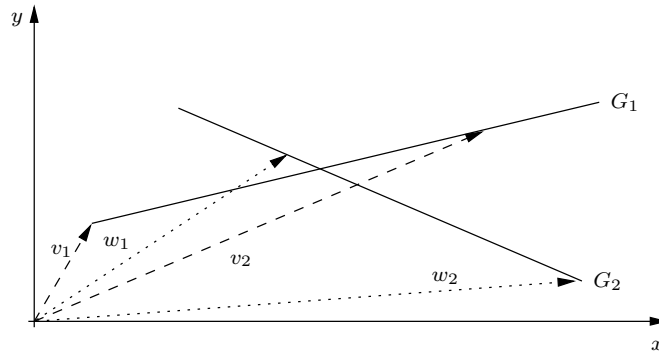
**Abbildung 3.2:** Darstellung der Geradenabschnitte für die Bewegungen zweier Geräte  $G_1$  und  $G_2$  samt zugehörigem Timingdiagramm.

abschnitte durchlaufen. Zur Berechnung der Konnektivität interessiert nur die Zeitspanne, in der sich beide Geräte bewegen. In Abbildung 3.2 ist dies die Zeitspanne zwischen  $t_1$  und  $t_2$ . Außerhalb dieses Zeitraums liegende Teile der Geradenabschnitte können ignoriert werden, da sie bei späteren Bewegungen der Geräte automatisch berücksichtigt werden. Die Konnektivätsereignisse werden dabei stets in beiden Richtungen berechnet, d.h. sowohl aus Sicht von Gerät  $G_1$  als auch aus Sicht von  $G_2$ .

Die Abstandsberechnung zwischen den Geradenabschnitten erfolgt anhand der zugehörigen Geradengleichungen. Aufgrund der linearen Bewegung der Geräte kann für jeden Zeitpunkt  $t$  berechnet werden, wo sich das Gerät gerade befindet. Berechnet man nun für die Zeitpunkte  $t_1$  und  $t_2$  jeweils, wo sich die Geräte  $G_1$  und  $G_2$  befinden, so erhält man pro Gerät zwei Vektoren  $v_1$  und  $v_2$  bzw.  $w_1$  und  $w_2$ , mit deren Hilfe man die zu den Geradenabschnitten passenden Geraden  $f$  und  $g$  beschreiben kann.

$$f(t) = \frac{t - t_1}{t_2 - t_1}(v_2 - v_1) + v_1$$

$$g(t) = \frac{t - t_1}{t_2 - t_1}(w_2 - w_1) + w_1$$



**Abbildung 3.3:** Ableitung der Geradengleichungen über die Positionen der Geräte  $G_1$  und  $G_2$  zu den Zeitpunkten  $t_1$  und  $t_2$  sowie die zugehörigen Vektoren  $v_1, v_2, w_1$  und  $w_2$ .

Setzt man den Abstand der beiden Geraden mit dem Senderadius  $r$  gleich, so erhält man eine quadratische Gleichung.

$$|f(t) - g(t)| = r$$

Die Lösungen der quadratischen Gleichung sind die Zeitpunkte, zu denen der Abstand zwischen den beiden Geraden gleich dem Funkradius  $r$  ist und damit die Konnektivität zwischen den Geräten beginnt bzw. endet. Zu beachten ist, daß diese Zeitpunkte vor  $t_1$  oder nach  $t_2$  liegen können, da nicht nur die Geradenabschnitte, sondern die gesamten Geraden betrachtet werden. In einem solchen Fall werden keine Konnektivitätsaktionen erzeugt.

Kombiniert mit einer Spezialbehandlung des Ein- und Austritts von Geräten läßt sich auf diese Weise die Konnektivität effizient berechnen.

Die Reduzierung der notwendigen Vergleiche mit anderen Geräten ist bei dieser Art der Konnektivitätsberechnung natürlich ebenfalls möglich, indem man die Simulationsfläche mittels eines Gitters unterteilt und nur die Geräte aus Gitterfeldern untersucht, die potentiell in Sendereichweite liegen. Allerdings dürfte der Geschwindigkeitsgewinn bei weitem nicht so groß ausfallen wie im Fall von ns-2, da die proaktive Berechnung über die Geradengleichungen schon erheblich effizienter als die reaktive Berechnung von ns-2 ist.

## 3.3 Netzwerkmodell

---

Das Netzwerkmodell der Simulationsumgebung stellt zwei Kommunikationsprimitive bereit: lokalen Unicast, über den mit einem benachbarten Gerät kommuniziert werden kann, und lokalen Broadcast, mit dessen Hilfe alle benachbarten Geräte erreicht werden können. Konnektivitätsaktionen aus der Dynamic Source werden dem Netzwerkmodell mitgeteilt; es stellt sie zusammengefaßt den übrigen Simulationskomponenten zur Verfügung.

Die konkrete Netzwerkimplementierung ist nicht festgelegt, so daß verschiedene Netzwerkmodelle je nach Bedarf eingebunden werden können. Die Informationen über den Verbindungsaufbau zwischen zwei Geräten werden dabei als potentielle Verbindungen behandelt, d.h. die konkrete Netzwerkimplementierung kann entscheiden, ob die entsprechende Verbindung wirklich zustandekommt. Auf diese Weise ist es möglich, daß z.B. bei hohen Gerätedichten zwar potentiell eine Verbindung zwischen zwei Geräten zustande käme, die Netzwerkimplementierung allerdings das Verhalten eines realen Netzwerks nachbildet und die Verbindung aufgrund zu hoher Gerätedichte nicht erlaubt. Diese Art des Netzwerkmodells erlaubt, neben einfachen Implementierungen (keine Verluste, keine Kollisionen, beliebige Kapazität) auch detailliertere statistische Modelle zu implementieren.

Nachrichten, die über das Netzwerk verschickt werden, können überwacht werden. Die Benachrichtigung des Senders bei Auslieferung aus der lokalen Queue, d.h. wenn die Nachricht wirklich versendet wird, ist in allen Netzwerkimplementierungen möglich. Zusätzlich können Netzwerkimplementierungen auch noch exakte Auslieferungsinformationen für Unicast-Nachrichten bereitstellen. Der Sender der Nachricht wird informiert, ob die Auslieferung sicher erfolgreich, sicher erfolglos oder undefiniert (es kann nicht mit Sicherheit gesagt werden, ob die Nachricht erfolgreich ausgeliefert wurde oder nicht) war.

### Einfaches Netzwerkmodell

---

Die einfache Netzwerkmodellimplementierung der Simulationsumgebung stellt ein idealisiertes Netzwerkmodell zur Verfügung, wie es z.B. bei der theoretischen Betrachtung von Algorithmen für Ad-Hoc-Netzwerke oft zugrundegelegt wird. Unabhängig von der Gerätedichte bleibt die Datenrate stets gleich, so daß jedes Gerät

immer mit dieser Datenrate senden kann. Das Netzwerk hat also eine beliebig große Kapazität. Verbindungen zwischen den Geräten sind (innerhalb der Sendereichweite) optimal, so daß gesendete Datenpakete nicht verlorengehen. Da auch Kollisionen nicht existieren, können gesendete Datenpakete nur aufgrund der Gerätemobilität verlorengehen.

Dieses idealisierte Netzwerkmodell läßt sich einfach simulieren und ist für die erste Analyse von Algorithmen und Anwendungen oft ausreichend; in manchen Fällen ist es sogar präzise genug, um das Verhalten der Anwendung genau vorherzusagen [38].

## Realistischere Netzwerkmodelle

---

Zusätzlich zur einfachen Netzwerkmodellimplementierung existieren noch zwei weitere, realistischere Netzwerkmodellimplementierungen: **●LinkReliabilityNetwork**<sup>2</sup> und **●SharedNetwork**.

Das **●LinkReliabilityNetwork** verfolgt, im Gegensatz zu einfachen Netzwerkmodellimplementierungen, die Grundidee, realistische Verbindungsqualitäten zu simulieren. In der einfachen Implementierung ist die Verbindungsqualität stets maximal, während sie im **●LinkReliabilityNetwork** über beliebige Funktionen über die Verbindungszeit definiert werden kann. Eine entsprechend erweiterte Implementierung des **LinkCalculators** berechnet für jede neue Verbindung eine entsprechende Funktion für die Verbindungsqualität, die dann vom **●LinkReliabilityNetwork** verwendet wird.

Dabei werden Unicasts entsprechend der Verbindungsqualität verzögert, d.h. je schlechter die Verbindungsqualität ist, desto länger dauert die Auslieferung der Pakete, allerdings werden alle Pakete ausgeliefert, sofern sie nicht aufgrund von Gerätemobilität verlorengehen. Bei Broadcasts hingegen kommt es zu der Verbindungsqualität entsprechenden Paketverlusten.

Das **●SharedNetwork** implementiert einen MAC-Layer mit RTS/CTS für Unicast und genereller Kollisionserkennung. Dazu verwaltet diese Netzwerkmodellimplementierung für jedes Gerät eine Liste mit Sendeanforderungen. Ein Gerät, das in den Listen der beteiligten Geräte vorne steht, darf Pakete senden. Die Listen werden mit Hilfe der Verbindungsaufbau-/Verbindungsabbau-Ereignisse sowie beim Ende

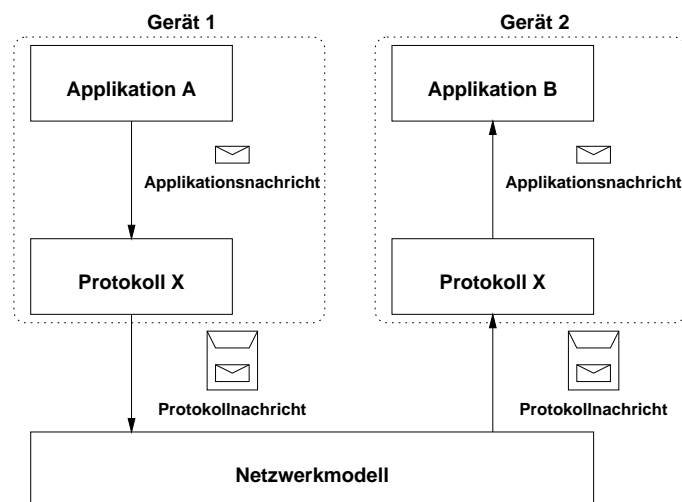
---

<sup>2</sup>Im folgenden werden Klassennamen durch ein **●** und Interfacenamen durch ein **Ⓜ** gekennzeichnet. Methodennamen werden in einer serifenlosen Schriftart dargestellt.

des Sendens aktualisiert. Im Fall eines Unicasts wird das sendende Gerät bei allen direkten Nachbarn und bei allen Nachbarn des empfangenden Geräts eingetragen. Für Broadcasts reicht die Eintragung bei allen direkten Nachbarn. Prinzipiell ist eine Berücksichtigung der Verbindungsqualitäten aus dem **LinkReliabilityNetwork**-Ansatz auch im **SharedNetwork** denkbar.

## Nachrichten und Protokolle

Applikationen nutzen niemals direkt die Netzwerkimplementierung der Simulation, sondern haben stattdessen Zugriff auf eine Netzwerkprotokollsammlung. Diese Netzwerkprotokolle können durch Nutzung der beiden Kommunikationsprimitive des Netzwerkmodells beliebig komplexe Verfahren implementieren (siehe Abbildung 3.4). Der Nutzer der Simulationsumgebung kann die Netzwerkprotokollsammlung durch eigene Protokollimplementierungen beliebig erweitern.



**Abbildung 3.4:** Zusammenspiel von Applikationen, Protokollen und Netzwerkmodellen.

Zur Beschleunigung der Applikations- und Protokollentwicklung sind Nachrichten in der Simulationsumgebung als Java-Objekte, die das **Message** Interface implementieren, realisiert. Diese Objekte „wissen“, welche Methode sie beim Empfänger aufrufen müssen und können so generisch durch das Netzwerkmodell behandelt werden. Auf diese Weise kann der Entwickler eigene Nachrichtentypen mit maßgeschneiderter Behandlung leicht implementieren. Die Datenstrukturen in den Nachrichten müssen nicht größenoptimiert werden, sondern können auf möglichst ein-



fache Nutzung ausgelegt werden, da eine explizite Größenangabe für die Gesamtnachricht zur Simulation verwendet wird.

Applikationsnachrichten werden beim Versand über ein Protokoll in eine entsprechende Protokollnachricht eingepackt. Auf der Empfängerseite wird dann die Protokollnachricht behandelt und löst bei Bedarf die Behandlung der eingebetteten Applikationsnachricht aus.

Aus Performancegründen können Nachrichten in der Simulation auch direkt als Referenzen weitergegeben werden, statt zuerst eine Kopie der Originalnachricht für jeden Empfänger zu erstellen. Der Applikations- bzw. Protokollentwickler kann dies für jeden Nachrichtentyp festlegen. Falls eine Nachricht unveränderbar ist, kann der Simulationsablauf durch den Verzicht auf die zahlreichen Kopien beschleunigt werden.

## 3.4 Hardwareabstraktion

---

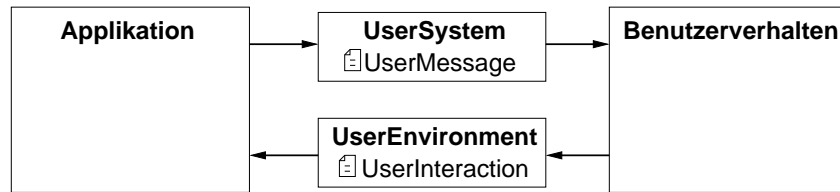
Die spätere Ausführung der im Simulationsteil der Workbench entwickelten Anwendungen auf einer realen Hardwareplattform ist nur möglich, wenn Implementierungsdetails der Simulationsumgebung wie das FES vor der Anwendung verborgen bleiben. Daher ermöglicht das *Operating System* in der Simulation Zugriff auf die für die Anwendungen notwendigen Informationen, ohne Interna der Implementierung der Simulationsumgebung zugreifbar zu machen. Über das Operating System kann die Anwendung Informationen über die benachbarten Geräte, die eigene Geräteadresse, den eigenen Senderadius, die eigene Bewegungsrichtung und die aktuelle Zeit erfragen. Zusätzlich kann sich die Applikation als *Connection Observer* registrieren, um direkt über neu hinzukommende oder wegfallende Verbindungen zu anderen Geräten informiert zu werden.

## 3.5 Benutzerverhalten und Applikationen

---

Die Zuordnung von Applikationen zu mobilen Geräten und die Definition des Benutzerverhaltens für eine Applikation wird über *Application User Sources* festgelegt. Der Grundgedanke ist, daß die Definition des Simulationsszenarios, d.h. die Geräte-mobilität, Größe der Simulationsfläche und Senderadien, völlig unabhängig von der

zu simulierenden Applikation und dem zugeordneten Benutzerverhalten sein sollte. In diesem Fall lassen sich die Vorteile der Vorausberechnung von Mobilitäts- und Konnektivitätsdaten besonders gut nutzen.



**Abbildung 3.5:** Interaktion zwischen Applikation und Benutzerverhalten bei gleichzeitiger Entkoppelung mittels UserSystem und UserInteraction.

Application User Sources liefern auf Anfrage für jedes Gerät in der Simulation ein Tupel aus auszuführender Applikation und zugehörigem Benutzerverhalten, d.h. sie leisten das Mapping  $\{\text{Applikationen}\} \times \{\text{Benutzerverhalten}\} \rightarrow \{\text{Geräte}\}$ . So kann der Entwickler genau definieren, welche Applikation und welches Benutzerverhalten auf bestimmten Geräten ausgeführt werden. Das simulierte Benutzerverhalten hat dabei Zugriff auf die Applikation und kann über *User Interactions* asynchron und unter Umständen zeitversetzt Methoden der Applikation aufrufen. Umgekehrt hat die Applikation die Möglichkeit, über das *User System* mittels *User Messages* asynchron Ereignisse im Benutzerverhalten auszulösen (siehe Abbildung 3.5).

## 3.6 Visualisierung

Die Möglichkeit der Visualisierung ist für eine Simulationsumgebung für mobile Ad-Hoc-Netzwerke besonders wichtig, da aufgrund der Mobilität und der noch relativ neuen Verfahren das Verhalten einer Anwendung weit weniger gut vorhersagbar ist als in traditionellen verteilten Systemen. Oft genügt ein Blick auf die Visualisierung, um den Grund für einen bestimmten Fehler oder ein nicht vorhergesehenes Verhalten zu finden. Dies erspart dem Entwickler zeitaufwendige Auswertungen von Logdateien.

Traditionelle Simulatoren schränken die Möglichkeit der Visualisierung oft ein, indem nur vordefinierte Symbole für die mobilen Geräte sowie die Darstellung der Verbindungen zwischen den einzelnen Geräten möglich sind. Zusätzlich kann die

Visualisierung oft nur *nach* dem Simulationslauf auf Basis einer entsprechenden Logdatei erfolgen.

Im Gegensatz dazu ermöglicht es die Simulationsumgebung der Workbench dem Entwickler, nahezu alle Komponenten der Simulation in beliebiger Art zu visualisieren. Dazu basiert die Visualisierung auf *Shapes*, die sich durch Zeichnen auf einer *Canvas* darstellen können. Die Canvas bietet dabei alle notwendigen Grundoperationen zum Zeichnen von Linien, Rechtecken, Polygonen, Ellipsen, Texten und Bildern. Die entsprechenden Shapes werden von der Simulationsumgebung zur Verfügung gestellt. Sie können beliebig auf der Canvas positioniert werden. Zusätzlich gibt es eine Sammlung von Shapes mit relativer Positionierung, über die mehrere Shapes zusammengefaßt werden können. Der Entwickler kann für die Visualisierung der einzelnen Komponenten entweder auf die vorgefertigten Shapes zurückgreifen und diese z.B. in einer Shapesammlung zusammenfassen oder aber eigene Shapes definieren.

Die Verwendung der Canvas ermöglicht es, die Visualisierung von der konkreten Ausgabetechnologie unabhängig zu machen. Denn die Canvas kann für die Ausgabe beliebig implementiert werden. Die Workbench liefert Implementierungen für Java Swing, für das *Standard Widget Toolkit (SWT)* [22] des Eclipse Projekts, für die OpenGL-Anbindung von SWT [77, 83] sowie für die PostScript- bzw. XML-Ausgabe mit. Aber auch eine Ausgabe in Form einer Videodatei wäre problemlos realisierbar.

Die Simulationsumgebung erlaubt, die Simulationsfläche, die einzelnen Geräte, das Netzwerk, Nachrichtenpakete, Protokolle und die Applikation zu visualisieren. Der Entwickler muß nur ein entsprechendes Shape (das auch eine Shapesammlung sein kann) definieren. Falls für eine Komponente keine Visualisierung gewünscht wird, so kann für diese Komponente ein „leeres“ Shape definiert werden. Die Darstellungsreihenfolge der Komponentenvisualisierung kann über *Shape Builder* beeinflusst werden. Zusätzlich können auch Ergebnisse der statistischen Analyse (siehe Abschnitt 3.7) dargestellt werden. Abbildung 3.6 zeigt die Visualisierung von UbiBay, einer Auktionsanwendung für mobile multihop Ad-Hoc-Netzwerke (siehe auch die Fallstudie zu UbiBay in Kapitel 6). Zu sehen ist die Visualisierung der Simulationsfläche mittels eines Stadtplans und der Geräte durch Abbildungen von PDAs. Die Anwendung selbst nutzt eine Agentenplattform und stellt ihren Zustand über verschiedenfarbige Agenten dar, die jeweils neben dem Gerät gezeichnet werden, auf dem sie sich gerade befinden. Zusätzlich werden noch bestimmte Bereiche auf der

Simulationsfläche wie der Marktplatz und die Homezones der Geräte durch Kreise dargestellt.



**Abbildung 3.6:** Visualisierung von Applikationen im Simulationsteil der Workbench am Beispiel von UbiBay.

Die Aktualisierung der Visualisierung wird über regelmäßiges Einfügen von Visualisierungsereignissen in das FES erreicht. Diese Visualisierungsereignisse sorgen dafür, daß die Shapes aller Komponenten berechnet werden und kombinieren diese in einem Frame, der in eine Warteschlange begrenzter Länge geschrieben wird. Dabei hat diese Warteschlange die Eigenschaft, daß sie nie ganz geleert werden kann, sondern der letzte Frame stets erhalten bleibt. Solange in der Warteschlange noch Platz ist, werden die berechneten Frames dort gespeichert. Wenn die Kapazität erschöpft ist, wird die Simulation solange blockiert, bis in der Warteschlange wieder Platz ist.

Die graphische Oberfläche entnimmt der Warteschlange mit fester Bildwiederhol­frequenz Frames und stellt sie dar. Dadurch wird die Warteschlange regelmäßig ge­leert und es können neue Frames in der Simulation berechnet werden. Aufgrund des garantierten Vorhandenseins mindestens eines Frames in der Warteschlange ist die Darstellung jederzeit möglich. Über die Steuerung der Entnahmerate und der Rate, mit der Visualisierungsereignisse in die FES eingefügt werden, kann die Geschwin­

digkeit der Simulation verlangsamt oder beschleunigt (soweit die Rechenleistung ausreicht) werden. Im Normalfall erfolgt die Visualisierung in Echtzeit während des Simulationslaufs. Über die Verwendung der Ausgabe in eine XML- oder Videodatei kann sie aber auch problemlos erst im Anschluß an einen Simulationslauf erfolgen.

Die gesamte Visualisierung ist mit hohem Aufwand verbunden, da mit hoher Frequenz die Shapes aller Komponenten berechnet und dargestellt werden müssen. Beschleunigen läßt sich dies zum einen durch Caching von Shapes, die sich nicht andauernd verändern, und durch effiziente Implementierung der Canvas. So läßt sich durch die Nutzung von OpenGL eine Beschleunigung um bis zu 25% erzielen. Falls für einen Simulationslauf keine Visualisierung gewünscht wird, läßt sich diese abschalten. Dann fällt dann auch kein zusätzlicher Berechnungsaufwand an. Insgesamt zeigt sich, daß auf aktuellen Rechnern die Visualisierung von wenigen hundert Geräten in Echtzeit problemlos möglich ist, was die Visualisierung bei der Entwicklung von neuen Anwendungen zu einem sehr hilfreichen und damit unverzichtbaren Werkzeug macht.

## 3.7 Statistik

---

Die Auswertung von Simulationsläufen wird durch ein Statistiksystem in der Simulationsumgebung erleichtert. Mit dessen Hilfe ist es möglich, sowohl global als auch lokal für jedes mobile Gerät Daten zu sammeln und auszuwerten. Die gesammelten Daten können dann nach Simulationsende in Textform ausgegeben und abgespeichert werden. Zusätzlich ist es dem Entwickler möglich, schon während des Simulationslaufs über das *Statistics Monitor System* auf Statistikdaten zuzugreifen und diese zu visualisieren. Dies ermöglicht oft eine einfachere und effizientere Auswertung als in traditionellen Simulatoren, wo die entsprechenden Informationen nach Ende des Simulationslaufs mit eigens dafür entwickelten Skripten aus der Logdatei extrahiert werden müssen.

Über das Statistiksystem können verschiedene Arten von Ereignissen beobachtet werden: *Event Observer* erlauben das inkrementelle Hochzählen, *Population Observer* erlauben es, die Größe von Populationen durch inkrementelles Hoch- und Herunterzählen zu protokollieren und *Time Observer* stellen eine „Stopuhr“ bereit. Es können beliebig viele dieser Observer über das Statistiksystem definiert werden;

sie werden eindeutig benannt und stehen dann lokal oder global zur Verfügung.

Ein Observer erhält bei Definition neben einem eindeutigen Namen auch eine Liste von *Statistical Calculators* und einen *Unit Converter*. Ein *Statistical Calculator* erlaubt, bestimmte Funktionen auf den gesammelten Werten zu berechnen. Diese Funktionen kann der Entwickler selbst definieren, die Simulationsumgebung liefert die gängigen zur Berechnung von Minimum, Maximum, Mittelwert, Varianz etc. mit. Wird über einen Observer ein neuer Wert hinzugefügt, so wird dieser an alle registrierten *Statistical Calculators* des Observers weitergegeben, die ihn entsprechend für ihre Berechnung verwenden können.

Die Auswertung der gesammelten Daten wird durch die *Unit Converter* erleichtert, die die gesammelten statistischen Werte eines Observers in Textform umwandeln. Auf diese Weise können z.B. gemessene Zeiten eines *Time Observers* einfach ausgegeben werden. Mittels eigener *Statistics Renderer* kann der Entwickler die Ausgabe der gesammelten Daten genau definieren.

## 3.8 Entwicklung einer Beispielapplikation

---

In diesem Abschnitt wird die Entwicklung einer einfachen Beispielapplikation im Simulationsteil der Workbench beschrieben. Die Evaluation dieser Beispielanwendung in der Emulation und die Ausführung auf mobilen Kleingeräten werden in Kapitel 4 und 5 erläutert. Abbildung 3.7 zeigt zum besseren Verständnis der Beispielanwendungsimplementierung eine vereinfachte UML-Darstellung der Architektur der Simulationsumgebung.

Die Beispielanwendung *Ping-Anwendung* ist bewußt einfach gehalten, erlaubt aber die Erläuterung der wesentlichen Teile der Workbench. Sie zeigt dem Nutzer alle Nachbargeräte in Funkreichweite an und ermöglicht es ihm, diese „anzupingen“. In diesem Fall antwortet das entsprechende Gerät mit einer „Pong-Nachricht“, die dem Nutzer anzeigt, daß die Kommunikation zwischen beiden Geräten möglich ist.

Für den Simulationsteil der Workbench entfällt die graphische Benutzeroberfläche, die erst im Emulationsteil entwickelt wird. Stattdessen wird das Nutzerverhalten simuliert: jedes zweite Gerät ist aktiv und versucht in regelmäßigen Abständen, einen Nachbarn anzupingen.

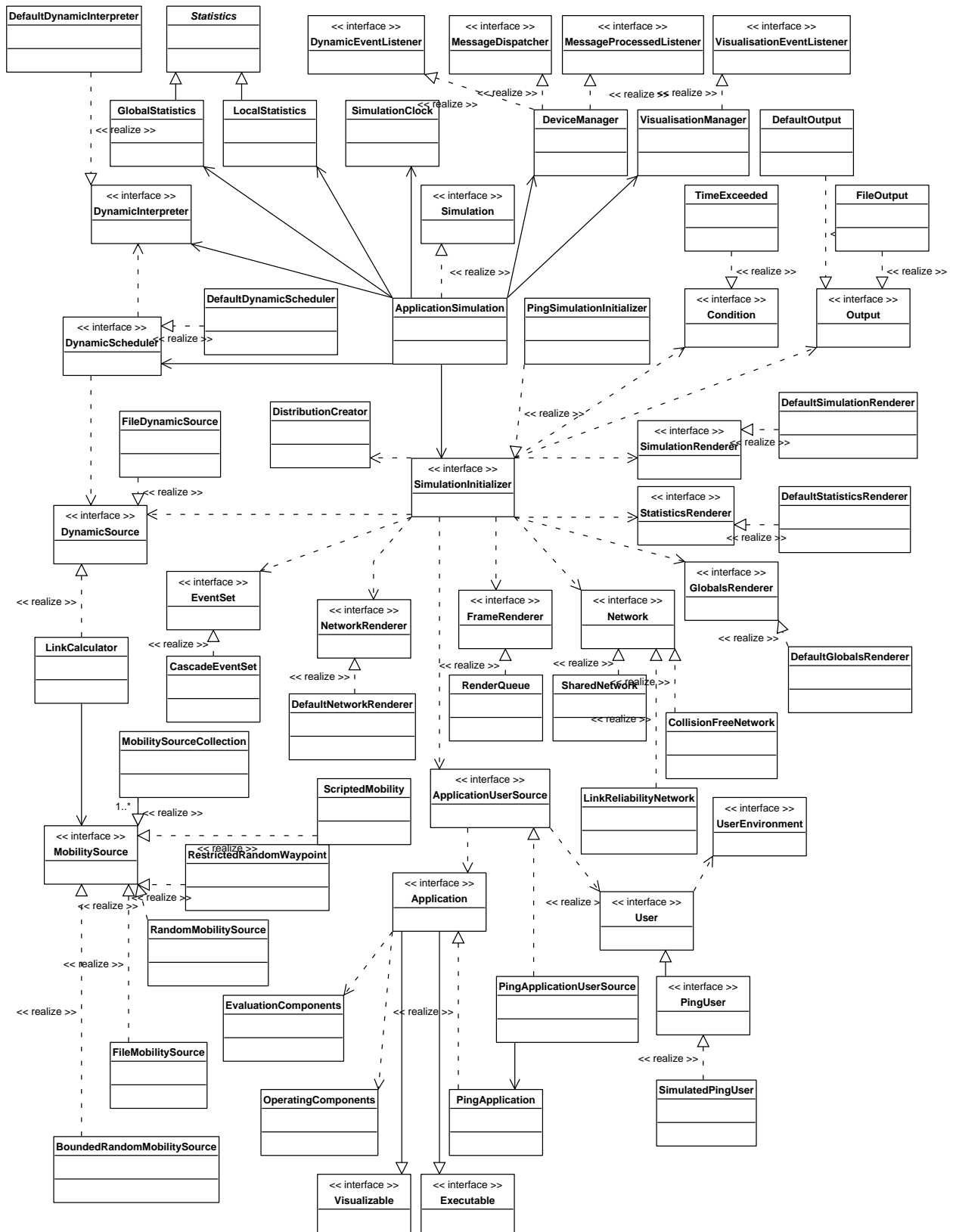


Abbildung 3.7: Vereinfachte UML-Darstellung der Architektur des Simulationsteils der Workbench.

Zur Umsetzung dieser Anwendung im Simulationsteil der Workbench sind drei Teilaufgaben zu erledigen. Die eigentliche Anwendung, die die Erkennung von benachbarten Geräten, das Verschicken von Ping-Nachrichten und das Empfangen von Pong-Nachrichten erlaubt, muß implementiert werden. Dazu kommt als zweites die Definition des simulierten Benutzerverhaltens und abschließend muß die Simulationsumgebung für die Simulation der Anwendung konfiguriert werden.

## Implementierung der Ping-Anwendung

---

Jede in der Workbench ausgeführte Anwendung muß das Interface **Application** implementieren. Die Workbench stellt einen Dienst zur Erkennung von benachbarten Geräten bereit. Indem eine Anwendung das Interface **ConnectionObserver** implementiert und sich entsprechend beim **ConnectionAnnouncer** registriert, wird sie automatisch über in den Sendebereich kommende bzw. ihn verlassende Geräte informiert. Diese beiden Interfaces erfordern die Bereitstellung verschiedener Methoden durch die Anwendung. Somit sieht das Gerüst der Anwendung wie folgt aus:

---

```

1 public class PingApplication implements Application,
2     ConnectionObserver {
3     // Implementation of Application
4     public void start(OperatingComponents operatingComponents,
5         EvaluationComponents evaluationComponents) {
6     }
7
8     public void handleEnter() {
9     }
10
11    public void handleExit() {
12    }
13
14    public Shape getShape() {
15    }
16
17    // Implementation of ConnectionObserver
18    public void attachUnidirectional(Address source) {
19    }
20
21    public void detachUnidirectional(Address source) {
22    }
23
24    public void attachBidirectional(Address other) {
25    }
26

```



```
27 public void detachBidirectional(Address other) {  
28 }  
29 }
```

---

Jede Applikation wird mittels der Methode `start` initialisiert. Dabei liefert die Workbench der Anwendung Referenzen sowohl auf Ressourcen, die zur Ausführung der Applikation benötigt werden (●`OperatingComponents`), als auch auf Evaluationsressourcen (●`EvaluationComponents`) wie das Statistiksystem. Üblicherweise merkt sich die Anwendung diese Referenzen, um später Zugriff auf die entsprechenden Komponenten zu haben. Zusätzlich registriert sich die Ping-Anwendung als `ConnectionObserver` und initialisiert eine ●`HashMap`, in der die gerade laufenden Ping-Versuche gespeichert werden.

---

```
1  
2private OperatingComponents operatingComponents;  
3private EvaluationComponents evaluationComponents;  
4private Map pingsInProgress;  
5  
6public void start(OperatingComponents operatingComponents,  
7                 EvaluationComponents evaluationComponents) {  
8     this.operatingComponents = operatingComponents;  
9     this.evaluationComponents = evaluationComponents;  
10    operatingComponents.getOperatingSystem().getConnectionAnnouncer().add(this);  
11    pingsInProgress = new HashMap();  
12 }
```

---

Desweiteren wird die Anwendung über den Eintritt in die Simulation (Methode `handleEnter`) und das Verlassen der Simulation (Methode `handleExit`) informiert. Für die ●`PingApplication` werden beide Informationen nicht benötigt, daher bleiben beide Methoden leer.

Mittels der Methoden `attachUnidirectional`, `detachUnidirectional`, `attachBidirectional` und `detachBidirectional` meldet der `ConnectionAnnouncer` der Anwendung, daß unidirektionale bzw. bidirektionale Verbindungen zu benachbarten Geräten zustandegekommen oder abgebrochen sind. Für die Ping-Anwendung interessieren nur bidirektionale Verbindungen zu benachbarten Geräten, da ansonsten die Antwort des anderen Gerätes beim Senden einer Ping-Nachricht nicht ankäme. Daher bleiben die Methoden `attachUnidirectional` und `detachUnidirectional` leer.

Im Fall einer hinzukommenden oder wegfallenden bidirektionalen Verbindung muß das simulierte Benutzerverhalten informiert werden, da es ja anhand der aktuellen

Menge von Nachbarn entscheidet, welches Gerät als nächstes „angepingt“ werden soll. Das **④UserSystem** im **④OperatingSystem** des mobilen Geräts, auf dem die Anwendung ausgeführt wird, erlaubt den Zugriff auf das jeweilige Benutzerverhalten. Die Anwendung definiert dazu eine passende **①UserMessage**, die – über das UserSystem abgesetzt – die gewünschte Methode im Benutzerverhalten aufruft. Im Fall einer wegfallenden bidirektionalen Verbindung entfernt die Anwendung zusätzlich noch die Referenzen auf aktive Pings zu dieser Adresse aus der **④HashMap**.

---

```

1 public void attachBidirectional(Address other) {
2     operatingComponents.getUserSystem().
3         handleMessage(new NeighborAddedUserMessage(other));
4 }
5
6 public void detachBidirectional(Address other) {
7     operatingComponents.getUserSystem().
8         handleMessage(new NeighborRemovedUserMessage(other));
9     pingsInProgress.remove(other);
10 }

```

---

Beispielhaft sei hier die **④NeighborRemovedUserMessage** dargestellt: das UserSystem ruft die Methode `handle` auf, die Zugriff auf das der Applikation zugeordnete Benutzerverhalten gibt. Dementsprechend wird die passende Methode `neighborRemoved` mit der Adresse des wegfallenden Nachbarn aufgerufen.

---

```

1 public class NeighborRemovedUserMessage implements UserMessage {
2     private Address address;
3
4     public NeighborRemovedUserMessage(Address address) {
5         this.address = address;
6     }
7     public void handle(User user) {
8         ((PingUser)user).neighborRemoved(address);
9     }
10 }

```

---

Zur Visualisierung muß die Anwendung der Workbench ein **①Shape** liefern, das zur Darstellung des Anwendungszustands verwendet wird. Dieses Shape wird immer wieder abgefragt, so daß die Anwendung ihre Visualisierung dynamisch ändern kann. Zwei Shapes visualisieren die Beispielanwendung: falls kein Ping aktiv ist, wird das erste Shape angezeigt, ansonsten das zweite. Die Ping-Anwendung definiert

dazu zwei Standardshapes (gefüllter Kreis in schwarz bzw. gelb), erlaubt aber auch, diese Shapes im Konstruktor zu übergeben<sup>3</sup>. Die **⊙**HashMap, in der alle aktiven Pings vermerkt werden, erlaubt, effizient den aktuellen Status der Applikation zu bestimmen.

---

```

1private final Shape idleShape;
2private final Shape pingingShape;
3private static final Shape DEFAULT_SHAPE =
4  new EllipseShape(new Position(0,0), new Extent(10,10), Color.BLACK, true);
5private static final Shape DEFAULT_PING_SHAPE =
6  new EllipseShape(new Position(0,0), new Extent(10,10), Color.YELLOW, true);
7
8public PingApplication() {
9  idleShape = DEFAULT_SHAPE;
10  pingingShape = DEFAULT_PING_SHAPE;
11}
12
13public PingApplication(Shape idleShape, Shape pingingShape) {
14  this.idleShape = idleShape;
15  this.pingingShape = pingingShape;
16}
17
18public Shape getShape() {
19  if (pingsInProgress.size() != 0) {
20    return pingingShape;
21  }
22  return idleShape;
23}

```

---

Damit sind alle aus der Implementierung der beiden Interfaces **ⓐ**Application und **ⓐ**ConnectionObserver herrührenden Anforderungen erfüllt. Es fehlt nur noch die eigentliche Funktionalität der Ping-Anwendung, die Möglichkeit, Ping-Nachrichten an einen Nachbarn zu senden und auf eingehende Ping- bzw. Pong-Nachrichten zu reagieren.

Dazu wird die Ping-Anwendung um drei Methoden ergänzt: ping, handlePingMessage und handlePongMessage. Die Methode ping sendet eine **⊙**PingMessage an die angegebene Adresse. Dazu muß sie zuerst eine eigene Nachricht definieren, die **⊙**PingMessage, die das allgemeine Nachrichteninterface **ⓐ**ApplicationMessage implementiert.

---

```

1public class PingMessage implements ApplicationMessage {
2  public void handle(Application application, Address sender) {

```

<sup>3</sup>Diese Möglichkeit wird später in der Emulation der Anwendung genutzt.

```

3    ((PingApplication) application).handlePingMessage(sender);
4  }
5
6  public ApplicationMessage copy() {
7      return this; // stateless!
8  }
9
10 public int getSize() {
11     return 100;
12 }
13
14 public Shape getShape() {
15     return EmptyShape.getInstance();
16 }
17}

```

---

Die Methode `handle` wird von der Workbench aufgerufen, wenn die Nachricht beim Empfänger eintrifft. Die Methode bietet Zugriff auf die empfangende Applikation und den Absender der Nachricht, so daß eine passende Methode in der Zielapplikation aufgerufen werden kann. Im Fall der Ping-Anwendung ist das die Methode `handlePingMessage`.

Zusätzlich muß eine `ApplicationMessage` auch noch die Methode `getSize` implementieren, die die für die Netzwerksimulation relevante Größe der Nachricht in Bytes zurückliefert. Für die Beispielanwendung, in der die Nachricht keine Daten transportiert, wird exemplarisch eine Länge von 100 Bytes festgelegt. Die Methode `copy` wird von der Workbench benutzt, wenn eine Applikationsnachricht von Gerät zu Gerät kopiert wird. Der Applikationsentwickler kann festlegen, ob die Nachricht geklont werden muß, da sie zustandsbehaftet ist, oder ob mehrere Geräte die gleiche Instanz der Nachricht (z.B. bei einem Broadcast) nutzen können. Die `PingMessage` ist zustandslos und muß daher nicht geklont werden. Die Workbench kann auch einzelne Nachrichten visualisieren. Dazu müssen sie ein geeignetes Shape zurückliefern. In der Beispielanwendung wird darauf verzichtet und daher ein leeres Shape zurückgeliefert.

Die `PingMessage` muß in der Methode `ping` an den angegebenen Nachbarn gesendet werden. Dazu bedient sich die Anwendung des `SendDispatchers` aus dem `OperatingSystem`, der Zugriff auf alle zur Verfügung stehenden Netzwerkprotokolle gibt.

---

```

1 public void ping(Address receiver) {
2     OperatingSystem operatingSystem = operatingComponents.getOperatingSystem();

```

```

3 PingMessage pingMessage = new PingMessage();
4 try {
5     operatingComponents.getSendDispatcher().sendLocalUnicast(receiver, pingMessage);
6     operatingSystem.write(operatingSystem.getAddress()+": pinging "+receiver);
7     addPing(receiver);
8 } catch (NetworkException e) {
9     // Ignore
10 }
11}
12

```

---

Im Fall der **●**PingMessage wird nur ein lokaler Unicast benötigt, der als Grundprimitiv des Netzwerkmodells immer zur Verfügung steht. Die Anwendung instanziiert eine neue **●**PingMessage und sendet sie über den **●**SendDispatcher an die spezifizierte Adresse. Zusätzlich gibt sie über die Simulationskonsole eine Meldung über die versendete Nachricht aus und fügt den aktiven Ping mittels der Hilfsmethode addPing der **●**HashMap hinzu. Die beim Versenden potentiell auftretende **●**NetworkException wird einfach ignoriert.

Zur Vervollständigung der Anwendung fehlen nun nur noch die beiden Methoden handlePingMessage und handlePongMessage, die beim Eintreffen dieser beiden Nachrichtentypen aufgerufen werden.

```

1public void handlePingMessage(Address sender) {
2    OperatingSystem operatingSystem = operatingComponents.getOperatingSystem();
3    operatingSystem.write(operatingSystem.getAddress()+": got ping from "+sender);
4    try {
5        operatingComponents.getSendDispatcher().sendLocalUnicast(sender, new PongMessage());
6    } catch (NetworkException e) {
7        // Ignore
8    }
9}
10
11public void handlePongMessage(Address sender) {
12    OperatingSystem operatingSystem = operatingComponents.getOperatingSystem();
13    operatingSystem.write(operatingSystem.getAddress()+": pong from "+sender);
14    removePing(sender);
15    operatingComponents.getUserSystem().
16        handleMessage(new PongMessageReceivedUserMessage(sender));
17}

```

---

Die Methode handlePingMessage gibt eine entsprechende Meldung auf der Konsole aus und sendet dann per lokalem Unicast eine **●**PongMessage an den Sender der

●PingMessage. Der Aufbau einer ●PongMessage ist identisch zur ●PingMessage, nur daß bei Behandlung der Nachricht die Methode `handlePongMessage` aufgerufen wird.

Beim Eintreffen einer ●PongMessage gibt die Methode `handlePongMessage` eine Meldung auf der Konsole aus und entfernt den Sender der Nachricht aus der Liste der aktiven Pings. Über eine ●PongMessageReceivedUserMessage wird das Benutzerverhalten über das Eintreffen der Nachricht informiert. Die ●PongMessageReceivedUserMessage ist dabei identisch zu der auf Seite 52 beschriebenen ●NeighborRemovedUserMessage aufgebaut, nur wird im Benutzerverhalten die Methode `handlePongMessage` aufgerufen.

## Implementierung des Benutzerverhaltens

---

Neben der eigentlichen Applikation muß auch ein simuliertes Benutzerverhalten für die Applikation definiert werden. Dies geschieht, indem das Interface ●User implementiert wird. Dieses Interface definiert nur eine Methode `start`, die zur Initialisierung des Benutzerverhaltens dient. Über diese Methode erhält das Benutzerverhalten eine Referenz auf das ●UserEnvironment, das Zugriff auf wichtige Informationen wie die Geräteadresse, die aktuelle Simulationszeit etc. bietet und die Interaktion mit der Applikation, die auf dem mobilen Gerät ausgeführt wird, ermöglicht.

Alle weiteren Methoden des Benutzerverhaltens, die von der Applikation aufgerufen werden, sind frei definierbar. Im Fall der Beispielanwendung sind sie in einem gesonderten Interface ●PingUser zusammengefaßt, das das Interface ●User erweitert.

---

```

1 public interface PingUser extends User {
2     public void neighborAdded(Address neighbor);
3     public void neighborRemoved(Address neighbor);
4     public void handlePongMessage(Address sender);
5 }

```

---

Das simulierte Benutzerverhalten ●SimulatedPingUser implementiert entsprechend dieses Interface. Über ein boolesches Flag im Konstruktor wird definiert, ob es sich um einen aktiven oder passiven Benutzer handelt. Aktive Benutzer pingen in regelmäßigen Abständen andere Geräte an. Zusätzlich wird im Konstruktor die Liste der benachbarten Geräte initialisiert.

---

```

1 public class SimulatedPingUser implements PingUser {
2     private boolean active;
3     private UserEnvironment userEnvironment;
4     private List neighbors;
5
6     public SimulatedPingUser(boolean active) {
7         this.active = active;
8         neighbors = new ArrayList();
9     }
10
11    public void start(UserEnvironment userEnvironment) {
12        this.userEnvironment = userEnvironment;
13        if (active) {
14            scheduleNextPing();
15        }
16    }

```

---

Um eine erste Ping-Nachricht eines aktiven Benutzers zu versenden, wird die Methode `scheduleNextPing` aufgerufen. Diese Methode nutzt einen Timer, um nach 30 Sekunden die Methode `handlePingTimeout` aufzurufen. Falls zu dem entsprechenden Zeitpunkt Nachbarn in Sendereichweite sind, wird über eine sogenannte **①**UserInteraction im **②**UserEnvironment das Versenden einer Ping-Nachricht durch die Ping-Anwendung ausgelöst.

---

```

1 private void scheduleNextPing() {
2     userEnvironment.getTimer().set(new Timeout(30) {
3         public void handle() {
4             handlePingTimeout();
5         }
6     });
7 }
8
9 private void handlePingTimeout() {
10    if (!neighbors.isEmpty()) {
11        userEnvironment.setInteraction(new PingInteraction((Address)neighbors.get(0)));
12    }
13    scheduleNextPing();
14 }

```

---

Das **②**UserEnvironment ruft die Methode `handle` der **③**PingInteraction auf, die in der Ping-Applikation die Methode `ping` aufruft.

---

```

1 public class PingInteraction implements UserInteraction {
2     private Address receiver;

```

```

3
4 public PingInteraction(Address receiver) {
5     this.receiver = receiver;
6 }
7
8 public void handle(Application application) {
9     ((PingApplication) application).ping(receiver);
10 }
11}

```

---

Die Liste der benachbarten Geräte wird über die beiden Methoden `neighborAdded` und `neighborRemoved` verwaltet, die – wie oben beschrieben – von der Applikation aufgerufen werden.

## Konfiguration der Workbench

---

Zur Simulation der Ping-Anwendung muß die Workbench konfiguriert werden. Die Parameter der Workbench werden über den **●** `PingSimulationInitializer`, der das Interface **Ⓢ** `SimulationInitializer` implementiert, definiert.

```


1 public interface SimulationInitializer {
2     public String getSimulationName();
3     public EventSet getEventSet();
4     public Network getNetwork();
5     public NetworkRenderer getNetworkRenderer();
6     public DistributionCreator getDistributionCreator();
7     public Condition getTerminalCondition();
8     public ApplicationUserSource getApplicationUserSource();
9     public DynamicSource getDynamicSource();
10    public Shape getBackgroundShape();
11    public FrameRenderer getFrameRenderer();
12    public Output getResultOutput();
13    public Output getConsoleOutput();
14    public SimulationRenderer getSimulationRenderer();
15    public StatisticsRenderer getStatisticsRenderer();
16    public GlobalsRenderer getGlobalsRenderer();
17    public ShapeBuilder getShapeBuilder();
18    public void initializeGlobals(GlobalsInitializer globalsInitializer);
19    public void initializeStatistics(StatisticsInitializer statisticsInitializer,
20                                    StatisticsMonitorSystem statisticsMonitorSystem);
21 }
22

```

---

Durch die Abfrage der Konfiguration über ein Interface kann der Anwendungsentwickler die Parameter der Workbench ganz nach seinen Wünschen definieren. Im




Fall der Ping-Anwendung werden viele der Standardimplementierungen benutzt, die mit der Workbench mitgeliefert werden. Es wäre aber problemlos möglich, selbst entwickelte Komponenten einzubinden. Hier sei nur der Konstruktor des  PingSimulationInitializers gezeigt; die dort definierten Komponenten werden in den entsprechenden Getter-Methoden zurückgeliefert.

---

```
1 public PingSimulationInitializer(boolean visualize, int numberOfDevices) {
2     simulationName = "ping simulation";
3     eventSet = new CascadeEventSet(10, 25);
4     network = new CollisionFreeNetwork(1024);
5     networkRenderer = new DefaultNetworkRenderer();
6     distCreator =
7         new DistributionCreator(new long[] { 1231, 6473, 69543, 56887,
8                                             4663783, 547894, 3654234,
9                                             958544, 4342, 4648501 });
10    terminalCondition = new TimeExceeded(eventSet, 600);
11    applicationUserSource = new PingApplicationUserSource();
12    resultOutput = new DefaultOutput();
13    consoleOutput = new DefaultOutput();
14    simulationRenderer = new DefaultSimulationRenderer();
15    statisticsRenderer = new DefaultStatisticsRenderer();
16    globalsRenderer = new DefaultGlobalsRenderer();
17    shapeBuilder = new DefaultShapeBuilder();
18    dynamicSource = new LinkCalculator(createMobilitySource(numberOfDevices), null);
19    backgroundShape = EmptyShape.getInstance();
20    if (visualize) {
21        frameRenderer =
22            new RenderQueue(16, 0.1, dynamicSource.getShape());
23    } else {
24        frameRenderer = null;
25    }
26 }
```

---

Es werden der Name der Simulation und die Implementierung des FES (Cascade-Implementierung [60]) festgelegt. Als Netzwerkmodell wird eine Implementierung ohne Kollisionen und beliebiger Kapazität (siehe Abschnitt 3.3) verwendet. Die Visualisierung des Netzwerks geschieht über die Standardimplementierung der Workbench, die die Senderadien der mobilen Geräte und die Verbindungen zwischen ihnen darstellt.

Der  DistributionCreator dient dazu, die Zufallszahlengeneratoren mit geeigneten Seed-Werten zu versorgen. In der Beispielanwendung sind dies nur wenige Werte, für wirkliche Simulationsexperimente sollten ausreichend viele Seed-Werte zur Verfügung gestellt werden. Die benötigte Anzahl hängt davon ab, wieviele unabhängige Zufallsverteilungen benötigt werden.

Das Simulationseende wird über eine **①**Condition definiert: die Bedingung **②**TimeExceeded ist erfüllt, wenn die Simulationszeit 600 erreicht wird. Über eine Implementierung des Condition-Interfaces sind aber auch erheblich kompliziertere Abbruchbedingungen denkbar.

---

```

1public class TimeExceeded implements Condition {
2    private EventSet eventSet;
3    private double time;
4
5    public TimeExceeded(EventSet eventSet, double time) {
6        this.eventSet = eventSet;
7        this.time = time;
8    }
9
10   public boolean reached() {
11       return eventSet.getTime() > time;
12   }
13}

```

---

Die Zuordnung von mobilen Geräten, Benutzerverhalten und Applikationen geschieht über Implementierungen des Interface **①**ApplicationUserSource.

---

```

1public class PingApplicationUserSource implements ApplicationUserSource {
2    private int count = 0;
3
4    public ApplicationUserPair getApplicationUserPair(double time, Address device) {
5        if (count++%2 == 0) {
6            return new ApplicationUserPair(new PingApplication(),
7                                           new SimulatedPingUser(true));
8        } else {
9            return new ApplicationUserPair(new PingApplication(),
10                                         new SimulatedPingUser(false));
11        }
12    }
13}

```

---

Für die Ping-Anwendung ist die entsprechende **②**PingApplicationUserSource sehr einfach: jedes Gerät erhält ein Tupel aus **③**PingApplication und **④**SimulatedPingUser, wobei über einen Zähler jedes zweite Gerät ein aktives Benutzerverhalten zugeteilt bekommt.

Die Ausgabe von Konsolenmeldungen und der Statistik (nicht verwendet in der Beispielanwendung) geschieht mittels der Standardimplementierungen der Workbench.

Der ❶ShapeBuilder regelt die Reihenfolge, in der die einzelnen Visualisierungskomponenten übereinander gemalt werden. Auch hier wird die Standardimplementierung verwendet.

Die Berechnung der Verbindungen zwischen den mobilen Geräten erfolgt mittels des ❷LinkCalculator, der auf die Daten des Bewegungsmodells zurückgreift. Dieses Bewegungsmodell wird in der Methode createMobilitySource erzeugt:

---

```

1private MobilitySource createMobilitySource(int numberOfDevices) {
2    ContinuousDistribution enterDist =
3        distCreator.getContinuousUniformDistribution(new ConstantDoubleMapping(0.0),
4                                                    new ConstantDoubleMapping(60.0));
5    ContinuousDistribution exitDist =
6        distCreator.getContinuousUniformDistribution(new ConstantDoubleMapping(540.0),
7                                                    new ConstantDoubleMapping(599.9));
8    RandomIntervalGenerator lifetime = new RandomIntervalGenerator(enterDist, exitDist);
9    ContinuousDistribution xDist =
10       distCreator.getContinuousUniformDistribution(new ConstantDoubleMapping(0.0),
11                                                  new ConstantDoubleMapping(1000.0));
12    ContinuousDistribution yDist =
13       distCreator.getContinuousUniformDistribution(new ConstantDoubleMapping(0.0),
14                                                  new ConstantDoubleMapping(1000.0));
15    RandomPositionGenerator posGen = new RandomPositionGenerator(xDist, yDist);
16    ContinuousDistribution pauseDist =
17       distCreator.getExponentialDistribution(new ConstantDoubleMapping(1.0 / 60.0));
18    ContinuousDistribution sendingDist =
19       distCreator.getContinuousUniformDistribution(new ConstantDoubleMapping(0.0),
20                                                  new ConstantDoubleMapping(100.0));
21    ContinuousDistribution speedDist =
22       distCreator.getContinuousUniformDistribution(new ConstantDoubleMapping(0.8),
23                                                  new ConstantDoubleMapping(1.0));
24    return new RandomMobilitySource(numberOfDevices, lifetime, posGen, pauseDist,
25                                   sendingDist, speedDist);
26}

```

---

Der ❸RandomPositionGenerator nutzt die X- und Y-Verteilungen, um zufällig Positionen zu erzeugen. Aufbauend auf diesen Positionen erzeugt die ❹RandomMobilitySource Bewegungsdaten, die dem Random Walk Mobility Model entsprechen, indem die Verteilungen für den Eintritts- und Austrittszeitpunkt, die Gerätegeschwindigkeit und die Länge der Pausen verwendet werden. Zusätzlich definiert die Verteilung sendingDist die Größe der Senderadien der mobilen Geräte. Mit den Daten der Beispielanwendung erhält man so Bewegungsdaten für die angegebene Zahl von Geräten auf einer Fläche von  $1000 \times 1000$  Metern, einer gleichverteilten

Gerätegeschwindigkeit von 0.8 bis 1 Meter pro Sekunde und einer durchschnittlichen Pausenzeit von 60 Sekunden (exponentialverteilt). Die Senderadien der Geräte sind gleichverteilt zwischen 0 und 100 Metern.

Im Konstruktor des Initializers (siehe Listing auf Seite 59, Zeile 19) folgt die Definition des Hintergrundbildes der Visualisierung. Für die Beispielanwendung wird kein Hintergrundbild, sondern ein leeres Shape definiert. Über einen Parameter kann man beim Initializer festlegen, ob die Visualisierung verwendet werden soll. Ist dies nicht der Fall, so bleibt die entsprechende Ausgabekomponente, der **FrameRenderer**, undefiniert (Zeile 24). Ansonsten wird die Standardimplementierung zur Visualisierung, die **RenderQueue**, verwendet. Dabei werden ein Puffer von 16 Frames und eine Framerate von 10 Bildern pro Sekunde definiert. Der Puffer wird initial mit einem Frame, bestehend aus der Visualisierung der **DynamicSource**, aber ohne Konsolenausgabe und zu visualisierende Statistikwerte gefüllt. Sobald die Simulation beginnt, wird der Puffer mit „echten“ Frames gefüllt. Bis dahin wird der initiale Frame verwendet.

Zur Ausführung der Simulation fehlt noch ein Hauptprogramm, das die Simulation über den Initializer definiert und startet.

---

```

1 public class PingSimulation {
2     private static final int NUMBER_OF_DEVICES = 50;
3     private static final boolean VISUALIZE = true;
4
5     public static void main(String[] args) {
6         SimulationInitializer initializer =
7             new PingSimulationInitializer(VISUALIZE, NUMBER_OF_DEVICES);
8         if (VISUALIZE) {
9             new SimulationFrame(initializer.getSimulationName(),
10                                (RenderQueue)initializer.getFrameRenderer(), 10.0);
11         }
12         Simulation simulation = new ApplicationSimulation(initializer);
13         simulation.run();
14     }
15 }

```

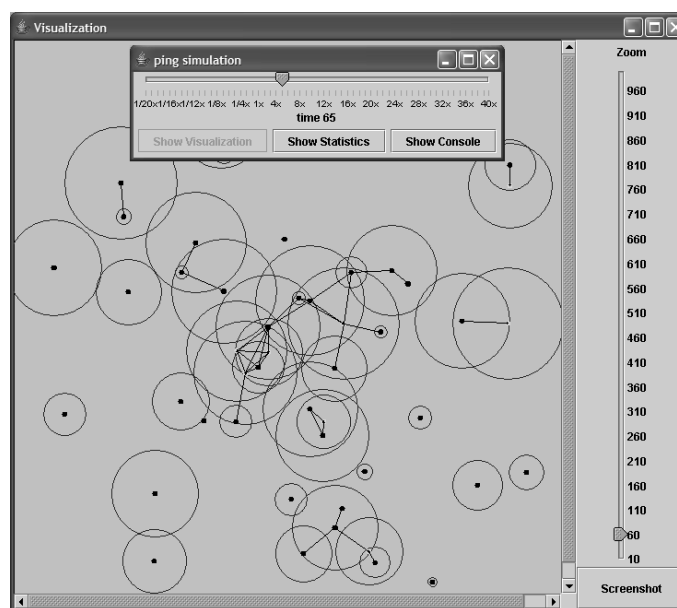
---

Der Initializer wird mit der gewünschten Anzahl von Geräten und dem Visualisierungsflag instanziiert, und im Fall der Visualisierung wird ein **SimulationFrame** erzeugt, die Defaultimplementierung der graphischen Simulationsoberfläche in Swing. Der Initializer wird genutzt, um die eigentliche Simulation, eine Instanz der Klasse **ApplicationSimulation** zu initialisieren und mittels der Methode `run` zu starten.

## Ausführung der Simulation

---

Zur Ausführung einer Simulation der Ping-Anwendung startet man das Hauptprogramm **●**PingSimulation. Daraufhin erscheint – falls man die Visualisierung aktiviert hat – die in Abbildung 3.8 dargestellte graphische Oberfläche. Zu sehen ist die Simulationsfläche mit den mobilen Geräten samt Senderadien und Verbindungen. Die Darstellung der Simulationsfläche kann beliebig skaliert werden. Außerdem ist die Simulationskontrolle zu sehen, bestehend aus der Geschwindigkeitsregelung der Workbench, mit der die Ausführungsgeschwindigkeit der Simulation beschleunigt (sofern die Rechenleistung ausreicht) und verlangsamt werden kann, und der Möglichkeit, die Konsolen- und Statistikausgaben ein- und auszublenden.



**Abbildung 3.8:** Visualisierung der Simulation der Ping-Anwendung.

## 3.9 Performancemessungen

---

Zur besseren Einschätzung der Skalierbarkeit und Performance der Simulationsumgebung der Workbench dienen die folgenden Messungen mit einer einfachen Informationsverteilungsanwendung: zu Beginn besitzt genau ein Gerät eine Textnachricht, die an alle Geräte verbreitet werden soll. Jedes Gerät, das die Textnachricht noch nicht empfangen hat, fragt in Reichweite kommende Geräte nach der Text-

nachricht. Falls ein Gerät, das die Textnachricht schon besitzt, eine solche Anfrage erhält, schickt es diese an das anfragende Gerät. Dieses verbreitet die soeben erhaltene Textnachricht per lokalem Broadcast an alle Nachbargeräte und wartet dann auf Anfragen von anderen Geräten.

Diese einfache Anwendung wird in Kombination mit dem Random Waypoint Mobility Model und einem einfachen Netzwerkmodell (keine Verluste, keine Kollisionen, beliebige Kapazität) für 10 Minuten simuliert. Der Senderadius der Geräte wird gleichverteilt zwischen 10 und 100 Metern gewählt, die Geschwindigkeit der Geräte liegt gleichverteilt zwischen 0.8 und 1.0 Metern pro Sekunde.

# Geräte	Ausführungszeit	Ausführungszeit (vorberechnet)
50	0.52	0.87
100	0.76	1.08
200	1.30	1.61
400	3.31	3.69
800	11.77	11.97
1600	47.90	40.72
3200	245.13	167.30
6400	3797.74 <sup>1</sup>	1173.89
12800	n/a <sup>2</sup>	n/a <sup>2</sup>

**Tabelle 3.1:** Ausführungszeiten (in Sekunden) für die Simulation der Beispielanwendung auf einer Simulationsfläche von  $500m \times 500m$  mit steigender Gerätedichte.

Zwei Meßreihen werden mit dieser Anwendung auf einem Pentium IV (HT) mit 3 GHz und Java 2 SDK 1.4.1 (IBM) unter Linux durchgeführt. Die erste Meßreihe mißt die Ausführungszeit der Simulation für eine wachsende Zahl von Geräten auf einer festen Simulationsfläche von  $500m \times 500m$ . Die durchschnittliche Gerätedichte wächst also in jedem Schritt. In der zweiten Meßreihe wächst die Größe der Simulationsfläche mit der Zahl der simulierten Geräte, so daß die durchschnittliche Gerätedichte über alle Schritte gleich bleibt. Die Tabellen 3.1 und 3.2 zeigen klar, daß die Gerätedichte ein wichtiger Faktor für die Performance der Simula-

<sup>1</sup>Im Gegensatz zu den anderen Simulationsläufen schnitt die ebenfalls getestete SUN JVM in diesem Fall erheblich besser ab und benötigte nur 2686.99 Sekunden.

<sup>2</sup>Der verfügbare Speicher des Testrechners (1792 MB) reichte nicht aus, um die Anwendung zu simulieren. Dies rührt aus der hohen Gerätedichte und der damit verbundenen großen Anzahl von Ereignissen her.

# Geräte	Simulationsfläche	Ausführungszeit	Ausführungszeit (vorberechnet)
125	250 × 250	1.24	1.81
250	354 × 354	2.20	2.81
500	500 × 500	4.78	5.23
1000	707 × 707	11.69	10.12
2000	1000 × 1000	29.50	19.87
4000	1414 × 1414	77.05	35.57
8000	2000 × 2000	233.10	66.32
16000	2828 × 2828	807.87	118.73
32000	4000 × 4000	4784.08	243.48

**Tabelle 3.2:** Ausführungszeiten (in Sekunden) für die Simulation der Beispielanwendung mit einer durchschnittlichen Gerätedichte von 0.002 Geräten pro  $m^2$ .

tionsumgebung ist. Zu beachten ist dabei, daß die rechnerische durchschnittliche Gerätedichte die Situation während der Simulation nicht richtig wiedergibt, da die Gerätedichte aufgrund der Eigenschaften des Random Waypoint Mobility Model in der Mitte der Simulationsfläche höher ist als in den äußeren Bereichen.

Zusätzlich wird im gleichen Umfeld eine dritte Meßreihe mit einer weiteren einfachen Anwendung durchgeführt, um die Auswirkung der Netzwerklast auf die Simulationsgeschwindigkeit zu ermitteln. In dieser Meßreihe wird die durchschnittliche Gerätedichte wie in der ersten Meßreihe schrittweise erhöht. Gleichzeitig wird aber auch die Netzwerklast erhöht: eine steigende Zahl von Geräten (#Sender in Tabelle 3.3) flutet jeweils eine Region im Umkreis von 50m mit konstanter Bitrate (alle 4s ein Datenpaket, das von allen Geräten in der Region per Broadcast weiterverteilt wird).

Im Fall der konstanten Gerätedichte skaliert die Simulationsumgebung gut mit der Geräteanzahl, da die Zahl der Verbindungen pro Gerät nicht steigt. In allen Fällen ist die Ausführungszeit ungefähr quadratisch zu der Zahl der Geräte, da alle Geräte miteinander verglichen werden müssen. Berechnet man allerdings die Bewegungs- und Verbindungsdaten vor, so ist die Ausführungszeit linear zur Anzahl der Geräte, da die Verbindungsanzahl pro Gerät konstant ist. In beiden Fällen ist die Simulation einer großen Gerätezahl in Echtzeit möglich. Selbst die Simulation von 8000 Geräten ist mehr als doppelt so schnell wie in Echtzeit möglich.

Für vorberechnete Mobilitäts- und Konnektivitätsdaten sinken die Ausführungs-

#Geräte	50	100	200	400	800	1600	3200
#Sender							
0	0.47	0.61	0.96	2.49	9.73	40.50	210.89
10	0.76	1.08	1.58	3.94	16.61	62.68	311.48
20	0.90	1.15	2.00	4.87	21.64	82.60	399.88
40	0.94	1.36	2.54	7.53	30.62	129.91	558.92
80	x	1.79	3.93	12.57	53.62	231.52	982.71
160	x	x	6.70	23.85	95.68	444.40	1936.53
320	x	x	x	44.29	190.65	861.43	3882.74
640	x	x	x	x	394.41	1703.27	7662.28

**Tabelle 3.3:** Ausführungszeiten (in Sekunden) der zweiten Beispielanwendung für steigende Gerätedichte und steigende Netzwerklast.

zeiten signifikant. Besonders im Fall der konstanten Gerätedichte ist der Geschwindigkeitsgewinn sehr groß. Ein Simulationslauf mit 32000 mobilen Geräten wird fast 20-mal so schnell ausgeführt (243.5s statt 4784.1s). In der Messung für steigende durchschnittliche Gerätedichten verringert sich die Ausführungszeit um 70% (1173.9s statt 3797.7s)

Nutzt man die Java2D-Visualisierungskomponente der Workbench, so ist die Visualisierung der Informationsverteilungsanwendung im Fall steigender Gerätedichte in Echtzeit (oder schneller) bis zu 3200 mobilen Geräten möglich. Für den Fall der konstanten Gerätedichte erhöht sich dieser Wert auf die maximale Zahl von 8000 mobilen Geräten, die noch in Echtzeit visualisiert werden können. Weitere Geschwindigkeitszugewinne sind durch die Verwendung vorberechneter Mobilitäts- und Konnektivitätsdaten möglich. Zu beachten ist insgesamt, daß die Geschwindigkeit der Visualisierung auch vom Detaillierungsgrad abhängt. Im Falle sehr aufwendiger Visualisierungen kann die Geschwindigkeit auch niedriger liegen und die maximale Zahl in Echtzeit visualisierbarer Geräte entsprechend geringer sein.

Erhöht man die Netzwerklast linear, so verlängert sich die Ausführungszeit der Simulation ebenfalls nur linear. Die erste Zeile der Tabelle 3.3 zeigt die Zeit, die die Simulationsumgebung für Konnektivitätsberechnung, Bewegungsmodellberechnung und generelle Simulation des Szenarios benötigt, wenn keines der mobilen Geräte auf das drahtlose Netzwerk zugreift. Rechnet man diese Grundlast aus den Zahlen für die Messungen mit Netzwerklast heraus, so erhöht eine Verdoppelung der Netzwerklast die Ausführungszeit nur um den Faktor 1.1 bis 2.2. Die Datenrate des Netzwerks (10240 bytes/s) und die Datenrate, mit der die Geräte senden



---

(128 *bytes/s*), sind so gewählt, daß das Netzwerk im Fall von 640 sendenden Geräten bei einer Gesamtzahl von 3200 Geräten nahezu saturiert ist.



# 4

## Workbench: Hybride Emulation

---

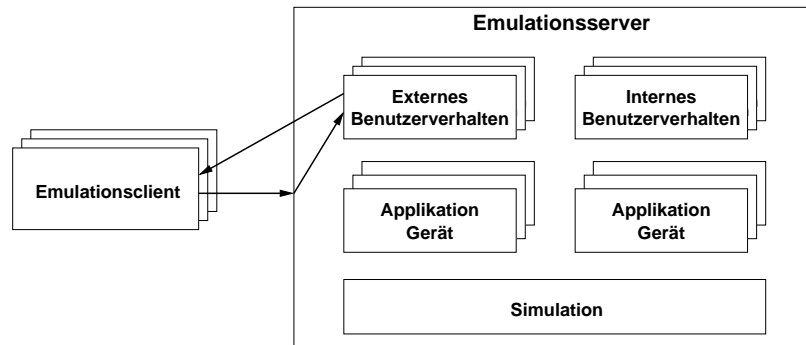
Die Simulation von Anwendungen bzw. Algorithmen für mobile multihop Ad-Hoc-Netze reicht normalerweise nicht aus, um das Verhalten abschließend beurteilen zu können. Mit der Komplexität der simulierten Anwendung steigt auch der Aufwand, der zur Simulation des Benutzerverhaltens in der Applikation notwendig ist. Doch auch komplexes simuliertes Benutzerverhalten kann echtes Benutzerverhalten nicht ganz ersetzen. Gleichzeitig sind Feldversuche mit mobilen Geräten sehr aufwendig und teuer – schließlich müssen entsprechend viele mobile Geräte und Versuchspersonen bereitstehen – und sollten erst dann unternommen werden, wenn das Funktionieren der Anwendung hinreichend gesichert ist.

Als Zwischenschritt zwischen der reinen Simulation und dem Feldversuch bietet die Workbench die Emulation der Anwendung mittels eines hybriden Ansatzes an. Die Grundidee besteht darin, das Benutzerverhalten in der Simulation an eine über Netzwerk angebundene graphische Oberfläche zu koppeln und auf diese Weise „echtes“ Benutzerverhalten in der Simulation zu erhalten. Gleichzeitig kann das Gesamtszenario visualisiert werden und die Benutzer erhalten ein „Gefühl“ für das Verhalten der Applikation in der Realität. Alle anderen Komponenten wie Bewegungs- und Netzwerkmodell werden weiterhin simuliert. Die folgenden Abschnitte stellen vor, wie dieser Ansatz in der Workbench umgesetzt wird.

## 4.1 Architektur

---

Abbildung 4.1 zeigt die Architektur der hybriden Emulationsumgebung der Workbench. Ein zentraler Emulationsserver steuert die ursprüngliche Simulation und vermittelt zwischen den extern angebotenen Clients und dem entsprechenden Benutzerverhalten in der Simulation.



**Abbildung 4.1:** Architektur der hybriden Emulationsumgebung der Workbench.

Die ursprüngliche, simulierte Anwendung wird für die Emulation unverändert übernommen. Das bestehende simulierte Benutzerverhalten wird ergänzt durch ein *externes Benutzerverhalten*, das eine identische Schnittstelle bereitstellt, aber mit dem Emulationsserver verbunden ist. Der Emulationsserver ist als RMI-Server [82] realisiert, zu dem die Clients per RMI eine Verbindung aufbauen und Callbacks registrieren. So können sie Aktionen im externen Benutzerverhalten auslösen und über Reaktionen der Applikation informiert werden.

Der Emulationsserver führt die Simulation in einem eigenen Thread aus. Bei Anmeldung eines externen Clients wird dieser an das externe Benutzerverhalten eines simulierten Geräts gekoppelt. Aus Sicht der Anwendung sind Geräte mit externem und internem Benutzerverhalten nicht zu unterscheiden, daher sind auch keine Änderungen an der simulierten Anwendung nötig.

## 4.2 Synchronisation

---

Die Simulationsumgebung der Workbench läuft als ereignisgesteuerte Simulation in einem Thread: in einer Schleife werden die Ereignisse aus dem FES sequentiell abgearbeitet. Dieses Single-Thread-Programmiermodell gilt somit auch innerhalb des

Applikationscodes der simulierten Anwendungen. Der Entwickler muß sich daher um Nebenläufigkeit und Synchronisation keine Gedanken machen.

Im Rahmen des hybriden Emulationsansatzes wird die Simulation nun allerdings mit per RMI angebotenen Clients gekoppelt, die beliebig nebenläufige Ereignisse in der Simulation auslösen können. Kritisch ist insbesondere häufig auftretender Code, der beispielsweise die aktuelle Simulationszeit abfragt, um eine Bedingung zu prüfen, und bei vorliegender Bedingung ein neues Ereignis, basierend auf dieser Simulationszeit, in das FES einfügt. In der Simulationsumgebung der Workbench kann der Entwickler sicher sein, daß der gesamte Codeblock atomar ausgeführt wird. In der Emulationsumgebung ist dies ohne besondere Vorkehrungen nicht gewährleistet. Der Zugriff auf die konkrete FES-Implementierung wird daher mittels eines generischen Wrappers synchronisiert. Ein Thread kann den Zugriff auf das FES für sich reservieren und atomar beliebig viele Ereignisse in das FES einfügen. Auf diese Weise können auch nebenläufige Anforderungen der Clients in das Single-Thread-Programmiermodell der Workbench eingebunden werden. Die Sperrung des FES führt dazu, daß die Simulation kurzfristig angehalten wird; daher sollten die Sperren nur für möglichst kurze Zeit genutzt werden, um die Ausführung der Emulation nicht zu verzögern.

## 4.3 Graphische Oberfläche

---

Die graphische Oberfläche für den Client der Emulation kann frei gestaltet werden; einzige Einschränkung ist die Anbindung des Clients per RMI an den Emulationsserver. Sinnvollerweise wird die graphische Oberfläche allerdings direkt im Hinblick auf die Einsetzbarkeit auf einem mobilen Gerät entwickelt, da der entstehende Code – wie in Kapitel 5 gezeigt wird – später direkt auf mobilen Geräten verwendet werden kann. Aktionen des Benutzers werden per RMI an den Emulationsserver weitergeleitet, während der Applikationszustand durch Reaktion auf Callbackaufrufe seitens des Emulationsservers aktuell dargestellt wird.

Die Entwicklung der graphischen Oberfläche ist der einzige Schritt, der die Neuentwicklung von Programmcode erfordert, wenn eine simulierte Anwendung aus der Simulationsumgebung der Workbench in der hybriden Emulationsumgebung der Workbench verwendet werden soll.

## 4.4 Hybride Emulation der Beispielanwendung

---

Um die hybride Emulationsumgebung der Workbench besser zu veranschaulichen, wird im folgenden die Umsetzung der in Abschnitt 3.8 beschriebenen Ping-Anwendung für die Emulationsumgebung beschrieben.

Zur Ausführung der Ping-Anwendung in der Emulationsumgebung muß die Anwendung sowohl um ein per RMI fernsteuerbares Benutzerverhalten als auch um eine graphische Oberfläche erweitert werden.

### Externes Benutzerverhalten

---

Die Steuerung des Benutzerverhaltens über RMI benötigt zwei Grundfunktionalitäten: ein externer Client muß sich anmelden können, um durch Callbacks über Ereignisse wie hinzukommende Nachbarn etc. informiert zu werden, und er muß die Möglichkeit haben, eine Ping-Nachricht an eine bestimmte Adresse zu schicken. Diese beiden Funktionalitäten werden im Interface **①ExternalPingUserInterface** zusammengefaßt.

---

```

1public interface ExternalPingUserInterface extends Remote, Serializable {
2    public RemoteClientId registerExternalClient(RemotePingClient client)
3        throws RemoteException, TooManyExternalClientsException;
4    public void sendPingMessage(RemoteClientId clientId, Address receiver)
5        throws RemoteException, UnknownRemoteClientIdException;
6}

```

---

Die Callbacks für die Ereignisse (`handlePongMessage`, `neighborAdded`, `neighborRemoved`) sind im Interface **①RemotePingClient** zusammengefaßt.

---

```

1public interface RemotePingClient extends Remote {
2    public void handlePongMessage(Address sender) throws RemoteException;
3    public void neighborAdded(Address neighbor) throws RemoteException;
4    public void neighborRemoved(Address neighbor) throws RemoteException;
5}

```

---

Ein RMI-Client, der das Interface **①RemotePingClient** implementiert, kann so das Benutzerverhalten eines Geräts in der Emulation steuern, wenn diese das Interface **①ExternalPingUserInterface** erfüllt.

Analog zum **SimulatedPingUser** in der Simulationsumgebung wird ein **ExternalUser** definiert, der diese Steuerung erlaubt. Dazu bietet diese Implementierung des Benutzerverhaltens neben den im Interface **PingUser** definierten Methoden noch die Möglichkeit, einen **RemotePingClient** zu registrieren und Ping-Nachrichten zu versenden.

---

```
1 public class ExternalUser implements PingUser {
2     private UserEnvironment userEnvironment;
3     private Object syncObject;
4     private RemotePingClient remotePingClient;
5
6     public ExternalUser(Object syncObject) {
7         this.syncObject = syncObject;
8     }
9
10    public void start(UserEnvironment userEnvironment) {
11        this.userEnvironment = userEnvironment;
12    }
13
14    public void registerExternalClient(RemotePingClient client) {
15        this.remotePingClient = client;
16    }
17
18    ...
```

---

Das im Konstruktor übergebene Sync-Objekt dient zur Synchronisation der Zugriffe auf die FES, da im Gegensatz zur reinen Simulation in der Emulation asynchrone Zugriffe der externen Clients möglich sind. Die Referenz auf den externen Client wird gespeichert, um die Callbacks aufrufen zu können.

---

```
1 public void sendPingMessage(Address receiver) {
2     synchronized(syncObject) {
3         userEnvironment.setInteraction(new PingInteraction(receiver));
4     }
5 }
6
7 public void handlePongMessage(Address sender) {
8     if (remotePingClient != null) {
9         try {
10            remotePingClient.handlePongMessage(sender);
11        } catch (RemoteException e) {
12            e.printStackTrace();
13        }
14    }
15 }
```

---

Die Methode `sendPingMessage` verwendet das Sync-Objekt, um sich den alleinigen Zugriff auf die FES zu sichern und setzt dann eine neue **●**`PingInteraction` ab. Die Methode `handlePongMessage` zeigt exemplarisch, wie ein Callback im Remote-Client aufgerufen wird; die übrigen Methoden für Callbacks sehen analog aus.

Die Emulation selbst wird in der Klasse **●**`PingEmulation` definiert. Diese Klasse funktioniert im Prinzip wie die **●**`PingSimulation` aus der Simulationsumgebung. Allerdings implementiert sie das Interface **①**`Runnable` statt eine Main-Methode zu enthalten, da sie in einem eigenen Thread ausgeführt wird. In Erweiterung der **●**`PingSimulation` leitet sie sich von **●**`UnicastRemoteObject` ab, um als RMI-Server zu dienen, und implementiert die Methoden aus **①**`ExternalPingUserInterface`, damit externe Clients das Benutzerverhalten einzelner Geräte steuern können.

---

```

1 public class PingEmulation extends UnicastRemoteObject
2     implements ExternalPingUserInterface, Runnable {
3     private static final int NUMBER_OF_EXTERNAL_USERS = 1;
4     private static final int NUMBER_OF_DEVICES = 59;
5     private static final boolean VISUALIZE = true;
6
7     private HashMap externalUserMap;
8     private List externalUsers;
9     private int externalClientCount = 0;
10
11    public PingEmulation() throws RemoteException {
12        externalUserMap = new HashMap();
13    }
14
15    public void run() {
16        Object syncObject = new Object();
17        externalUsers = new ArrayList(NUMBER_OF_EXTERNAL_USERS);
18        for(int i=0; i<NUMBER_OF_EXTERNAL_USERS; i++) {
19            externalUsers.add(new ExternalUser(syncObject));
20        }
21        PingEmulationInitializer initializer =
22            new PingEmulationInitializer(VISUALIZE, syncObject, new ArrayList(externalUsers),
23                NUMBER_OF_DEVICES);
24        if (VISUALIZE) {
25            new SimulationFrame(initializer.getSimulationName(),
26                (RenderQueue)initializer.getFrameRenderer(), 10.0);
27        }
28        new ApplicationSimulation(initializer).run();
29    }
30
31...

```

---

Der Konstruktor initialisiert die **●**`HashMap`, in der die registrierten externen Clients verwaltet werden. In der `run`-Methode wird ein Sync-Objekt erzeugt und die



Liste mit **ExternalUser**-Objekten wird angelegt. Der Rest der Methode ist identisch zur Main-Methode der **PingSimulation**. Der **PingEmulationInitializer** ist eine leicht modifizierte Variante des **PingSimulationInitializer**: zusätzlich wird eine Referenz auf das Sync-Objekt sowie auf die Liste mit den **ExternalUser**-Objekten übergeben. Im Vergleich zum Simulationsinitializer ändern sich nur drei Zeilen:

---

```

1simulationName = "ping emulation";
2eventSet = new SynchronizedEventSet(new CascadeEventSet(10, 25), syncObject);
3applicationUserSource = new PingEmulationApplicationUserSource(externalUsers);

```

---

Der Name wird angepaßt und es wird ein **SynchronizedEventSet** in Verbindung mit dem übergebenen Sync-Objekt verwendet, um die Zugriffe auf die FES zu synchronisieren. Außerdem wird eine modifizierte **ApplicationUserSource** verwendet, die **PingEmulationApplicationUserSource**, die die Zuteilung des externen Benutzerverhaltens zu bestimmten Geräten erlaubt. Dabei unterscheidet sie sich nur geringfügig von der originalen **PingApplicationUserSource**: bis zur definierten Anzahl von externen Geräten erhalten die ersten Geräte ein externes Benutzerverhalten, alle übrigen erhalten wie gehabt abwechselnd aktives und passives Benutzerverhalten.

---

```

1public class PingEmulationApplicationUserSource implements ApplicationUserSource {
2    private int count = 0;
3    private List externalUsers;
4    private final static Shape EXTERNAL_IDLE_SHAPE =
5        new EllipseShape(new Position(0,0), new Extent(20,20), Color.RED, true);
6    private final static Shape EXTERNAL_PINGING_SHAPE =
7        new EllipseShape(new Position(0,0), new Extent(20,20), Color.ORANGE, true);
8
9    public PingEmulationApplicationUserSource(List externalUsers) {
10        this.externalUsers = externalUsers;
11    }
12
13    public ApplicationUserPair getApplicationUserPair(double time, Address device) {
14        if (externalUsers.size() > 0) {
15            return new ApplicationUserPair(new PingApplication(EXTERNAL_IDLE_SHAPE,
16                                                                EXTERNAL_PINGING_SHAPE),
17                                         (ExternalUser)externalUsers.remove(0));
18        }
19        if (count++%2 == 0) {
20            return new ApplicationUserPair(new PingApplication(),
21                                         new SimulatedPingUser(true));
22        } else {
23            return new ApplicationUserPair(new PingApplication(),
24                                         new SimulatedPingUser(false));
25        }
26    }
27}

```

---

Die Geräte mit externem Benutzerverhalten werden dabei auch mit abweichenden Shapes initialisiert, so daß sie in der Visualisierung besser zu erkennen sind.

In der Implementierung der **●**PingEmulation (siehe Seite 74) fehlen noch die Verwaltung der externen Clients und die Weiterleitung von Ping-Nachrichten-Sendewünschen an das externe Benutzerverhalten des entsprechenden Geräts.

---

```

1public RemoteClientId registerExternalClient(RemotePingClient client)
2  throws TooManyExternalClientsException {
3  if (externalUsers.size() == 0) {
4    throw new TooManyExternalClientsException();
5  }
6  RemoteClientId clientId = new RemoteClientId(externalClientCount++);
7  ExternalUser externalUser = (ExternalUser) externalUsers.remove(0);
8  externalUserMap.put(clientId, externalUser);
9  externalUser.registerExternalClient(client);
10 return clientId;
11}
12
13public void sendPingMessage(RemoteClientId clientId, Address receiver)
14  throws UnknownRemoteClientIdException {
15  ExternalUser externalUser = (ExternalUser) externalUserMap.get(clientId);
16  if (externalUser == null) {
17    throw new UnknownRemoteClientIdException();
18  }
19  externalUser.sendPingMessage(receiver);
20}

```

---

Wenn sich ein externer Client registriert, so wird ihm – falls noch nicht alle externen Clients belegt sind – ein externes Benutzerverhalten zugewiesen und diese Zuordnung wird in der entsprechenden **●**HashMap vermerkt. Über die zurückgelieferte **●**RemoteClientId lassen sich spätere Aufrufe des externen Clients identifizieren. Dies geschieht in der Beispielanwendung in der Methode sendPingMessage: es wird das entsprechende externe Benutzerverhalten aus der **●**HashMap herausgesucht und der Aufruf wird dorthin delegiert.

Zur Ausführung der Emulation fehlt noch ein einfaches Startprogramm: es startet die Emulation in einem separaten Thread und registriert sie als RMI-Server.

---

```

1public class PingEmulationStarter {
2  public static void main(String[] args) throws RemoteException, MalformedURLException {
3    LocateRegistry.createRegistry(1099);
4    PingEmulation pingEmulation = new PingEmulation();

```

```
5   Thread simRunner = new Thread(pingEmulation);
6   Naming.rebind("PingRMIServer", pingEmulation);
7   simRunner.start();
8 }
9}
```

---

Der Anwendungscode der Ping-Anwendung bleibt auch für die Emulation völlig unverändert. Die beschriebenen, überschaubaren Änderungen betreffen nur das Benutzerverhalten und die Initialisierung der Workbench.

## Graphische Benutzeroberfläche

---

Die graphische Oberfläche der Ping-Anwendung wird in SWT entwickelt, um bei der späteren Übertragung auf echte mobile Geräte (wie z.B. Pocket-PCs) eine effiziente Ausführung zu gewährleisten. Die genaue Implementierung soll hier nicht dargestellt werden, da sie sich nicht von der üblichen Programmierung graphischer Oberflächen unterscheidet. Interessant sind nur der Verbindungsaufbau zur Emulation und die Behandlung der Callbacks, die vom externen Benutzerverhalten in der Emulationsumgebung an die graphische Oberfläche zurückgeliefert werden.

---

```
1try {
2  externalPingUserInterface =
3    (ExternalPingUserInterface) Naming.lookup("//"+hostname+"/PingRMIServer");
4  UnicastRemoteObject.exportObject(this);
5  clientId = externalPingUserInterface.registerExternalClient(this);
6} catch (Exception e) {
7  e.printStackTrace();
8}
```

---

Zur Verbindungsaufnahme mit der Emulationsumgebung wird über den RMI-Namensdienst eine Referenz auf das **①ExternalPingUserInterface** der Emulation der Ping-Anwendung ermittelt und dort die graphische Oberfläche als **①RemotePingClient** registriert. Dies ist möglich, da die Klasse **②PingEmulationClient** dieses Interface implementiert. Gleichzeitig wird der **②PingEmulationClient** für eingehende Remote-Method-Calls freigeschaltet, damit die Callbacks aus dem externen Benutzerverhalten in der Emulation auch in der graphischen Oberfläche ankommen. Außerdem wird die **②RemoteClientId** gespeichert, um sie bei späteren Methodenaufrufen zu nutzen. Die potentiell auftretenden Exceptions werden in der Beispielanwendung der Einfachheit halber ignoriert.

Falls der Benutzer über die graphische Oberfläche eine Ping-Nachricht an ein anderes Gerät senden will, so wird die Referenz auf das **❶**ExternalPingUserInterface in Verbindung mit der gespeicherten **❷**RemoteClientId genutzt, um die Methode sendPingMessage in der Emulation aufzurufen.

---

```

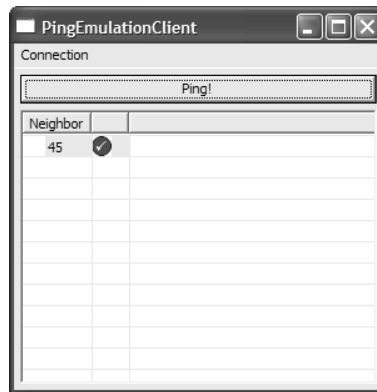
1 try {
2   externalPingUserInterface.sendPingMessage(clientId, receiver);
3 } catch (RemoteException e) {
4   e.printStackTrace();
5 } catch (UnknownRemoteClientIdException e) {
6   e.printStackTrace();
7 }

```

---

## Ausführung der Emulation

---



**Abbildung 4.2:** Graphische Oberfläche des externen Ping-Clients.

Zur Ausführung der Emulation wird zuerst das Startprogramm **❸**PingEmulationStarter ausgeführt. Danach können sich dann die Emulationsclients über RMI verbinden und die entsprechenden mobilen Geräte mit externem Benutzerverhalten versorgen. Der Emulationsserver bietet dabei die Möglichkeit, das Gesamtszenario wie in der Simulationsumgebung zu visualisieren und die Geschwindigkeit der Ausführung zu steuern. Auch die Auswertungsmechanismen aus der Simulationsumgebung stehen unverändert zur Verfügung.

Abbildung 4.2 zeigt die graphische Oberfläche eines mit der Emulationsumgebung verbundenen externen Clients. Zu sehen ist die Liste der Adressen der aktuell bekannten Nachbarn. Falls ein Nachbar erfolgreich „angepingt“ wurde, erscheint in der Liste ein grünes Häkchen.

# 5

## Workbench: Übertragung auf mobile Geräte

---

Die Übertragbarkeit des Applikationscodes aus der Simulation und Emulation auf mobile Geräte wird durch eine spezielle Ausführungsplattform der Workbench sichergestellt. Diese Ausführungsplattform stellt auf den mobilen Geräten Implementierungen der Simulationsabstraktionen zur Verfügung, so daß der Applikationscode auf die gleichen Interfaces wie in der Simulation und Emulation zugreift.

Die Ausführungsplattform muß dazu die Netzwerkkommunikation zwischen den Geräten, die Nachbarschaftserkennung und die Positionsbestimmung sowie das ereignisbasierte Programmiermodell der Simulation zur Verfügung stellen. Aufgrund des Single-Thread-Programmiermodells der Simulationsumgebung muß dabei insbesondere auch die Synchronisation der nebenläufigen Ereignisse wie Empfang von Nachrichten oder Benutzereingaben beachtet werden.

Nachfolgend werden die Implementierungen dieser Abstraktionen für mobile Geräte mit einer Java Virtual Machine<sup>1</sup> vorgestellt.

---

<sup>1</sup>Die derzeitige Implementierung orientiert sich am Umfang der J9 VM von IBM [45], die alle Klassen des JDK 1.1, erweitert um das Collection-Framework und einige andere Klassen aus dem JDK 1.2, bereitstellt.

## 5.1 Operating System

---

Die Betriebssystemimplementierung läuft in einem eigenen Thread ab. Sie verarbeitet zum einen die Ergebnisse der Positionsbestimmung (siehe Abschnitt 5.4), zum anderen wird das Single-Thread-Programmiermodell der Simulation durch eine lokale Ereigniswarteschlange realisiert, auf die die anderen Komponenten synchronisiert zugreifen können. Auf diese Weise können die Komponenten nebenläufig Ereignisse in diese Warteschlange einfügen, die dann vom Operating System ausgeführt werden. Timeouts werden über Java-Timer abgebildet.

## 5.2 Netzwerkimplementierung

---

Die Netzwerkimplementierung basiert auf Wireless LAN (IEEE 802.11b) und realisiert die beiden von der Simulation geforderten Kommunikationsprimitive lokaler Unicast und lokaler Broadcast über UDP. Alle anderen höheren Protokolle der Simulation bzw. der Anwendungen sind mittels dieser Kommunikationsprimitive verwirklicht und können unverändert in der Ausführungsplattform benutzt werden.

In einer lokalen synchronisierten Ereigniswarteschlange werden Sendeereignisse gesammelt. Ein eigener Thread arbeitet die Ereignisse der Netzwerkereigniswarteschlange ab. Dabei werden die zu sendenden Protokollnachrichten mit den entsprechenden Funktionen von Java serialisiert, in UDP-Datagramme verpackt und per UDP-Unicast oder -Broadcast versendet. Eventuell registrierte Observer der Nachricht werden informiert. Die Übertragung der Nachrichten könnte durch die Verwendung einer anderen Art der Serialisierung (z.B. mit Kompression) in der Ausführungsplattform optimiert werden.

Der Empfang von Nachrichten geschieht in einem eigenen Thread, der ständig auf einkommende UDP-Datagramme wartet. Der Inhalt der Datagramme wird deserialisiert und – falls es sich um Protokollnachrichten handelt – als entsprechendes Empfangsereignis in die lokale Ereigniswarteschlange des Operating Systems eingefügt. Auf diese Weise wird die Behandlung der empfangenen Protokollnachricht vom Empfangsthread der Netzwerkimplementierung entkoppelt.

## 5.3 Nachbarschaftserkennung

---

Die Erkennung neu hinzukommender oder außer Reichweite geratener benachbarter Geräte wird ebenfalls von der Netzwerkimplementierung übernommen. In festen Abständen werden dazu *Beacon-Nachrichten* per lokalem Broadcast verschickt, die Informationen über das sendende Gerät wie die aktuelle Position und Bewegungsrichtung enthalten. Der Empfang einer solchen Beacon-Nachricht liefert dem empfangenden Gerät neben den Positionsinformationen des anderen Gerätes auch die Information, daß eine unidirektionale Verbindung zwischen den beiden Geräten besteht. Damit auch bidirektionale Verbindungen erkannt werden können, antwortet das empfangende Gerät mit einer Unicastnachricht auf die Beacon-Nachricht.

Informationen über benachbarte Geräte werden zwischengespeichert und der Applikation zur Verfügung gestellt. Die gespeicherten Informationen werden regelmäßig auf Aktualität geprüft: hat ein Gerät während einer bestimmten Anzahl von Beaconintervallen kein neues Beacon von einem Gerät empfangen, so wird die entsprechende Information gelöscht und die Verbindung als abgebrochen betrachtet.

Im Fall einer Implementierung der Ausführungsplattform für andere Funktechnologien wie z.B. Bluetooth, die eine geeignete Form von Device Discovery direkt bereitstellen, kann auf das beschriebene Beaconing verzichtet und stattdessen direkt die „eingebaute“ Device Discovery der Funktechnologie genutzt werden.

## 5.4 Positionsbestimmung

---

Die Positionsbestimmung der mobilen Geräte erfolgt über das *Global Positioning System* (GPS) [51]. Die Geräte nutzen einen GPS-Empfänger, um ihre Position kontinuierlich zu erfassen. Prinzipiell ist auch die Verwendung anderer Positionsbestimmungssysteme [11, 37] möglich, allerdings ist die derzeitige Implementierung der Ausführungsplattform auf GPS ausgerichtet. Problematisch an GPS ist, daß es nur im Freien funktioniert und damit die Ausführungsplattform in der derzeitigen Form innerhalb von Gebäuden nicht genutzt werden kann, wenn Positionsinformationen benötigt werden.

Ein eigener Thread parst die vom GPS-Empfänger gelieferten NMEA-Nachrichten [67]. Die Workbench arbeitet mit zweidimensionalen kartesischen Koordinaten, so

daß die aus den NMEA-Nachrichten extrahierten sphärischen Koordinaten (geographische Länge und Breite, Höhe) transformiert werden müssen. Dies geschieht mit Hilfe der Gauß-Krüger-Projektion [92], wobei der Nullpunkt ungefähr auf die eigene Position verschoben wird. Auf diese Weise liegen die ermittelten Positionen immer im gleichen Meridianstreifen und lassen sich entsprechend einfach vergleichen. Die transformierten Ortsinformationen werden dann an das Operating System geliefert, wo die jeweils letzte Position gespeichert wird, um auch bei fehlendem Satellitenkontakt kurzfristig noch Positionsinformationen liefern zu können und die Berechnung der Bewegungsrichtung zu ermöglichen. Anstatt bei fehlendem Satellitenkontakt immer nur die letzte bekannte Position zurückzugeben, wäre es auch möglich, die zuletzt bekannte Bewegungsrichtung und die Dauer des fehlenden Kontakts bei der Berechnung zu berücksichtigen.

## 5.5 Graphische Oberfläche

---

Die graphische Oberfläche für die Applikation auf der Ausführungsplattform kann aus der Emulation übernommen werden. An die Stelle der RMI-Aufrufe zum Emulationsserver treten in der Version für die mobilen Geräte direkte Aufrufe im „externen“ Benutzerverhalten. Diese Änderung ist bei entsprechender Vorbereitung des Codes leicht durchführbar. Weitere Änderungen sind nicht notwendig, allerdings kann es sinnvoll sein, die Oberfläche um Konfigurationsmöglichkeiten für die Ausführungsplattform zu erweitern. Durch die Wiederverwendung der graphischen Oberfläche entfällt der sonst notwendige Entwicklungsaufwand, und es kann auf bereits getesteten Code zurückgegriffen werden.

Prinzipiell ist es natürlich auch möglich, weiterhin simuliertes Benutzerverhalten auf der Ausführungsplattform zu verwenden. Dies kann genutzt werden, um bestimmte Situationen einfacher nachstellen zu können und aus der Simulation bekannte Phänomene nachzuvollziehen.

## 5.6 Ausführung einer Beispielanwendung

---

Die graphische Oberfläche der externen Ping-Clients aus der Emulationsumgebung kann fast unverändert in die Ausführungsplattform übernommen werden. Es sind



nur zwei Veränderungen notwendig: die Ausführungsplattform muß beim Applikationsstart initialisiert werden und die Methoden zum RMI-Verbindungsaufbau müssen entfernt werden.

Die Initialisierung der Ausführungsplattform geschieht im Konstruktor der  Ping-App.


---

```

1Console console = new SystemOutConsole();
2DistributionCreator dist =
3  new DistributionCreator(new long[] { 1231, 6473, 69543, 56887, 4663783, 547894,
4                                     3654234, 958544, 4342, 4648501 });
5OperatingSystem operatingSystem = new OperatingSystem(dist, console);
6SimpleNetwork network = null;
7try {
8  network = new SimpleNetwork(operatingSystem, 10034, 10035, 3.0);
9} catch (SocketException e) {
10 console.println("Ex: "+e.getMessage());
11 return;
12}
13operatingSystem.setNetwork(network);
14Application application = new PingApplication();
15ProtocolCollection protocolCollection = new ProtocolCollection(operatingSystem);
16protocolCollection.initialize(application, operatingSystem, network);
17network.initialize(protocolCollection);
18network.start();
19externalUser = new ExternalUser(new Object());
20externalUser.start(new UserEnvironment(application, operatingSystem));
21externalUser.registerExternalClient(this);
22application.start(new OperatingComponents(operatingSystem,
23                                           protocolCollection.getSendDispatcher(),
24                                           new UserSystem(externalUser)), null);
25operatingSystem.start();

```

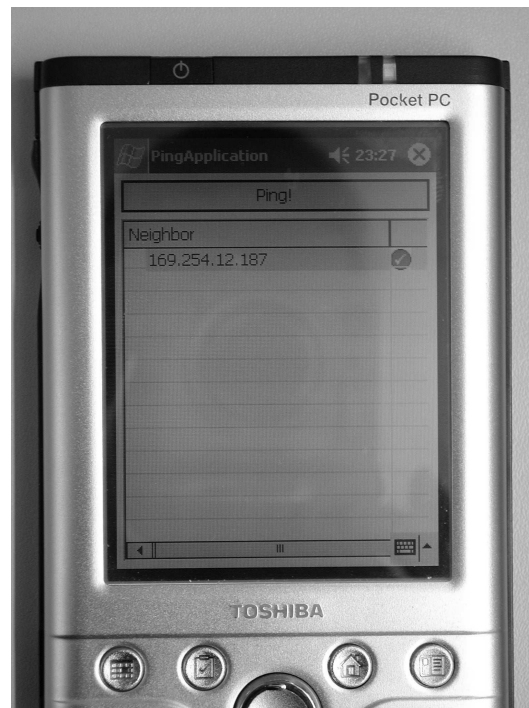
---

Im Gegensatz zur Simulations- und Emulationsumgebung sind Komponenten wie das  OperatingSystem in der Ausführungsplattform nicht automatisch initialisiert. Die Netzwerkimplementierung der Ausführungsplattform sendet auf UDP-Port 10034 und empfängt auf UDP-Port 10035. Alle drei Sekunden wird ein Beacon zur Erkennung benachbarter Geräte versendet. Die Zeilen 19 bis 21 zeigen, daß das externe Benutzerverhalten aus der Emulationsumgebung unverändert übernommen werden kann, indem die Anwendung als „externer“ Client registriert wird. Dadurch ruft das externe Benutzerverhalten beim Eintreffen von Nachrichten automatisch die richtigen Methoden auf.

Die Referenz auf das externe Benutzerverhalten wird benutzt, um Ping-Nachrichten an andere Geräte zu senden. Sie tritt damit an die Stelle der Referenz auf den



**Abbildung 5.1:** Praktische Erprobung der Ping-Anwendung auf zwei Pocket-PCs.



**Abbildung 5.2:** Ausführung der Ping-Anwendung auf einem Pocket-PC.

RMI-Server. Zur Ausführung der Applikation muß man eine geeignete Java-Implementierung auf dem Pocket-PC installieren, z.B. die J9 VM von IBM. Der Code für die Ping-Anwendung, die Ping-App sowie der Code für die Ausführungsplattform werden auf den Pocket-PC transferiert und dort gestartet. Abbildung 5.1 zeigt die

---

praktische Erprobung der Anwendung mit zwei Geräten; in Abbildung 5.2 ist ein Screenshot der Anwendung auf dem Pocket-PC zu sehen.



# 6

## Fallstudie: UbiBay

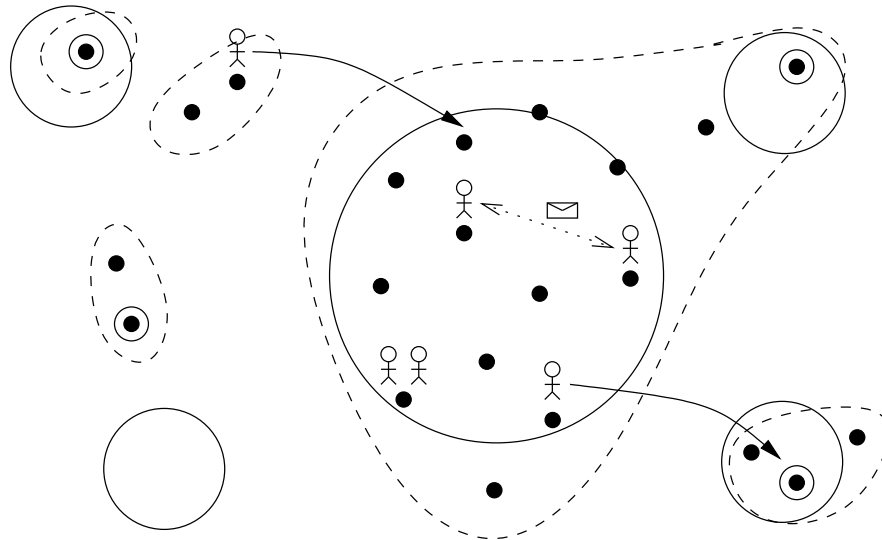
---

Die in dieser Arbeit vorgestellte einheitliche Workbench und der mit der Workbench verbundene Entwicklungsprozeß sind in einer Fallstudie anhand von UbiBay, einer selbstorganisierenden Auktionsanwendung für mobile multihop Ad-Hoc-Netzwerke, erprobt worden. Die folgenden Abschnitte erläutern die Middleware *SELMA* (*Self-organized Marketplace-based Middleware for Mobile Ad-hoc Networks*) [36] für mobile multihop Ad-Hoc-Netzwerke, die Anwendung UbiBay, die basierend auf SELMA entwickelt wurde, und gehen auf die dabei gewonnenen Erfahrungen mit der Workbench ein.

### 6.1 SELMA – Middleware für Ad-Hoc-Netze

---

Das Marktplatzkommunikationspattern [35] ist ein Ansatz, ein Grundproblem von Anwendungen in mobilen multihop Ad-Hoc-Netzen zu lösen: wie können sich Anbieter und Kunden – im Fall einer Auktion also Auktionatoren und Bieter – im Netzwerk finden und effizient miteinander kommunizieren? Die Grundidee besteht darin, zu diesem Zweck – wie in der Realität – „Marktplätze“ einzusetzen. Ein *Marktplatz* im Sinne des Marktplatzkommunikationspatterns ist ein Ort, an dem eine hohe Dichte von mobilen Geräten herrscht, und der in seiner Ausdehnung klar begrenzt

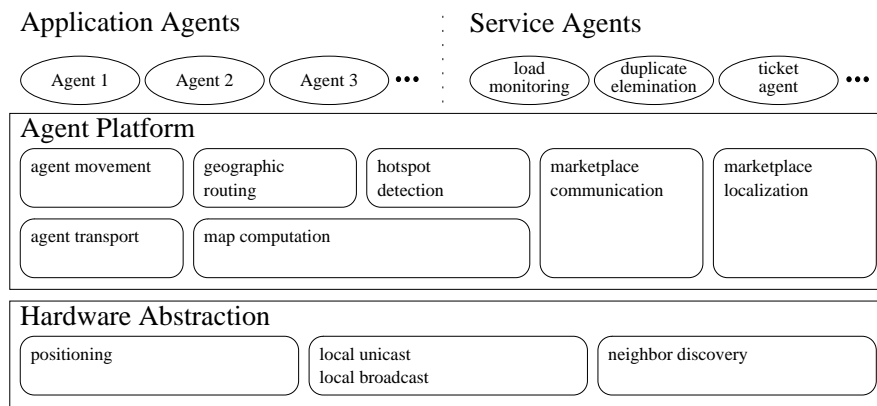


**Abbildung 6.1:** Ein einfaches Marktplatzbeispiel mit einem Marktplatz im Zentrum und vier Homezones von Benutzern. Ein Stellvertreter versucht den Marktplatz zu erreichen während ein anderer auf dem Weg zu seiner Homezone ist. Zwei Stellvertreter verhandeln und kommunizieren gerade miteinander. Die gestrichelte Linie markiert die aktuellen Netzwerkpartitionen.

ist (siehe Abbildung 6.1). Die mobilen Geräte kommunizieren nicht direkt miteinander, sondern senden stattdessen Stellvertreter<sup>1</sup> per geographischem Routing zum Marktplatz.

Auf dem Marktplatz sammeln sich somit Stellvertreter von Anbietern und Kunden, die nun – aufgrund der begrenzten Ausdehnung des Marktplatzes und der höheren Gerätedichte – effizient und über wenige Zwischenstationen miteinander kommunizieren und verhandeln können. Durch die Abgrenzung des Marktplatzes wird das Gesamtnetzwerk durch die Marktplatzkommunikation nicht zusätzlich belastet. Anstelle von simplem Fluten wird innerhalb des Marktplatzes ein *Neighbor Knowledge Broadcast* [93] für die Kommunikation zwischen den Stellvertretern genutzt. Zusätzlich können die Geräte auf dem Marktplatz für die Unicast-Kommunikation topologiebasiertes Routing einsetzen. Nach Ende der Verhandlung zwischen den Stellvertretern kehren diese per geographischem Routing zu ihrem Eigentümer zurück. Dabei finden sie das mobile Gerät ihres Eigentümers, indem sie dessen *Homezone*

<sup>1</sup>Die Implementierung dieser Stellvertreter hängt von einer grundlegenden Entscheidung ab: falls die benötigte Software auf allen mobilen Geräten installiert ist, genügen einfache Nachrichten, die von der Software auf den Geräten entsprechend behandelt wird. Andernfalls kann man auf mobile Agenten zurückgreifen und mit ihrer Hilfe den benötigten Code direkt zum Marktplatz verschicken.



**Abbildung 6.2:** Architektur von SELMA, einer Middleware für mobile multihop Ad-Hoc-Netzwerke.

ansteuern, d.h. einen Bereich, in dem sich der Eigentümer relativ regelmäßig aufhält (z.B. Haus, Arbeitsplatz etc.).

SELMA faßt die zur Realisierung des Marktplatzkommunikationspatterns notwendigen Basisdienste mit weiteren, oft benötigten Diensten zu einer Middleware für mobile multihop Ad-Hoc-Netzwerke zusammen. Abbildung 6.2 zeigt den Aufbau von SELMA.

Mittels einer Abstraktionsschicht wird die eigentliche Gerätehardware vor der Anwendung und den Middlwarediensten verborgen. Diese Schicht stellt die beiden grundlegenden Kommunikationsprimitive, lokaler Unicast und lokaler Broadcast, die Erkennung benachbarter Geräte, sowie die Möglichkeit der Positionsbestimmung zur Verfügung.

Darauf aufbauend gibt es mit *Agent movement*, *Agent transport* und *Geographic routing* diejenigen Dienste, die die Bewegung eines Agenten bzw. Stellvertreters zwischen zwei benachbarten Geräten und hin zu einem bestimmten Ort im Ad-Hoc-Netzwerk ermöglichen. Die *Hotspot detection* erlaubt es, aufbauend auf einer verteilt erstellten Karte mit Gerätehäufigkeiten (geliefert von *Map computation*), Orte im Netzwerk zu finden, die sich aufgrund der dort vorherrschenden Gerätedichte besonders gut als Marktplätze eignen. Die *Marketplace communication* stellt die Kommunikationsverfahren der Stellvertreter auf dem Marktplatz bereit und *Marketplace localization* ermöglicht es mobilen Geräten, Marktplätze im Ad-Hoc-Netzwerk zu finden.

Alle diese Dienste können über die Agentenplattform genutzt werden. Hier können applikationsspezifische Agenten implementiert werden. Zusätzlich gibt es noch die

Möglichkeit, Serviceagenten z.B. auf Marktplätzen zu nutzen. Diese Serviceagenten haben keinen Eigentümer, bleiben auf dem Marktplatz und erfüllen dort Serviceaufgaben wie z.B. die Eliminierung von Agentenduplikaten, die beim Senden der Agenten zum Marktplatz entstehen können. Ausfallende Serviceagenten werden von der Middleware eines beliebigen Geräts, die das Fehlen des entsprechenden Serviceagenten bemerkt, neu erzeugt.

## 6.2 UbiBay

---

UbiBay realisiert eine Auktionsplattform für mobile multihop Ad-Hoc-Netzwerke, die auf dem Prinzip der Selbstorganisation aufbaut und völlig ohne zentrale Infrastruktur auskommt.

Ausgehend von einer frühen ersten Version [33], die noch auf einem Hintergrundinformationsverbreitungsdienst basierte, wurde UbiBay konzeptuell ständig weiterentwickelt und nutzt in der der Fallstudie zugrundeliegenden Version [30, 31] die oben beschriebene Middleware SELMA und das ihr zugrundeliegende Marktplatzkommunikationspattern.

Die Auktionsplattform wird durch drei verschiedene Arten von Agenten realisiert: Auktions-, Biet- und Informationsagenten. Ein Benutzer, der eine Auktion starten möchte, sendet einen Auktionsagenten mit einer Beschreibung der Auktion, einem Mindestgebot und einer Laufzeit zum Marktplatz. Andere Benutzer können jederzeit Informationsagenten starten und diese zum Marktplatz schicken. Dort erfragen sie bei den anwesenden Auktionsagenten den Stand der aktuell laufenden Auktionen und überbringen diese Informationen an den Benutzer. Falls er auf einer Auktion bieten möchte, startet er einen Bietagenten für die betreffende Auktion, gibt diesem ein Höchstgebot mit und schickt ihn zum Marktplatz. Auf dem Marktplatz verhandelt dieser Bietagent solange mit dem entsprechenden Auktionsagenten, bis er überboten wird oder die Auktionszeit abgelaufen ist. Er kehrt dann mit dem Auktionsergebnis zurück zum Benutzer.

UbiBay tritt nicht an, um mit herkömmlichen Auktionsplattformen im Internet zu konkurrieren oder diese zu ersetzen. Vielmehr ist UbiBay zur Versteigerung geringwertiger Dinge in einem relativ begrenzten Bereich wie z.B. einem Universitätscampus, einer Firma oder der Fußgängerzone einer Stadt gedacht. Da sich die Auktionsteilnehmer recht einfach persönlich treffen können, entfällt der bei Internet-



auktionen notwendige Aufwand für Verpackung und Versand. Dies ermöglicht die Versteigerung auch von Kleinigkeiten für wenige Euro. Die begrenzten Nutzerkreise z.B. auf einem Universitätscampus haben zudem den Vorteil, daß die Wahrscheinlichkeit steigt, Kaufinteressenten zu finden (gleiche Interessenslage, gleiches Alter etc.). Selbst wenn sich die Auktionsteilnehmer nicht persönlich zur Übergabe der versteigerten Sachen treffen können oder wollen, wäre es denkbar, daß z.B. Supermärkte oder andere gut besuchte Orte eine Art Treuhänderfunktion übernehmen.

## 6.3 Erfahrungen

---

Aufbauend auf einer bestehenden Implementierung von SELMA wurde auch die Anwendung UbiBay für den Simulationsteil der Workbench implementiert. Basierend auf einem einfachen simulierten Benutzerverhalten, bei dem ein gewisser Teil der Benutzer regelmäßig Artikel aus einer festen Artikelbibliothek versteigert und alle Benutzer mit einer gewissen Wahrscheinlichkeit an Auktionen teilnehmen und auf Artikel bieten, konnte die Anwendung ersten Tests unterzogen werden. Dabei wurden die Visualisierungsmöglichkeiten (siehe auch Abbildung 3.6) und das Statistiksystem der Simulationsumgebung genutzt, um das Verhalten der Anwendung besser zu verstehen.

Dieser erste Entwicklungsschritt wurde genutzt, um eine in der Simulation funktionierende Auktionsanwendung herzustellen. Darauf aufbauend wurde in einem zweiten Schritt eine graphische Oberfläche für UbiBay entwickelt, um die Anwendung in der hybriden Emulationsumgebung der Workbench mit einer Kombination von realem und simuliertem Benutzerverhalten zu testen. Der restliche Programmcode mußte für die Ausführung in hybriden Emulationsumgebung nicht verändert werden, insbesondere die Anwendungslogik blieb völlig unverändert.

Abbildung 6.3 zeigt einen Screenshot der graphischen Oberfläche von UbiBay. Der Benutzer kann neue Auktionen starten, Informationsagenten senden und auf Auktionen bieten. Der Stand der aktuellen Auktionen wird in einer Tabelle ständig aktualisiert; so werden Auktionen, die abgelaufen sind und an denen der Benutzer nicht beteiligt war, aus der Liste entfernt. Einziges „Zugeständnis“ an die Ausführung in der hybriden Emulationsumgebung ist das Menü „Connection“, mit dessen Hilfe sich der Benutzer per RMI mit dem Emulationsserver verbinden kann.



**Abbildung 6.3:** Screenshot der graphischen Oberfläche von UbiBay.

Die graphische Oberfläche wurde mit SWT realisiert. Dieses Widget-Set bietet erheblich mehr Möglichkeiten als AWT, eignet sich aber im Gegensatz zu Swing und anderen Alternativen auch zur Ausführung auf mobilen Geräten (z.B. in Verbindung mit der IBM J9 Java VM auf gängigen PocketPCs wie dem HP iPAQ 5450). Da die Oberfläche später auch bei der Ausführung der Anwendung auf realen Geräten verwendet werden soll, bot sich die Verwendung von SWT auch für die graphische Oberfläche in der Emulationsumgebung an.

Die Tests von UbiBay in der hybriden Emulationsumgebung unter Nutzung von Arbeitsplatzrechnern in einem herkömmlichen stationären Netzwerk erwiesen sich als sehr hilfreich. Aufgrund des simulierten Benutzerverhaltens fallen Unstimmigkeiten und kleinere Fehler in der Simulationsumgebung der Workbench nicht direkt auf. Bedient man hingegen die emulierte Anwendung über die graphische Benutzeroberfläche und interagiert mit anderen menschlichen Benutzern, so stellen sich Unstimmigkeiten oder Fehler in der Anwendung sehr schnell heraus. Gleichzeitig bemerkt man in dieser Phase auch „fehlende“ Funktionen, die für einen realen Betrieb der Anwendung notwendig sind, in der Simulation der Anwendung aber nicht als fehlend auffallen. Beispielsweise lieferte eine frühe Version bei einem Gebot den Namen des Bieters nicht mit. In der Simulation war dies unproblematisch, da die Gebote über eindeutige IDs unterschieden werden konnten. Zur wirklichen Nutzung der Anwendung ist es aber sehr wichtig, daß der Name des Bieters bekannt ist. Ein weiterer Fehler war, daß im Falle des Überbietens zwar die Information über den

Vorgang an sich an den Überbotenen zurückgeliefert wurde, nicht aber die Höhe des aktuellen Gebots. Zwar könnte ein Benutzer diese Informationen mittels eines Informationsagenten beschaffen, allerdings ist es in der Praxis viel zweckmäßiger, diese Information direkt mitzuliefern. Nur durch die Interaktion mit der Anwendung bekommt man als Anwendungsentwickler ein richtiges Gefühl für die Anwendung. Die Kombination von simuliertem und echtem Benutzerverhalten erlaubt es, verschiedenste Anwendungssituationen gezielt herbeizuführen.

Die auf diese Weise fehlerbereinigte und weiter getestete Anwendung wurde in einem letzten Entwicklungsschritt auf die Ausführungsplattform für reale Geräte der Workbench übertragen. Dabei wurden PocketPCs mit Microsoft Pocket PC 2002, GPS Empfänger und die IBM J9 Java VM als Plattform genutzt. Aufgrund der einheitlichen APIs in allen drei Teilen der Workbench konnte der Anwendungscode unverändert auf die Ausführungsplattform übertragen werden. Nur die graphische Oberfläche mußte leicht angepaßt werden. Die RMI-Zugriffe wurden durch direkte Zugriffe auf den entsprechenden Anwendungscode ersetzt und die GUI-Elemente an die kleineren Bildschirmdimensionen der mobilen Geräte angepaßt. Diese Modifikationen erwiesen sich allerdings als problemlos und in kurzer Zeit durchführbar.

Aufgrund der begrenzten Gerätezahl, die in der Arbeitsgruppe zur Verfügung steht, konnten keine großen Feldversuche durchgeführt werden. Allerdings konnte das prinzipielle Funktionieren der Anwendung bei Versuchen mit einigen wenigen Geräten erfolgreich überprüft werden.



# 7

## Fazit

---

Diese Arbeit führt eine einheitliche Workbench zur Entwicklung von Applikationen in mobilen multihop Ad-Hoc-Netzwerken ein. Das Design und die Architektur der dreigeteilten, auf einer Simulations-, einer Emulations- und einer Ausführungsumgebung basierenden Workbench werden erläutert und evaluiert. Begleitend dazu wird ein dreistufiger Entwicklungsprozeß, der nacheinander auf die einzelnen Teile der Workbench aufbaut, eingeführt.

Erfahrungen mit dem Entwicklungsprozeß und der einheitlichen Workbench werden exemplarisch in zwei Fällen dargestellt. Eine einfache Beispielanwendung wird Schritt für Schritt in den drei Umgebungen der Workbench umgesetzt. Weiterhin werden die Erfahrungen bei der Umsetzung einer Auktionsanwendung für mobile multihop Ad-Hoc-Netze in einer Fallstudie erläutert. In beiden Fällen zeigt sich, daß die Verwendung der Workbench den Entwicklungsprozeß vereinfacht und die Konzentration des Entwicklers nicht von der Anwendung abgelenkt wird. Dank der Workbench lassen sich Anwendungsszenarien einfach visualisieren und evaluieren. Der dabei entstehende Applikationscode läßt sich mit geringen Änderungen in der Emulationsumgebung der Workbench erproben und mit echtem Benutzerverhalten verbinden. Auch die Übertragung auf mobile Geräte ist aufgrund der Ausführungsplattform mit nahezu identischem Code möglich.

Die Simulationsumgebung der Workbench beschränkt sich auf die topologischen Eigenschaften der simulierten Netze, bietet im Gegenzug aber eine sehr hohe Ska-

lierbarkeit und ein hohes Abstraktionsniveau. Die Simulation tausender mobiler Geräte ist schneller als in Echtzeit möglich. Das hohe Abstraktionsniveau erlaubt eine komfortable Anwendungsentwicklung und die einfache spätere Übernahme des entstehenden Anwendungscodes auf mobile Geräte. Gleichzeitig wird die Evaluation der simulierten Anwendung über vielfältige Visualisierungsmöglichkeiten und komfortable Statistikfunktionen erleichtert. Das modulare Design der Simulationsumgebung erlaubt eine einfache Austauschbarkeit und Erweiterbarkeit einzelner Komponenten. Die hohe Skalierbarkeit wird u.a. durch eine spezielle Bewegungsmodellabstraktion, die Gerätebewegungen über eine Folge von Geradenteilstücken approximiert, und eine dafür entwickelte, besonders effiziente Konnektivitätsberechnung möglich. Aufgrund dieser Art der Konnektivitätsberechnung können alle Konnektivitätsereignisse für ein Geradenteilstück auf einen Schlag berechnet und so die Zahl der notwendigen Konnektivitätsberechnungen stark verringert werden. Gleichzeitig erlaubt die Bewegungsmodellabstraktion die problemlose Einbindung externer Bewegungsmodellgeneratoren sowie die Vorausberechnung und Wiederverwendung aller Bewegungs- und Konnektivitätsereignisse.

Mit diesem Ansatz unterscheidet sich die Simulationsumgebung deutlich von traditionellen Netzwerksimulatoren wie ns-2, die auf eine möglichst exakte Simulation aller Netzwerkschichten setzen und nicht auf die Simulation ganzer Anwendungen abzielen. Die Erfahrung in der Entwicklung kleiner Prototypenanwendungen und in der Fallstudie „UbiBay“ zeigen, daß der Workbenchansatz zur Entwicklung von Anwendungen in Ad-Hoc-Netzen sinnvoll ist. Hier sind aufgrund der guten Skalierbarkeit und des hohen Abstraktionsniveaus schnell erste Ergebnisse sichtbar; das Gesamtverhalten der Anwendung kann vielfältig visualisiert und Fehler können somit schneller gefunden werden.

Auch in der Evaluation von Routingalgorithmen hat sich die Simulationsumgebung der Workbench inzwischen bewährt. Über einen speziellen Layer, der auf dem Applikationsinterface der Simulationsumgebung aufsetzt, wird die Simulation dieser Algorithmen erleichtert. Hannes Frey hat diese Umgebung erfolgreich zur Simulation einer Vielzahl bereits existierender Routingalgorithmen sowie zur Entwicklung eines neuen Routingalgorithmus genutzt [28, 29]. Auch in diesem Fall war die hohe Skalierbarkeit und die komfortable Umgebung insbesondere hinsichtlich der Visualisierung viel wichtiger als die detaillierte Simulation aller Netzwerklayer.

Steffen Rothkugel und seine Mitarbeiter setzen die Workbench zur Entwicklung

---

von Anwendungen für mobile multihop Ad-Hoc-Netzwerke an der Universität du Luxembourg im Projekt SONI (Self Organising Network Infrastructure) [78] ein.

Dennoch gibt es Situationen, in denen man auf die traditionellen Simulatoren zurückgreifen muß. Falls eine möglichst genaue Simulation aller Netzwerkschichten zur Evaluation eines Algorithmus' notwendig ist, so kann (und will) die Workbench dies nicht leisten. In diesem Fall muß der Entwickler einen der traditionellen Simulatoren nutzen.

Die Emulationsumgebung erlaubt, ohne aufwendige Feldversuche erste praktische Erfahrungen mit einer neu entwickelten Anwendung zu sammeln. Versuchspersonen können die Anwendung über eine graphische Oberfläche von per RMI angebotenen Clients steuern und so echtes Benutzerverhalten liefern, während die notwendige kritische Masse an mobilen Geräten in der Emulationsumgebung mittels simuliertem Benutzerverhalten realisiert wird. Dies läßt eine einfache Evaluierung der Anwendung zu. Die Erfahrung aus der Fallstudie zeigt, daß gerade dieser Schritt viele Fehler noch vor dem aufwendigen Feldversuch offenlegen kann. Weiterhin erlaubt die Visualisierungskomponente der Emulationsumgebung, den Zustand der Gesamtanwendung auch bei Anbindung externer Clients komfortabel darzustellen.

Das Risiko scheiternder Feldversuche wird durch die Einheitlichkeit der Workbench reduziert, da Feldversuche stets auf in der Emulationsumgebung getestetem Code basieren.

Java als Implementierungssprache und -umgebung hat sich als sehr vorteilhaft erwiesen. Es erlaubt ein modulares, objektorientiertes Design und die mächtigen, mitgelieferten Bibliotheken erlauben eine einfache und schnelle Umsetzung komplexer Komponenten wie z.B. der Visualisierung. Die Ausführungsgeschwindigkeit der Simulations- und Emulationsumgebung ist unproblematisch, die Optimierungen moderner Java-Virtual-Machines erlauben eine sehr effiziente Ausführung. Zudem hat sich gezeigt, daß – wie so oft – prinzipielle Änderungen an den Basisalgorithmen, beispielsweise in der Konnektivitätsberechnung, einen um Größenordnungen höheren Einfluß auf die Ausführungsgeschwindigkeit haben als die Wahl der Implementierungssprache. Gleichzeitig existiert dank der Plattformunabhängigkeit von Java die gleiche Umgebung auf den mobilen Geräten. Bei Verwendung geeigneter Tools (SWT als Widget-Set) und einer geeigneten Java VM (IBM J9) ist die Geschwindigkeit der Ausführungsplattform auf aktuellen Pocket-PCs ebenfalls völlig ausreichend.



Zusammenfassend läßt sich sagen, daß die einheitliche Workbench ein mächtiges und effizientes Werkzeug zur Entwicklung von Anwendungen in mobilen Ad-Hoc-Netzwerken bereitstellt, das die schnelle Erprobung und Umsetzung von Ideen ermöglicht.



# Literaturverzeichnis

---



- [1] G. ANASTASI und A. PASSARELLA: *Towards a Novel Transport Protocol for Ad hoc Networks*. In: *Proc. of the 8th IFIP International Conference on Personal Wireless Communications (PWC 2003)*, Venice, Italy, September 2003.
- [2] KEN ARNOLD und JAMES GOSLING: *The Java Programming Language*. Addison-Wesley, Reading, MA, USA, 2. Auflage, 1998.
- [3] R. BAGRODIA, R. MEYER, M. TAKAI, Y. CHEN, X. ZENG, J. MARTIN, B. PARK und H. SONG: *Parsec: A Parallel Simulation Environment for Complex Systems*. *IEEE Computer*, 31(10):77–85, Oktober 1998.
- [4] HARI BALAKRISHNAN, VENKATA N. PADMANABHAN, SRINIVASAN SESHAN und RANDY H. KATZ: *A comparison of mechanisms for improving TCP performance over wireless links*. *IEEE/ACM Transactions on Networking*, 5(6):756–769, 1997.
- [5] STEFANO BASAGNI, IMRICH CHLAMTAC, VIOLET R. SYROTIUK und BARRY A. WOODWARD: *A distance routing effect algorithm for mobility (DREAM)*. In: *Proceedings of the fourth annual ACM/IEEE international conference on Mobile computing and networking*, Seiten 76–84. ACM Press, 1998.
- [6] CHRISTIAN BETTSTETTER: *Smooth is Better than Sharp: A Random Mobility Model for Simulation of Wireless Networks*. In: *Proceedings of the 4th ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, Seiten 19–27. ACM Press, 2001.





- [7] JAN BLUMENTHAL: *Middleware für mobile spontan vernetzte Sensornetzwerke*, 2002.  [http://rtl.e-technik.uni-rostock.de/~bj/publications/2002\\_10\\_25\\_-\\_DFG\\_Basissoftware\\_Startkolloquium.pdf](http://rtl.e-technik.uni-rostock.de/~bj/publications/2002_10_25_-_DFG_Basissoftware_Startkolloquium.pdf).
- [8] PROSENJIT BOSE, PAT MORIN, IVAN STOJMENOVIC und JORGE URRUTIA: *Routing with Guaranteed Delivery in Ad Hoc Wireless Networks*. *Wireless Networks*, 7(6):609–616, 2001.
- [9] JOSH BROCH, DAVID A. MALTZ, DAVID B. JOHNSON, YIH-CHUN HU und JORJETA JETCHEVA: *A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols*. In: *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM-98)*, Seiten 85–97, New York, Oktober 25–30 1998. ACM Press.
- [10] N. BROWN und C. KINDEL: *Distributed Component Object Model Protocol - DCOM/1.0*. Internet draft, January 1998.  [www.globecom.net/ietf/draft/draft-brown-dcom-v1-spec-03.html](http://www.globecom.net/ietf/draft/draft-brown-dcom-v1-spec-03.html).
- [11] NIRUPAMA BULUSU, JOHN HEIDEMANN und DEBORAH ESTRIN: *GPS-less Low Cost Outdoor Localization For Very Small Devices*. *IEEE Personal Communications Magazine*, 7(5):28–34, October 2000.
- [12] LEVENTE BUTTYÁN und JEAN-PIERRE HUBAUX: *Enforcing service availability in mobile ad-hoc WANS*. In: *Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, Seiten 87–96. IEEE Press, 2000.
- [13] T. CAMP, J. BOLENG und V. DAVIES: *A Survey of Mobility Models for Ad Hoc Network Research*. *Wireless Communications and Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.
- [14] SRDJAN CAPKUN, JEAN-PIERRE HUBAUX und LEVENTE BUTTYÁN: *Mobility helps security in ad hoc networks*. In: *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, Seiten 46–56. ACM Press, 2003.
- [15] DAVID CAVIN, YOAV SASSON und ANDRÉ SCHIPER: *On the accuracy of MANET simulators*. In: *Proceedings of the Second ACM International Workshop*







- on Principles of Mobile Computing (POMC-02)*, Seiten 38–43, New York, Oktober 30–31 2002. ACM Press.
- [16] WILLIAM CHAO, JOSEPH P. MACKER und JEFFERY W. WESTON: *NRL Mobile Network Emulator*. Technischer Bericht FR-10054, Naval Research Laboratory, Washington, DC, USA, 2003.
- [17] CMU MONARCH PROJECT: *ad-hockey and ns FAQ*, 2000. 🌐 <http://www.monarch.cs.cmu.edu/ns-faq/ns-faq/faq.html>.
- [18] JAMES H. COWIE: *Scalable Simulation Framework API Reference Manual*, März 1999. 🌐 <http://www.ssfnet.org/SSFdocs/ssfapiManual.pdf>.
- [19] DAIMLER-CHRYSLER AG: *FleetNet – Internet on the Road*, Januar 2002. 🌐 <http://www.fleetnet.de/>.
- [20] DARTMOUTH COLLEGE: *SWAN: Simulator for Wireless Ad Hoc Networks*, 2003. 🌐 <http://www.cs.dartmouth.edu/research/SWAN/>.
- [21] DFG SPP 1140, Meeting zum Thema NS-2 Simulator, 13. April 2003. 🌐 <http://www.tm.uka.de/forschung/SPP1140/meetings/ns2/veranstaltungen.html>.
- [22] ECLIPSE PROJECT - UNIVERSAL TOOL PLATFORM: *SWT: Standard Widget Toolkit*, 2003. 🌐 <http://www.eclipse.org/platform/index.html>.
- [23] ETSI: *Universal Mobile Telecommunications System (UMTS): Selection procedures for the choice of radio transmission technologies of the UMTS (UMTS 30.03 version 3.2.0)*. Technischer Bericht TR 101 112 V3.2.0, European Telecommunications Standards Institute, April 1998.
- [24] KEVIN FALL und KANNAN VARADHAN: *The ns Manual*. The VINT Project – A collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC, 1989-2003.
- [25] KLAUS FINKENZELLER: *RFID-Handbuch*. Carl Hanser Verlag München, 3. Auflage, September 2002.
- [26] HOLGER FÜSSLER, JÖRG WIDMER, MICHAEL KÄSEMANN, MARTIN MAUVE und HANNES HARTENSTEIN: *Beaconless Position-Based Routing for Mobile Ad-Hoc Networks*. Technischer Bericht TR-03-001, Universität Mannheim, 2003.

- [27] JUAN FLYNN, HITESH TEWARI und DONAL O' MAHONY: *JEmu: A Real-Time Emulation System for Mobile Ad-Hoc Networks*. In: *Proceedings of the First Joint IEI/IEE Symposium on Telecommunications Systems Research*, Dublin, Ireland, November 2001.
- [28] HANNES FREY: *Illustrating videos*, 2004. ☞ <http://www.syssoft.uni-trier.de/~frey/gfg/>.
- [29] HANNES FREY und DANIEL GÖRGEN: *Planar Graph Routing on Geographical Clusters*. Ad Hoc Networks – Special Issue on Data Communication and Topology Control in Ad Hoc Networks, 2004.
- [30] HANNES FREY, DANIEL GÖRGEN, JOHANNES K. LEHNERT und PETER STURM: *Erfahrungsbericht zur praktischen Umsetzung eines Auktionssystems für großflächige mobile multihop Ad-hoc-Netzwerke*. Frühjahrstreffen der Fachgruppe Betriebssysteme der Gesellschaft für Informatik, 2003.
- [31] HANNES FREY, DANIEL GÖRGEN, JOHANNES K. LEHNERT und PETER STURM: *Auctions in mobile multihop ad-hoc networks following the marketplace communication pattern*. In: *Proceedings of the Third International Workshop on Wireless Information Systems (WIS)*, Porto, Portugal, 2004.
- [32] HANNES FREY, DANIEL GÖRGEN, JOHANNES K. LEHNERT und PETER STURM: *A Java-based uniform workbench for simulating and executing distributed mobile applications*. In: *Scientific Engineering of Distributed Java Applications. Third International Workshop, FIDJI 2003*, Band 2952 der Reihe LNCS. Springer, 2004.
- [33] HANNES FREY, JOHANNES K. LEHNERT und PETER STURM: *UbiBay: An auction system for mobile multihop ad-hoc networks*. Workshop on Ad hoc Communications and Collaboration in Ubiquitous Computing Environments (AdHocCCUCE'02), 2002.
- [34] MATTHEW S. GAST: *802.11 Wireless Networks – The Definitive Guide*. O'Reilly & Associates, Inc., 2002.
- [35] DANIEL GÖRGEN, HANNES FREY, JOHANNES K. LEHNERT und PETER STURM: *Marketplaces as communication patterns in mobile ad-hoc networks*. In: *Kommunikation in Verteilten Systemen (KiVS)*, 2003.

- [36] DANIEL GÖRGEN, HANNES FREY, JOHANNES K. LEHNERT und PETER STURM: *SELMA: A middleware platform for self-organizing distributed applications in mobile multihop ad-hoc networks*. In: *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation (CNDS) 2004*, San Diego, CA, USA, 19.-22. Januar 2004.
- [37] MAHER HAMDI, S. CAPKUN und J. P. HUBAUX: *GPS-free Positioning in Mobile Ad-Hoc Networks*. In: *Proceedings of HICSS*, Hawaii, January 2001.
- [38] JOHN HEIDEMANN, NIRUPAMA BULUSU, JEREMY ELSON, CHALERMEK INTANAGONWIWAT, KUN CHAN LAN, YA XU, WEI YE, DEBORAH ESTRIN und RAMESH GOVINDAN: *Effects of Detail in Wireless Network Simulation*. In: *SCS Communication Networks and Distributed Modeling and Simulation Conference*, September 2000. <http://www.isi.edu/~johnh/PAPERS/Heidemann00d.pdf>.
- [39] HORST HELLBRÜCK: *ANSim - Ad-Hoc Network Simulation Tool*. <http://www.i-u.de/schools/hellbrueck/ansim/start.htm>.
- [40] HORST HELLBRÜCK und STEFAN FISCHER: *Basic analysis and simulation of ad-hoc networks*. Technischer Bericht, International University in Bruchsal, Deutschland, 2001.
- [41] DANIEL HERRSCHER und KURT ROTHERMEL: *A Dynamic Network Scenario Emulation Tool*. In: *Proceedings of the 11th International Conference on Computer Communications and Networks (ICCCN 2002)*, Seiten 262–267, Miami, Florida, USA, Oktober 2002.
- [42] HIGHLAND SYSTEMS, INC.: *Bluetooth Simulation Model Suite for OPNET*, 2001. <http://www.highsys.com/products/Suitetooth.pdf>.
- [43] XIAOYAN HONG, MARIO GERLA, GUANGYU PEI und CHING-CHUAN CHIANG: *A group mobility model for ad hoc wireless networks*. In: *Proceedings of the 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, Seiten 53–60. ACM Press, 1999.
- [44] J. P. HUBAUX, L. BUTTYAN und S. CAPKUN: *The quest for security in mobile ad hoc networks*. In: *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, 2001.

- [45] IBM: *WebSphere Studio Device Developer – Product Overview*, 2004.  <http://www-306.ibm.com/software/wireless/wsdd/>.
- [46] *IEEE 802.15 WPAN<sup>TM</sup> Task Group 4 (TG4)*, April 2004.  <http://www.ieee802.org/15/pub/TG4.html>.
- [47] INTERNATIONAL STANDARD ISO/IEC 8802-11: *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications*, 1999.
- [48] AMIT JARDOSH, ELIZABETH M. BELDING-ROYER, KEVIN C. ALMERTH und SUBHASH SURI: *Towards Realistic Mobility Models for Mobile Ad hoc Networks*. In: *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom 2003)*, San Diego, California, USA, 2003.
- [49] DAVID B. JOHNSON, JOSH BROCH, YIH-CHUN HU, JORJETA JETCHEVA und DAVID A. MALTZ: *The CMU Monarch Project’s Wireless and Mobility Extensions to NS*. In: E. TANG (Herausgeber): *Proceedings of the forty-second internet engineering task force*, Chicago, IL, USA, 1998.
- [50] D.B. JOHNSON und D.A. MALTZ: *Dynamic Source Routing in Ad Hoc Wireless Networks*. In: *Mobile Computing*, Band 353 der Reihe *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, 1996.
- [51] E. D. KAPLAN: *Understanding GPS-Principles and Applications*. Artech House Publisher, Boston, 1996.
- [52] BRAD KARP und H. T. KUNG: *GPSR: greedy perimeter stateless routing for wireless networks*. In: *Proceedings of the sixth annual international conference on Mobile computing and networking*, Seiten 243–254. ACM Press, 2000.
- [53] SRINIVASAN KESHAV: *REAL: A Network Simulator*. Technischer Bericht CSD-88-472, University of California, Berkeley, Dezember 1988.
- [54] J. KONG, P. ZERFOS, H. LUO, S. LU und L. ZHANG: *Providing robust and ubiquitous security support for mobile ad-hoc networks*. In: *9th International Conference on Network Protocols (ICNP’01)*, 2001.

- [55] WOLFGANG KREUTZER: *System Simulation. Programming Styles and Languages*. Addison-Wesley, 1986.
- [56] APURVA KUMAR: *BlueHoc: Bluetooth Performance Evaluation Tool*, 2001.  <http://oss.software.ibm.com/bluehoc/index.html>.
- [57] JOHANNES K. LEHNERT, HANNES FREY, DANIEL GÖRGEN und PETER STURM: *A scalable workbench for implementing and evaluating distributed applications in mobile ad-hoc networks*. In: *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation (CNDS) 2004*, San Diego, CA, USA, 19.-22. Januar 2004.
- [58] BEN LIANG und ZYGMUNT J. HAAS: *Predictive Distance-Based Mobility Management for PCS Networks*. In: *Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Infocom 1999)*, Seiten 1377–1384, März 1999.
- [59] JASON LIU und DAVID M. NICOL: *DaSSF 3.1 User's Manual*, April 2001.  <http://www.cs.dartmouth.edu/~jasonliu/projects/ssf/papers/dassf-manual-3.1.ps>.
- [60] MARTIN LUCKOW und NORBERT TH. MÜLLER: *Cascade: A Simple and Efficient Algorithm for Priority Queues*. Technischer Bericht 94-15, Universität Trier, Abteilung Informatik, 1994.  <http://www.informatik.uni-trier.de/~mueller/Forschung/cascade.ps>.
- [61] JAROSLAW MALEK: *Tracegraph, Trace graph - network simulator ns trace files analyser*, 2004.  <http://www.geocities.com/tracegraph/>.
- [62] DAVID A. MALTZ, QIFA KE und DAVID B. JOHNSON: *Emulation of Ad Hoc Networks*. Delivered at the VINT Project Retreat, Asilomar Retreat Center, Pacific Grove, CA, USA, Juni 1999.
- [63] MARTIN MAUVE, JÖRG WIDEMER und HANNES HARTENSTEIN: *A Survey on Position-Based Routing in Mobile Ad-Hoc Networks*. *IEEE Network Magazine*, 15(6):30–39, 2001.
- [64] B.A. MILLER und C. BISDIKIAN: *Bluetooth Revealed*. Prentice Hall, Upper Saddle River, NJ, 2000.

- [65] SHREE MURTHY und J. J. GARCIA-LUNA-ACEVES: *An Efficient Routing Protocol for Wireless Networks*. *Mobile Networks and Applications*, 1(2):183–197, 1996.
- [66] *Nam: Network Animator*.  <http://www.isi.edu/nsnam/nam/>.
- [67] NATIONAL MARINE ELECTRONICS ASSOCIATION: *NMEA 0183 Interface Standard, Version 3.01*, Januar 2002.  <http://www.nmea.org/pub/0183/>.
- [68] OBJECT MANAGEMENT GROUP: *Common Object Request Broker Architecture: Core Specification*. Technical Report Version 3.0.2 Editorial update, December 2002.  [www.omg.org/cgi-bin/doc?formal/02-12-02](http://www.omg.org/cgi-bin/doc?formal/02-12-02).
- [69] OPNET TECHNOLOGIES INC.: *OPNET Modeler*, 2002.  <http://www.opnet.com/products/modeler/home.html>.
- [70] *OTcl*.  <http://otcl-tclcl.sourceforge.net/otcl/>.
- [71] V.D. PARK und M.S. CORSON: *A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks*. In: *Proc. of the Conf. on Computer Communications (IEEE INFOCOM'97)*, Seiten 1405–1413, 1997.
- [72] C. PERKINS und P. BHAGWAT: *Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers*. In: *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, Seiten 234–244, 1994.
- [73] CHARLES E. PERKINS und ELIZABETH M. ROYER: *Ad-Hoc On Demand Distance Vector Routing*. In: *2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, Louisiana, USA, 25.-26. Februar 1999.
- [74] KAY RÖMER, THOMAS SCHOCH, FRIEDEMANN MATTERN und THOMAS DÜBENDORFER: *Smart Identification Frameworks for Ubiquitous Computing Applications*. *Wireless Networks*, 10(6), Dezember 2004.
- [75] ELIZABETH M. ROYER und CHAI-KEONG TOH: *A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks*. *IEEE Personal Communications*, 6(2):46–55, April 1999.
- [76] SCALABLE NETWORK TECHNOLOGIES: *QualNet Family of Products*, 2003.  <http://www.qualnet.com/products/qualnet.php>.



- [77] MARK SEGAL und KURT AKELEY: *The OpenGL Graphics System: A Specification (Version 1.4)*, 2002. 🌐 <http://www.opengl.org/developers/documentation/version1.4/glspec14.pdf>.
- [78] *SONI project homepage*, 2003. 🌐 <http://www.ist.lu/html/projets/de/soni/index.html>.
- [79] SSF RESEARCH NETWORK: *Scalable Simulation Framework*, 1999-2002. 🌐 <http://www.ssfnet.org/homePage.html>.
- [80] IVAN STOJMENOVIC (Herausgeber): *Handbook of Wireless Networks and Mobile Computing*. Wiley-Interscience, 2002.
- [81] KARTHIKEYAN SUNDARESAN, VAIDYANATHAN ANANTHARAMAN, HUNG-YUN HSIEH und RAGHUPATHY SIVAKUMAR: *ATP: a reliable transport protocol for ad-hoc networks*. In: *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, Seiten 64–75. ACM Press, 2003.
- [82] SUN MICROSYSTEMS, INC.: *Java Remote Method Invocation Specification (Revision 1.8)*, 2003. 🌐 <ftp://ftp.java.sun.com/docs/j2se1.4/rmi-spec-1.4.pdf>.
- [83] *SWT Experimental OpenGL Plug-in*, 2003. 🌐 <http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/platform-swt-home/opengl/opengl.html>.
- [84] HIDEAKI TAKAGI und LEONARD KLEINROCK: *Optimal Transmission Ranges for Randomly Distributed Packet Radio Terminals*. IEEE Transactions on Communications, 32(3):246–257, März 1984.
- [85] THE CMU MONARCH PROJECT: *The CMU Monarch Project's ad-hockey Visualization Tool For ns Scenario and Trace Files*, 1998. 🌐 <http://www.monarch.cs.rice.edu/ftp/monarch/wireless-sim/ad-hockey.ps>.
- [86] THE NETFILTER/IPTABLES PROJECT, 2003. 🌐 <http://www.netfilter.org/>.
- [87] JING TIAN, JÖRG HÄHNER, CHRISTIAN BECKER, ILLYA STEPANOV und KURT ROTHERMEL: *Graph-Based Mobility Model for Mobile Ad Hoc Network Simulation*. In: *Proceedings of the 35th Annual Simulation Symposium*, Seiten 337–345. IEEE Computer Society Press, 2002.

- [88] F. A. TOBAGI und L. KLEINROCK: *Packet Switching in Radio Channels: Part II - the Hidden Terminal Problem in Carrier Sense Multiple-Access Modes and the Busy-Tone Solution*. IEEE Transactions on Communications, 23(12):1417–1433, 1975.
- [89] CHRISTIAN DE WAAL: *BonnMotion: A mobility scenario generation and analysis tool*, 2002-2003. <http://web.informatik.uni-bonn.de/IV/Mitarbeiter/dewaal/BonnMotion/>.
- [90] KEVIN WALSH und EMIN GÜN SIRER: *Staged Simulation for Improving the Scale and Performance of Wireless Network Simulations*. In: *Proceedings of the Winter Simulation Conference*, New Orleans, LA, Dezember 2003.
- [91] MARK WEISER: *The Computer for the 21st Century*. Scientific American, 265(3):94–104, September 1991.
- [92] WIKIPEDIA: *Gauß-Krüger-Koordinatensystem*, 2004. <http://de.wikipedia.org/wiki/Gauß-Krüger-Koordinatensystem>.
- [93] B. WILLIAMS und T. CAMP: *Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks*. In: *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, Seiten 194–205, 2002.
- [94] JUNGKEUN YOON, MINGYAN LIU und BRIAN NOBLE: *Random Waypoint Considered Harmful*. In: *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Infocom 2003)*, San Francisco, CA, USA, April 2003.
- [95] JUNGKEUN YOON, MINGYAN LIU und BRIAN NOBLE: *Sound Mobility Models*. In: *Proceedings of the Ninth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM 2003)*, San Diego, CA, USA, September 2003.
- [96] MANEL GUERRERO ZAPATA: *Secure Ad hoc On-Demand Distance Vector Routing*. ACM Mobile Computing and Communications Review (MC2R), 6(3):106–107, July 2002.
- [97] X. ZENG, R. BAGRODIA und M. GERLA: *GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks*. In: *Proceedings of the 12th*

---

*Workshop on Parallel and Distributed Simulation (PADS-98)*, Seiten 154–161,  
Los Alamitos, 1998. IEEE Computer Society.

[98] *ZigBee Alliance*, Juni 2004.  <http://www.zigbee.org/>.



# Tabellarischer Bildungsgang

---

---

29.03.1974	geboren in Koblenz
14.06.1993	Abitur am Friedrich-Wilhelm-Gymnasium in Trier
01.10.1993	Studienbeginn Mathematische Informatik an der Universität Trier
20.10.1995	Vordiplom Schwerpunkte im Hauptstudium: Datenbanken und Compilerbau
01.09.1998	Abschluß der Diplomprüfungen
14.09.1998 – 31.01.1999	Austauschstudent im Rahmen des Sokrates-/Erasmus- programms an der University of Southampton, Großbritannien
01.04.1999 – 28.09.1999	Diplomarbeit Thema: „Verteilungs-, Leistungs- und Zuverlässigkeits- aspekte elektronischer Diskussionsforen“
28.09.1999 – 17.12.2004	Doktorand in der Arbeitsgruppe „Systemsoftware und Verteilte Systeme“ von Prof. Dr. Peter Sturm an der Universität Trier
17.12.2004	Disputation und Abschluß der Promotion

---