

Universität Hannover

Institut für Technische Informatik, Rechnergestützte Wissensverarbeitung

Diplomarbeit

**Erweiterung eines Mobilfunksimulators um marktbasierete
dynamische Ressourcenvergabeverfahren**

Mohamed Silmi Khanfir

September 2000

Erstprüfer : Prof. Dr. techn. Dipl.-Ing. W. Nejd
Zweitprüfer : Prof. Dr.-Ing. K. Jobmann
Betreuer : Dipl.-Inform. Dr. techn. F. Steimann

Inhaltsverzeichnis

Inhaltsverzeichnis	1
1 Einleitung.....	5
2 Grundlagen der Mobilfunknetze	7
2.1 Grundbegriffe der zellularen Netze.....	7
2.2 Signal-Störabstand.....	8
2.3 Cluster	9
2.4 Vielfachzugriffsverfahren.....	10
2.4.1 Frequenzvielfachzugriff FDMA	10
2.4.2 Zeitvielfachzugriff TDMA	10
2.5 Verkehrstheoretische Grundlagen.....	11
3 Verfahren zur Ressourcenverteilung in Mobilfunknetzen	13
3.1 Statische Ressourcenverteilung.....	13
3.2 Dynamische Ressourcenverteilung	14
3.2.1 Verkehrsadaptive Verfahren	15
3.2.2 Signaladaptive Verfahren	17
3.2.3 Interferenzadaptive Verfahren	17
3.2.4 Proaktive Verfahren: CMRA und STBR	18
4 Marktbasierte Regelung (Market Based Control)	21
4.1 Auktionen.....	22
4.1.1 English Auction	23
4.1.2 Dutch Auction	23
4.1.3 First-Price, Sealed Bid Auction	24
4.1.4 Vickrey Auction (Uniform second-price).....	24
4.1.5 Double Auction	25
4.2 Preisgestaltung	25
5 Das realisierte marktbasierete Ressourcenverteilungsverfahren.....	27
5.1 Marktbasiertes Modell für das Mobilfunknetz	27
5.2 Ressourcenbewertung.....	27
5.3 Auktionen.....	31

5.3.1	Aufbau der Transaktionstabelle	31
5.3.2	Bearbeitung der Transaktionstabelle	33
5.3.3	Update der Transaktionstabelle	34

6 Der Mobilfunksimulator MOSIT 37

6.1	Aufbau.....	37
6.2	Der Netzmanagement-Block.....	38
6.2.1	Erweiterbarkeit des Netzmanagementblocks	39
6.2.2	Der Prozeß Netzmanagement_Steuerung	40
6.2.3	Der Prozess NM_Area_Manager.....	40
6.2.4	Der Prozess BS_Agent	40
6.3	SDL Schnittstellen zur Umgebung.....	41
6.3.1	SDL und das SDT-Tool.....	41
6.3.2	Kommunikation über CORBA	44
6.3.3	Das SDT Postmaster Interface.....	45
6.3.4	Environment Funktionen	47
6.3.4.1	Nötige Funktionen	47
6.3.4.2	Hilfsfunktionen	48
6.3.5	Java Native Interface (JNI).....	49
6.3.5.1	Lokale und globale Referenzen	50
6.3.5.2	Threads und JNI.....	51
6.3.5.3	Das Invocation API.....	52
6.3.5.4	Speichermanagment.....	54

7 Implementierung..... 56

7.1	Architektur der implementierten Lösung.....	56
7.2	Die SDL-Seite.....	57
7.2.1	Senden der Netz- und Verkehrsdaten	59
7.2.2	Empfang und Bearbeitung der Transaktionsinformationen	60
7.3	Die Schnittstelle über die Environment-Funktionen und JNI.....	63
7.3.1	Die Funktion XoutEnv.....	64
7.3.1.1	Die relevanten SDL-Datenstrukturen und ihre Manipulation	64
7.3.1.2	Die Schnittstelle über JNI.....	66
7.3.2	Die Funktion XinEnv.....	67
7.4	Die JAVA-Seite.....	68
7.4.1	Einleitung	68
7.4.2	Datenstrukturen für die Netzdaten.....	68
7.4.2.1	Klasse <i>Temp_Network</i>	68
7.4.2.2	Klasse <i>SektorStateT</i>	69
7.4.2.3	Klasse <i>Sektor_TF</i>	70
7.4.2.4	Klasse <i>TF</i>	70
7.4.2.5	Klasse <i>Koordinaten</i>	71
7.4.3	Datenstrukturen für den marktbasierten Algorithmus	71
7.4.3.1	Klasse <i>Transaktion</i>	71
7.4.3.2	Klasse <i>End_Transaktion</i>	72
7.4.3.3	Klasse <i>Result</i>	72
7.4.4	Das Hauptprogramm.....	73
7.4.4.1	Die Methode <i>init</i> und die Initialisierungsdaten.....	73
7.4.4.2	Die Methoden zur Realisierung des marktbasierten Algorithmus	74

8 Simulation und Ergebnisse..... 79

8.1	Simulationsumgebung.....	79
8.1.1	Netztopologie	81
8.1.2	Bewegungsprofile der Mobilstationen.....	82
8.1.2.1	2 Züge mit Ringen	82
8.1.2.2	2 Züge	83
8.2	Simulationsergebnisse.....	83
8.2.1	Erste Konfiguration: 16 Basisstationen, Clustergröße 4.....	83
8.2.2	Zweite Konfiguration: 25 Basisstationen, Clustergröße 7.....	90
8.2.2.1	Simulation 1-4: inhomogen verteilter Verkehr (Hotspot).....	92
8.2.2.2	Simulation 5-6: Homogen verteilter Verkehr	93
8.2.2.3	Simulation 7-8: Einfluß der Verhandlungsrandbedingungen.....	93
8.2.2.4	Häufigkeit der ausgeführten Transaktionen.....	95

9 Zusammenfassung und Ausblick 97

10Anhang..... 101

10.1	Literaturverzeichnis.....	101
10.2	Abbildungsverzeichnis.....	103
10.3	Abkürzungsverzeichnis.....	105

1 Einleitung

Der Mobilfunkmarkt ist in den letzten Jahren stark gewachsen. Die Schätzungen der zukünftigen Teilnehmeranzahl werden ständig nach oben hin korrigiert. Darüber hinaus zeigen die neu eingeführten Mobilfunkdienste einen hohen Bedarf an Übertragungskapazitäten. Der Bedarf an Ressourcen steigt also stetig und dies bei einer begrenzten Kapazität des Übertragungsmediums, was vom Netzbetreiber fordert, die zur Verfügung stehenden Ressourcen am effektivsten zu nutzen.

Einer der wichtigsten Lösungsansätze für eine effektivere Nutzung der Ressourcen der Luftschnittstelle ist die dynamische Ressourcenverteilung. Dieser Ansatz soll die klassische aufwendige Netzplanung ersetzen. Statt die Ressourcen einmalig bei der Inbetriebnahme des Netzes statisch den Mobilfunkzellen zuzuweisen, soll sich die Ressourcenverteilung dynamisch an Verkehrssituationen anpassen können. Durch den dynamischen Aspekt können plötzliche Überlastungen bestimmter Teile des Netzes effizient aufgelöst werden.

Die Frage, die sich stellt ist, nach welchen Kriterien und nach welchem Algorithmus werden die Ressourcen dynamisch verteilt sprich, wie wird die Kontrolle der Ressourcenverteilung realisiert?

Eine zentrale Kontrolle würde zwar die optimalsten Ergebnisse liefern, führt aber wegen der Komplexität der Mobilfunknetze zu einem hohen Signalisierungsaufwand bzw. Ausfallrisiko, was für eine dezentrale Kontrolle spricht.

Die marktbasierter Regelung (Market Based Control) bietet sich als guter Ansatz für eine dezentrale Kontrolle der dynamischen Ressourcenverteilung. Unter einem Markt versteht man ein System, in dem eine Anzahl von Komponenten basierend auf Kauf/Verkauf Transaktionen lokal interagieren, um ein globales zusammenhängendes Verhalten zu erreichen.

Im Rahmen dieser Arbeit wird der im IANT entworfene Mobilfunksimulator (MOSIT) um ein dynamisches marktbasierter Ressourcenverteilungsverfahren erweitert. Das Verfahren wird als externes Programm in Java implementiert und kommuniziert über C-Environment-Funktionen mit dem in SDL implementierten Mobilfunksimulator MOSIT. Diese Implementierung soll als Basis für spätere Arbeiten bezüglich Ressourcenverteilung dienen.

Das anschließende 2. Kapitel führt in die grundlegende Begriffe der Mobilfunknetze und Verkehrstheorie ein. Einen Überblick über existierende Ressourcenverteilungsverfahren in Mobilfunknetzen gibt das 3. Kapitel. Im 4. Kapitel werden die Grundlagen der marktbasierter

Regelung vorgestellt. Das 5. Kapitel enthält eine Beschreibung des realisierten dynamischen marktbasieren Ressourcenverteilungsverfahrens. Im 6. Kapitel werden die technischen Hintergründe der Arbeit erläutert. Dabei wird zuerst auf den allgemeinen Aufbau und die für diese Arbeit relevanten Teile von MOSIT kurz eingegangen, dann werden die verschiedenen Alternativen der Schnittstellen zwischen Java und MOSIT vorgestellt. Das 7. Kapitel bietet eine umfassende Beschreibung der Implementierung und der Architektur der Realisierung. Im 8. Kapitel werden die durchgeführten Simulationen, die das implementierte Verfahren hinsichtlich seiner dynamischen Eigenschaften und seiner Auswirkungen auf die Netz-Performance untersuchen, ausgewertet und beschrieben. Eine Zusammenfassung und ein Ausblick auf zukünftige Aufgaben sind im 9. Kapitel zu finden. Die Ausarbeitung schließt im 10. Kapitel mit einem Anhang, wo sich unter anderem ein Abkürzungs- und Literaturverzeichnis befinden.

2 Grundlagen der Mobilfunknetze

Dieses Kapitel dient zur Einführung in die Grundlagen zellularer Netze, die bei der Betrachtung von Ressourcenverteilungsverfahren benutzt werden. Darüber hinaus werden einige Vielfachzugriffsverfahren und die für diese Arbeit relevanten Definitionen und Formeln der Verkehrstheorie vorgestellt.

2.1 Grundbegriffe der zellularen Netze

Aufgrund der sehr begrenzten Kapazität der Luftschnittstelle steht einem Mobilfunknetz nur eine relativ kleine Anzahl von Trägerfrequenzen zur Verfügung. Um trotzdem mehrere Hunderttausend bis Millionen Teilnehmer zu bedienen, müssen die Frequenzen räumlich (geographisch) mehrfach genutzt werden. Diese räumliche Frequenzwiederverwendung (*spatial frequency reuse*) führte zur Entwicklung der zellularen Netze.

Bei einem zellularen Netz wird die abzudeckende Fläche in *Zellen* (einzelne Funkzonen) aufgeteilt. Diese Zellen werden jeweils von einer Basisstation (*BS*) versorgt und werden vereinfacht als Hexagone modelliert, was den realen Bedingungen nur bedingt entspricht. Im Realfall haben die Funkzellen aufgrund topologischer und verkehrsbedingter Umstände eine unregelmäßige Form und überlappen sich zum Teil.

Im Normalfall wird jeder Zelle eine Untermenge von Frequenzen aus der verfügbaren Gesamtmenge zugewiesen (siehe Kapitel 3.1). Zwei direkt benachbarte Zellen dürfen keinesfalls dieselben Frequenzen verwenden, da sonst starke *Gleichkanalstörungen* aus den Nachbarzellen zu erwarten wären. Erst in einem bestimmten Abstand D (*Frequenzwiederholabstand*) wird eine Frequenz erneut verwendet. Wenn dieser Abstand ausreichend groß gewählt wird, bleiben die Gleichkanalstörungen klein genug, um die Sprachqualität nicht zu beeinträchtigen. Die räumliche Wiederholung der Frequenzen erfolgt regelmäßig, so daß jede Zelle ihre nächsten Nachbarn mit gleichen Frequenzen wieder im Abstand D sieht (siehe Abbildung 1).

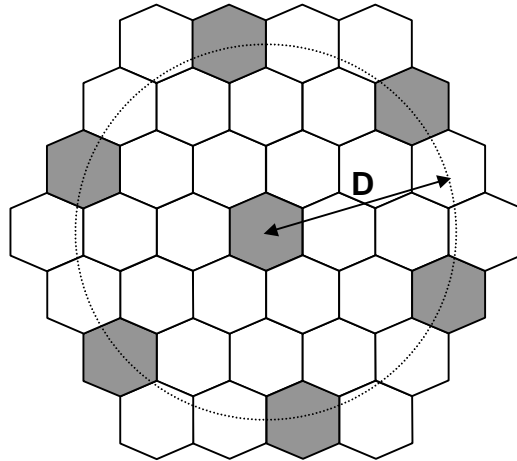


Abb. 1: Beispiel eines zellularen Netzes mit Frequenzwiederholung

2.2 Signal-Störabstand

Die Störung, die von den Nachbarzellen verursacht wird, wird im Nutzsignal-zu-Störsignal-Verhältnis gemessen [EV99]:

$$W = \frac{\text{Nutzsignal}}{\text{Störsignal}} = \frac{\text{Nutzsignal}}{\text{Nachbarzelleninterferenz} + \text{Rauschen}} \quad (\text{Gl. 2.1})$$

Dieses Nutzsignal-zu-Störsignal-Verhältnis wird üblicherweise als *Signal-Störabstand* oder Träger-zu-Interferenzverhältnis (*Carrier to Interference Ratio C/I*) bezeichnet. Die Intensität der Störung durch Gleichkanalinterferenzen wird in Abhängigkeit von den Abständen der Störzellen zur Mobilstation bestimmt. Die Gleichkanalstörungen aus Sicht einer Mobilstation erzeugen die Basisstationen im Abstand d_i von der aktuellen Basisstation. Für eine Mobilstation am Rand ihres aktuellen Versorgungsgebietes im Abstand R zur Basisstation kann unter der Annahme, daß alle Z benachbarten Störsender mit der gleichen Sendeleistung P_0 betrieben werden und der Abstand einer Störzelle zur Mobilstation näherungsweise gleich d_i ist (Abstand d_i groß gegenüber Zellradius R), eine worst-case Abschätzung für den Signal-Störabstand W vorgenommen werden:

$$W = \frac{P_S R^{-\alpha}}{\sum_i^Z P_i d_i^{-\alpha} + N_0} \approx \frac{P_0 R^{-\alpha}}{\sum_i^Z P_0 d_i^{-\alpha} + N_0} \quad (\text{Gl. 2.2})$$

P_S bzw. P_i stehen für die Sendeleistung des Nutzsignals bzw. der Störsender. N_0 ist die Rauschleistung und α ist ein Parameter, der abhängig von der Umgebung zwischen 3-5 liegt. Bei Vernachlässigung des Rauschens N_0 gilt somit näherungsweise für das Träger-zu-Interferenzverhältnis:

$$\frac{C}{I} = W \approx \frac{R^{-\alpha}}{\sum_i d_i^{-\alpha}} \cong \frac{1}{\sum_i d_i^{-\alpha}} \quad (\text{Gl. 2.3})$$

Die in Gl. 2.3 gezeigte Proportionalität wird in den späteren Betrachtungen bei der Wahl einer Kostenfunktion für die Ressourcenverteilung ihren Einsatz finden.

2.3 Cluster

Aufgrund der regelmäßigen Wiederholung der Frequenzen werden die Zellen in Gruppen aufgeteilt. Jede so entstandene Gruppe von Zellen (*Cluster*) kann den gesamten Frequenzbereich besitzen. Die Größe des Clusters wird in "Anzahl der Zellen pro Cluster" angegeben. Innerhalb eines Clusters wird keine Frequenz mehrfach verwendet. Die Frequenzen einer Zelle innerhalb eines Clusters werden frühestens im benachbarten Cluster wiederverwendet. Je größer ein Cluster ist, desto größer wird der Frequenzwiederholabstand und desto größer auch der Signal-Störabstand. Der Einfluß der Interferenzen durch die Störsender wird also geringer. Allerdings ist die Anzahl von Kanälen und damit auch die Teilnehmerzahl je Zelle umso geringer, je größer der Cluster ist. Dies ist dadurch begründet, daß eine höhere Anzahl von Zellen mit dem selben gesamten Frequenzbereich zu versorgen ist. Die selbe Anzahl von Frequenzen wird folglich auf mehreren Zellen verteilt.

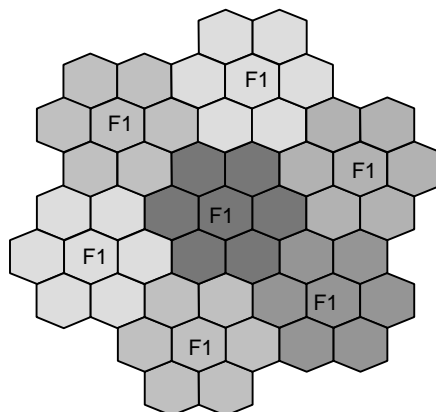


Abb. 2: Beispiel eines 7-Zellen-Cluster zellularen Netzes

2.4 Vielfachzugriffsverfahren

Im Mobilfunk nutzen alle Teilnehmer in einer Zelle ein gemeinsames Übertragungsmedium: die Luftschnittstelle. Die Mobilstationen konkurrieren miteinander um die Ressource Frequenz, um ihre Informationen zu übertragen. Um die verfügbaren physikalischen Ressourcen eines Mobilfunksystems, d.h. die Frequenzbänder, in möglichst vielen Gesprächskanälen aufzuteilen, werden spezielle Vielfachzugriffsverfahren (*multiple access*) eingesetzt. In GSM sind der Frequenzvielfachzugriff und der Zeitvielfachzugriff realisiert, die im folgenden kurz vorgestellt werden. Für eine umfassende Beschäftigung mit den Vielfachzugriffsverfahren wird auf [EV99] verwiesen.

2.4.1 Frequenzvielfachzugriff FDMA

Der Frequenzvielfachzugriff (*Frequency Division Multiple Access* FDMA) ist eines der herkömmlichsten Vielfachzugriffsverfahren. Das Frequenzband wird dabei in gleich große Kanäle zerlegt, so daß alle Gesprächsverbindungen auf unterschiedlichen Frequenzen geführt werden. Im GSM900-System werden beispielsweise 25 MHz Bandbreite im Frequenzbereich um 900 MHz benutzt. Mit dieser Bandbreite können bei einer Trägerbandbreite von 200 kHz maximal 125 Trägerfrequenzen bereitgestellt werden.

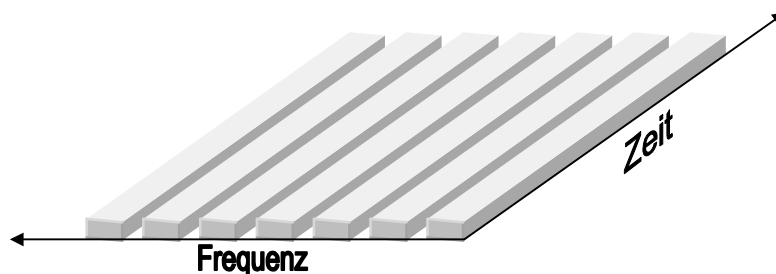


Abb. 3: Kanäle in einem FDMA-System

2.4.2 Zeitvielfachzugriff TDMA

Der Zeitvielfachzugriff (*Time Division Multiple Access* TDMA) wird im digitalen Mobilfunk angewandt. Die einzelnen Mobilstationen erhalten die Frequenz für die Dauer eines TDMA-Zeitschlitzes (*time slot*) zyklisch exklusiv zugewiesen. Dabei wird nicht die gesamte Bandbreite eines Systems für einen Zeitschlitz einer Mobilstation zugeteilt, sondern nur ein Unterband, das als Trägerfrequenz bezeichnet wird. Das digitale System GSM verwendet eine Kombination von FDMA und TDMA. In einem Band von 25 MHz Breite werden 124

einzelne Kanäle mit 200 kHz Bandbreite (125 Unterbänder – 1 Schutzband = 124 einzelne Trägerfrequenzen) untergebracht, wobei jeder dieser Frequenzkanäle wiederum 8 TDMA-Gesprächskanäle enthält. Die Folge von Zeitschlitzten, die einer Mobilstation zugewiesen werden, ist damit der physikalische Kanal eines TDMA-Systems.

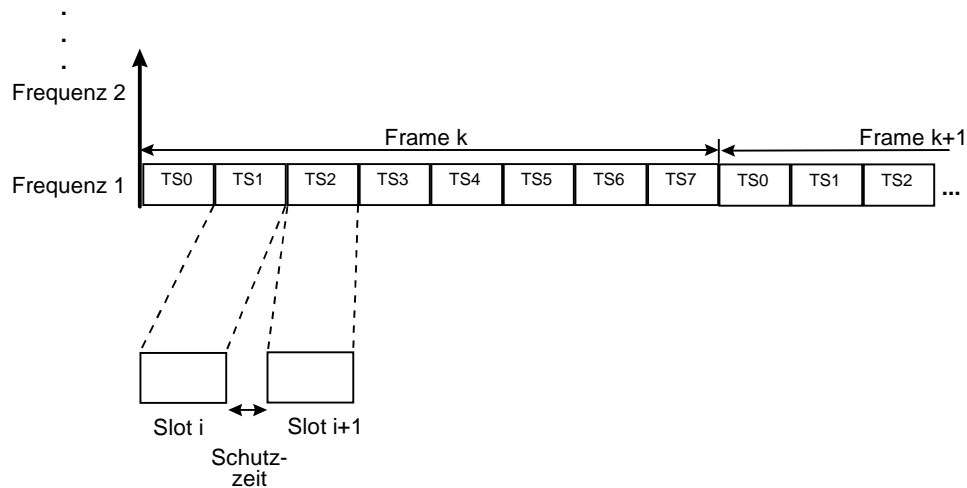


Abb. 4: Zeitvielfachzugriff TDMA

2.5 Verkehrstheoretische Grundlagen

Eine Zelle kann näherungsweise als verkehrstheoretisches Verlustsystem mit n Bedieneinheiten (Kanälen) modelliert werden, mit einem Rufankunftsprozeß, dessen Zwischenankunftsabstände negativ exponentiell verteilt sind (Poissonprozeß), und ebenfalls einem Poissonprozeß als Bedienprozeß (näheres zur Verkehrstheorie findet man in [DAV96]). Im folgenden werden die wichtigsten Definitionen aus der Verkehrstheorie kurz erläutert.

Das *Angebot* A ist ein statistischer Wert, der den zu bedienenden Verkehr darstellen soll. Es läßt sich durch die Summe aller Belegungsauern dividiert durch den betreffenden Beobachtungszeitraum berechnen.

$$A = \frac{1}{T} \sum_{t_0}^{t_0+T} n_t \cdot \Delta t \quad (\text{Gl. 2.4})$$

n_t : Anzahl der Belegungsversuche im Meßzeitraum Δt .

T : Beobachtungszeitraum beginnend zum beliebigen Zeitpunkt t_0 .

Nimmt man poissonverteilten Verkehr an, so vereinfacht sich (Gl. 2.4) zu

$$A = \frac{t_m}{a_m} \quad (\text{Gl. 2.5})$$

t_m : mittlere Belegungsdauer.

a_m : mittlerer Abstand der ankommenden Anrufe.

Das Angebot ist also die relative Dauer einer Belegung pro Zeiteinheit. Sie kann dann auch zusammengesetzt werden aus der Multiplikation der mittleren Ankunftsrate λ mit dem Kehrwert der mittleren Bedienrate $1/\mu$.

$$A = \frac{\lambda}{\mu} \quad (\text{Gl. 2.6})$$

λ : mittlere Ankunftsrate.

μ : mittlere Bedienrate.

Die Einheit des Verkehrsangebotes ist Erlang [Erl]. Weil die Einheit der mittleren Ankunftsrate λ [1/s] dieselbe ist wie die der mittleren Bedienrate μ , handelt es sich um eine dimensionslose Einheit, die zeigt, daß eine verkehrstheoretische Größe betrachtet wird.

Die *Blockierwahrscheinlichkeit* B ist bei Verlustsystemen ein Maß für die abgewiesenen Belegungswünsche. Den Zusammenhang zwischen Angebot A und Blockierwahrscheinlichkeit B bei der Gesamtzahl der Kanäle N beschreibt die Erlang-Blockierungswahrscheinlichkeitsformel:

$$B = \frac{A^N}{N! \sum_{k=0}^N \frac{A^k}{k!}} \quad (\text{Gl. 2.7})$$

3 Verfahren zur Ressourcenverteilung in Mobilfunknetzen

Die Anzahl der GSM-Teilnehmer steigt kontinuierlich, dies bei einer konstanten Kapazität der Luftschnittstelle als Übertragungsmedium. Die Netzbetreiber sind gefordert neue Verfahren einzusetzen, um eine bessere Ausnutzung der zur Verfügung stehenden Ressourcen bei einer guten Dienstqualität (QoS) zu erreichen. Die Ressourcen müssen also so über das Netz verteilt werden, daß der Verlust sowie die mittlere Wartezeit einen bestimmten Wert nicht überschreiten. Ressourcen in Mobilfunksystemen sind die Anzahl zur Verfügung stehender Funkkanäle. In der Literatur wird bei der Ressourcenverteilung auch von Kanalvergabe gesprochen; im folgenden wird für die Verteilung der zur Verfügung stehenden logischen Kanäle der Luftschnittstelle über das Mobilfunknetz von *Ressourcenverteilung* gesprochen. Unter *Kanalvergabe* versteht man die Zuordnung einer Verbindung einer Mobilstation mit einer Basisstation. Dabei ist es bei bestimmten dynamischen Verfahren (interferenzadaptive Verfahren) aber durchaus möglich, daß eine solche Unterscheidung nicht möglich ist, da die Ressourcenverteilung erst im Augenblick der Kanalvergabe vorgenommen wird.

Dieses Kapitel liefert einen Überblick über die in heutigen Mobilfunknetzen eingesetzten Ressourcenverteilungs / Kanalvergabeverfahren. Sie lassen sich in zwei Gruppen einteilen: Statische (FCA, *Fixed Channel Allocation*) und dynamische Ressourcenverteilung / Kanalvergabe (DCA, *Dynamic Channel Allocation*). Beide sollen abhängig von der Lastverteilung und der Interferenzsituation die begrenzte Kapazität der Luftschnittstelle möglichst gut nutzen.

Eine ausführliche Beschreibung der Algorithmen der verschiedenen eingesetzten Ressourcenverteilungsverfahren liefern [BEN96] und [GER98].

3.1 Statische Ressourcenverteilung

Bei den statischen Ressourcenverteilungs / Kanalvergabeverfahren sind die Kanäle den Basisstationen fest zugeordnet. Eine Basisstation hat damit einen festen Vorrat an Kanälen. Bei einer Überlastung werden die Belegungswünsche, die nicht bedient werden können, abgewiesen. Es wird nicht versucht die Ressourcenverteilung dynamisch an den Verkehr anzupassen. Die Zuordnung der Trägerfrequenzen auf die Basisstationen wird in aufwendigen Netzplanungen einmalig durchgeführt. Dabei wird versucht, die voraussichtliche

Teilnehmerdichte und den daraus abzuleitenden Verkehr zu ermitteln. Der Vorteil der statischen Ressourcenverteilung ist der geringe Signalisierungsaufwand im Vergleich zu den folgend erläuterten dynamischen Ressourcenverteilungsverfahren.

3.2 Dynamische Ressourcenverteilung

Bei den dynamischen Ressourcenverteilungsverfahren / Kanalvergabeverfahren sind die Kanäle den Basisstationen nicht fest zugeordnet und können somit über die Netzfläche verschoben werden. Daraus erreicht man insbesondere bei inhomogener Verkehrsverteilung im Mobilfunknetz, den „Hotspots“, deutlich geringere Blockierungsraten. Bei hoher homogener Verteilung der Belastung des Mobilfunknetzes ist hingegen bei den meisten DCA-Verfahren mit einer höheren Blockierungswahrscheinlichkeit zu rechnen [BEN96], weil bei jedem Verschieben von Ressourcen sicher ein Teil des Netzes (Donatoren) benachteiligt wird. Alle Basisstationen haben nämlich ungefähr den gleichen Bedarf an Ressourcen.

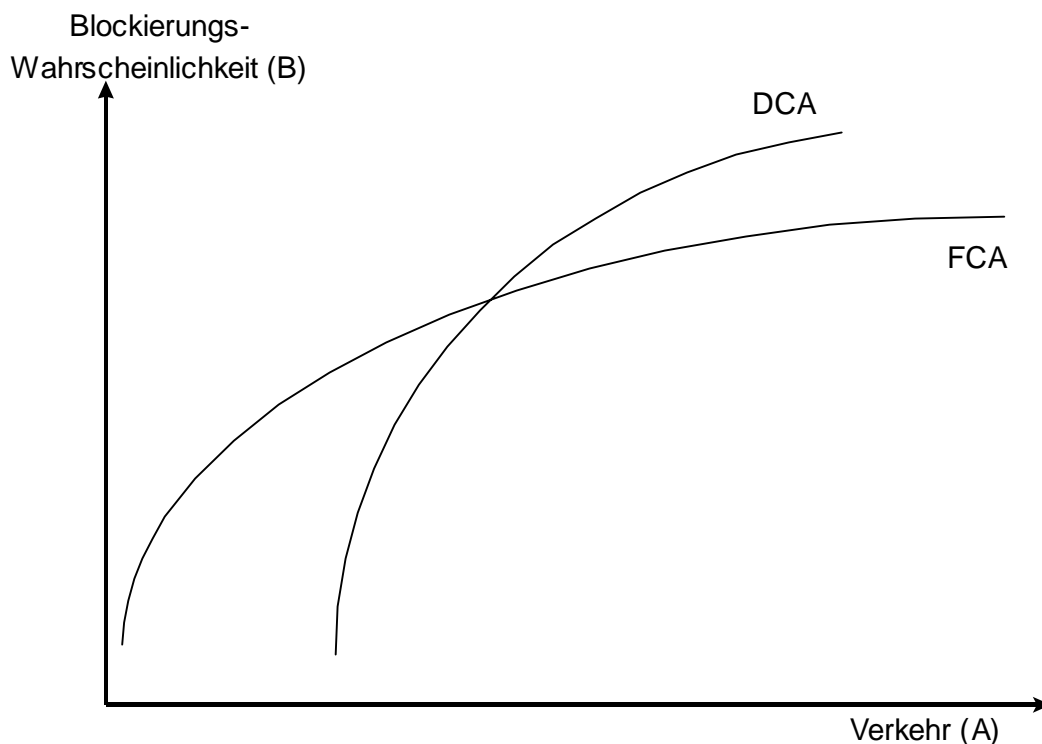


Abb. 5: Zusammenhang zwischen angebotenenem Verkehr und Blockierungswahrscheinlichkeit bei FCA und DCA [BEN96]

Der Signalisierungsaufwand und die Ausfallsicherheit sind die wichtigsten Kriterien zur Beurteilung der verschiedenen dynamischen Verfahren. Daher spielt die Entscheidungsinstanz der Verteilungsverfahren eine große Rolle. Zentrale Verfahren (*centralized DCA schemes*) setzen Kenntnis über große Bereiche des Mobilfunknetzes voraus; daraus ergibt sich ein

großer Signalisierungsaufwand und das Halten einer zentralen Datenbank, mit deren Hilfe die Ressourcen optimal über das Netz verteilt werden können. Wenn z.B. eine solche Datenbank ausfällt, fallen große Teile des Netzes aus. Verteilte Verfahren (*distributed DCA schemes*) bedürfen einem geringeren Signalisierungsaufwand und sind ausfallsicherer, liefern aber nicht die optimal möglichen Ergebnisse [DEL97]. Im folgenden werden verschiedene Verfahren, die die dynamische Ressourcenverteilung / Kanalvergabe vorgestellt.

3.2.1 Verkehrsadaptive Verfahren

Verkehrsadaptive Verfahren reagieren flexibel auf die Lastsituationen im Netz. So kann sich z.B. eine überlastete Zelle Kanäle von ihren Nachbarn ausleihen. Man kann in diesen Verfahren zwei Klassen unterscheiden: Bei Kanal-Leih-Verfahren wird eine initiale Netzplanung vorausgesetzt, die den Zellen eine variable Anzahl von Kanälen zur Verfügung stellt. Bei den anderen Verfahren entnehmen die Zellen ihren Bedarf an Kanälen aus einem gemeinsamen Pool, wobei auf Interferenzen geachtet wird.

Die Kanal-Leih-Verfahren basieren auf einer zu Beginn fest vorgegebenen Verteilung der Ressourcen über das Mobilfunknetz. Bei Bedarf kann sich eine Basisstation bei ihren Nachbarn Ressourcen leihen, die dort dann nicht mehr verwendet werden dürfen. Außerdem muß gewährleistet werden, daß der geliehene Kanal keine Störungen verursacht. Der Kanal muß deswegen u.U. auch in anderen Basisstationen gesperrt werden, damit der Frequenzwiederholabstand eingehalten wird.

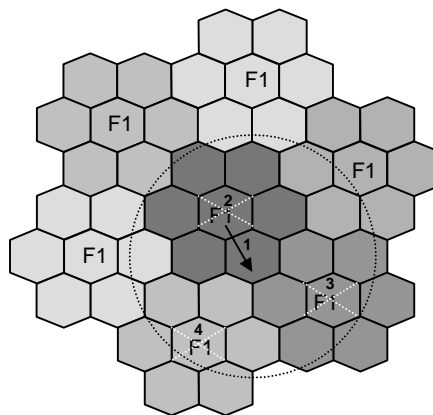


Abb. 6: Prinzip der Kanal-Leih-Verfahren

In Abbildung 6 ist ein Beispiel eines Leih- und Sperrvorgangs in einem Mobilfunknetz dargestellt, der auf einfachem Leihen beruht (SB, *Simple Borrowing*). Es wurde eine Clustergröße von $N=7$ gewählt, so daß die Zellen mit der Beschriftung F1 den gleichen Frequenzvorrat zur Verfügung gestellt haben. Zelle 1 sucht neue Ressourcen und wird fündig

in Zelle 2. Damit sich Zelle 1 die Frequenz F1 aus Zelle 2 leihen darf, muß diese Frequenz nicht nur in Zelle 2 sondern auch in den Zellen 3 und 4 frei sein. Bei einem erfolgtem Leihvorgang wird die Frequenz F1 in den Zellen 2, 3 und 4 gesperrt (*Channel Locking*), um den Frequenzwiederholabstand einzuhalten. Ein einfaches Leihverfahren ist das BFA-Verfahren (*Borrowing First Available*), bei dem der erste gefundene freie Kanal geliehen wird (sofern die Randbedingungen eingehalten werden). Dieses Verfahren kann verbessert werden, indem man von den Zellen mit den meisten verfügbaren freien Kanälen leiht (SBR, *Borrow from the Richest*). Diese beiden Leihverfahren können auch in Kombination mit einer statischen Ressourcenverteilung benutzt werden; man spricht dann von hybriden Kanal-Leih-Verfahren (HCA, *Hybrid Channel Assignment*). Dabei verfügt jede Zelle über eine feste Anzahl von Kanälen, die nicht verliehen werden dürfen, und zusätzlich über eine Anzahl von Leih-Kanälen, die verliehen werden dürfen. Der Nachteil der Leih-Verfahren ist die Reduzierung der Kapazität des Netzes. Bei dem Verleihen eines Kanals durch eine Mobilstation muß mindestens ein weiterer Nutzkanal gesperrt werden. Man kann die Auswirkung dieses Nachteils vermindern, indem den Kanälen Ordnungsnummern zugeordnet werden (BCO, *Borrowing with Channel Ordering*). Die aufgebauten Verbindungen belegen Kanäle mit möglichst niedriger Ordnungsnummer. Kanäle mit hoher Ordnungsnummer werden zuerst geliehen. So können geliehene Kanäle wieder zurückgegeben werden, bevor sie in den Zellen, in denen sie gesperrt sind, gebraucht werden. Die Umordnung einer bestehenden Verbindung auf einen Kanal mit niedrigerer Ordnungsnummer wird als *Intracell-Handover* bezeichnet. Weitere Varianten des BCO-Verfahrens sind BDCL (*Borrowing with Directional Channel Locking*), FBCA (*Forced Borrowing Channel Assignment*) und ABCO (*Adaptive Borrowing Channel Ordering*). Beim BDCL-Verfahren wird die Richtung, in die ein Kanal verliehen wird, bei der Sperrung berücksichtigt. Das FBCA-Verfahren dehnt den Leihvorgang über die erste Reihe der Nachbarzellen hinaus aus. Schließlich verhält sich das ABCO-Verfahren wie die HCA-Verfahren, indem von einer festen Grundverteilung ausgegangen wird.

Eine zweite Alternative der verkehrsadaptiven Verfahren ist durch einen Frequenz-Pool realisiert, aus dem sich alle Zellen unter Berücksichtigung der Randbedingungen bedienen können. Das FA-Verfahren (*First Available*) vergibt den ersten zu vergebenden Kanal, der weder in der eigenen Zelle noch in den Zellen innerhalb des erlaubten Frequenzwiederholabstandes benutzt wird.

3.2.2 Signaladaptive Verfahren

Signaladaptive Verfahren nutzen die Tatsache daß, eine sich in der Nähe einer Basisstation befindende Mobilstation weniger anfällig gegen Gleichkanalstörungen ist als eine, die sich in größerer Entfernung befindet [BEN96]. Bei signaladaptiven Verfahren wird die Zellstruktur in einen inneren und einen äußeren Bereich unterteilt, die unterschiedliche Frequenzwiederholabstände haben dürfen (*Reuse Partitioning*). In der folgenden Abbildung ist ein Beispiel einer Zellunterteilung dargestellt. Die inneren Zellbereiche A_x werden in dreier Clustern zusammengefaßt, die äußeren B_x in siebener Clustern.

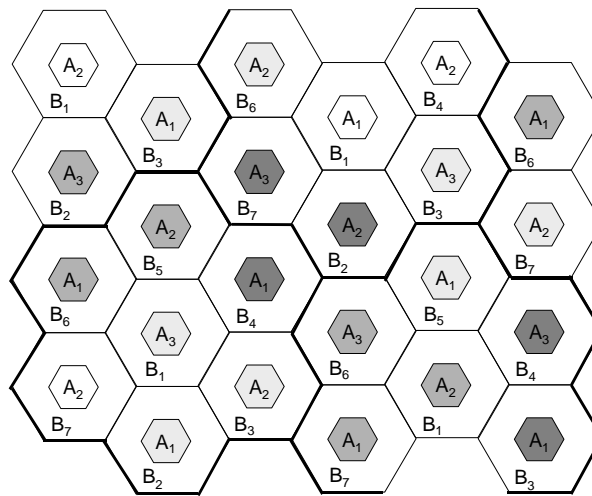


Abb. 7: Reuse Partitioning [BEN96]

Das „*Selection with Maximum Usage on the Reuse Ring*“ (RING) ist ein weiteres signaladaptives Verfahren. Dabei stehen jeder Zelle alle verfügbaren Kanäle zur Verfügung. Wenn ein Kanal belegt werden soll, wird der auf dem Frequenzwiederholabstandsring (Reuse Ring) am meisten eingesetzte Kanal ausgesucht. Die Anzahl gleicher Kanäle auf dem Reuse Ring wird dadurch optimiert. Dieses verfahren zeigt Grenzen bei geringer Belastung des Reuse Ringes und zufälliger Kanalvergabe. In dem Fall nutzt dieses Verfahren nicht immer die maximal mögliche Netzkapazität aus.

3.2.3 Interferenzadaptive Verfahren

Die Verteilung der Kanäle bei den interferenzadaptiven Verfahren wird aufgrund von Messungen der Interferenzen realisiert. Dabei steht jeder Basisstation der gesamte Vorrat an Frequenzen zur Verfügung. Eine Ressourcenvergabe findet erst bei der tatsächlichen Kanalvergabe statt. Man kann zwischen zentralen und dezentralen Verfahren unterscheiden.

Bei den zentralen Verfahren werden die von den Mobilstationen gemessenen Pfadverluste zu den Basisstationen zu einer zentralen Instanz gesendet, die daraufhin eine Verbindung mit einem ausreichenden Frequenzwiederholabstand aussucht. Dabei wird gewährleistet, daß die ausgewählte Verbindung möglichst geringe Störungen zu den bestehenden Verbindungen verursacht.

Bei den dezentralen Verfahren entscheiden die Mobilstationen autonom über die Kanalvergabe. Im folgenden werden einige dezentrale interferenzadaptive Verfahren kurz aufgezählt. Eine umfassende Beschreibung der Verfahrensalgorithmen ist in [GER98] zu finden:

- *SCS (Sequential Channel Search)*

Bei einem Belegungswunsch testet das Verfahren die für die Mobil- und Basisstation verfügbaren Kanäle und weist dem Belegungswunsch den Kanal mit dem akzeptabelsten Interferenzpegel zu.

- *MSIR (Minimum Signal-to-Noise)*

Das verfahren sucht für eine Verbindung den Kanal mit dem größten Frequenzwiederholabstand aus.

- *CSEG (Channel Segregation)*

Bei diesem Verfahren wird für jede Basisstation eine Kanalliste geführt, in der die Kanäle in der Reihenfolge ihrer Störanfälligkeit aufgelistet werden. Diese gewisse Priorität der Kanäle wird über die gesamte Vergangenheit bestimmt. Die Kanäle mit der schlechteren Qualität werden nach unten geschoben, während die, die wenig Störungen aufweisen, nach oben rücken und damit zuerst belegt werden.

3.2.4 Proaktive Verfahren: CMRA und STBR

Im folgenden wird auf die im IANT implementierten Ressourcenverteilungsverfahren, die in späteren Arbeiten als Vergleichsbasis mit den im KBS-Institut entwickelten Verfahren dienen werden, kurz eingegangen. Eine ausführliche Beschreibung dieser beiden Verfahren findet man in [FET00].

Das erste Verfahren basiert auf einem Regelkreis (CMRA-Verfahren, *Control Method based Resource Allocation Algorithm*), das zweite Verfahren auf einer schwellenbasierten Anforderung neuer Ressourcen (STBR-Verfahren, *Simple Threshold Based Resource*

allocation algorithm). Beide Verfahren sind aus der Anforderung, daß sich das gesamte Netzmanagement proaktiv verhalten soll, entwickelt worden.

Beim CMRA-Verfahrens basiert der Ressourcen-Anforderungsalgorithmus auf einem Standard-Regelkreis, der an das Problem der Ressourcenverteilung angepaßt wird (Abb. 7).

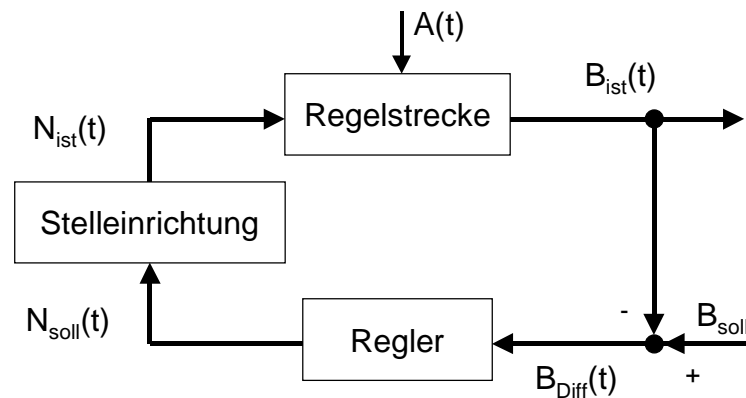


Abb. 8: Regelkreis CMRA-Verfahren [FET00]

B : Erlang'scher Verlust.

A : Angebot.

N : Anzahl der Kanäle.

Das STBR-Verfahren ist ein Verfahren, das auf den gleichen Prinzipien wie das CMRA-Verfahren basiert und sich auch durch einen Regelkreis darstellen läßt (Abb. 9). Dieses Verfahren fordert genau dann einen neuen Kanal an, wenn die Belegungsquote eine bestimmte Schwelle überschreitet.

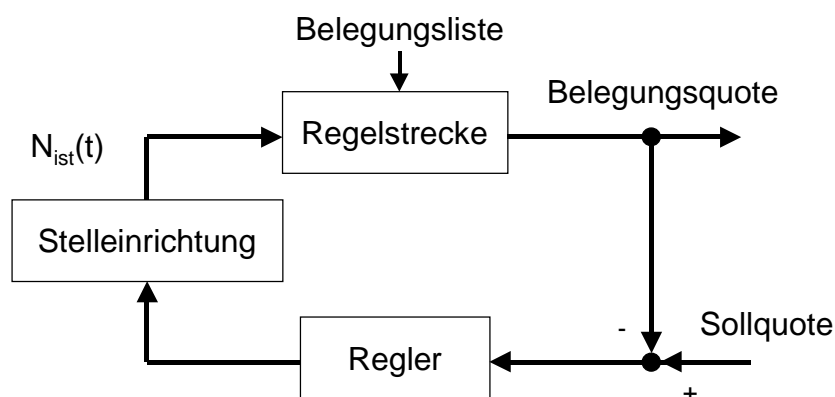


Abb. 9: Regelkreis STBR-Verfahren [FET00]

Beide Verfahren benutzen eine Kostenfunktion, die die Kosten für den geplanten Leihvorgang abhängig vom Verkehr, von der Kanalanzahl und von einer Gewichtung der Kosten errechnet.

Die Kostenfunktion für den Leihvorgang hat folgende Form:

$$K_{Leih}(t) = \sum_i w_i B_{i,neu}(t) - \sum_i w_i B_i(t) \quad (\text{Gl. 3.1})$$

i : Index für die jeweils beteiligten Zellen (Akzeptor oder Donatoren).

w_i : Gewichte, die den Einfluß der Zellen auf den Gesamtkosten $K_{leih}(t)$ bestimmen.

$B_i(t)$, $B_{i,neu}(t)$: Erlangsche Verluste der Zellen vor und nach dem Leihvorgang.

Der Leihvorgang wird nur dann ausgeführt, wenn folgende Bedingung erfüllt wird:

$$K_{Leih}(t) \leq 0 \quad (\text{Gl. 3.2})$$

Beim CMRA-Verfahren werden geliehene Ressourcen nach einer Mindestverweildauer nur dann zurückgegeben, wenn sie nicht belegt sind und der Verlust an der aktuellen Zelle kleiner ist als in ihrer Ursprungszelle.

Beim STBR-Verfahren kann zum einen ein geliehener Kanal zurückgegeben werden, wenn er nicht belegt ist, zum anderen, wenn dazu die Belegungsquote an der aktuellen Zelle kleiner ist als an der Ursprungszelle.

4 Marktbasierte Regelung (Market Based Control)

Die Erfahrung im Management der Ressourcen in großen und komplexen Systemen hat gezeigt, daß hierarchische Managementstrukturen eher ungeeignet sind. Dies insbesondere dann, wenn die einzelnen Ressourcenbesitzer einen hohen Grad an Autonomie aufweisen. Zentrale Verwaltungsinstanzen haben in solchen Situationen Mühe, eine optimale und flexible Nutzung der Ressourcen zu garantieren, da sie selber meist ungenügend über den sich permanent ändernden, globalen Systemzustand orientiert sind.

Eine mögliche Organisationsform, die sich für die Koordination der Ressourcenverteilung und -nutzung in offenen Systemen anbietet, sind Märkte.

Unter einem Markt versteht man ein System, in dem eine Anzahl von Komponenten lokal interagieren, um ein globales zusammenhängendes Verhalten zu erreichen [CLE96]. Es ist ein System, das Begriffe wie Dezentralisation, agierende Agenten und zu verteilende Ressourcen umfaßt.

Ein Markt basiert auf Kauf/Verkauf Transaktionen, durch die ein gesamter Effekt erreicht wird. Dabei verfügen die einzelnen Agenten nur über die für sie relevanten Daten und Informationen. Sie brauchen nicht den kompletten Zustand des Systems zu kennen. Aus diesem wichtigen Grund bietet sich der marktbasierter Ansatz als dezentrales Verfahren für die Regelung und Verteilung der Systemressourcen an. Die wichtigsten Anforderungen an der marktbasierter Regelung (*market based control*) sind [BOR97]:

- Das System muß sich an den sich ändernden Gegebenheiten und Zuständen anpassen.
- Vermeiden der Kosten und Risiken der zentralisierten Kontrolle.
- Nutzen der Vorteile der dezentralisierten Entscheidungen.
- Effizienz: Das gesamte Verhalten des Systems soll dadurch optimiert werden.

Der marktbasierter Ansatz zeigt aber Grenzen. Das angestrebte Systemverhalten und die Ressourcenverteilung sind nicht unbedingt optimal. Zentralisierte Verfahren können prinzipiell bessere Ergebnisse erreichen, weil sie beim Treffen von Entscheidungen ein breites Wissen über das gesamte System anwenden.

Wie in realen Märkten, werden marktbasierte Systeme durch Handelsbeziehungen bestimmt. Es wird versucht reale Marktgesetze und -verfahren an komplexen Systemen anzuwenden. Weil es unnötig ist, die Realität völlig exakt nachzubilden, wird auf einige Aspekte des realen Markts verzichtet. In einem realen Markt versucht jeder Beteiligte seine Gewinne zu maximieren, auch wenn er dabei den anderen hohe Schäden verursacht. Es wird getäuscht, geblufft und gelogen. In einem durch marktbasierte Verfahren kontrollierten System wird dagegen fair und ehrlich gehandelt. Es wird sogar von ehrlichen Agenten und transparenten Handelsaktionen gesprochen. Jeder Agent versucht zwar seine eigenen Gewinne zu optimieren, strebt aber dabei unter Berücksichtigung aller anderen Agenten nach einem möglichst hohen Gesamtgewinn des Systems. Ziel ist immerhin die Verbesserung des Gesamtgewinns. Man könnte hier von einem föderalen System sprechen, in dem alle beteiligten Agenten für das Interesse des gesamten Systems agieren.

Die Voraussetzung einen virtuellen Markt zu schaffen, ist das Vorhandensein von Käufern, Verkäufern, Preisen, Zahlungsmitteln, Angebot und Anfrage. Das sind die grundlegenden Aspekte eines realen Markts.

In virtuellen Märkten wird von virtuellem Geld als Zahlungsmittel gesprochen, wobei das rein virtuelle Geld in manchen Fällen einen engen Zusammenhang mit dem realen Geld haben könnte. Im Fall des Mobilfunknetzes könnte das Zahlungsmittel für Ressourcen einen engen Bezug zu den durch sie eingegangenen Gesprächsgebühren haben.

Damit ein Handel zwischen den Komponenten des Systems möglich wird, muß ein Protokoll festgelegt werden, das die Handelsbeziehungen und die verschiedenen Aktionen und Abläufe innerhalb eines Markts abbildet. Allgemein gibt es zwei Möglichkeiten, nach denen sich die Preise bestimmen lassen, und der Handel stattfindet. Entweder treten die Handelspartner direkt miteinander in Kontakt, oder es findet eine Auktion statt, bei der ein zusätzlicher, am Handel unbeteiligter Agent die Rolle des Auktionators übernimmt. Ein wichtiger Grundsatz für alle Arten von Auktionen ist, daß kein Verkäufer zu einem Preis unter seinem Angebot verkauft und kein Käufer zu einem Preis über seinem Gebot kauft. Im folgenden werden die verschiedenen Auktionsarten vorgestellt [AGO00].

4.1 Auktionen

Unter Auktion versteht man eine auf Konkurrenz basierende Methode zur Vergabe knapper Ressourcen, bei der der Verkäufer so viel wie möglich verdienen und der Käufer so wenig

wie möglich ausgeben möchte. Auktionen sind effiziente Mittel, um die knappen Ressourcen nahezu optimal zu verteilen. Das ausgewählte Verhandlungsprotokoll beeinflusst stark die Markteigenschaften. Die durch den Einsatz von Auktionen erreichten Vorteile sind:

- Einfaches Bestimmen der marktbasieren Preise
- Flexibilität statt festgesetzte Preisen
- Zeitersparnis durch ein bestimmtes Verhandlungsprotokoll im Vergleich zu zufälligen Verhandlungen
- Garantie der Zuteilung einer Ressource zu dem höchst bietenden Agenten

Während einer Auktion verfügen Käufer und Verkäufer über bestimmte Informationen, die sie zur Bewertung und Einschätzung der Lage nutzen. Diese Informationen können symmetrisch oder asymmetrisch sein d.h., die Anbieter haben die gleichen Informationen oder nicht. Darüber hinaus wird eine Auktion von der Rolle des Auktionators beeinflusst. Je nachdem, wie die Verhandlungen stattfinden und wie stark die Strategien der Käufer, Verkäufer und des Auktionators die Auktion prägen, werden viele Auktionsarten unterschieden.

William Vickrey [AGO00] hat eine grundlegende Systematik der Auktionen aufgestellt, die auf der Reihenfolge, in der Preise und Angebote betrachtet und bearbeitet werden, basiert. Er stellte vier hauptsächliche Auktionsarten auf. Im folgenden wird auf die wichtigsten Auktionsarten kurz eingegangen.

4.1.1 English Auction

Die „English Auction“ ist die geläufigste Auktionsart und ist bekannt als „open-outcry Auction“ oder „ascending-price Auction“. Der Auktionator eröffnet die Auktion mit dem niedrigsten zulässigen Preis, „the reserve price“, und wartet auf höhere Gebote von den potentiellen Käufern. Der Preis steigt, bis kein Käufer mehr das Gebot erhöht. Der zu verkaufende Gegenstand wird dann dem Käufer mit dem höchsten Gebot zugewiesen. In machen Fällen wird der Gegenstand überhaupt nicht verkauft, weil das Gebot einen bestimmten Wert nicht erreicht hat. Dies könnte geschehen, wenn kein „reserve price“ festgelegt wurde.

4.1.2 Dutch Auction

Bei der „Dutch Auction“, auch bekannt als „descending-price Auction“, fängt der Auktionator im Gegensatz zur „English Auction“ mit einem extrem hohen Preis an. Dieser Preis wird dann

gesenkt, bis ein potentieller Käufer zufrieden mit dem vorgeschlagenen Preis ist. Dieser meldet also seinen Wunsch, den Gegenstand mit dem vereinbarten Preis zu kaufen.

Der Vorteil für den Verkäufer gegenüber die „English Auction“ ist, daß der Käufer immer zu einem für ihn maximalen Preis die Ware kauft. Weil der interessierte Käufer kein Risiko eingehen will, wird er natürlich nicht lange warten und gleich beim ersten für ihn zulässigen Preis zusagen. Bei einer „English Auction“ kann der Käufer eine Ware zu einem für ihn relativ geringen Preis kaufen, obwohl er theoretisch mehr zahlen könnte.

4.1.3 First-Price, Sealed Bid Auction

Die dritte Auktionsart hat die Eigenschaft, daß sie versiegelt (sealed) ist, d.h. ein gewinnender Käufer zahlt genau den Preis, den er geboten hat. Jeder Käufer bietet unabhängig von den anderen Konkurrenten einen Preis an. Dabei versucht er die Güter zu bewerten. Über Informationen über die anderen potentiellen Käufer verfügt er nicht. Eine „Sealed Bid Auction“ hat zwei Phasen: eine Gebotsphase, in der die Beteiligten ihre Gebote einreichen, und eine Resolutionsphase, in der die Gebote veröffentlicht werden, und der Gewinner bestimmt wird. In dem Fall, bei dem viele Einheiten zu verkaufen sind, und die Käufer nicht alle Einheiten brauchen, werden viele Käufer gewinnen. In diesem Fall spricht man von einer „Discriminatory Auction“ und nicht von einer „First-Price Auction“. Die Gebote werden vom höchsten zum niedrigsten sortiert und dann bearbeitet. Jeder Käufer zahlt sein Gebot, bis die Einheiten erschöpft sind. Wichtig hier ist, daß gewinnende Käufer zu verschiedenen Preisen Güter kaufen können.

Vom Standpunkt des Käufers aus, wird ein hohes Gebot seine Gewinnchancen erhöhen aber seinen Profit im Fall einen Gewinnes senken.

4.1.4 Vickrey Auction (Uniform second-price)

Die „Uniform second-price Auction“, bekannt als „Vickrey Auction“, ist nach William Vickrey, der den Nobelpreis für Wirtschaftswissenschaft 1996 erhielt, benannt.

Wie bei der „first-price Auction“ werden die Gebote versiegelt, und die Käufer ignorieren die anderen Gebote. Der Unterschied ist, daß der gewinnende Käufer nicht sein Gebot zahlt, sondern das zweithöchste Gebot. In anderen Worten zahlt ein Gewinner weniger als den von ihm vorgeschlagenen Preis. Ein interessanter Punkt ist, daß in Auktionen für mehrere Einheiten alle gewinnende Käufer denselben Preis pro Einheit zahlen (das höchstverlierende Gebot).

4.1.5 Double Auction

Die Auktionsart ist eigentlich nicht klassifiziert. Sie wird aber seit Jahrhunderten in den amerikanischen Finanzinstitutionen angewendet. In dieser Auktion reichen sowohl Käufer ihre Gebote als auch Verkäufer ihre Angebote ein. Diese werden vom höchsten zum niedrigsten sortiert, um Profile der Nachfrage und Angebot zu erhalten. Von diesen Profilen wird das Maximum an Transaktionen und Austauschen erreicht, nachdem die Preisspanne für alle Transaktionen geschätzt wird. Dafür werden Angebote (beginnend mit dem niedrigsten Preis) mit Geboten (beginnend mit dem höchsten Preis) verglichen. Die geschätzte Preisspanne liegt dann zwischen dem Angebotspreis der letzt möglichen Transaktion und dem Gebotspreis der ersten Transaktion, die nicht ausgeführt werden kann [AGO00].

4.2 Preisgestaltung

Es gibt für die Ressourcenmanager hauptsächlich zwei Methoden, den Preis der Ressourcen zu ermitteln. Entweder wird bei der Initialisierungsphase anhand von Festpreisgestaltungsstrategien (*fixed-pricing*) ein fester Preis der Ressourcen festgelegt, der dann möglicherweise durch menschliche Manager geändert wird, oder es werden die Preise ermittelt, indem dynamische Preisgestaltungsstrategien angewendet werden. Diese passen die Preise an den aktuellen Zustand des Systems (Angebot und Nachfrage) an.

Die dynamische Preisgestaltung ist aus verschiedenen Gründen wichtig. Erstens wird davon ausgegangen, daß manuell festgelegte Preise nicht zu einem stabilen und zuverlässigen System führen können. Zweitens wird erwartet, daß sich der Systemzustand ständig ändert, so daß statische Preise nicht mehr der aktuellen Lage entsprechen würden und von daher inkonsistent wären

Es gibt vier verschiedene Ansätze, die der dynamischen Preisgestaltung zugrunde liegen [BKR97]:

1. Verkäuferangepaßte Preisgestaltung: Die Verkäufer setzen ihre Preise abhängig von den Käufern fest. Preise steigen bei hoher Anfrage und sinken bei niedriger Anfrage. Die Preise der Konkurrenten werden dabei berücksichtigt.
2. Verhandlung: Bei diesem Ansatz verhandeln Käufer und Verkäufer bei jeder möglichen Transaktion. Dieser Ansatz kann mit Hilfe der „Double Auction“ implementiert werden.
3. Verkäuferbasierte Auktionen: Auktionen sind ausführbar, wenn mehrere Käufer um die gleichen Ressourcen konkurrieren besonders, wenn die Anfrage größer als die Ressourcenmenge ist.

4. Käuferbasierte Auktionen: Bei diesen Auktionen gibt es mehrere Verkäufer und relativ wenige Käufer. Die Verkäufer verkünden ihre Preise, und die Käufer wählen die günstigsten Angebote.

5 Das realisierte marktbasierende Ressourcenverteilungsverfahren

5.1 Marktbasierendes Modell für das Mobilfunknetz

Bei der derzeit verwendeten statischen Ressourcenverteilung in Mobilfunknetzen werden den einzelnen Basisstationen nach einer aufwendigen Netzplanung Frequenzen zugewiesen. Jede Basisstation verfügt über eine Anzahl von Frequenzen, auf denen sie die Verbindungen unterbringen kann. Wenn alle Frequenzen in einer Zelle belegt sind, werden die ankommenden Verbindungswünsche abgewiesen, auch wenn in der Nachbarzelle noch freie Frequenzen oder Kanäle sind. Diese Problematik könnte durch den Einsatz von marktbasierenden Verfahren als dynamische Ressourcenverteilungsverfahren gelöst werden.

Man könnte sich das Mobilfunknetz als Markt vorstellen, auf dem die Frequenzen oder Kanäle die Güter oder die Ware darstellen. Die Zellen repräsentieren Verkäufer und Käufer. Als Folge daraus finden die Verhandlungen zwischen Zellen statt. Frequenzen werden dann durch Marktgesetze verteilt. Abhängig vom Verkehr im Netz stellt sich ein Nachfrage/Angebot Zustand ein. Die Zellen sollen abhängig von ihrem Bedarf und ihrer Kaufkraft nach einem bestimmten Protokoll verhandeln, Frequenzen kaufen und verkaufen, so daß die Ressourcenverteilung optimiert wird und dadurch der Gewinn des Netzbetreibers maximiert.

In den durchgeführten Simulationen besteht jede Zelle aus drei Sektoren, die sich autonom wie eine einzelne Zelle verhalten. Die tatsächlichen Verhandlungen finden dann eine Hierarchie tiefer statt, sprich zwischen den Sektoren. In den nächsten Abschnitten wird der Einfachheit halber von Zellen statt Sektoren gesprochen.

5.2 Ressourcenbewertung

Um effiziente Ergebnisse zu erzielen und dafür zu sorgen, daß Handelsbeziehungen optimal verlaufen, muß die Preisgestaltung gut einstudiert werden. Die Ressourcen, in diesem Fall die Frequenzen, sollen richtig geschätzt und bewertet werden. Jede Zelle muß genau wissen, was ihre Frequenzen kosten und dadurch einen Preis ermitteln. Die Preise der Frequenzen sollen

einen engen Zusammenhang mit dem Gewinn, den sie durch Gespräche erbringen könnten, haben.

Je höher der Preis einer Frequenz ist, desto größer ist der durch sie eventuell erreichbare Gewinn. Durch eine optimierte Kostenfunktion kann der Preis einer Frequenz bestimmt werden. Die Kostenfunktion soll möglichst viele Randbedingungen in Kauf nehmen, um nach den Transaktionen unnötige Signalisierungen zu vermeiden. Darüber hinaus darf die Kostenfunktion nicht zu einem instabilen Netz führen. Daß die Frequenzen hin und her verkauft werden, sollte vermieden werden.

Im folgenden wird genauer auf die in dieser Arbeit eingesetzten Kostenfunktion, durch die der Preis einer Frequenz ermittelt wird, eingegangen.

Durch die Integration verschiedener Bewertungsgrößen, die das Mobilfunknetz jeweils von einem anderen Gesichtswinkel betrachten, setzt diese Kostenfunktion die Prinzipien der verkehrsadaptiven und interferenzadaptiven Ressourcenverteilungsverfahren durch. Sie enthält nämlich ein *Verkehrsglied*, das sich auf die Blockierungswahrscheinlichkeiten bezieht, und ein *Störglied*, das ein Maß für die Qualität der Frequenz und für die durch die Nachbarzellen verursachten Störeffekte darstellt.

Darüber hinaus zeichnet sich die Kostenfunktion dank der durchgeführten Prädiktion durch einen proaktiven Aspekt aus. Es wird agiert statt reagiert. Das Verfahren erkennt den Bedarf einer Zelle und bewertet durch eine Vorhersage den eventuellen neuen Zustand im Falle einer Frequenztransaktion. Abhängig von dieser Prädiktion wird entschieden, ob die Transaktion tatsächlich ausgeführt wird.

Der letzte Aspekt der Kostenfunktion ist der Einsatz eines *Reichtumsgliedes*. Durch diese Bewertungsgröße gewinnt das Verfahren die Eigenschaft, die Ressourcen fair und ausgeglichen zu verteilen. Eine gewisse Priorität wird zwischen den Zellen beachtet: die reicheren Zellen benehmen sich eher als Donatoren und die ärmeren als Akzeptoren. Wenn zwei Zellen, die den gleichen Bedarf zeigen, um die selbe Ressource konkurrieren, wird die ärmere bevorzugt.

Die Formel zur Preisgestaltung läßt sich durch die folgende Kostenfunktion definieren.

$$P_{i,f} = w_B \cdot \underbrace{\left[B_{i,f \notin i}(N_{i,f \notin i}, A_i) - B_{i,f \in i}(N_{i,f \in i}, A_i) \right]}_{\text{Verkehrsglied}} - w_R \cdot \underbrace{(N_i - V_i)}_{\text{Reichtumsglied}} - w_I \cdot \underbrace{I_f}_{\text{Störglied}} \quad (\text{Gl. 5.1})$$

oder anders ausgedrückt

$$P_{i,f} = w_B \cdot |B_i(t + \Delta t) - B_i(t)| - w_R \cdot (N_i - V_i) - w_I \cdot I_f \quad (\text{Gl. 5.2})$$

- $P_{i,f}$: Der von Zelle i berechnete Preis für die Frequenz f . Es spielt keine Rolle, ob die Frequenz der Zelle schon gehört oder nicht. Die Kostenfunktion gilt für beide Fälle. Der berechnete Preis ist also entweder der angebotene oder der gebotene Preis, je nachdem, ob die Frequenz verkauft oder gekauft wird.
- w_B : Gewicht für das Verkehrsglied.
- $B_{i,f \in i}$: Erlang'scher Verlust der Zelle i , wenn die Frequenz f mitbetrachtet wird.
- $B_{i,f \notin i}$: Erlang'scher Verlust der Zelle i , wenn die Frequenz f nicht mitbetrachtet wird.
- B_i : Erlang'scher Verlust der Zelle i .
- t : Aktuelle zeit.
- Δt : Dauer eines Zeitschrittes.
- $N_{i,f \in i}$: Anzahl der Kanäle der Zelle i mit der Frequenz f .
- $N_{i,f \notin i}$: Anzahl der Kanäle der Zelle i ohne die Frequenz f .
- A_i : Angebot in der Zelle i .
- w_R : Gewicht für das Reichtumsglied.
- N_i : Anzahl der Kanäle in der Zelle i .
- V_i : Anzahl der belegten Kanäle (bestehende Verbindungen + Signalisierungskanäle + reservierte Kanäle) in der Zelle i .
- w_I : Gewicht für das Störglied.
- I_f : Störglied der Frequenz f (Maß für die bestehende oder eventuelle Störung durch interferierende Frequenzen).

Für die Käuferzellen gelten folgende Gleichungen

$$B_i(t) = B_{i,f \notin i}(N_{i,f \notin i}, A_i) = B_i(N_i, A_i) \quad (\text{Gl. 5.3})$$

$$B_i(t + \Delta t) = B_{i,f \in i}(N_{i,f \in i}, A_i) = B_i(N_i + 1, A_i) \quad (\text{Gl. 5.4})$$

und für die Verkäuferzellen folgende

$$B_i(t) = B_{i,f \in i}(N_{i,f \in i}, A_i) = B_i(N_i, A_i) \quad (\text{Gl. 5.5})$$

$$B_i(t + \Delta t) = B_{i,f \notin i}(N_{i,f \notin i}, A_i) = B_i(N_i - 1, A_i) \quad (\text{Gl. 5.6})$$

Das Angebot wird für den kurzen Zeitraum als konstant angesehen

$$A_i(t + \Delta t) = A_i(t) = A_i \quad (\text{Gl. 5.7})$$

Das Störglied soll ein Maß für die Qualität der Frequenz sein. Es sagt aus, wie stark die Störfrequenzen der anderen Zellen die betrachtete Frequenz beeinträchtigen. Die

Interferenzstörung steht in einem direkten Zusammenhang zu den Abständen zwischen den betroffenen Zellen. Das Störglied läßt sich durch die folgende Formel beschreiben.

$$I_f = \sum_{\text{Störzellen}} (1/d)^4 \quad (\text{Gl. 5.8})$$

d ist der Abstand zwischen der betrachteten und der störenden Zelle. Aus der Formel ist zu ersehen, daß das Störglied um so größer bzw. kleiner wird, je näher bzw. weiter die Störstationen von der betrachteten Zelle sind.

Die vorgestellte Formel wurde von der folgenden Formel hergeleitet, die in Kapitel 2.2 erläutert wurde:

$$\frac{C}{I} = W \approx \frac{R^{-\alpha}}{\sum_i d_i^{-\alpha}} \cong \frac{1}{\sum_i d_i^{-\alpha}} \quad (\text{Gl. 5.9})$$

Für die Auswahl des Parameters α wurde von der folgenden Formel, die den Zusammenhang zwischen der Sende- und Empfangsleistung darstellt [WAL00], ausgegangen:

$$P_E = P_S \left(\frac{h_E h_S}{d^2} \right)^2 G_S G_E . \quad (\text{Gl. 5.10})$$

Wenn man der Einfachheit halber davon ausgeht, daß die Antennengewinne G_S und G_E , die Höhen von Sende- und Empfangsantenne h_S und h_E und die Sendeleistung P_S konstant für das ganze Netz sind, fällt auf, daß die Empfangsleistung mit der Entfernung mit vierter Potenz abnimmt.

Die Gewichte der Kostenfunktion erteilen dem Netzmanager sehr hohe Freiheitsgrade. Abhängig von der Strategie und den Zielen des Netzbetreibers können die Gewichte so festgelegt werden, daß das gewünschte Resultat erreicht wird. Wenn der Netzbetreiber mehr Wert auf die Qualität der Dienste legt (QoS), muß das Gewicht für das Störglied erhöht werden, so daß die Käuferzellen ganz wenig für eine schlechte Frequenz bieten. Dadurch werden Transaktionen vermieden, die zu einer Verschlechterung der Netzqualität führen könnten. Wenn sich der Netzbetreiber aber eher für die Erhöhung der Performance des Netzes und dadurch des Gewinns interessiert, muß man das Gewicht des Verkehrsgliedes vergrößern. Man kann aber auch einen Kompromiß zwischen allen Aspekten finden, indem die Gewichte für ein bestimmtes Ziel optimiert werden.

Die Grundideen, die mich zur Auswahl der vorgestellten Kostenfunktion gebracht haben, sind Flexibilität und Integration. Man kann das gesamte Verhalten des Netzes in eine bestimmte Richtung ändern, indem einfach die Gewichte einer einheitlichen Kostenfunktion entsprechend angepaßt werden.

5.3 Auktionen

Die Verhandlungen zwischen den Zellen werden durch eine Kombination von „Double Auction“ und käuferbasierten Auktionen realisiert. Im folgenden werden die wichtigsten Schritte des Algorithmus, auf dem die Verhandlungen basieren, grob erklärt. Eine detaillierte Beschreibung der Funktionsweise des Verfahrens erfolgt in den nächsten Abschnitten.

5.3.1 Aufbau der Transaktionstabelle

Es wird eine Schleife über alle Zellen durchlaufen. Jede Zelle verlangt von ihren Nachbarzellen eine Preisliste der Frequenzen, die sie besitzen. Für alle Frequenzen, die sie selbst nicht besitzt, bietet die kaufende Zelle einen Preis. Wenn der gebotene Preis für eine Frequenz höher als der von der Nachbarzelle vorgeschlagene Preis ist, wird diese Transaktion gemerkt und in eine Transaktionstabelle eingetragen.

Die Transaktionstabelle ist eine sortierte Liste. Die Transaktion, die den größten Gewinn darstellt, steht ganz oben in der Liste. Der Gewinn ist die Differenz zwischen dem vom Käufer gebotenen Preis und dem vom Verkäufer vorgeschlagenen Preis. Er reflektiert die Wichtigkeit der Transaktion und ist ein Maß für den Gewinn, der für den Netzbetreiber durch diese Transaktion erreicht wird.

Natürlich kann man abhängig von der Großzügigkeit und der Dynamik des Verfahrens eine Schwelle für die Transaktionen festlegen. Eine Transaktion muß einen höheren Gewinn als die Schwelle aufweisen, damit sie überhaupt in die Transaktionstabelle eingetragen und damit in Betracht gezogen wird. Darüber hinaus macht es auch Sinn, eine maximale und minimale Anzahl von Frequenzen pro Zelle festzulegen, so daß nicht alle Ressourcen irgendwann bei einer Zelle landen oder, daß eine Zelle über keine Ressourcen für die Signalisierung verfügt.

Ein minimaler Vorrat an Ressourcen muß immer verfügbar sein.

Nach der Schleife ist die Transaktionstabelle vollständig. Sie enthält alle eventuellen Transaktionen, die in Frage kommen könnten. Am Anfang enthält die Transaktionstabelle Transaktionen, die nicht zusammen ausgeführt werden können, weil das Ausführen einer dieser Transaktionen die anderen sinnlos machen würde. Diese redundanten Transaktionen werden in den nächsten Schritten entfernt.

Wie oben erwähnt, sind die Transaktionen nach dem Gewinn sortiert. Jeder Eintrag enthält die Käuferzelle, die Verkäuferzelle, die Frequenz und der durch die Transaktion erreichte

Gewinn: (KZ, VZ, F, G). Die folgenden Abbildungen erklären an einem Beispiel den Inhalt und die Struktur der Transaktionstabelle:

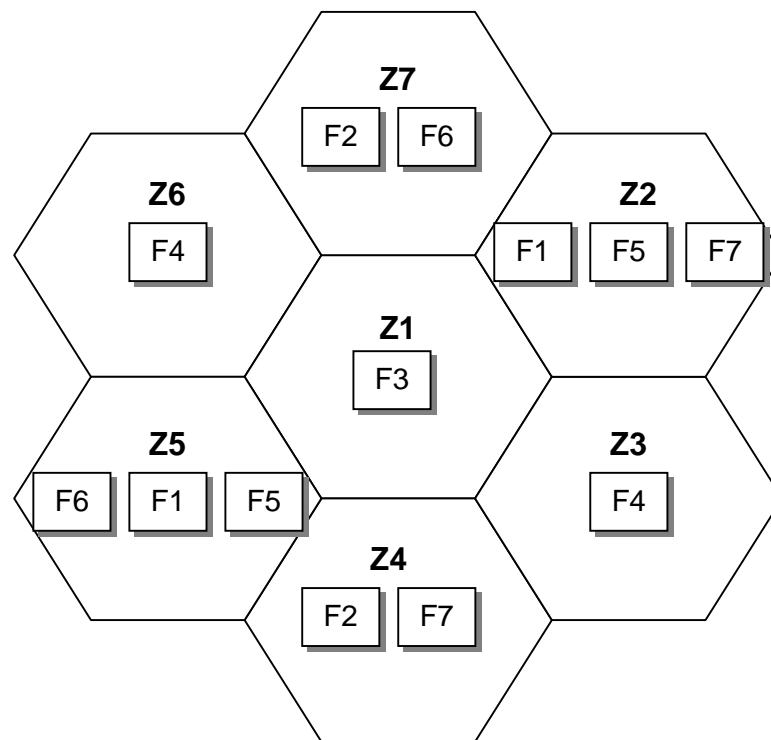


Abb. 10: Beispiel einer Verteilung

Käuferzelle	Verkäuferzelle	Frequenz	Gewinn
Z1	Z2	F1	100
Z1	Z4	F2	75
Z1	Z5	F1	74
Z2	Z7	F2	70
Z3	Z4	F7	60
Z3	Z2	F1	57
Z4	Z5	F1	50
Z3	Z2	F5	45
Z5	Z4	F7	35

Abb. 11: Transaktionstabelle

5.3.2 Bearbeitung der Transaktionstabelle

Die Transaktion, die ganz oben in der Transaktionstabelle steht, wird durchgeführt. Das wäre in diesem Fall (Abb. 11) die Transaktion (Z1, Z2, F1, 100). Die Frequenz wechselt den Besitzer. Die kaufende Zelle hat dadurch eine Frequenz mehr ($N_i \rightarrow N_i + 1$). Die verkaufende Zelle eine weniger ($N_i \rightarrow N_i - 1$).

Danach wird der durchgeführte Eintrag gelöscht, und anschließend wird die Transaktionstabelle gefiltert, indem die redundanten Einträge entfernt werden. Dies geschieht folgendermaßen:

Weil die kaufende Zelle Z1 jetzt über die Frequenz F1 verfügt, werden alle Einträge, in denen Z1 als Käufer und F1 als Frequenz stehen (Z1, X, F1, X), aus der Tabelle entfernt. X steht hier für einen beliebigen Ausdruck.

Folglich werden alle Einträge, in denen Z2 als Verkäufer und F1 als Frequenz stehen (X, Z2, F1, X), aus der Tabelle entfernt, denn Z2 verfügt nun nicht mehr über die Frequenz F1.

Käuferzelle	Verkäuferzelle	Frequenz	Gewinn
Z1	Z2	F1	100
Z1	Z4	F2	75
Z1	Z5	F1	74
Z2	Z7	F2	70
Z3	Z4	F7	60
Z3	Z2	F1	57
Z4	Z5	F1	50
Z3	Z2	F5	45

Abb. 12: Transaktionstabelle nach der ersten Transaktion und dem Filtern

Die in der Abbildung in grau dargestellten Einträge werden durch das Filtern von der Transaktionstabelle entfernt, wobei die in weiß dargestellten Ausdrücke Grund des Entfernens sind.

Nach dem Filtern muß die Transaktionstabelle auf den neusten Stand gebracht werden (Update), weil die ausgeführte Transaktion den Inhalt vieler mit ihr zusammenhängenden Transaktionen beeinflußt.

5.3.3 Update der Transaktionstabelle

Um das Update der Transaktionstabelle zu erläutern, wird wieder vom oben gezeigten Beispiel ausgegangen. Weil Z1 nach der Transaktion (Z1, Z2, F1, 100) eine Frequenz mehr hat, wird ihr Verlust kleiner und ihre Kanalzahl höher. Aus dem Grund müßte die Zelle ihre Frequenzpreise ändern. Die Zelle hat einen Teil ihres Bedarfs gedeckt und bietet jetzt als Käuferzelle für weitere Frequenzen weniger als sie vor dem Ausführen der Transaktion geboten hat. Als Verkäuferzelle ist Z1 bereit, ihre Ressourcen günstiger abzugeben, weil sie nun über mehr Ressourcen verfügt. Deswegen werden alle Einträge, in denen Z1 erscheint, wieder überarbeitet (Update der Gewinne).

Auf der anderen Seite hat Z2 nach der Transaktion eine Frequenz weniger. Deswegen wird ihr Verlust größer und ihre Kanalzahl niedriger. Aus dem Grund müßte sie ihre Frequenzpreise ändern. Die Zelle hat eine Ressource verloren und verlangt jetzt für weitere Frequenzen mehr als sie vor dem Ausführen der Transaktion als Verkäuferzelle verlangt hat. Als Käuferzelle ist Z2 bereit, neue Ressourcen teurer zu kaufen, weil sie jetzt über weniger Ressourcen verfügt. Deswegen werden alle Einträge mit Z2 erneut bearbeitet.

Dadurch, daß die durchgeführte Transaktion eine Frequenz verschiebt, ändert sich die Lage der Interferenzen, weil sich neue Störeffekte einstellen. Aus dem Grund müssen alle Einträge, die die Frequenz F1 enthalten, ebenfalls neu bearbeitet werden.

Käuferzelle	Verkäuferzelle	Frequenz	Gewinn
Z2	Z7	F2	74
Z1	Z4	F2	66
Z3	Z4	F7	60
Z4	Z5	F1	52
Z3	Z2	F5	37

Abb. 13: Transaktionstabelle nach dem Update

Die Gewinne der in der Abbildung in grau dargestellten Einträge werden während dem Update auf den neuen Stand gebracht, wobei die in weiß dargestellten Ausdrücke Grund des Updates sind.

Wenn der Gewinn einer Transaktion nach dem Update die festgelegte Schwelle unterschreitet oder, wenn die Käuferzelle bzw. die Verkäuferzelle ihre maximale bzw. minimale Anzahl von Frequenzen erreicht hat, wird diese Transaktion von der Tabelle entfernt.

Bei diesen Vorgängen wird natürlich die Sortierung der Tabelle ständig gewährleistet, damit immer die Transaktion mit dem höchsten Gewinn an erster Stelle ist und als nächste zur Bearbeitung bereit steht. Diese beiden Schritte (Bearbeitung und Update) laufen iterativ in einer Schleife, bis die Transaktionstabelle keinen Eintrag mehr enthält. Bei jedem Schritt wird die Länge der Tabelle kleiner, weil mindestens ein Eintrag entfernt wird, was ein terminiertes Verfahren garantiert.

Im folgenden werden alle ausgeführten Transaktionen in einer Tabelle zusammengefaßt und im Netzbild abgebildet, wobei die Pfeile die verschiedenen Transaktionen darstellen:

Käuferzelle	Verkäuferzelle	Frequenz	Gewinn
Z1	Z2	F1	100
Z2	Z7	F2	74
Z1	Z4	F2	66
Z4	Z5	F1	60
Z3	Z2	F5	45

Abb. 14: Die während der Auktion ausgeführten Transaktionen



Abb. 15: Abbildung der während der Auktion ausgeführten Transaktionen

Die folgende Abbildung dient zur Veranschaulichung des Ressourcenverteilungsverfahrens. Sie zeigt die Verteilung der Frequenzen über das Netz vor und nach der Auktion:

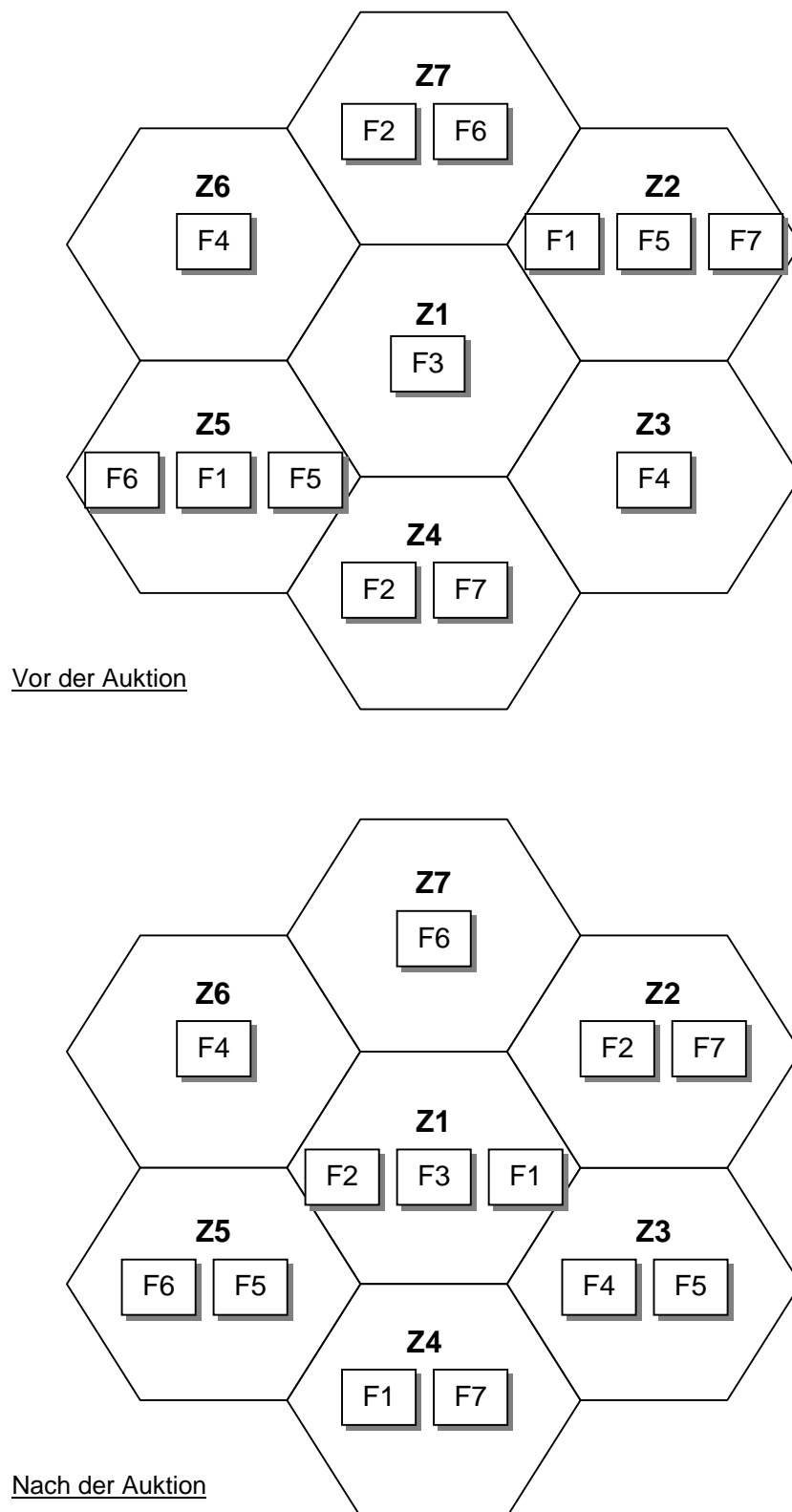


Abb. 16: Ressourcenverteilung vor und nach einer Auktion

6 Der Mobilfunksimulator MOSIT

6.1 Aufbau

Der Mobilfunksimulator MOSIT (Mobilfunk Simulationstool) wird am Institut für Allgemeine Nachrichtentechnik entwickelt. Er dient zur Simulation eines Mobilfunknetzes von den sich bewegenden Mobilstationen bis zur Luftschnittstelle.

Der Simulator wurde im Rahmen mehrerer Studien- und Diplomarbeiten in SDL (Specification and Description Language), eine graphische Programmiersprache, implementiert und wird immer noch weiterentwickelt. Die folgende Abbildung zeigt den groben Aufbau von MOSIT.

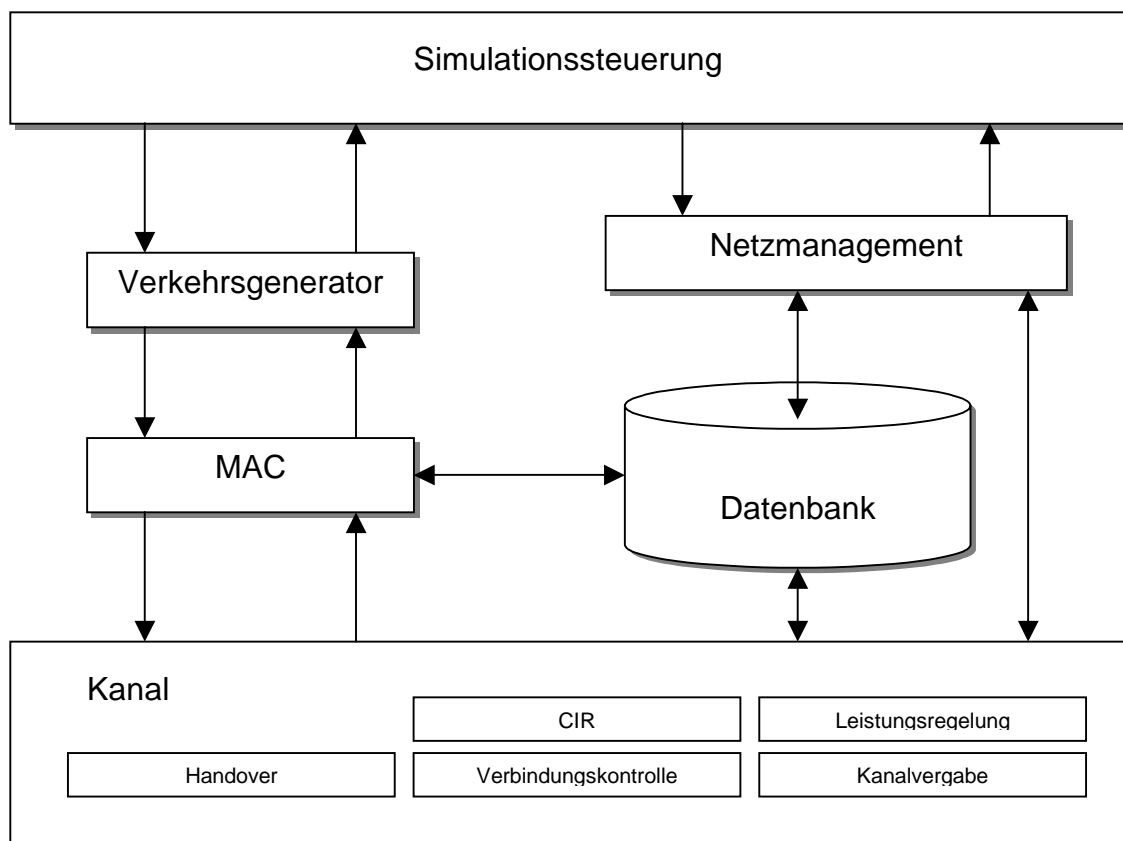


Abb. 17: Aufbau von MOSIT

Der Verkehrsgenerator erzeugt den Verkehr nach bestimmten Vorgaben. Er bestimmt, wann welche Mobilstation wie lange telefonieren möchte. Dabei kann er eine gewählte mittlere Belegungsdauer einhalten. Der Verkehrsgenerator kann einen statistisch verteilten Verkehr erzeugen, der einem bestimmten mittleren Angebot entspricht.

Der Block MAC entspricht der Schicht 2, der Media Access Control. In diesem Block werden die MACs der Mobil- und Basisstationen simuliert.

Der Block Kanal gewährleistet die Kommunikation zwischen den einzelnen MACs. Der Kanal simuliert einen Funkkanal mit seinen physikalischen Eigenschaften. Der Block Kanal ist in weitere Bereiche unterteilt. Der Übersichtlichkeit halber sind nur einige in der Abbildung gezeigt. Die Verbindungskontrolle z.B. überwacht ständig die Qualität der Verbindungen um eventuell einen Handover durch den Block Handover durchzuführen. Der Block CIR dient zur Berechnung der Qualität der Verbindungen über die CIR (Carrier to Interference Ratio) der Kanäle. Das erfolgt unter Verwendung eines Pfadverlustmodells und abhängig von Position von Mobil- und Basisstation.

Die Datenbank enthält alle Daten, die für die Simulation relevant sind, wie Zustand der Mobil- und Basisstationen, Frequenzzuordnungen, Verbindungslisten, Antennendaten etc.

Der letzte Block Netzmanagement ist der Teil von MOSIT, der in dieser Diplomarbeit um das marktbasierete Verfahren erweitert worden ist. Aus diesem Grund ist es von Vorteil, die Aufgaben dieses Blocks zu untersuchen. Die eigentliche Implementierung und Integration des Verfahrens in diesen Block wird in den nächsten Kapiteln beschrieben.

6.2 Der Netzmanagement-Block

In diesem Kapitel werden die wichtigsten Aufgaben des Netzmanagementblocks, die für die Diplomarbeit relevant sind, erläutert. Auf eine ausführliche Behandlung aller Details wird der Übersichtlichkeit halber verzichtet. Für eine umfassende Beschäftigung mit den Details wird auf [SCH00] verwiesen.

Der Block Netzmanagement besteht aus den drei Prozessen Netzmanagement_Steuerung, NM_Area_Manager und BS_Agent. Die beiden letzteren Prozesse können mehrfach instanziiert werden (siehe Abb. 18)[SCH00]. Der Prozeß BS_Agent verwaltet die Sektoren einer Basisstation.

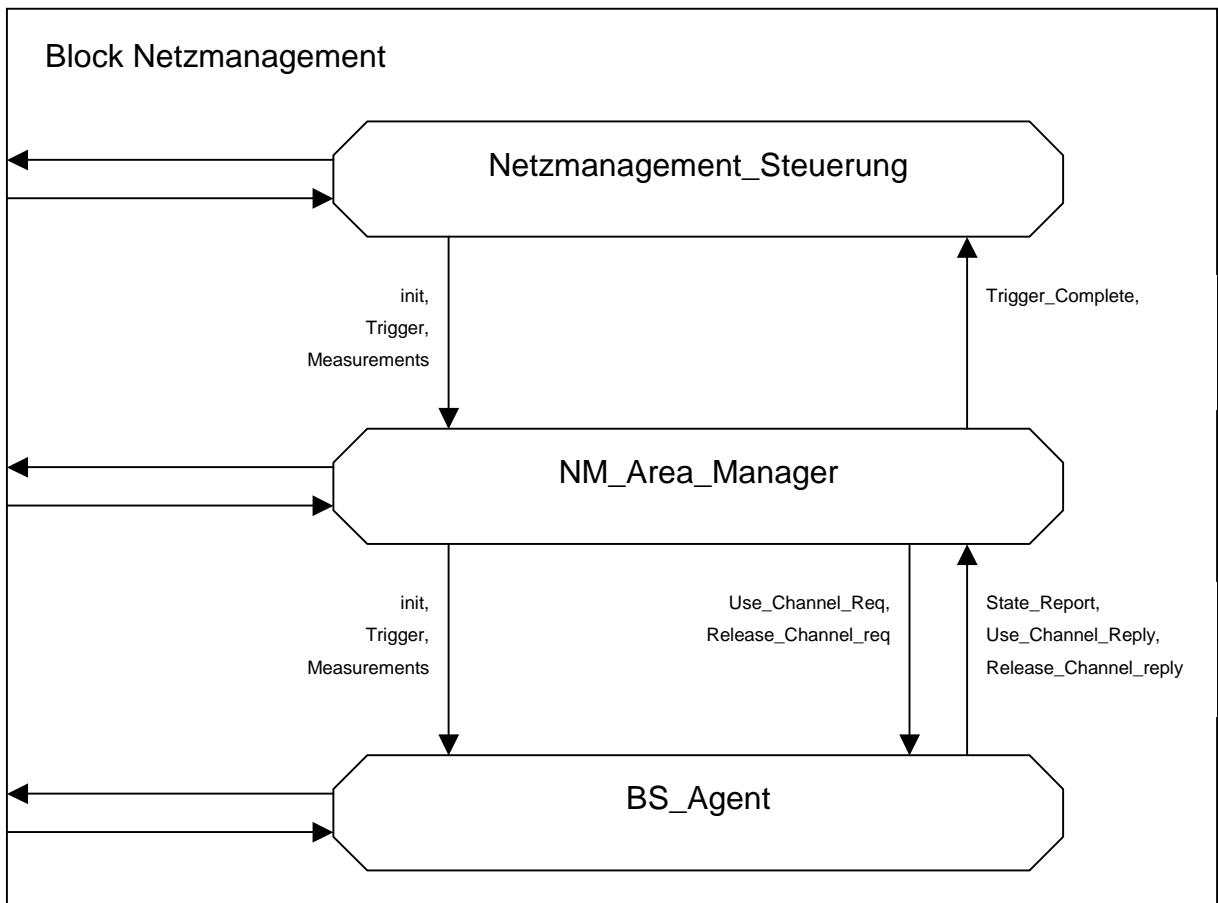


Abb. 18: Struktur des Blocks Netzmanagement

Die Kommunikation zwischen Manager und Agent sind in zwei Gruppen unterteilt. Auf der linken Seite sind die Signale gruppiert, die die Schnittstelle zwischen dem Netzmanagement und MOSIT darstellen. Die Signale auf der rechten Seite sind für die eigentliche Kommunikation zwischen Agent und Manager zuständig.

6.2.1 Erweiterbarkeit des Netzmanagementblocks

Der Block Netzmanagement erlaubt es, verschiedene Ressourcenverteilungsalgorithmen zu implementieren. Die Prozesse NM_Area_Manager und BS_Agent sind in SDL als Prozeß-

Typen definiert. Man kann deswegen von diesen Prozessen neue Prozesse ableiten, die Eigenschaften und Funktionen von den Grundtypen erben. Bei einer Erweiterung müssen nur die Teile der Prozesse neu definiert werden, die neue Aufgaben und Funktionalitäten erfüllen müssen. Die Grundprozesse BS_Agent und NM_Area_Manager bzw. die dort definierten Prozeduren übernehmen die restlichen Aufgaben. Die Prozeduren dieser Prozesse, die als VIRTUAL definiert sind, können in vererbten Prozeß-Typen überschrieben (REDEFINED) werden. Einige dieser virtuellen Prozeduren sind im Rahmen dieser Diplomarbeit zur Erweiterung des Netzmanagementsystems um eine marktbasierende Ressourcenverteilung verwendet worden. Zunächst wird auf die Aufgaben der drei Prozesse kurz eingegangen.

6.2.2 Der Prozeß Netzmanagement_Steuerung

Der Prozeß Netzmanagement_Steuerung führt initialisierende Funktionen wie z.B. das Lesen von Konfigurationsdateien oder die Erzeugung der Areamanager-Prozesse durch. Darüber hinaus leitet er von anderen Teilen MOSITs kommende Signale an die entsprechenden Netzmanagementinstanzen weiter. Zusätzlich sammelt der Prozeß Informationen über aufgebaute Verbindungen, Verbindungsabbrüche oder -blockierungen, Handover, Bitfehlerraten und CIR-Messungen und reicht sie an die Agenten weiter.

6.2.3 Der Prozess NM_Area_Manager

Der Areamanager verwaltet die Ressourcen seiner Area. Er beinhaltet alle Prozeduren, die der Ressourcenverteilung dienen. Die Informationen über den Bedarf und Zustand der einzelnen Zellen bekommt er von den Zellagenten geliefert. Anhand dieser Informationen und abhängig von der aktuellen Ressourcenverteilung soll der Areamanager entscheiden, was für Aktionen ausgeführt werden müssen, um eine neue Verteilung vorzunehmen. Die Informationen über die durchzuführenden Änderungen in der Ressourcenverteilung werden den Zellagenten weitergereicht.

6.2.4 Der Prozess BS_Agent

Der BS_Agent bearbeitet die Entscheidungen bezüglich der Ressourcenverteilung, die er vom Areamanager bekommt. Eine Entscheidung ist entweder ein Slot-Entzug oder ein Slot-Einfügen. Bei einem Slot-Entzug prüft der Agent, ob der Slot frei ist. Ist das der Fall, entfernt er diesen Slot aus seinen Datenstrukturen und meldet den Erfolg oder Mißerfolg dieser Aktion dem Manager. Das Slot-Einfügen erfolgt analog dazu. Der Agent fügt den Slot in seinen Datenstrukturen ein und meldet dem Manager eine Antwort über den Erfolg dieser Aktion.

6.3 SDL Schnittstellen zur Umgebung

Wie im vorherigen Kapitel erwähnt wurde, ist MOSIT in SDL implementiert. Für die Programmierung wurde das von der Firma Telelogic entwickelte SDT Tool angewendet. Dieses Tool stellt mehrere Techniken und Alternativen zur Verfügung, die es dem SDL-Programm erlauben, über eine bestimmte Schnittstelle mit der äußeren Umgebung zu kommunizieren. Weil die Untersuchung der verschiedenen Alternativen der Schnittstellen ein großer Teil der Diplomarbeit ist, werden im folgenden die wichtigsten technischen Grundlagen ausführlich behandelt.

6.3.1 SDL und das SDT-Tool

SDL (Specification and Description Language) ist eine von ITU-T standardisierte Programmiersprache, die zur Spezifikation und Beschreibung von Systemen dient.

SDL erlaubt eine hierarchische graphische Programmierung, die von einer Systemebene als höchste Ebene anfängt. Ein System besteht aus zusammenhängenden Modulen (Blöcke). Ein Block kann ebenfalls innerhalb einer Blockhierarchie in mehreren Blöcken rekursiv geteilt werden. Die Kommunikation zwischen Blöcken oder mit der Umgebung erfolgt über Kommunikationspfade, die als Kanäle bezeichnet werden. Jeder Kanal enthält eine Warteliste, welche die zu transportierenden Signale beinhaltet. Das Verhalten der letzten Blöcke in der Hierarchie ist durch einen oder mehrere mit sich kommunizierenden Prozesse beschrieben. Die Prozesse sind durch endliche Zustandsmaschinen (FSM: finite state machine) beschrieben, die eventuell Prozeduren anwenden.

SDL unterstützt auch die Konzepte der Objektorientierung wie Vererbung, Instanzen, Klassen mit ihren Methoden und Attributen etc.

Die folgende Abbildung zeigt die vier hierarchischen Hauptebenen in SDL: System, Block, Prozeß und Prozedur [TEL].

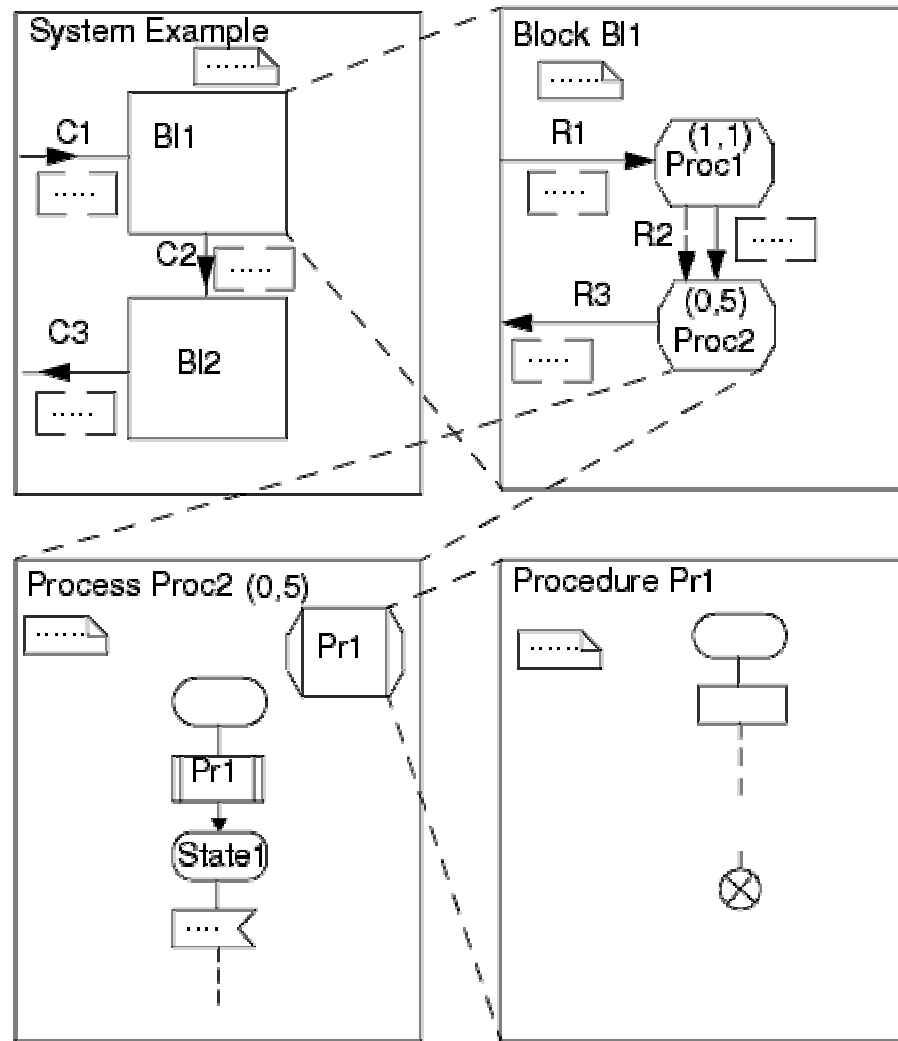


Abb. 19: Architektur eines SDL Systems

In SDL gibt es keine globalen Daten. Statt dessen werden Informationen zwischen den Prozessen oder mit der Umgebung über Signale mit optionalen Parametern getauscht. Die Signale sind asynchron gesendet, so daß der sendende Prozeß mit der Ausführung seiner Aufgaben fortfahren kann, ohne auf eine Quittung von dem empfangenden Prozeß warten zu müssen. Um ein synchrones Verhalten zu erzwingen, muß ein Wartezustand eingefügt werden, in dem der Prozeß auf ein Quittierungssignal wartet.

Die folgende Abbildung zeigt die Kommunikation zwischen zwei Prozessen über Signale. Der Prozeß $Proc1$ sendet das Signal $Sig1$, welches den Parameter 5 vom Typ Integer beinhaltet, zu dem empfangenden Prozeß $Proc2$. Die Variable Number in $Proc2$ bekommt also den Inhalt des empfangenen Signals $Sig1$ zugewiesen.

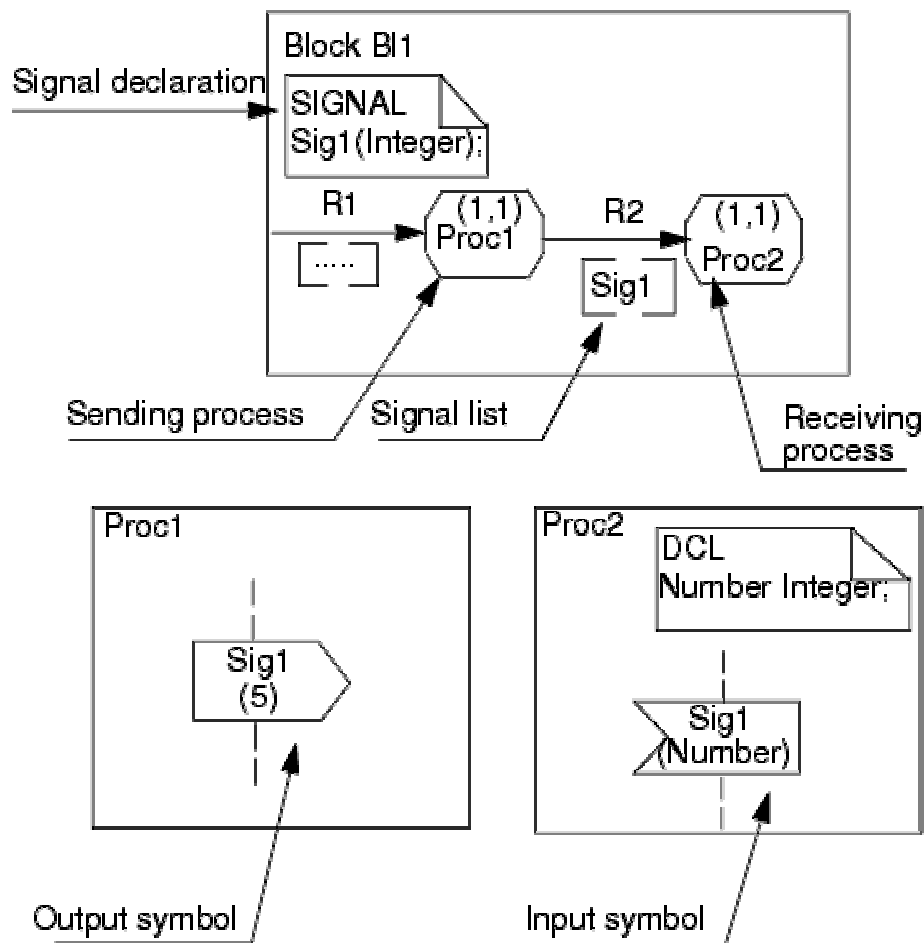


Abb. 20: Kommunikation zwischen Prozessen über Signale

Das von der Firma Telelogic entwickelte SDT-Tool stellt eine Entwicklungsumgebung für SDL-Programmierer dar [TEL]. Mit seiner Hilfe können SDL-Programme graphisch dargestellt werden. Darüber hinaus bietet SDT eine Menge von Funktionalitäten wie die Übersetzung, Gültigkeitsprüfung und Simulation des Programms. Die wichtigste Aufgabe von SDT ist die Codeerzeugung aus den SDL-Diagrammen. Daraus wird eine Anwendung gewonnen, die in einer Hochsprache geschrieben ist. Das Tool verfügt über einen C-Codegenerator, der SDL in C übersetzt. Dann übernimmt ein C-Compiler die Übersetzung in die Maschinensprache.

Ein interessantes Merkmal des SDT-Tools, welches für die Implementierung des Ressourcenverteilungsverfahrens eingesetzt wurde, ist seine Fähigkeit, die SDL-Programme auf verschiedenen Arten mit der äußeren Umgebung kommunizieren zu lassen. Im folgenden werden diese verschiedenen Möglichkeiten vorgestellt.

6.3.2 Kommunikation über CORBA

CORBA, die Common Object Request Broker Architecture wurde von der Object Management Group (OMG), einem Standardisierungsgremium mit mehr als 700 Mitgliedern, 1991 in der ersten Version definiert. CORBA war eine Antwort auf die starke Zunahme von Hardware- und Software-Produkten. Das Ziel war, eine Middleware zu schaffen, welche eine orts-, plattform- und implementations-unabhängige Kommunikation zwischen Applikationen erlaubt. Wirklich interessant geworden ist CORBA seit der Verabschiedung der Version 2.0 im Dezember 1994. Diese Version brachte das Kommunikationsprotokoll IIOP, welches den Meldungs-austausch zwischen Object Request Brokern (ORB) verschiedener Hersteller und vor allem auch über das Internet ermöglicht [TIN98].

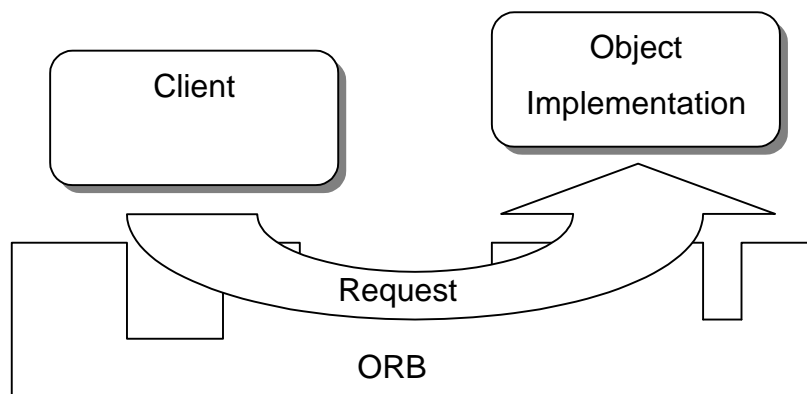


Abb. 21: CORBA Architektur

ORBs sind die technischen Implementierungen des Standards CORBA. Ein ORB ermöglicht es einem Client, eine Meldung transparent an ein Serverobjekt zu senden, wobei das Serverobjekt auf derselben oder einer anderen Maschine laufen kann. Der ORB ist dafür zuständig, das Serverobjekt zu finden, dort die Funktion aufzurufen, die Parameter zu übergeben und das Resultat an den Client zurückzureichen. Dadurch wird die bereits erwähnte nahtlose Interoperabilität zwischen Applikationen erreicht, welche in einem völlig heterogenen Umfeld betrieben werden können.

Bisher wurde in einem heterogenen Umfeld typischerweise jede Schnittstelle spezifisch programmiert. Dabei müssen Plattform, Betriebssystem, Programmiersprache und anderes in Betracht gezogen werden. Bei Änderungen ist die Anpassung solcher Schnittstellen entsprechend schwerfällig. Bei CORBA-basierenden Systemen ist dies anders. Es besteht eine strikte Trennung zwischen der Schnittstellendefinition eines Objektes und deren

Implementation. Beim CORBA-Vorgehen wird zunächst die öffentliche Schnittstelle eines Objektes (d.h. die Funktionen) in der Interface Definition Language (IDL) definiert. IDL ist eine implementationsunabhängige Beschreibungssprache. In einem zweiten Schritt erst wird diese Definition ausprogrammiert, und zwar sowohl für den Client als auch für den Server-Teil. Dabei kann der Client beispielsweise in Java implementiert werden, während der Server in C++ programmiert wird. CORBA-IDL kann aber nicht nur auf Implementationssprachen „gemapped“ werden, sondern auch auf andere Komponentenmodelle wie ActiveX/DCOM. Somit ist die Freiheit gegeben, die am besten passende Implementations-Strategie zu wählen.

Neben der Kern-Architektur sind eine Reihe von zusätzlichen Services definiert worden, welche CORBA zu einer vollständigen Middleware wachsen ließen. Nur um die wichtigsten zu nennen:

- Naming-Service: hilft bei der Suche und Identifikation von Objekten
- Event-Service: definiert die Schnittstellen für Messaging-Systeme
- Transaktions-Service: Schnittstellen für Two-Phase-Commit und andere Verfahren
- Datenbank-Services: einheitliche Schnittstellen für Datenbankoperationen

Wie schon beim CORBA-Kern definiert OMG vorerst einen Standard. Typischerweise 1 – 2 Jahre später sind dann die ersten Implementationen auf dem Markt erhältlich. Da praktisch alle Service-Definitionen 1995 verabschiedet worden sind, liegen für die meisten von ihnen schon Implementationen verschiedener Anbieter vor.

Das SDT-Tool bietet eigentlich CORBA als guten Ansatz zur Kommunikation zwischen SDL-Programmen und externen Programmen, nur das Institut für Allgemeine Nachrichtentechnik verfügte nicht über eine Lizenz für den ORB, was mich gezwungen hat auf diese Alternative zu verzichten.

6.3.3 Das SDT Postmaster Interface

Die Telelogic SDT Tools besitzen eine definierte Schnittstelle zur Kommunikation zwischen den einzelnen Tools und der äußeren Umgebung. Diese Schnittstelle, der sog. Postmaster, steht auch für die Kommunikation mit eigenen Software Tools und Programmen offen [TEL]. Der Postmaster wird sofort nach dem Öffnen des SDT Organizers aktiviert. Alle Tools wie der Editor oder Simulator können untereinander mit einem festgelegten Befehlssatz kommunizieren. Auch ein Programm, was unter SDT programmiert wurde, aber nicht mehr in einem dieser Tools gestartet wird, sondern vielmehr von der Kommandozeilenebene aus, kann

noch über den Postmaster mit anderen Tools, die über eine derartige Schnittstelle verfügen, kommunizieren.

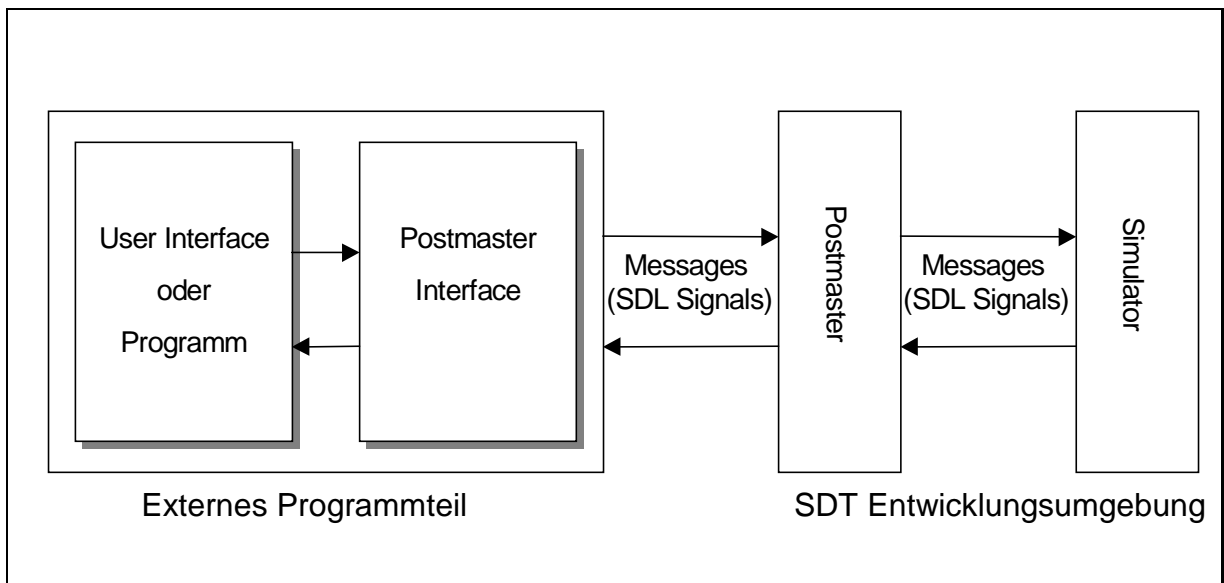


Abb. 22: Postmaster

Im folgenden wird der Aufbau und die Funktionsweise eines Programms beschrieben, das mit einem SDL Simulator kommunizieren soll. Es laufen alle Kommunikationsprozesse über das Environment ab, d.h. es werden nur die SDL Signale zum UI (User Interface) oder zum externen Programm geschickt, die in der Systemebene in das Environment zeigen und umgekehrt nur die Daten vom UI eingelesen, für die ein Signal auf der SDL Systemebene zur Verfügung steht.

Zu bearbeiten sind die zwei Dateien `env.c` und `Programmname.c`. Dabei sind die folgenden Funktionen zu implementieren:

- `Init_UI`, `Exit_UI` Initialisierung bzw. Beendigung der Initialisierung.
- `Send_To_PM(char* sdl_signal_name)` erzeugt und sendet das SDL-Signal zum Postmaster des SDT-Systems. Diese Funktion kann erweitert werden, um auch Parameter mit Signalen zu übergeben.
- `Receive()` empfängt Signale mit Parametern vom SDT-Postmaster und reicht sie an externe Programme weiter.

Der Einsatz vom Postmaster für diese Arbeit hätte den Nachteil, daß die ausgetauschten Signale nur Zeichenketten als Parameter enthalten könnten, was die Sache viel komplizierter gemacht hätte. Die Signalparameter hätten nämlich bei jeder Kommunikation zwischen dem SDL-Programm und der Umgebung geparkt werden sollen, um die Datenstrukturen wieder zu gewinnen.

Aus dem Grund wurde diese Alternative bei der Realisierung der Schnittstelle zu MOSIT nicht eingesetzt.

6.3.4 Environment Funktionen

Ein SDL-System (oder Partitionen eines SDL-Systems) kommuniziert mit seiner Umwelt, indem Signale nach außen geschickt werden, bzw. von außen in das System eingespeist werden.

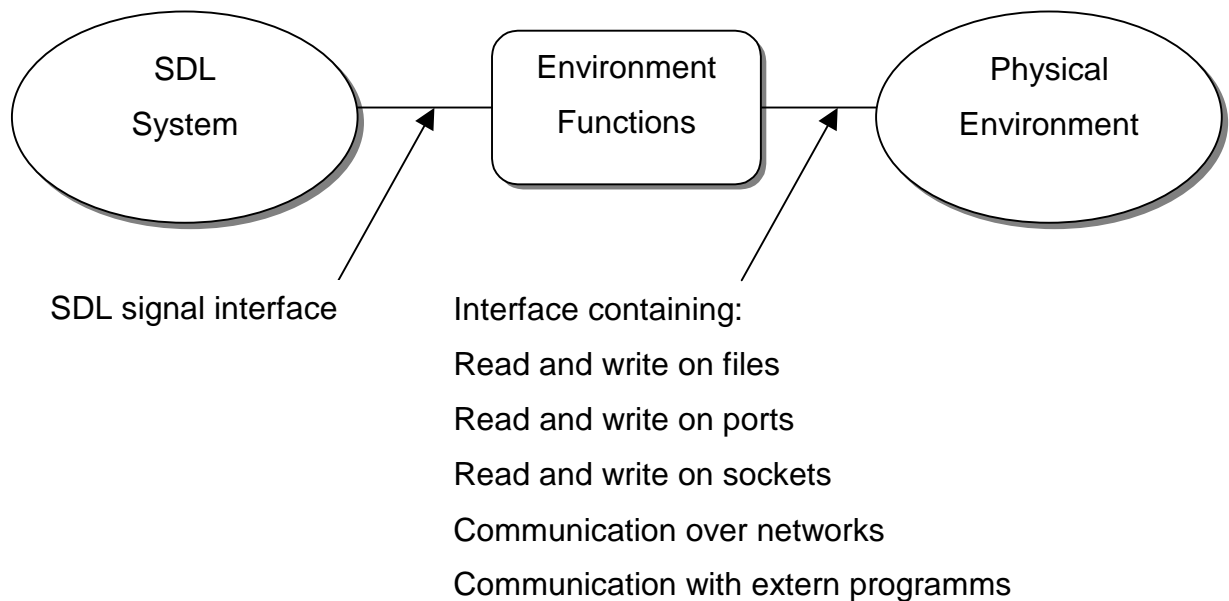


Abb. 23: Schnittstelle über Environment-Funktionen

Bei einer konkreten Implementierung hängt die Realisierung stark von der Anwendung, dem verwendeten Betriebssystem und den verwendeten Kommunikationsmitteln ab. Der Code-Generator des SDT-Tools kann nicht für jede dieser Situationen Code erzeugen. Daher muß der Benutzer (der Applikationsentwickler) in der Form sogenannter „Umgebungsfunktionen“ die Kommunikation mit der Außenwelt „von Hand“ implementieren [TEL].

Im folgenden wird beschrieben, wie die Umgebungsfunktionen implementiert werden können.

6.3.4.1 Nötige Funktionen

Um mit dem SDL-System kommunizieren zu können, müssen die folgenden C-Funktionen implementiert werden:

1. `xGlobalNodeNumber`

Ordnet jedem separaten Teil einer SDL-Applikation eine eindeutige Nummer zu.

2. `XinitEnv`

Initialisiert die Kommunikationsbeziehungen zur Umgebung. Hier können beispielsweise Ports geöffnet, Verbindungen aufgebaut, Prozesse gesucht werden etc.

3. `XcloseEnv`

Baut die per `xInitEnv` aufgebauten Kommunikationsverbindungen wieder ab.

4. `XinEnv`

Dient dazu, Signale aus der Umgebung in das SDL-System einzuspeisen.

5. `XoutEnv`

Dient dazu, Signale, die aus dem SDL-System kommen, an die Umgebung weiterzureichen.

Diese Funktionen werden automatisch von dem Code-Generator als „Skeleton“ in einer C-Datei zur Verfügung gestellt. Der Entwickler muß dann den Körper dieser Funktionen implementieren.

6.3.4.2 Hilfsfunktionen

SDT stellt eine Reihe von Hilfsfunktionen zur Verfügung, die bei der Implementierung der Umgebungsfunktionen verwendet werden sollten.

- `XgetSignal` erzeugt eine neue Signalinstanz.
- `XreleaseSignal` gibt eine Signalinstanz wieder frei.

`SDL_Output` dient zum Absetzen einer Signalinstanz in das SDL-System.

Nach einem Vergleich aller Alternativen zur Erweiterung von MOSIT um ein externes Programm habe ich mich für die letzte Möglichkeit entschieden, nämlich die Schnittstelle zu MOSIT über die C-Environment-Funktionen. Diese Technik garantiert die Erhaltung der Struktur der Signalparameter. Aufwendig ist aber die Typenumwandlung von SDL nach C und umgekehrt. Die in den Signalen enthaltenen Parameter müssen bei jeder Kommunikation von SDL nach C entpackt und in entsprechenden Typen umgewandelt und wieder gepackt werden. Das gleiche betrifft den anderen Fall, sprich die Kommunikation von C nach SDL.

Als Beispiel kann man den Typ `ARRAY` in SDL nennen. Ein Feld in SDL ist als verkettete Liste implementiert. Die Größe des Feldes ist bei der Deklaration nicht festgelegt. Es könnte theoretisch eine unendliche Liste werden. In C ist dies nicht der Fall, weil ein Feld schon bei der Entstehung eine vorbestimmte Größe haben soll. Wenn ein Parameter vom Typ `SDL ARRAY` in die Umgebung kommt, ist er in der C-Funktion als Zeiger auf eine verkettete Liste

definiert. Um die Umwandlung zu einem C-Feld durchzuführen, muß zuerst die Länge des Feldes ermittelt werden, und dann müssen alle Knoten der verketteten Liste durchlaufen werden. Dabei werden die Inhalte der Listeneinträge in einem C-Feld gemappt. Bei der anderen Kommunikationsrichtung verläuft die Umwandlung der Datenstrukturen analog.

Als nächstes stellt sich die Frage, wie man von den Environment-Funktionen aus externe Programme erreichen kann. Die Environment-Funktionen sollen im Endeffekt nur eine Schnittstelle bilden und nicht den Kern eines Programmes. Weil sie in C geschrieben sind, wäre das Anhängen eines externen C-Programmes am einfachsten zu realisieren. Es werden dann nur C-Funktionen und Bibliotheken von den Environment-Funktionen aus aufgerufen. Diese C-Funktionen haben dann die Aufgabe, Daten zu verarbeiten und Ergebnisse über die Schnittstelle ans SDL-Programm zu senden.

Im Rahmen dieser Diplomarbeit sollte ein externes Programm an MOSIT angehängt werden, das in der Zukunft als Basis und Rahmen für weitere Studien dient. Das Programm soll eine leichte und wenig aufwendige Erweiterung oder Implementierung weiterer dynamischen Ressourcenvergabeverfahren erlauben. Es soll darüber hinaus portierbar und plattformunabhängig sein. Aus diesen Gründen sollte die Implementierung des externen Programms in Java erfolgen.

Das nächste Problem, das sich stellt, ist die Kommunikation zwischen C und Java. Zwei Alternativen standen mir zur Auswahl: Kommunikation über Sockets oder JNI (Java Native Interface). Weil die erste Alternative die Implementierung eines Kommunikationsprotokolls voraussetzt, das die übertragenen Bits rekonstruiert, stellt sich JNI als bessere Möglichkeit, C mit Java und umgekehrt kommunizieren zu lassen.

Im folgenden werden die Grundlagen von JNI ausführlich behandelt, weil sie einen wichtigen Bestandteil der Implementierung des marktbasieren Ressourcenvergabeverfahren darstellen.

6.3.5 Java Native Interface (JNI)

Das Java Native Interface (JNI) wurde entwickelt, um Java-Programmen, die innerhalb der Java-VM laufen, die Möglichkeit zu geben, mit Programmen zusammenzuarbeiten oder Bibliotheksfunktionen zu nutzen, die in einer anderen Programmiersprache (z.B. C oder Assembler) geschrieben sind [SUN00]. JNI besorgt die Umwandlung der Datenformate bei der Übergabe von Parametern zwischen dem Java-Programm und der Fremdfunktion (native method). Der Fremdprozeß hat zudem die Möglichkeit, auf Klassen und Objekte des Java-Programms zuzugreifen und kann Ausnahmen erzeugen. Der Einsatz des JNI ist erforderlich,

wenn eine Java-Anwendung plattformabhängige Funktionen nutzen möchte, die nicht von den Standardklassen des JDK unterstützt werden. Ein weiteres Szenario für die Nutzung des JNI ist der Fall, daß bestimmte Funktionen bereits von einer bestehenden Bibliothek bereitgestellt werden und diese von dem Java-Programm genutzt werden sollen.

6.3.5.1 Lokale und globale Referenzen

JNI-Variablen mit Datentypen wie `jobject`, `jclass` oder `jstring` dienen als Referenz auf Java-Objekte. Diese werden als Argumente der native-Funktion zu einer Java-Methode übergeben oder entstehen als Resultat einer JNI-Funktion wie `GetObjectClass()`. Diese Referenzen zählen als vollwertige Referenzen für die `garbage collection`.

Das JNI unterscheidet lokale und globale Referenzen. Normalerweise programmiert man nur mit lokalen Referenzen. Alle Referenzen auf Objekte, die über ein Argument in die Funktion gelangen oder die als Resultat eines JNI-Funktionsaufrufs gewonnen werden, sind lokale Referenzen. Diese Verweise verlieren ihre Gültigkeit am Ende der Funktion, so daß die `garbage collection` dann möglicherweise die vom Objekt verwendeten Ressourcen freigeben kann. Deshalb ist es z.B. nicht erlaubt, sich das Ergebnis eines Aufrufs von `GetObjectClass()` in einer C-static-Variablen zu merken, um so ein wiederholtes Erzeugen des Klassenobjekts zu vermeiden.

Mit Hilfe der JNI-Funktion `DeleteLocalRef()` kann der Entwickler die Referenzen innerhalb der Funktion bereits selbst freigeben.

```
jobject NewGlobalRef(JNIEnv *env, jobject obj);  
void DeleteGlobalRef(JNIEnv *env, jobject globalRef);  
void DeleteLocalRef(JNIEnv *env, jobject localRef);
```

Abb. 24: Die Funktionen zum Umgang mit globalen Referenzen

Abbildung 24 zeigt die genaue Deklaration der Funktion. Werden in einer JNI-Funktion sehr viele lokale Referenzen angelegt und keiner der Verweise wird freigegeben, so kann es zum Überlauf der internen Tabelle von lokalen Verweisen kommen. Eine weitere denkbare Konstellation ist, daß man innerhalb einer JNI-Funktion die einzige Referenz auf ein Objekt mit sehr großem Ressourcenverbrauch besitzt und dieses Objekt nicht mehr benötigt wird, die Abarbeitung der Funktion aber noch relativ lange dauert. Dann ist es sinnvoll, mit `DeleteLocalRef()` die Ressourcen des Objekts freizugeben.

Das JNI bietet die Möglichkeit, globale Referenzen auf Objekte zu definieren. Die Methode `NewGlobalRef()` erhält neben dem `JNIEnv`-Zeiger einen lokalen Verweis auf ein Java-Objekt und hat einen globalen Verweis als Resultat. Diesen globalen Verweis darf man sich im Gegensatz zu lokalen Referenzen über Funktionsaufrufe hinweg merken und anstelle der lokalen Verweise benutzen.

Diese Technik hat zwei Nachteile. Zum einen stellt eine globale Referenz einen ständigen Verweis auf ein Objekt dar und entzieht damit dieses Objekt der garbage collection. Zum anderen müssen diese globalen Verweise explizit wieder durch Aufruf von `DeleteGlobalRef()` entsorgt werden.

6.3.5.2 Threads und JNI

Java ist multi-threaded, also eine nebenläufige Programmiersprache, und daher müssen auch die native-Funktionen thread-sicher programmiert werden.

Jeder Thread bekommt innerhalb einer JNI-Funktion einen eigenen `JNIEnv`-Zeiger. Daher darf dieser Zeiger auch nicht über Threads hinweg verwendet werden.

Jeder Thread bekommt dagegen aber immer den gleichen Zeiger pro Aufruf der Funktion.

Es ist nicht erlaubt, lokale Referenzen über Threads hinweg zu nutzen. Globale Referenzen dürfen dagegen von verschiedenen Threads geteilt werden.

Mehrere Threads können gleichzeitig den Code der JNI-Funktionen abarbeiten und um globale Daten konkurrieren. Ist dagegen die zugehörige Deklaration der Methode im Java-Programm mit dem Wort `synchronized` versehen, so kann immer nur ein Thread zur Zeit die Funktion abarbeiten. Alle anderen Funktionen müssen den Zugriff der einzelnen Threads auf die kritischen Daten selbst koordinieren. Dazu bietet das JNI zwei Funktionen, die analog zu einem `synchronized block` unter Java arbeiten:

In Java führt genau ein Thread den durch

```
synchronized(obj) {  
    ...  
}
```

geschützten Block aus, wenn er den Zugriff auf den zum Objekt `obj` gehörenden Monitor bekommt. Eine äquivalente Synchronisation der Threads kann innerhalb von JNI-Funktionen durch die Funktionen `MonitorEnter()` und `MonitorExit()` erreicht werden, siehe Abbildung 25.

```
jint MonitorEnter(JNIEnv *env, jobject obj);  
jint MonitorExit(JNIEnv *env, jobject obj);
```

Abb. 25: Die Funktionen zur Synchronisation von Threads

Lediglich ein Thread auf einmal kann den Code der Funktion hinter einem Aufruf der Funktion `MonitorEnter()` ausführen. Mit jedem Java-Objekt ist ein Monitor assoziiert, und `MonitorEnter()` regelt den Zugriff auf diesen Monitor. Ein Monitor hat intern einen Zähler. Bekommt ein Thread durch `MonitorEnter()` den exklusiven Zugriff auf den Monitor, so wird der Zähler auf Eins gesetzt. Alle weiteren Aufrufe von `MonitorEnter()` dieses Threads inkrementieren den Zähler um eins. Führt der Thread `MonitorExit()` aus, so wird der Zähler um eins verringert. Wird der Zähler so auf Null gesetzt, gibt der Thread damit den Zugriff auf den Monitor frei, und andere Threads können den kritischen Teil der Funktion betreten. Für die Java-Methoden `wait()`, `notify()` und `notifyAll()` gibt es keinen Ersatz im JNI. Man muß sie über den beschriebenen Mechanismus als Java-Methoden aufrufen.

6.3.5.3 Das Invocation API

Das JNI bietet Funktionen, um aus beliebigen eigenen Programmen heraus eine virtuelle Maschine zu erzeugen, zu initialisieren und in dieser Java-Programme auszuführen. Dafür steht das sogenannte Invocation API zur Verfügung. Damit kann jedes Programm Java-Programme zur Ausführung bringen. So wird das Kommando `java` zu einem einfachen C-Programm, welches die Kommandozeile parst, eine virtuelle Maschine erzeugt und anschließend mit dem auf der Kommandozeile angegebenen Java-Programm startet. Abbildung 26 zeigt den Java-Quelltext einer Klasse `HelloWorld`. Die Klasse enthält lediglich eine Klassenmethode `main()` und gibt in der Methode den Text `Hello World!` aus, gefolgt von dem optionalen ersten Element des Arrays `args`.

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!" +
            (args.length>0?args[0]:""));
    }
}
```

Abb. 26: HelloWorld.java

Abbildung 27 zeigt ein C-Programm, welches nun eine virtuelle Maschine erzeugt und die `main`-Methode der Klasse `HelloWorld` aktiviert.

```

int main(int argc, char ** argv) {
    JNIEnv *env;
    JavaVM *jvm;
    JDK1_1InitArgs vm_args;
    jclass cls;
    jmethodID mid;
    jstring jstr;
    jobjectArray args;
    ...
    JNI_GetDefaultJavaVMInitArgs(&vm_args);
    vm_args.classpath="/usr/local/java/lib/classes.zip:.";
    JNI_CreateJavaVM(&jvm,&env,&vm_args);
    cls = (*env)->FindClass(env, "HelloWorld");
    mid = (*env)->GetStaticMethodID(env,
        cls,"main", "([Ljava/lang/String;)V");
    jstr = (*env)->NewStringUTF(env, " from C!");
    args = (*env)->NewObjectArray(env, 1,
        (*env)->FindClass(env, "java/lang/String"), jstr);
    (*env)->CallStaticVoidMethod(env, cls, mid, args);
    (*jvm)->DestroyJavaVM(jvm);
    return 0;
}

```

Abb. 27: Die Erzeugung einer eigenen virtuellen Maschine

Die Funktion `JNI_GetDefaultJavaVMInitArgs()` hat als Resultat einen Wert vom Typ `JDK1_1InitArgs`. Diese Struktur definiert die initialen Werte bei der späteren Erzeugung der virtuellen Maschine. In ihr können unter anderem der Klassenpfad, die Stackgrößen, Optionen für die garbage collection usw. eingestellt werden. Abbildung 28 zeigt wichtige Teile der Struktur.

```

typedef struct JavaVMInitArgs {
    jint version;
    char **properties;
    jint checkSource;
    jint nativeStackSize; jint javaStackSize;
    jint minHeapSize; jint maxHeapSize;
    jint verifyMode;
    const char *classpath;
    ...
} JDK1_1InitArgs;

```

Abb. 28: Der Aufbau der Struktur `JDK1_1InitArgs`

Die von `JNI_GetDefaultJavaVMInitArgs()` gelieferte Initialisierungs-Struktur enthält im Eintrag `classpath` den Klassenpfad der späteren Java-Maschine. Unter Solaris enthält die Komponente alle Standard-Pfade. Unter Linux dagegen ist der Zeiger ein Null-Zeiger, und dem Entwickler bleibt es überlassen, die Standardpfade in den Klassenpfad aufzunehmen.

Als nächster Schritt erfolgt die Erzeugung und Initialisierung einer virtuellen Maschine durch den Aufruf der Funktion `JNI_CreateJavaVM()`. Diese Funktion erhält als drittes Argument einen Zeiger auf die Initialisierungs-Struktur. Die ersten beiden Argumente liefern bei einer erfolgreichen Erzeugung der Java-Maschine Zeiger auf Funktionstabellen.

`DestroyJavaVM()` zerstört die erzeugte Java-Maschine wieder. Mit `AttachCurrentThread()` können andere native-Threads sich an eine existierende virtuelle Maschine (erstes Argument) anhängen. Über das zweite Argument bekommt der native-Thread einen neuen `JNIEnv`-Zeiger. Dabei kann ein Thread nicht gleichzeitig an mehrere virtuelle Maschinen ankoppeln. Mit `DetachCurrentThread()` kann der Thread sich wieder von der Maschine loslösen. Die letzten beiden Funktionen funktionieren unter JDK 1.1.5 laut Sun zur Zeit nur unter Windows, nicht aber unter Solaris. Man kann aber für Solaris über die Sun-Internetseiten ein native-Thread-Paket laden, mit welchem dann auch diese beiden Funktionen problemlos arbeiten sollen.

Die beiden einzigen weiteren neuen JNI-Funktionen im Beispiel sind `FindClass()` und `NewObjectArray()`. `FindClass()` sucht eine lokale Klasse auf dem Klassenpfad, lädt diese zur virtuellen Maschine dazu und liefert eine Referenz auf das Klassenobjekt dieser Klasse als Resultat. `NewObjectArray()` erzeugt ein neues Array von Objekten, hier von String-Objekten. Die genauen Aufrufe der Funktion sind der Java Native Interface Specification zu entnehmen.

Nach dem erfolgreichen Aufruf von `JNI_CreateJavaVM()` arbeitet der native-Thread, als wäre er ein Java-Thread, der von der Java-Seite zu einer native-Funktion wechselt. Da der Thread aber nie von dieser native-Funktion in die Java-Seite zurückkehrt, werden die lokalen Referenzen nicht recycelt. Dies geschieht erst nach Aufruf von `DestroyJavaVM()`.

6.3.5.4 Speichermanagment

In Java wird die Speicherverwaltung automatisch vom Garbage-Collector erledigt, indem er die Methode `dispose()` aus der Java-Klasse ausführt. Wird in einer C-Funktion einer nativen

Methode Speicher angelegt, so hängt die Vorgehensweise des Programmierers beim Freigeben, von der Art des Speichers ab. Bei normalen Speicherbereichen für die interne C-Anwendung muß der Programmierer sich explizit um die Freigabe kümmern. Da der Java-Interpreter von diesem Speicher nichts weiß, müssen eigene native Methoden implementiert werden um ihn freizugeben. In C erzeugte Speicherbereiche von Java-Objekten werden nicht unbedingt vom Garbage-Collector gefunden. Diese Java-Objekte, also Instanzen einer bestimmten Java-Klasse, können auch aus Java heraus freigegeben werden. Eine einfache Variante der Freigabe ist der explizite Aufruf der Methode `dispose()` wenn die Instanz nicht mehr benötigt wird.

7 Implementierung

7.1 Architektur der implementierten Lösung

In den vorigen Abschnitten wurden die verschiedenen möglichen Schnittstellen zu SDL vorgestellt. Für die Erweiterung von MOSIT um das marktbasierende Verfahren habe ich mich für die Schnittstelle über die Environment-Funktionen entschieden. Weil diese Funktionen in C geschrieben sind, sah ich JNI als geeignete Technik, um das eigentliche Verfahren in Java zu implementieren.

Im folgenden werden Architektur und Abläufe der implementierten Lösung anhand Abbildung 29 im Detail erläutert.

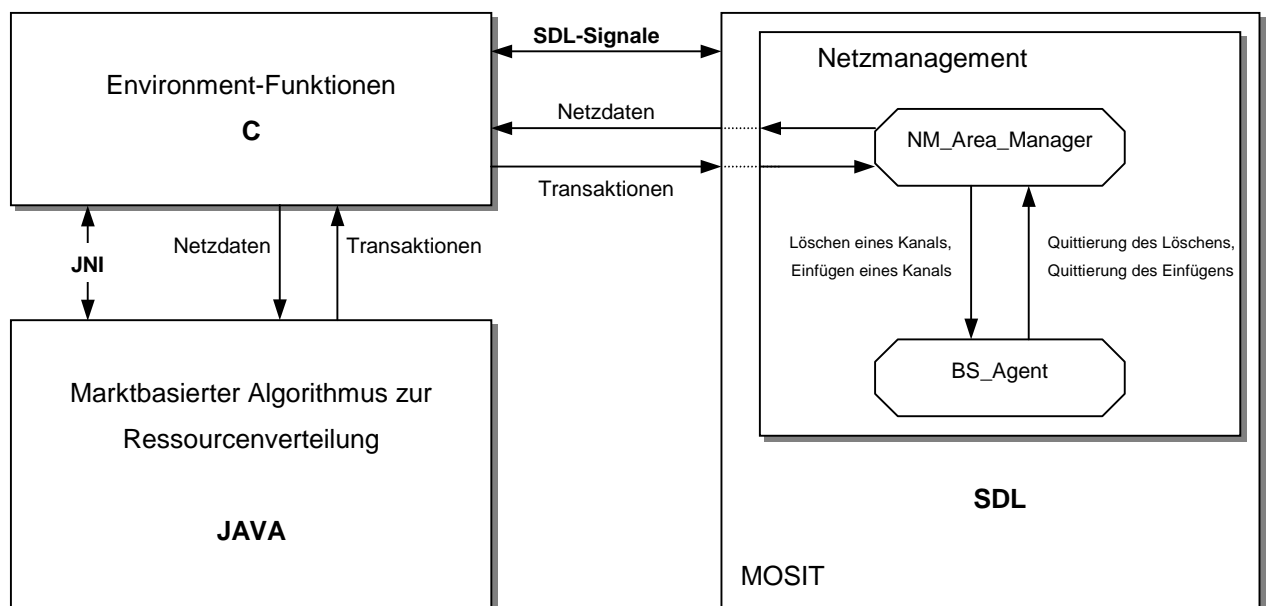


Abb. 29: Architektur der implementierten Lösung

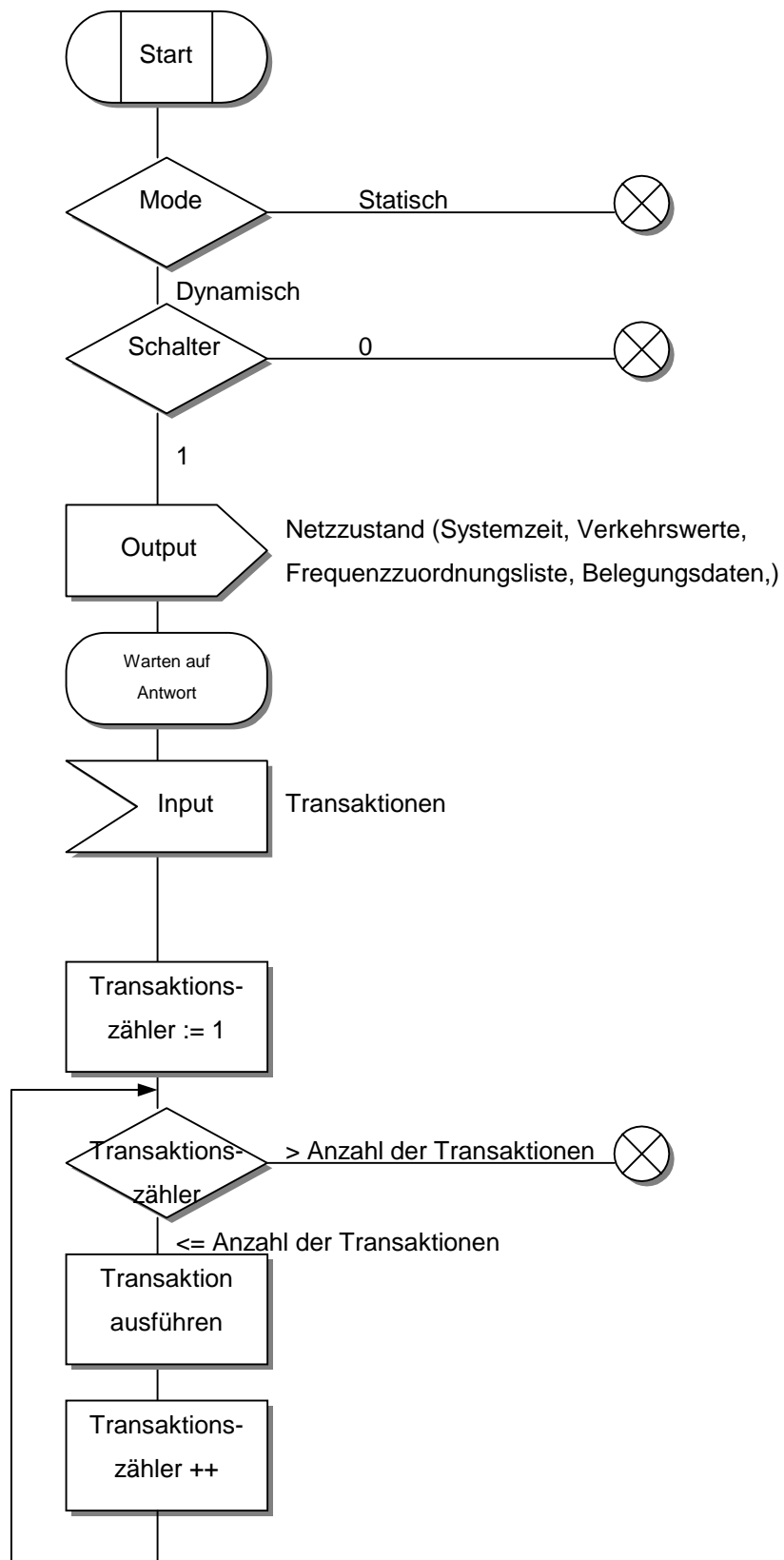
Zuerst ist es aber wichtig zu erwähnen, daß MOSIT eine diskrete Simulation durchführt. Das bedeutet, daß die verschiedenen kontinuierlichen Abläufe eines realen Mobilfunknetzes in MOSIT in Zeitschritten abgebildet werden. Bei jedem Zeitschritt besitzt das Netz einen neuen

Zustand. Die folgend beschriebenen Abläufe stehen also für einen Zeitschritt. Bei der gesamten Simulation finden diese Abläufe bei jedem Zeitschritt statt:

Die für die Entscheidungen des marktbasieren Verfahren relevanten Netzdaten werden auf der SDL-Seite gesammelt und anschließend über ein Signal zu den Environment-Funktionen als Anfrage gesendet. Dann geht die SDL-Seite in einen Wartezustand über, und wartet auf die Antwort der Environment-Funktionen. In diesen Funktionen werden die Rohdaten des Mobilfunknetzes von dem empfangenen Signal extrahiert und in JNI-Datenstrukturen abgebildet. Diese neu gewonnenen Informationen stellen die Eingabedaten für das aufzurufende Java-Programm. Die aufgerufenen Java-Methoden übernehmen also die Aufgabe der eigentlichen Bearbeitung der Mobilfunkdaten. Anhand dieser Daten entscheidet das Java-Programm nach dem Durchführen des marktbasieren Algorithmus, welche Transaktionen ausgeführt werden müssen. Die Entscheidungen werden in einer Datenstruktur zusammengefaßt und als Ausgabe zu den Environment-Funktionen über JNI gesendet. Diese erzeugen von den empfangenen Entscheidungen eine mit SDL-kompatible Datenstruktur, die anschließend über ein Signal zur SDL-Seite gesendet wird. Dieses Signal ist die Antwort der Umgebung auf die SDL-Anfrage. Die SDL-Seite verläßt dann den Wartezustand und bearbeitet die von dem Java-Programm getroffenen Entscheidungen. Jede Transaktion bedeutet für den SDL-Manager (NM_Area_Manager) zwei Aktionen: Ein Kanalverkauf entspricht einem Löschvorgang bei der verkaufenden Zelle und einem Einfügevorgang bei der kaufenden Zelle. Die eigentlichen Lös- und Einfügevorgänge übernehmen die BS-Agenten, die von dem NM_Area_Manager getriggert werden. Nach der Ausführung aller Transaktionen ist der Prozeß der Ressourcenverteilung zu Ende, und MOSIT fährt mit der Simulation fort.

7.2 Die SDL-Seite

Bei der Simulation wird die Prozedur *RsrcDistribution* vom *NM_Area_Manager* bei jedem Schritt aufgerufen. Diese Prozedur sollte von mir neu implementiert werden, so daß sie die Ressourcenverteilung nach den marktbasieren Konzepten durchführt. Ihre Aufgabe ist also das Zusammenfassen der relevanten Roh- und Verkehrsdaten des Netzes und ihre Weiterleitung zu den Environment-Funktionen. Nachdem die Entscheidungen vom Java-Programm getroffen sind, bearbeitet die Prozedur *RsrcDistribution* die Transaktionen, indem sie die betroffenen Kanäle von den Verkäuferzellen löscht und zu den Käuferzellen zufügt. Das folgende Flußdiagramm soll die Funktionsweise dieser Prozedur grob erklären.

Abb. 30: Ablaufdiagramm der Prozedur *RsrcDistribution*

In der Prozedur *RsrcDistribution* wird zuerst geprüft, ob das dynamische oder das statische Verfahren eingeschaltet ist (Mode). Im Falle des statischen Verfahrens führt die Prozedur keine Aktionen aus. Die Ressourcen werden also nicht verteilt. Im zweiten Fall wird dazu geprüft, ob der Schalter für das dynamische Verfahren eingeschaltet ist (Schalter). Dieser Schalter dient dazu die Ressourcenverteilung zyklisch nach einer bestimmten Anzahl von Schritten durchzuführen. Es ist nämlich nicht sinnvoll die Ressourcen bei jedem Schritt neu zu verteilen. Das soll eher in größeren Abständen erfolgen, sonst wird das Netz auf jede kleine Änderung reagieren, was die Stabilität des Verfahrens gefährdet. Das Verfahren ist dazu gedacht, anhand von statistischen Daten die Ressourcen zu verteilen und nicht anhand von augenblicklichen Werten.

Die beiden frei zu konfigurierenden Parameter (Mode und Fenster) werden von der Initialisierungsdatei `Parameter/Ini_Dateien/Market_Based.ini` eingelesen (siehe Abbildung 31).

```
Market_Based.ini
MarketBased.Dynamic.Algorithm.Schalter.Mode(0=aus.1=ein).: 1
Schalter.....: 10
```

Abb. 31: Die Initialisierungsdatei Marked_Based.ini

7.2.1 Senden der Netz- und Verkehrsdaten

Im Gegensatz zu einem realen Mobilfunksystem werden in MOSIT die simulierten Verbindungen an zentraler Stelle verwaltet und bearbeitet. Die BS_Agent-Prozesse benötigen diese Informationen, um Verkehrswerte wie Angebot oder Verlust ermitteln zu können. Dazu müssen die Daten an die Stationsagenten geleitet werden. Die Daten durchlaufen dabei die Managementsteuerung, anschließend die Manager und erreichen schließlich die Agenten in aufbereiteter Form.

Diese Meßwerte werden mit jedem Zeitschritt in einem Buffer gesichert. In diesem Buffer wird zusätzlich die Anzahl der freien Slots und die gesamte Anzahl der Slots dieser Zelle im aktuellen Zeitschritt abgelegt. Der Buffer speichert nur die letzten n Meßwerte. Die Länge des Buffers (die Fenstergröße) ist frei konfigurierbar.

Der Stationsagent verwaltet für jeden Sektor, d.h. für jede Zelle, ein eigenes Fenster. In jedem Zeitschritt errechnet er Verkehrswerte wie z.B. Ankunftsrate oder Angebot aus den Daten im Fenster. Dabei wird über das gesamte Fenster gemittelt. Diese Werte meldet der Agent an

seinen Areamanager zurück. Weil die Prozedur *RsrcDistribution* zum Areamanager gehört, verfügt sie über diese statistischen Netz- und Verkehrsdaten.

Diese Daten werden, wie in Abbildung 30 gezeigt ist (Output), über ein Output-Signal zu den Environment-Funktionen gesendet. In der folgenden Tabelle wird die Struktur des Output-Signals und seiner Parameter, wie sie in SDL implementiert worden sind, erläutert.

Das SDL-Signal heißt `Out_ClusterState` und beinhaltet folgende Parameter:

Parameter	Datentyp	Beschreibung
SystemTime	Real	Aktuelle Simulationszeit
BsList	BSListT	Die Liste aller dem Manager zugeordneten Zellen
ClusterState	ClusterStateT	Der Zustand der Area. In dieser Struktur sind die aktuellen Verkehrswerte aller Zellen gespeichert
TfList	TF_Zuordnungs_Feld	Trägerfrequenzzuordnungsliste
BelUp	Kanal_bel_Feld_Up	Aktuelle Kanalbelegungslisten
BelDown	Kanal_bel_Feld_Down	Aktuelle Kanalbelegungslisten

Abb. 32: Parameter des Output-Signals `Out_ClusterState`

Die relevanten SDL-Datentypen von MOSIT werden in den nächsten Abschnitten im Detail beschrieben.

7.2.2 Empfang und Bearbeitung der Transaktionsinformationen

Nach dem Senden der relevanten Netzdaten wartet die *RsrcDistribution* auf die Antwort der Environment-Funktionen. Diese Antwort bekommt sie, wie in Abbildung 30 gezeigt ist (Input), über ein Input-Signal, das die Transaktionsinformationen als Parameter enthält.

Das SDL-Signal heißt `Out_Result` und beinhaltet folgende Parameter:

Parameter	Datentyp	Beschreibung
Result	Transactions	Verkauf- und Kaufliste mit den Informationen über die zu verteilenden Kanäle (BS, Sektor, Frequenz, Slot).

Abb. 33: Parameter des Input-Signals `In_Result`

Die Datentypen, die zur Behandlung von Transaktionen dienen, wurden von mir komplett in SDL implementiert und in MOSIT eingefügt. In der folgenden Tabelle wird die Struktur dieser Datentypen beschrieben.

Datentyp	Definition	Beschreibung
Transactions	<pre> NEWTYP Transactions struct Length Integer; Kauf_List TransactionList; Verkauf_List TransactionList; ENDNEWTYP; </pre>	Enthält die Kauf- und Verkaufslisten. Die Länge steht für beide Listen. Es wird soviel verkauft wie gekauft.
TransactionList	<pre> NEWTYP TransactionList ARRAY(Integer, Transaction); ENDNEWTYP; </pre>	Transaktionsliste: Ein Feld, das Transaktionen enthält.
Transaction	<pre> NEWTYP Transaction struct Bs Integer; Sek Integer; Freq Real; Slot Integer; Gewinn Real; ENDNEWTYP; </pre>	Enthält die eindeutige Referenz des Slots, das von einer Transaktion betroffen wird. Je nachdem, ob es sich um einen Kauf oder Verkauf handelt, wird das betroffene Slot entweder gekauft oder verkauft.

Abb. 34: Datentypen der Transaktionen

Nach dem Empfangen des Transaktionssignals wird eine Schleife über beide Transaktionslisten durchlaufen. Die Transaktionen werden, wie in Abbildung 30 gezeigt ist, ausgeführt. Die Einträge der Transaktionslisten enthalten alle Informationen, die eine eindeutige Lokalisierung und Kennung der Slots erlauben. So kann man gezielt auf ein bestimmtes Slot im ganzen Netz zugreifen. Die Slots, die sich in der Verkaufsliste bzw. Kaufliste befinden, werden von bzw. zu den zugehörigen Zellen entfernt bzw. zugefügt. Das erfolgt durch die Prozeduren *TakeSlotFrom* bzw. *GiveSlotTo*, die anhand der folgenden Tabelle beschrieben werden.

Prozedur	Parameter	Beschreibung
TakeSlotFrom	UpDownMode Integer	Kennzeichnet, ob es sich um ein Uplink- oder Downlink-Slot handelt.
	SlotInd Index_Struct	Index und eindeutige Kennung des zu entfernenden Slots.
	Target PId	Die Prozeß-ID des Agentenprozesses, welchem das Slot gehört.
	RETURNS Boolean	Liefert ein true bei erfolgreicher und ein false bei fehlgeschlagener Aktion.
GiveSlotTo	UpDownMode Integer	Kennzeichnet, ob es sich um ein Uplink- oder Downlink-Slot handelt.
	SlotInd Index_Struct	Index und eindeutige Kennung des zuzufügenden Slots.
	Target PId	Die Prozeß-ID des Agentenprozesses, welchem das Slot gehört.
	RETURNS Boolean	Liefert ein true bei erfolgreicher und ein false bei fehlgeschlagener Aktion.

Abb. 35: Die Prozeduren TakeSlotFrom und GiveSlotTo

Die relevanten SDL-Datentypen von MOSIT werden in den nächsten Abschnitten im Detail beschrieben.

Wichtig zu erwähnen ist, daß alle Abläufe einmal für den Uplink und einmal für den Downlink ablaufen. Die Transaktionen, die vom marktbasieren Verfahren empfangen werden, stehen eigentlich nur für den Uplink. Weil jede Sprachverbindung genau einen Uplink- und einen Downlink-Kanal voraussetzt, werden die Kanäle immer paarweise verteilt. Zu jedem Uplink-Slot gehört ein Downlink-Slot. Deswegen werden pro Transaktion zwei Kanäle verschoben. Zuerst wird der Uplink-Kanal zwischen den beiden betroffenen Zellen ausgetauscht, dann der entsprechende Downlink-Kanal.

7.3 Die Schnittstelle über die Environment-Funktionen und JNI

Wie in den vorigen Kapiteln erwähnt wurde, stellen die Environment-Funktionen die Schnittstelle zwischen der SDL- und der Java-Seite dar. Im folgenden werden die von den Environment-Funktionen erfüllten Aufgaben und die von mir implementierten Teile erläutert.

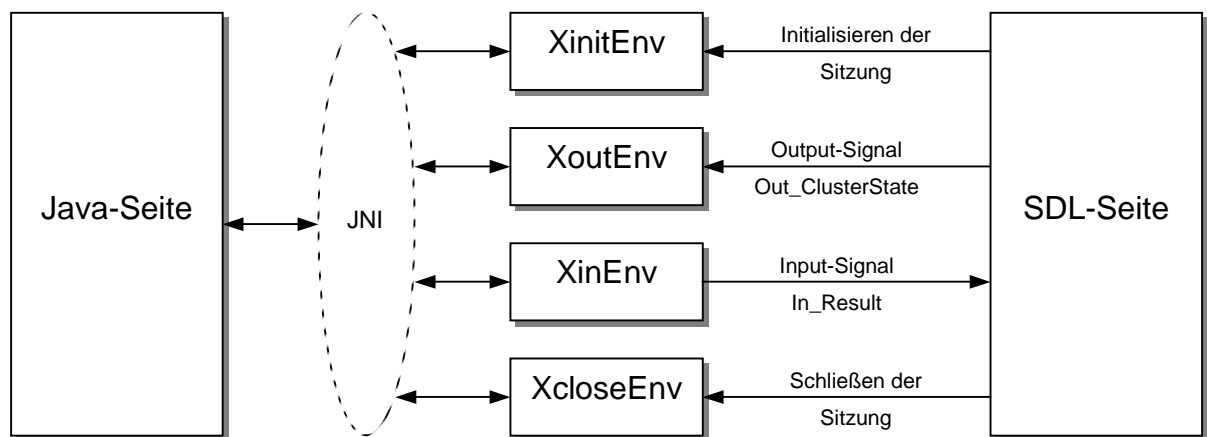


Abb. 36: Architektur der Schnittstelle über die Environment-Funktionen und JNI

Am Anfang der Simulation wird die Funktion `XinitEnv` von der `SDL-Seite` aufgerufen, um eine Sitzung zu initialisieren. Dabei werden die Aktionen ausgeführt, die nur einmal am Anfang ausgeführt werden müssen, wie das Lesen von Initialisierungsdateien und -parameter und das Initialisieren von Zählern. Folglich wird am Ende der Simulation die Funktion `XcloseEnv` aufgerufen, um Aktionen, wie das Schließen von Dateien, das Freimachen vom belegten Speicherplatz und das Schließen der Sitzung mit der `SDL-Seite`, auszuführen.

Der Kern der Schnittstelle sind die Funktionen `XoutEnv` bzw. `XinEnv`. Sie sind zuständig für das Empfangen bzw. Senden von Signalen von bzw. zu der `SDL-Seite`. Den größten Implementierungsaufwand habe ich in die Funktion `XoutEnv` investiert, weil sie das Mappen der Datenstrukturen und die Kommunikation über JNI mit der `Java-Seite` übernimmt. Die Funktion `XinEnv` hat nur die Aufgabe die Ergebnisse des marktbasierten Verfahrens in ein Signal zu packen und zur `SDL-Seite` zu senden. Aus dem Grund wird mehr auf die Funktion `XoutEnv` Akzent gesetzt.

7.3.1 Die Funktion XoutEnv

Die Funktion `XoutEnv` wird allgemein immer aufgerufen, wenn das SDL-Programm ein Signal zur Umgebung sendet. In unserem Fall wird sie in jedem Simulationsschritt von der SDL-Seite aufgerufen, in dem sie das Signal `Out_ClusterState` empfängt. Dieses Signal enthält als Parameter fünf Netzdaten von verschiedenen SDL-Typen. Diese Netzdaten werden dann eingelesen und in neue Datenstrukturen von JNI-Typen umgewandelt, damit sie von der Java-Seite bearbeitet werden können. In diesem Stadium ist eine ausführliche Beschreibung der relevanten SDL-Datenstrukturen und der Techniken zu ihrer Manipulation in den Environment-Funktionen unentbehrlich geworden.

7.3.1.1 Die relevanten SDL-Datenstrukturen und ihre Manipulation

In der folgenden Tabelle werden die relevanten SDL-Datenstrukturen, die ich zur Implementierung der Schnittstelle angewendet habe, zusammengefaßt.

Die relevanten SDL-Datenstrukturen von MOSIT, die zur Implementierung der Schnittstelle angewendet wurden, sind tief verschachtelt. Um einen besseren Überblick zu verschaffen, werde ich bei der Beschreibung mit den folgenden oft angewendeten grundlegenden MOSIT-Datenstrukturen anfangen. Die Abkürzungen BS, TF und TS stehen jeweils für Basisstation, Trägerfrequenz und Timeslot.

NEWTYPE feld_i ARRAY (Integer, Integer) ENDNEWTYPE;	NEWTYPE feld_r ARRAY (Integer, Real) ENDNEWTYPE;	NEWTYPE Ort_Index_Struct STRUCT BS Integer; Sektor Integer; ENDNEWTYPE;	NEWTYPE Index_Struct STRUCT BS Integer; Sektor Integer; TF Real; TS Integer; Code Integer ENDNEWTYPE;
---	--	--	---

Abb. 37: Die grundlegenden MOSIT-Datenstrukturen

Die Datenstrukturen der Signalparameter `BsList`, `ClusterState`, `TfList`, `BelUp` und `BelDown` lassen sich durch die folgenden Tabellen beschreiben.

```
NEWTYPE BSListT struct
  list feld_i;
  length Integer;
ENDNEWTYPE;
```

Abb. 38: Die Datenstruktur BSListT

```

NEWTYPE ClusterStateT
  ARRAY (Integer,BSSStateT);
ENDNEWTYPE;

NEWTYPE BSSStateT
  ARRAY (Integer,SectorStateT);
ENDNEWTYPE;

NEWTYPE SectorStateT STRUCT
  regularLambda Real;
  regularMu Real;
  regularA Real;
  regularB Real;
  regularB_Erlang Real;
  hoLambda Real;
  hoMu Real;
  hoA Real;
  hoB Real;
  hoB_Erlang Real;
  allLambda Real;
  allMu Real;
  allA Real;
  allB Real;
  allB_Erlang Real;
  nrOfSlots Integer;
  berUp Real;
  berDown Real;
ENDNEWTYPE;

```

Abb. 39: Die Datenstruktur ClusterStateT

```

NEWTYPE TF_Zuordnung_Feld
  ARRAY (Ort_Index_Struct,TF_Inhalt_Struct)
ENDNEWTYPE;

NEWTYPE TF_Inhalt_Struct STRUCT
  Anz_TF_Sekt_up Integer;
  Anz_TF_Sekt_down Integer;
  TF_up   feld_r;
  TF_down feld_r;
  TF_Nr_up   TS_Feld;
  TF_Nr_down TS_Feld;
ENDNEWTYPE;

NEWTYPE TS_Feld
  ARRAY (Integer, TS_Struct)
ENDNEWTYPE;

NEWTYPE TS_Struct STRUCT
  Anz_TS Integer;
  TS feld_i;
ENDNEWTYPE;

```

Abb. 40: Die Datenstruktur TF_Zuordnung_Feld

```

NEWTYPE Kanal_Bel_Feld_up
  ARRAY (Index_Struct,Inhalt_Struct_up)
ENDNEWTYPE;

NEWTYPE Inhalt_Struct_up STRUCT
  MS Integer:=0;
  Anf_Bel Real:=500000;
  reserviert Integer:=0;
  belegte_Slots Integer:=0;
  Sig_flag Integer:=1;
  Spreiz_Faktor Integer:=0;
ENDNEWTYPE;

```

Abb. 41: Die Datenstruktur Kanal_Bel_Feld_Up

```

NEWTYPE Kanal_Bel_Feld_down
  ARRAY (Index_Struct,Inhalt_Struct_down)
ENDNEWTYPE;

NEWTYPE Inhalt_Struct_down STRUCT
  BCCH_TF Real;
  MS Integer:=0;
  Anf_Bel Real:=500000;
  reserviert Integer:=0;
  belegte_Slots Integer:=0;
  Sig_flag Integer:=1;
  Spreiz_Faktor Integer:=0;
ENDNEWTYPE;

```

Abb. 42: Die Datenstruktur Kanal_Bel_Feld_Down

Diese SDL-Datenstrukturen werden vom Code-Generator des SDT-Tools in C übersetzt. In den Environment-Funktionen kann man sie genau so anwenden wie andere C-Datenstrukturen. Man muß aber exakt wissen, wie sie implementiert sind, um auf bestimmte Daten innerhalb dieser Strukturen zugreifen zu können. Dafür steht eine vom SDT-Tool automatisch erzeugte Bibliothek zur Verfügung. Im folgenden werden die wichtigsten Funktionen dieser Bibliothek anhand von Beispielen vorgestellt:

- `systemTime_help = (((yPDP_Out_ClusterState)(*S))->Param1);`

Diese Funktion extrahiert den ersten Parameter des SDL-Signals `Out_ClusterState`. Die allgemeine Form lautet `((yPDP_SDL_Signal)(*S))->Paramx`.

- `yExtr_feld_i((bsList_help.list),bsagent_index);`

Diese Funktion extrahiert den Eintrag mit dem Index `bsagent_index` aus dem SDL-Feld `bsList_help.list`. Um auf einen Eintrag eines SDL-Feldes zuzugreifen, wird die Funktion `yExtr_SDL_Feld_Name(SDL_Feld,index)` eingesetzt.

- `result = yMake_Transactions(3,SDL_kaufliste,SDL_verkaufliste);`

Diese Funktion erzeugt eine neue Instanz der SDL-Datenstruktur `Transactions`.

- `(*yAddr_TransactionList((&result.kauf_list),trans))=Trans_help;`

Diese Funktion fügt den Eintrag `Trans_help` zum SDL-Feld `result.kauf_list` vom SDL-Typ `TransactionList` an der Stelle `trans`. Die allgemeine Form lautet `(*yAddr_SDL_Feld_Typ((&SDL_Feld),index)) = Eintrag`.

Mit Hilfe dieser Funktionen werden die SDL-Datenstrukturen entpackt. Anschließend werden diese Daten kurz bearbeitet, um sie an die Java-Datenstrukturen anzupassen: Die Informationen über die Belegung der Kanäle sind zum Beispiel in verschiedenen SDL-Datenstrukturen gespeichert. In der Funktion `XoutEnv` werden diese Informationen interpretiert, zusammengefaßt und für die Java-Seite vorbereitet. Schließlich kommt JNI zum Einsatz, um die entpackten Datenstrukturen und die daraus gewonnenen Informationen zur Java-Seite schicken zu können und dann die Ergebnisse zu empfangen.

7.3.1.2 Die Schnittstelle über JNI

Die wichtigsten JNI-Funktionen, die zur Implementierung angewendet wurden, wurden im vorigen Kapitel behandelt. Diese Funktionen rufen Java-Methoden auf, erzeugen Java-Klasseninstanzen und Objekte oder bearbeiten die von der Java-Seite kommenden Objekte. Die eigentlichen Java-Klassen und Methoden werden im nächsten Kapitel ausführlich beschrieben.

Die wichtigsten Aktionen, die die Funktion `XoutEnv` mit Hilfe von JNI ausführt, werden durch die folgende Tabelle erläutert.

Ausgeführte Aktionen	Beschreibung
<pre>JNIEnv *env; JavaVM *jvm; JDK1_1InitArgs vm_args; vm_args.version = 0x00010001; JNI_GetDefaultJavaVMInitArgs(&vm_args); vm_args.classpath="path"; jint res; res = JNI_CreateJavaVM(&jvm,&env,&vm_args); if (res < 0) { fprintf(stderr, "Can't create Java VM\n"); exit(1); }</pre>	<p>Eine virtuelle Maschine JVM wird gestartet. Diese Aktion wird nur beim ersten Schritt der Simulation ausgeführt. Die JVM wird am Ende der Simulation automatisch gestoppt.</p>
<pre>Jclass cls; cls = (*env)->FindClass(env, "Prog"); jmethodID mid_init; mid_init = (*env)->GetStaticMethodID(env,cls, "init", "()V"); if (mid_init == 0) {fprintf(stderr, "Can't find Prog.init\n"); exit(1);} (*env)->CallStaticObjectMethod(env, cls, mid_init);</pre>	<p>Die Methode <code>init</code> der Klasse <code>Prog</code> wird beim ersten Schritt der Simulation aufgerufen. Sie dient zur Initialisierung der Java-Seite.</p>
<pre>jobject Network; jmethodID mid_Network_Init; mid_Network_Init = (*env)->GetMethodID(env, (*env)-> FindClass(env,"Network"),"<init>","()V"); if (mid_Network_Init == 0) {fprintf(stderr, "Can't find Prog.Network\n"); exit(1);} Network = (*env)->NewObject(env,(*env)->FindClass (env,"Network"),mid_Network_Init); //Schleife über alle Sektoren (*env)->CallVoidMethod(env,Network , mid_Set_Network_Node,C_J_SectorStateT,basis_index, sektor_index);</pre>	<p>Eine Klasseninstanz <code>Network</code> wird bei jedem Simulationsschritt erzeugt. Alle Netzdaten werden in ihr durch eine Schleife über alle Netzsektoren gepackt.</p>
<pre>jobject Result; jmethodID mid_Process_Network; mid_Process_Network = (*env)->GetStaticMethodID(env, cls,"Process_Network","(LNetwork;)LResult;"); if (mid_Process_Network == 0) {fprintf(stderr, "Can't find Prog.Process_Network\n"); exit(1);} Result=(*env)->CallStaticObjectMethod(env, cls, mid_Process_Network,Network);</pre>	<p>Das ist der Hauptaufruf. Das Objekt <code>Network</code> wird als Parameter zur Methode <code>Process_Network</code>, die der Kern des marktbasierten Verfahrens darstellt, übergeben. Der Rückgabeparameter wird im Objekt <code>Result</code> gespeichert.</p>

Abb. 43: Die wichtigsten JNI Abläufe

7.3.2 Die Funktion `XinEnv`

Die Funktion `XinEnv` dient zum Senden von Signalen zur SDL-Seite. In unserem Fall fügt sie den Parameter `result`, der die durchzuführenden Transaktionen enthält, ins SDL-Signal `In_Result` ein und sendet es zum SDL-Programm weiter:

```
S = xGetSignal( In_Result, xNotDefPid, xEnv);
((yPDP_In_Result)S)->Param1 = result;
SDL_Output( S, xSigPrioPar(xDefaultPrioSignal) (xIdNode *)0 );
```

7.4 Die JAVA-Seite

7.4.1 Einleitung

Der Kern des marktbasierten Ressourcenverteilungsverfahrens ist auf der Java-Seite implementiert. Der marktbasierte Algorithmus wird während der Simulation in konstanten Abständen von den Environment-Funktionen aufgerufen. Die Java-Seite verfügt dann über die Netzdaten und trifft abhängig von ihnen Entscheidungen zur Verteilung der Mobilfunk-Ressourcen. Diese Entscheidungen werden anschließend zur SDL-Seite geschickt, wo sie schließlich ausgeführt werden. Im Rahmen dieser Diplomarbeit sollte die Java-Seite so implementiert werden, daß sie als Rahmenwerk für nachfolgende Studien angewendet werden kann. Aus dem Grund wurde bei der folgenden Implementierung auf Wiederverwendbarkeit der Funktionalitäten geachtet, um bei späteren Arbeiten zum Thema dynamische Ressourcenverteilung in Mobilfunksystemen eine aufwendige Einarbeitung in MOSIT und seinen Schnittstellen zu vermeiden. Die Implementierten Klassen und Methoden bilden ein Framework mit einer Bibliothek, die die Entwicklung eines dynamischen Ressourcenverteilungsverfahrens stark vereinfacht. Das von mir entwickelte und realisierte Verfahren kann mit minimalen Aufwand durch ein anderes ersetzt werden, indem die Hauptmethode geändert wird.

Im folgenden werden die implementierten Klassen und Methoden ausführlich beschrieben.

7.4.2 Datenstrukturen für die Netzdaten

Die aktuellen Netzdaten werden bei jedem Aufruf des marktbasierten Verfahrens neu erzeugt und in einer Variable vom Klassentyp *Temp_Network* gespeichert. Diese Variable stellt ein Abbild des Netzes in einem bestimmten Zeitschritt dar. Sie enthält alle für den Verteilungsalgorithmus relevanten Daten der Sektoren. Weil die Struktur dieser Daten verschachtelt ist, war eine Implementierung verschiedener Klassen unentbehrlich. Die folgenden Abschnitte dienen zur Erläuterung der Klassen, die zur Beschreibung der Netzdaten angewendet worden sind.

7.4.2.1 Klasse *Temp_Network*

Variablen:

<code>int BS_Anzahl;</code>	Anzahl der Basisstationen im Netz.
<code>int Sektor_Anzahl;</code>	Anzahl der Sektoren pro Basisstation.
<code>SectorStateT List[][];</code>	Zweidimensionale Liste, die die Daten der Sektoren enthält.

Konstruktor:

<pre>public Temp_Network(){ BS_Anzahl=Prog.ANZBS; Sektor_Anzahl=Prog.ANZSEK; List=new SectorStateT[Prog.ANZBS+1][Prog.ANZSEK +1]; }</pre>	Die Anzahl der Basisstationen und die Anzahl der Sektoren werden von den Klassenvariablen <i>ANZBS</i> und <i>ANZSEK</i> der Klasse <i>Prog</i> eingelesen.
---	---

Methoden:

<pre>public void Set_Network_Node(SectorStateT node, int bs, int sektor){ List[bs][sektor]=node;}</pre>	Fügt einen Sektor in die Liste ein. Ein Sektor wird durch eine Sektornummer und die entsprechende Basisstationsnummer indexiert.
---	--

7.4.2.2 Klasse *SectorStateT***Variablen:**

int bs;	Basisstationsnummer
int sektor;	Sektornummer
float regularLambda;	Ankunftsrate (ohne Handover)
float regularMu;	Bedienrate (ohne Handover)
float regularA;	Angebot (ohne Handover)
float regularB;	Realer Verlust
float regularB_Erlang;	Erlang'scher Verlust
float hoLambda;	Handover- Ankunftsrate
float hoMu;	Handover- Bedienrate
float hoA;	Handover- Angebot
float hoB;	Realer Handover-Verlust
float hoB_Erlang;	Erlang'scher Handover-Verlust
float allLambda;	Gesamte Ankunftsrate
float allMu;	Gesamte Bedienrate
float allA;	Gesamtes Angebot
float allB;	Gesamter realer Verlust
double allB_Erlang;	Gesamte Erlang'scher verlust
int nrOfSlots;	Anzahl der Kanäle (Frequenzen)
float berUp;	Bitfehlerrate Uplink
float berDown;	Bitfehlerrate Downlink
Sektor_TF sektorTF;	Frequenzen, Belegungen des Sektors

Konstruktor:

<pre>public SectorStateT(){ regularLambda=0; sektorTF=new Sektor_TF(0,0);}</pre>	Beim Erzeugen einer neuen Instanz werden alle Variablen initialisiert und gleich null gesetzt.
---	--

7.4.2.3 Klasse *Sektor_TF***Variablen:**

int bs;	Basisstationsnummer
int length;	Länge der Trägerfrequenzliste des Sektors
Vector TF_List;	Trägerfrequenzliste (Einträge vom Typ TF)

Konstruktor:

<pre>public Sektor_TF(int lengthh,int bss){ bs=bss; length=lengthh; TF_List=new Vector();}</pre>	Erzeugt eine neue Instanz der Klasse <i>Sektor_TF</i> . Die Länge der Frequenzliste des Sektors und die Basisstationsnummer der Basisstation, zu der der Sektor gehört, werden dabei initialisiert.
--	---

7.4.2.4 Klasse *TF***Variablen:**

float TS_Freq;	Trägerfrequenz
int TS_Anzahl;	Anzahl der Timeslots (Kanäle) der Trägerfrequenz
Vector TS_List;	Liste der Timeslots (Einträge vom Typ Integer). Die Liste enthält Werte zwischen 1 und 8.
Vector TS_Bel;	Liste der Belegungsinformation der Timeslots (0: frei oder 1: belegt). Diese Liste ist analog zu TS_List.

Konstruktor:

<pre>public TF(float freq,int length){ TS_Freq=freq; TS_Anzahl=length; TS_List=new Vector(); TS_Bel=new Vector();}</pre>	Erzeugt eine neue Instanz der Klasse <i>TF</i> . Die Trägerfrequenz und die Anzahl der Timeslots dieser Frequenz werden dabei initialisiert.
--	--

7.4.2.5 Klasse *Koordinaten*

Variablen:

<code>double r;</code>	Radius der Zelle
<code>double x;</code>	Position der Zelle auf der x-Achse
<code>double y;</code>	Position der Zelle auf der y-Achse

Konstruktor:

<code>public Koordinaten(double r_help, double x_help, double y_help)</code>	Erzeugt eine neue Koordinateninstanz und setzt die übergebenen Werte ein.
--	---

7.4.3 Datenstrukturen für den marktbasierten Algorithmus

Außer den Klassen für die Netzdaten wurden Klassen für den marktbasierten Algorithmus implementiert. Sie dienen der Beschreibung der auszuführenden Transaktionen und des finalen Ergebnisses des Verfahrens mit den verschiedenen auszuführenden Entscheidungen.

7.4.3.1 Klasse *Transaktion*

Variablen:

<code>int k_bs;</code>	Nummer der Käuferzelle (BS)
<code>int k_sek;</code>	Nummer des Käufersektors
<code>int v_bs;</code>	Nummer der Verkäuferzelle (BS)
<code>int v_sek;</code>	Nummer des Verkäufersektors
<code>float freq;</code>	Zu verkaufende Frequenz
<code>double gewinn;</code>	Durch die Transaktion erzielter Gewinn
<code>int updated;</code>	Flag, ob die Transaktion beim Update upgedated wurde. 1: upgedated, 0: noch nicht upgedated

Konstruktor:

<code>public Transaktion(int kbs, int ksek, int vbs, int vsek, float freq_help, double gewinn_help, int upd)</code>	Erzeugt eine neue Transaktionsinstanz und setzt die übergebenen Werte ein. Die Klasse <i>Transaktion</i> steht für die Zwischenergebnisse (Transaktionen, die noch nicht endgültig zum finalen Ergebnis gehören)
---	--

7.4.3.2 Klasse *End_Transaktion*

Variablen:

<code>int bs;</code>	Nummer der betroffenen Zelle (Käufer/Verkäufer)
<code>int sek;</code>	Nummer des betroffenen Sektors (Käufer/Verkäufer)
<code>int freq;</code>	Betroffene Frequenz
<code>int slot;</code>	Der Zu verkaufende/kaufende Slot
<code>double gewinn;</code>	Durch die Transaktion erzielter Gewinn

Konstruktor:

<code>public End_Transaktion(int bs_help, int sek_help, float freq_help, int slot_help, double gewinn_help)</code>	Erzeugt eine neue Transaktionsinstanz und setzt die übergebenen Werte ein. Die Klasse <i>End_Transaktion</i> steht für die Endergebnisse (Transaktionen, die endgültig zum finalen Ergebnis gehören). Sie stellt einen Kauf/Verkaufseintrag der Kauf/Verkaufsliste dar.
--	--

7.4.3.3 Klasse *Result*

Variablen:

<code>int size;</code>	Länge der Transaktionslisten
<code>End_Transaktion kaufliste[];</code>	Kaufliste: Einträge vom Typ <i>End_Transaktion</i>
<code>End_Transaktion verkaufliste[];</code>	Verkaufsliste: Einträge vom Typ <i>End_Transaktion</i>

Konstruktor:

<code>public Result(Vector kauf, Vector verkauf)</code>	Erzeugt eine Result-Instanz und füllt die Kauf- und Verkaufslisten mit den Daten der übergebenen Listen auf. Die Klasse stellt die finalen Ergebnisse des Algorithmus dar, die zur SDL-Seite geschickt werden.
---	--

7.4.4 Das Hauptprogramm

Die tatsächlichen Funktionalitäten und Methoden sind in der Klasse *Prog* implementiert worden. Mit Hilfe dieser Klasse wird mit den Environment-Funktionen kommuniziert. Beim ersten Aufruf des marktbasierten Verfahrens wird die Methode *init* der Klasse *Prog* aufgerufen. Diese Methode dient zur Initialisierung der globalen Variablen und zum Lesen einiger Simulationsparameter aus den Initialisierungsdateien. Die Hauptmethode der Klasse *Prog*, die den marktbasierten Algorithmus realisiert, ist die Methode *Process_Network*. In dieser werden die Netzdaten bearbeitet und finden die Auktionen statt. Der Eingabeparameter enthält alle relevanten Netzdaten und ist vom Klassentyp *Network*. Diese Klasse ist im Inhalt mit der Klasse *Temp_Network* identisch. Beide Klassen unterscheiden sich nur in der Struktur. Die erste nähert sich eher den SDL-Datenstrukturen, während die zweite für die weitere Bearbeitung der Netzdaten im Java-Programm zur Verfügung steht. Mit der Methode *Get_Temp_Network* wird eine Variable vom Typ *Temp_Network* aus einer vom Typ *Network* erzeugt. Der Rückgabeparameter der Methode *Process_Network* ist vom Typ *Result* und enthält die auszuführenden Transaktionen, für die sich der marktbasierte Algorithmus anhand von Hilfsmethoden entschieden hat. Im folgenden werden der Algorithmus und die eingesetzten Methoden beschrieben.

7.4.4.1 Die Methode *init* und die Initialisierungsdaten

Diese Methode wird nur beim ersten Aufruf des marktbasierten Verfahrens aufgerufen. In ihr werden Initialisierungsparameter gelesen und Klassenvariablen gesetzt. Die eingelesenen Initialisierungsdateien werden im folgenden aufgezählt:

- *Simulation.ini*: befindet sich unter dem Wurzelpfad und enthält den Parameterpfad, unter dem sich die Parameterdateien befinden.
- *pfade_unix.ini*: befindet sich unter dem Parameterpfad und enthält den Bewegungsprofilpfad, den Basisstationenpfad, unter dem sich die initialen Basisstationendaten befinden und den Inipfad, unter dem sich die Initialisierungsdateien befinden.
- *Netz.ini*: befindet sich unter dem Parameterpfad und enthält die Initialisierungsparameter des Netzes (Anzahl der Basisstationen, Anzahl der Mobilstationen, Anzahl der sich nach einem Bewegungsprofil bewegendenden Mobilstationen...).
- *Java.ini*: befindet sich unter dem Inipfad und enthält die Gewichte der Kostenfunktion, die Schwelle des marktbasierten Algorithmus und die minimale und maximale Anzahl der Frequenzen, die ein Sektor besitzen darf. Darüber hinaus enthält die Datei einen

Schalter, der das Verhalten des marktbasierten Verfahrens stark beeinflusst. Anhand dieses Schalters wird die Definition der Nachbarschaft festgelegt. Entweder werden alle Sektoren der Nachbarzellen als Nachbarsektoren betrachtet oder nur die Sektoren der Nachbarzellen mit der gleichen Ausrichtung.

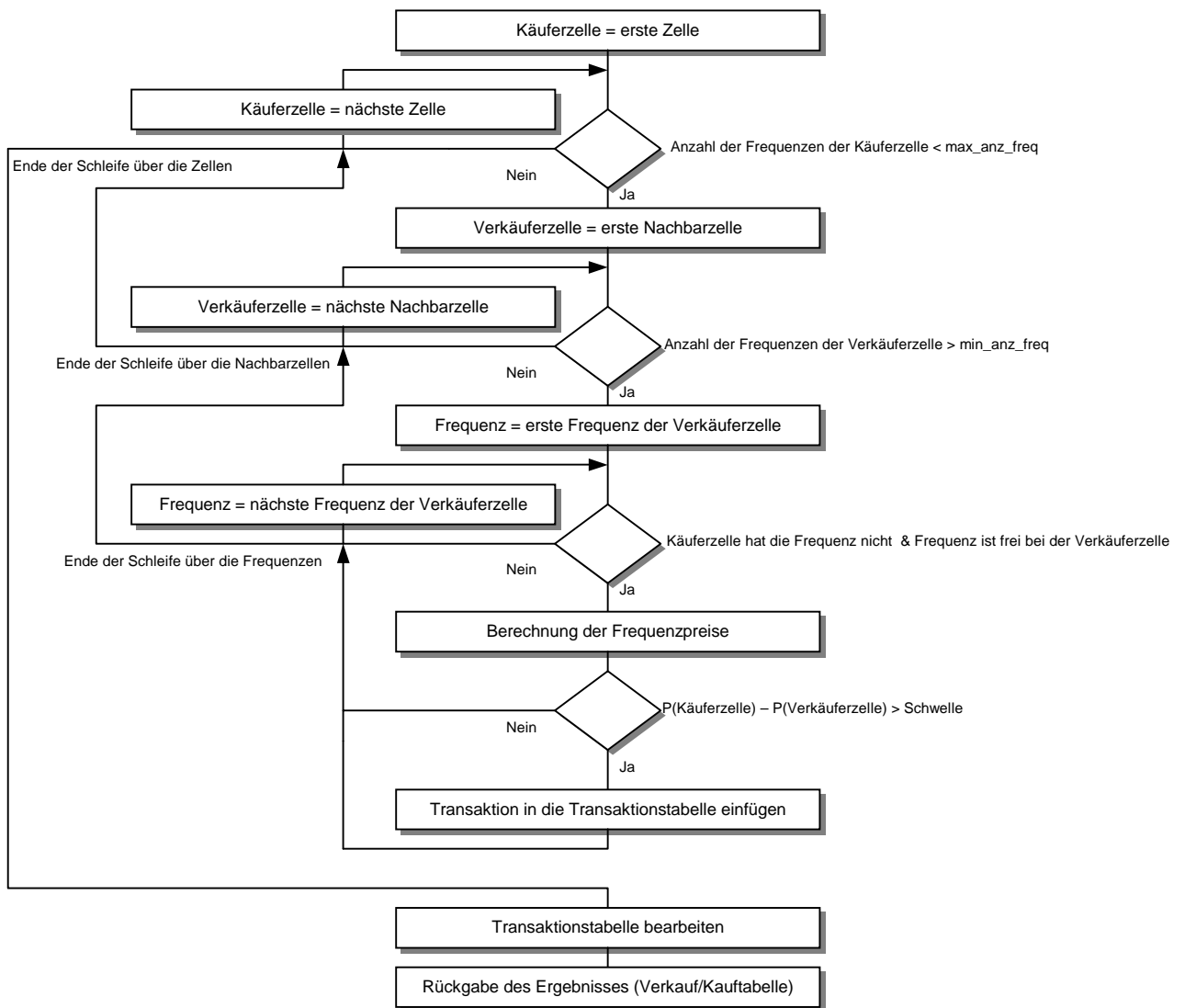
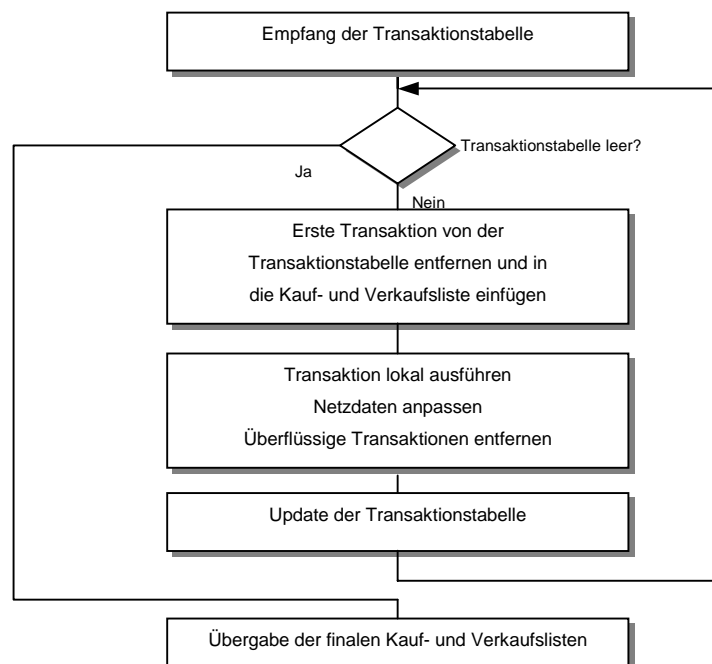
- `geom_nachbarn.ini`: befindet sich im Verzeichnis Netzmanagement unter dem Parameterpfad und enthält die Topologie des Netzes. Die Nachbarschaftsinformationen werden aus dieser Datei eingelesen und in eine Datenstruktur (Liste von Vektoren) gespeichert. Jede Zelle erhält einen Vektor an der entsprechenden Stelle in der Liste, der die Nummern der Nachbarzellen beinhaltet.
- `bsx.par`: befinden sich unter dem Basisstationenpfad und enthalten die Basisstationsdaten. Wichtig für die Berechnung des Störgliedes sind die Koordinaten der Zellen, die aus diesen Dateien eingelesen und in einer Datenstruktur (Liste von Objekten der Klasse *Koordinaten*) gespeichert werden.

7.4.4.2 Die Methoden zur Realisierung des marktbasierten Algorithmus

Diese Methode stellt den Kern des marktbasierten Algorithmus dar. Sie wird in regelmäßigen Abständen von den Environment-Funktionen aufgerufen. Sie bekommt die Netzdaten übergeben, setzt anschließend den marktbasierten Algorithmus ein und liefert schließlich die Ergebnisse zu den Environment-Funktionen. In der Methode *Process_Network* werden alle möglichen Transaktionen in eine Transaktionstabelle eingefügt, indem alle Zellen als Käuferzellen durchlaufen werden. Dabei wird eine Sortierung gewährleistet, so daß immer die Transaktion mit dem höchsten Gewinn an erster Stelle steht. Die Verkäuferzellen sind die Nachbarzellen, wobei der Begriff Nachbar verschiedene Definitionen haben kann. Anhand eines Initialisierungsparameters wird entschieden, was mit Nachbarn gemeint ist. Es können alle Sektoren, die sich in einer Nachbarzelle befinden, sein oder die entsprechenden Sektoren in den Nachbarzellen, die in die gleiche Richtung zeigen.

Nachdem die Transaktionstabelle vollständig ist, wird sie von der Methode *Process_Tabelle* bearbeitet, um die überflüssigen Transaktionen zu entfernen und die finalen Ergebnisse zu liefern. Innerhalb der Methode *Process_Tabelle* werden die Transaktionen iterativ auf den neuen Stand gebracht. Die Update-Funktionalitäten werden von der Methode *Update_Tabelle* übernommen.

Im folgenden werden diese Methoden und die dahintersteckenden Algorithmen anhand von Ablaufdiagrammen erläutert.

Abb. 44: Ablaufdiagramm der Methode *Process_Network*Abb. 45: Ablaufdiagramm der Methode *Process_Tabelle*

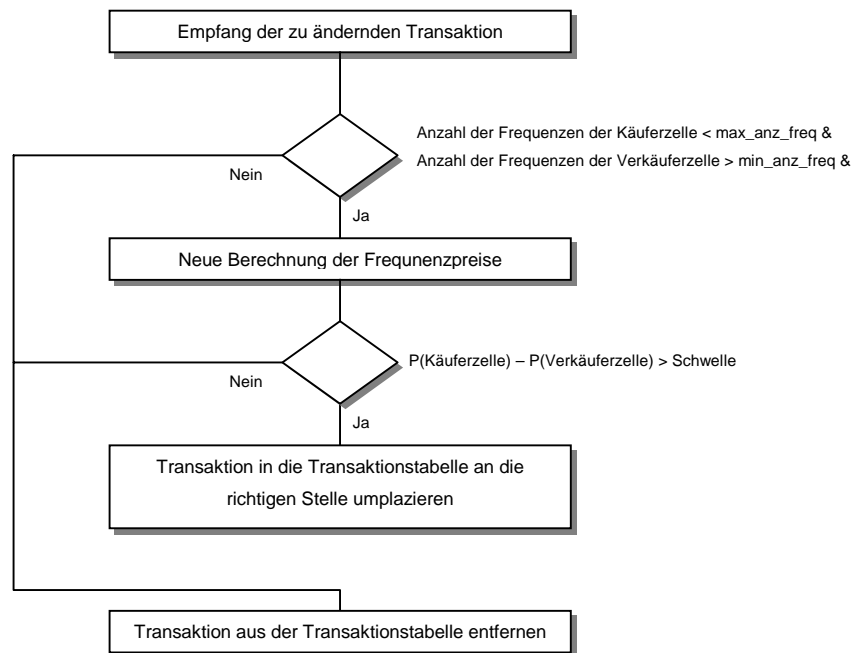


Abb. 46: Ablaufdiagramm der Methode *Update_Tabelle*

Nachdem der Algorithmus des marktbasierten Verfahrens im Detail beschrieben worden ist, werden im folgenden die Methoden der Klasse *Prog*, die zur Realisierung des Verfahrens implementiert worden sind, kurz beschrieben. Weil sie allgemeine Funktionalitäten bieten, stellen diese Methoden eine Bibliothek für weitere Arbeiten dar:

- *Process_Network*

```
public static Result Process_Network(Network net)
```

Hauptmethode, die von den Environment-Funktionen aufgerufen wird; bekommt die Netzdaten *net* als Eingabeparameter und liefert das die auszuführenden Transaktionen zurück.

- *Process_Tabelle*

```
public static void Process_Tabelle(Vector transaktionstabelle, Vector
kauf_tabelle, Vector verkauf_tabelle)
```

bearbeitet die Transaktionstabelle, und daraus füllt sie die Kauf- und Verkaufslisten auf. Die Parameter *kauf_tabelle* und *verkauf_tabelle* sind in/out Variablen.

- *Update_Tabelle*

```
public static int Update_Tabelle(Vector transaktionstabelle,int index)
```

bearbeitet den Eintrag der Transaktionstabelle mit dem Index *index* neu und fügt ihn eventuell an einer neuen Stelle der Transaktionstabelle ein. Der Rückgabeparameter ist die Stelle der Transaktionstabelle, ab der noch nicht upgedated wurde.

- `Get_Freq_Index`

```
public static int Get_Freq_Index(int bs, int sek, float freq)
```

liefert den Index der Frequenz *freq* in den Datenstrukturen des Sektors *sek* der Basisstation *bs*.

- `Get_BS_List_With_Freq`

```
public static Vector Get_BS_List_With_Freq(float freq)
```

liefert eine Liste der Basisstationen, die die Frequenz *freq* besitzen.

- `insertTransaktion`

```
public static int insertTransaktion(Vector tt, Transaktion t)
```

fügt eine Transaktion in die Transaktionstabelle ein. Der Rückgabeparameter ist der Index des Eintrages. Die Transaktionstabelle ist nach dem Gewinn fallend sortiert.

- `Get_Freq_Price`

```
public static double Get_Freq_Price(int bs, int sek, float freq)
```

berechnet den Preis der Frequenz *freq* des Sektors *sek* der Basisstation *bs* und liefert ihn als Ergebnis zurück.

- `Freq_Free`

```
public static boolean Freq_Free(int bs, int sek, int freq_index)
```

liefert den wert *true*, wenn die Frequenz mit dem Index *freq_index* des Sektors *sek* der Basisstation *bs* frei ist, sonst liefert sie den Wert *false* (Frequenz belegt).

- `Have_Freq`

```
public static boolean Have_Freq(int bs, int sek, float freq)
```

liefert den wert *true*, wenn der Sektor *sek* der Basisstation *bs* die Frequenz *freq* besitzt, sonst liefert sie den Wert *false*.

- `Neighbours_Have_Freq`

```
public static boolean Neighbours_Have_Freq(int bs, int sek, int nachbar_bs, float freq)
```

liefert den wert *true*, wenn mindestens eine der Nachbarstationen der Basisstation *bs* außer der Basisstation *nachbar_bs* die Frequenz *freq* besitzt, sonst liefert sie den Wert *false*.

- `Get_Freq_Interference`

```
public static double Get_Freq_Interference(int bs, int sek, float freq)
```

berechnet für den Sektor *sek* der Basisstation *bs* die Störung der Frequenz *freq*. Dieser Wert wird zurückgeliefert und dient zur Berechnung des Störgliedes des Frequenzpreises.

- `Get_Co_Interference`

```
public static double Get_Co_Interference(int bs1, int sek1, int bs2, int sek2)
```

liefert einen Wert, der aussagt, wie stark die Sektoren *sek1* und *sek2* der Basisstationen *bs1* und *bs2* sich gegenseitig stören können.

- `Get_Free_Channel_Nr`

```
public static int Get_Free_Channel_Nr(int bs, int sek)
```

liefert die Anzahl der freien Frequenzen des Sektors *sek* der Basisstation *bs* zurück. Das wird für die Berechnung des Reichtumsgliedes angewendet.

- `Get_B`

```
public static double Get_B(float A,int N)
```

berechnet den sich aus dem Angebot *A* und der Kanalanzahl *N* ergebenden Erlang'schen Verlust.

8 Simulation und Ergebnisse

Nach der Implementierung des Verfahrens sind verschiedene Simulationen durchgeführt worden, um den marktbasierten Ansatz hinsichtlich seiner dynamischen Eigenschaften und seiner Auswirkungen auf das Verhalten des Netzes zu untersuchen. Dabei wurden verschiedene Szenarien ausprobiert. Darüber hinaus wurden die Gewichte der Kostenfunktion so variiert, daß eine optimale Bewertung der Ressourcen erreicht wird. Die gewählten Gewichte haben verschiedene Größenordnungen, so daß alle Glieder der Kostenfunktion von vergleichbarer Größenordnung sind. Das Gewicht des Reichtumsgliedes ist relativ gering gehalten, so daß eine Entscheidung anhand des Reichtumsgliedes erst beeinflußt wird, wenn zwei Zellen ohne Berücksichtigung des Reichtumsgliedes einen fast gleichen Preis für die selbe Frequenz bieten. In dem Fall wird das Reichtumsglied entscheidend, und die ärmere Zelle, d.h. die mit dem niedrigeren Vorrat an freien Frequenzen, gewinnt das Duell um die Frequenz. Eine analoge Überlegung gilt für zwei Zellen, die die selbe Frequenz verkaufen wollen. In dem Fall setzt sich dank des Reichtumsgliedes die reichere Zelle durch.

8.1 Simulationsumgebung

Es wurde mit zwei verschiedenen Netzkonfigurationen simuliert. Die erste ist durch ein Vierercluster-Netz mit 16 Basisstationen mit jeweils 3 Sektoren beschrieben, während die zweite von einem größeren Siebenercluster-Netz mit 25 Basisstationen, die jeweils 3 Sektoren versorgen, ausgeht. Dabei verhält sich jeder Sektor autonom wie eine Zelle. Der Unterschied zwischen beiden Konfigurationen ist die gesamte Kapazität. Der Vorrat an Frequenzen ist beim größeren Netz höher als beim kleineren. Im Initialzustand verfügt ein Sektor bei beiden Konfigurationen über 5 Frequenzen. Ein Sektor darf Maximal über 8 und minimal über 3 Frequenzen verfügen. Die untere Grenze wurde eingestellt, um ein relativ statisches Verhalten des Netzes zu erzwingen. Die Zellen sollen nämlich auch bei einer niedrigen Last über eine minimale Reserve verfügen.

Bei den Simulationen wird Ressourcenverteilung in gleichmäßigen Zeitabständen durchgeführt, um auf einer Seite eine gewisse Stabilität des gesamten Systems zu gewährleisten und um auf der anderen Seite aussagekräftige statistische Meßwerte interpretieren zu können. Eine wiederholte Ressourcenverteilung bei jedem Schritt ist daher sinnlos.

Im folgenden werden die Parameter, die die Simulationsumgebung der durchgeführten Simulationen beschreiben, aufgezählt.

Die MOSIT-Parameter:

- Anzahl der zu simulierenden Zeitschritte.
- Gesamte Anzahl der Mobilstationen.
- Anzahl der Mobilstationen (BP-Mobilstationen), die sich nach einem bestimmten festgelegten Bewegungsprofil innerhalb der Netzfläche bewegen.
- Größe des mittleren Verkehrsangebotes.

Die für das marktbasierete Verfahren eingeführten Parameter:

- Schalter des marktbasiereten Verfahrens.
- Zeitabstand (Fenster) zwischen den Zeitschritten bei denen das Verfahren ausgeführt wird.
- Gewicht des Verkehrsgliedes.
- Gewicht des Störgliedes.
- Gewicht des Reichtumsgliedes.
- Maximale Anzahl der Frequenzen pro Sektor.
- Minimale Anzahl der Frequenzen pro Sektor.
- Schwelle der Transaktionen.
- Nachbarschaftdefintion: hier soll entschieden werden, welche Sektoren als Nachbarn betrachtet werden sollen; entweder alle Sektoren der Nachbarbasisstation oder nur diejenigen, die die gleiche Ausrichtung haben.

Wegen des hohen Zeitaufwandes der Simulationen (zirka 72 Stunden bei den alten Sun Workstations, zirka 30 Stunden bei der letztlich neu installierten Workstation) wurde die Anzahl der zu simulierenden Zeitschritte bei allen Simulationen auf 1500 Schritte gesetzt. Aus dem selben Grund sind nur diejenigen Parameter variiert worden, die eine Aussage über die Effizienz des implementierten Verfahrens geben. Bei der Auswertung der Ergebnisse werden die Einflüsse dieser Parameter untersucht.

Um die Effizienz des Verfahrens messen zu können, ist eine relativ hohe Last eingestellt worden. Das wird erreicht, indem das Angebot und die Anzahl der Mobilstationen hoch gewählt werden. Um das Verhalten des marktbasiereten Ansatzes bei einem Hotspot zu untersuchen, wurden bestimmte Szenarien simuliert, in denen zwei Züge sich in der Mitte des Netzes kreuzen.

In der folgenden Tabelle sind die bei den verschiedenen Simulationen eingenommenen Parameterwerte aufgezeichnet.

Parameter	Eingenommene Werte
Anzahl der zu simulierenden Zeitschritte	1500
Gesamte Anzahl der Mobilstationen	200
Anzahl der BP-Mobilstationen	0, 40, 134
Größe des mittleren Verkehrsangebotes	200, 150 Erlang
Schalter des marktbasierten Verfahrens	0, 1
Fenster des marktbasierten Verfahrens	10
Gewicht des Verkehrsgliedes	2
Gewicht des Störgliedes	2500000, 3000000
Gewicht des Reichtumsgliedes	0.0000002
Maximale Anzahl der Frequenzen pro Sektor	8
Minimale Anzahl der Frequenzen pro Sektor	3
Schwelle der Transaktionen	0.00005
Nachbarschaftdefintion	0 (nur gleich gerichtete Sektoren), 1(alle)

8.1.1 Netztopologie

Bei beiden Konfigurationen besteht jede Zelle aus drei Sektoren (siehe Abb. 47). Bei der ersten Konfiguration ist das Netz in Frequenzcluster der Größe 4 unterteilt, bei der zweiten in Frequenzcluster der Größe 7.

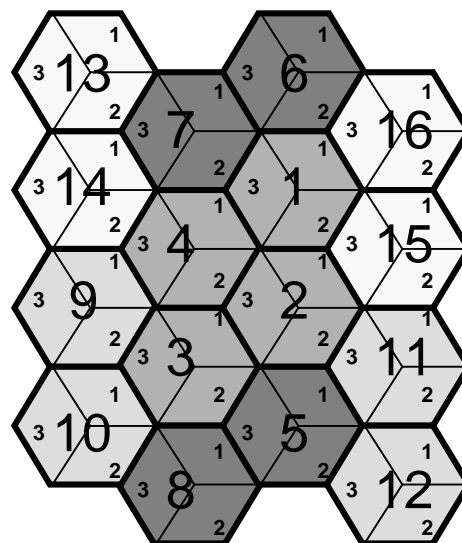


Abb. 47: Abbild des simulierten Netzes, 1. Konfiguration (16 Basisstationen)

8.1.2 Bewegungsprofile der Mobilstationen

8.1.2.1 2 Züge mit Ringen

Anhand dieses Bewegungsprofils lassen sich Verhältnisse in einem Stadtzentrum simulieren, in welchem die meisten GSM-Teilnehmer morgens ins Zentrum fahren und dieses am Abend verlassen. Dadurch steigt die Verkehrslast in der Netzmitte zunächst stark, um im Anschluß wieder zu fallen.

Es gibt drei Weisen, auf die die Teilnehmer sich innerhalb des Netzes bewegen. Ein Teil der Teilnehmer ist auf zwei Züge verteilt, die erst in das Zentrum fahren und es später wieder verlassen. Ein anderer Teil bewegt sich auf konzentrisch zur Stadtmitte angeordneten Ringen mit leicht unterschiedlichen Radien in Richtung Zentrum und dann in umgekehrter Richtung. Diese beiden Gruppen von Teilnehmern treffen sich irgendwann im Stadtzentrum, um es später wieder zu verlassen. Der Rest der Teilnehmer bewegt sich rein zufällig innerhalb des Netzes.

Abbildung 48 stellt eine laufende Simulation mit der ersten Konfiguration (16 BS) dar. Die Basisstationen werden durch numerierte Kreise, die Mobilstationen durch Punkte dargestellt. Die beiden Züge werden abgebildet. Die sich auf den konzentrischen Ringen befindenden Mobilstationen sind in der Abbildung zu erkennen. Die Mobilfunkverbindungen werden durch Verbindungslinien zwischen Mobil- und Basisstationen dargestellt.

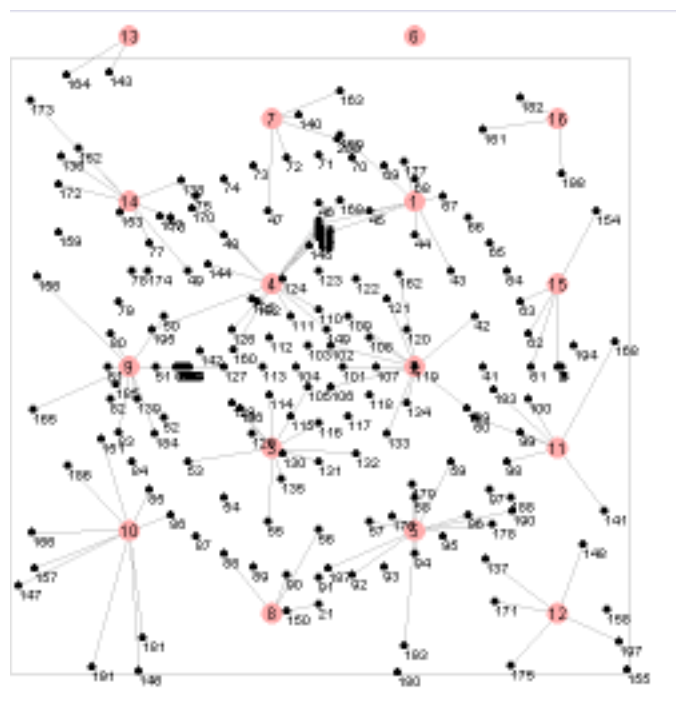


Abb. 48: Beispiel einer Simulation mit zwei Zügen und Ringen

8.1.2.2 2 Züge

Dieses Bewegungsprofil unterscheidet sich vom vorherigen dadurch, daß nur die Teilnehmer, die sich in den Zügen befinden, eine Überlastung im Stadtzentrum verursachen. Der Rest der Teilnehmer wird zufällig über das gesamte Netz verteilt.

Bei beiden Bewegungsprofilen ist die Anzahl der Teilnehmer jeder Gruppe frei konfigurierbar, so daß man bestimmte Szenarien simulieren kann.

8.2 Simulationsergebnisse

Es sind zwei Gruppen von Simulationen mit verschiedenen Szenarien und Parametern durchgeführt worden, um das Verhalten des implementierten marktbasierenden Verfahrens unter verschiedenen Aspekten zu untersuchen. Grundlage dieser beiden Gruppen waren zwei Konfigurationen mit zwei verschiedenen Netztopologien.

8.2.1 Erste Konfiguration: 16 Basisstationen, Clustergröße 4

Die ersten Simulationen nutzten die erste Netztopologie (16 BS Clustergröße 4). 200 Mobilstationen wurden simuliert, davon haben sich 134 nach dem ersten Bewegungsprofil (2 Züge mit Ringen) bewegt. Das Verkehrsangebot wurde auf 200 Erlang gesetzt, was einer relativ hohen Last entspricht. Durch das Bewegungsprofil und die hohe Last wurde ein Hotspot in der Netzmitte erzwungen. Mit diesen Initialisierungsparameter wurde eine Simulation mit der statischen und mehrere mit der dynamischen Ressourcenverteilung durchgeführt.

Bei den Simulationen mit der dynamischen Ressourcenverteilung wurde von den meisten Freiheitsgraden profitiert: Die Nachbarsektoren waren alle Sektoren, die zu den Nachbarzellen gehören, so daß die Käufersektoren viele Transaktionsalternativen hatten. Außerdem wurde bei den Transaktionen nicht explizit ausgeschlossen, daß zwei benachbarte Zellen eine selbe Frequenz benutzen. Dieser Fall wird implizit durch die Preisgestaltung (Störglied) vermieden aber nicht ausgeschlossen. Aber gerade bei der gewählten Konfiguration ist dieser Fall nicht zu vermeiden. Wegen der dichten Wiederholung der Frequenzen (vierer Cluster) befindet sich jede bei einer Transaktion verschobene Frequenz nämlich von mindestens einer Nachbarzelle umgeben, die die gleiche Frequenz besitzt (siehe Abbildung 49).

Man ist in der Situation zwischen zwei Möglichkeiten wählen zu müssen. Entweder läßt man zu, daß sich eine solche Konstellation bildet und führt Transaktionen aus, so daß der Verlust

abgemildert wird oder man bleibt bei einem relativ statischen Verhalten, damit die Interferenzen nicht steigen, dafür aber der Verlust beinahe unverändert bleibt. Um das Verfahren bezüglich des Verlustes zu testen, habe ich die erste Möglichkeit simuliert und das Verhalten des Netzes bei der dynamischen Ressourcenverteilung mit dem bei einer statischen verglichen. Daß die Bitfehlerwahrscheinlichkeit bei so einer Konfiguration (siehe Abbildung 49) steigt war zu erwarten. Wichtig war die Effizienz des Verfahrens bei einem Hotspot und seine Wirkung auf die Blockierungsrate und die realen Verluste.

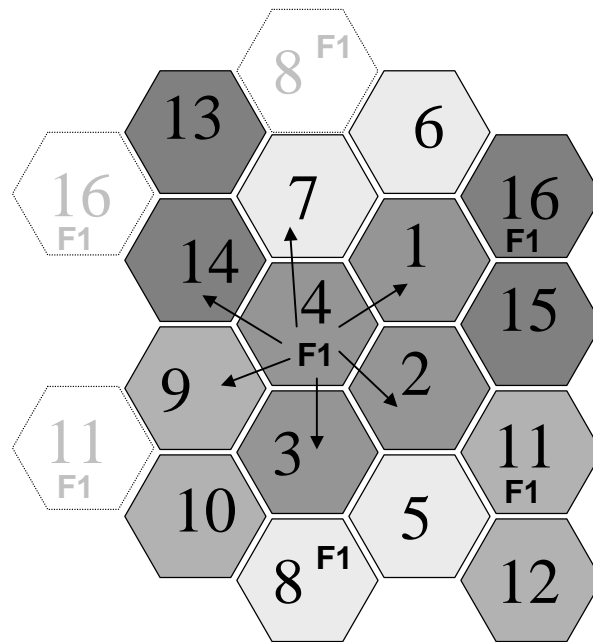


Abb. 49: Wirkung der Clustergröße auf die Interferenzlage nach einer Transaktion

Im folgenden werden zwei Sektoren untersucht, die den Zustand des Netzes typisch beschreiben. Der erste Sektor (Basisstation 4, Sektor 3) liegt in der Mitte des Netzes und wird vom Zug durchfahren. Außerdem treffen sich die sich auf den Ringen bewegenden Teilnehmer unmittelbar im Stadtzentrum (Netzmitte), u.a. in diesem Sektor (siehe Abbildungen 47 und 48). Die Last in diesem Sektor steigt also durch die Belegungswünsche der Teilnehmer, die sich konzentriert in der Netzmitte befinden. Der Sektor 3 der Basisstation 4 verhält sich deswegen hauptsächlich als Käufer. Der zweite Sektor (Basisstation 16, Sektor 3) liegt außerhalb des Hauptverkehrs (siehe Abbildungen 47 und 48) und verhält sich aus diesem Grund als Verkäufer.

Abbildung 50 dient zur Veranschaulichung des Verkehrsangebotes in beiden Sektoren. Es ist klar zu erkennen, daß Sektor 3 der Basisstation 4 wegen den oben erwähnten Gründen eine deutlich höhere Last hat.

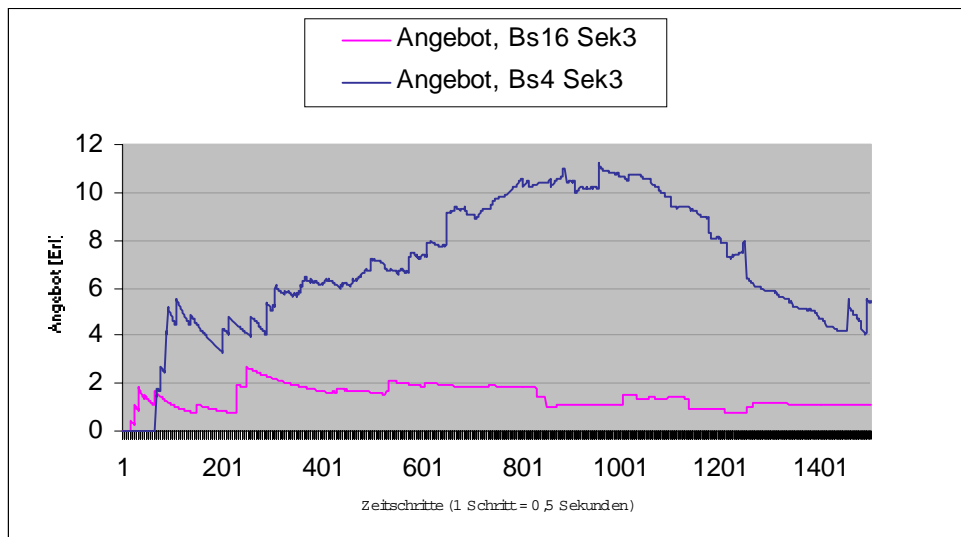


Abb. 50: Vergleich des Angebotes in einem Sektor in der Stadtmitte und außerhalb

In Abbildung 51 sind die Verluste des Sektors 3 der Basisstation 4 bei einer statischen und einer dynamischen Ressourcenverteilung dargestellt. Es sind jeweils die realen und die Erlang'schen Verluste abgebildet. Der reale Verlust ist das eigentliche Maß für die blockierten Verbindungen. Er läßt sich durch den prozentuellen Anteil der blockierten Verbindungen bezüglich der Summe der erfolgreich durchgeführten und der blockierten Verbindungen ermitteln. Der Erlang'sche Verlust ist eine statistische Größe, die die Wahrscheinlichkeit

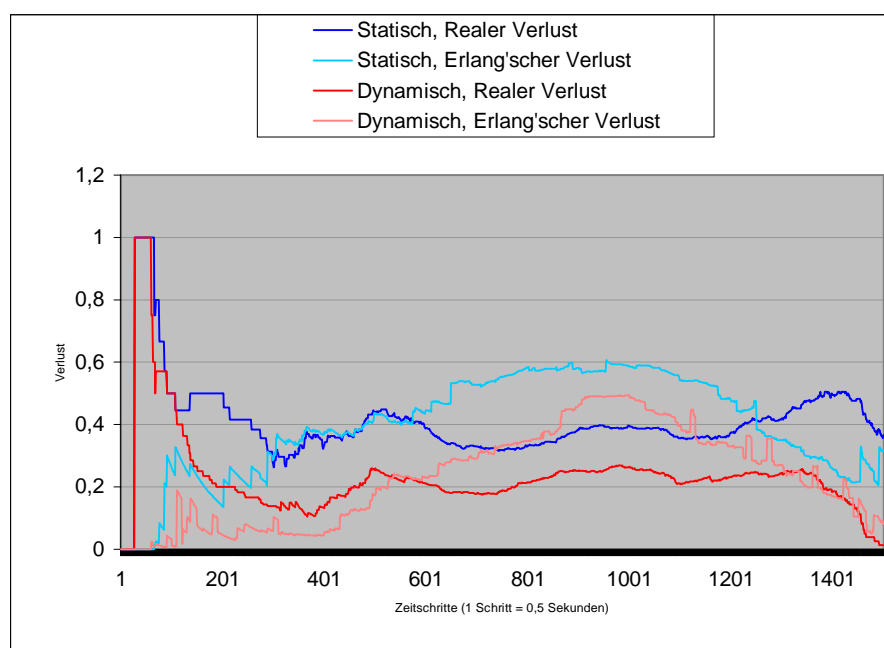


Abb. 51: Realer und Erlang'scher Verlust des Sektors 3 der BS 4 (statisch/dynamisch)

einer Blockierung darstellt. Man kann also annehmen, daß der Erlang'sche Verlust eine Vorhersage des realen Verlustes ist.

Wenn man die Verluste bei einer statischen und einer dynamischen Simulation vergleicht, zeigt sich, daß der marktbasierter Algorithmus eine deutliche Verbesserung bringt. In Abbildung 51 ist zu sehen, daß der Verlust im dynamischen Fall über die ganze Simulation niedriger ist als im statischen. Man erreicht eine Verbesserung von bis zu 50%. Das zeigt, daß der Algorithmus effektiv auf die vorhergesagten hohen Verluste reagiert. Abbildung 52 belegt diese Tatsache. Die Anzahl der Frequenzen steigt nämlich mit dem Erlang'schen Verlust, bis sie einen maximalen Wert erreicht (8 Frequenzen). Weil der Verlust am Ende der Simulation deutlich niedriger wurde, hat der Sektor entsprechend Frequenzen verkauft.

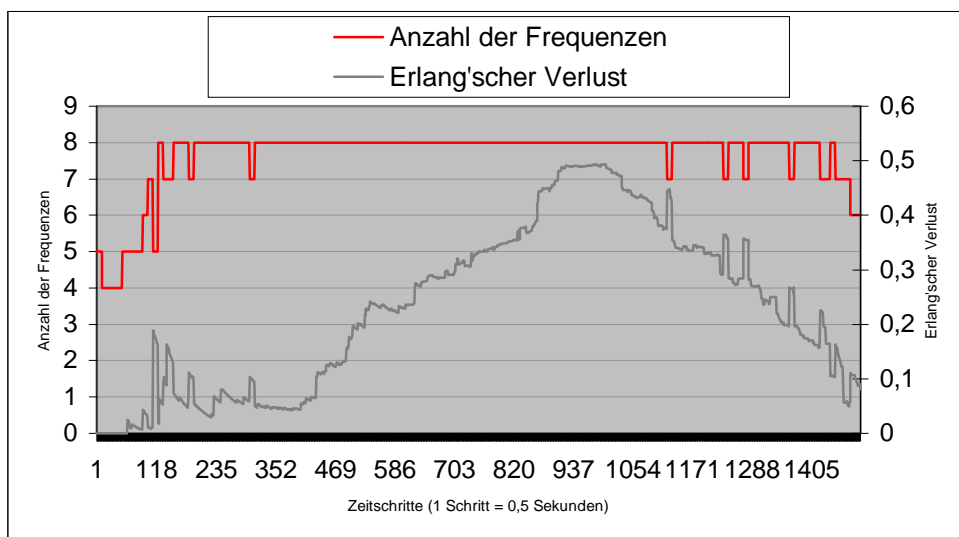


Abb. 52: Variation der Anzahl der Frequenzen in Abhängigkeit von dem Erlang'schen Verlust im Sektor 3 der BS 4

Das marktbasierter Verfahren scheint also bei einer Zelle mit einem hohen Bedarf große Vorteile bezüglich der Verluste zu bringen. Allerdings sollte nicht nur der Verlust als Maß für die Qualität des Algorithmus dienen. Es ist auch zu untersuchen, inwieweit die Ressourcenverteilung die Qualität der einzelnen Verbindungen beeinflusst. Ein gutes Beurteilungskriterium dafür liefert die Bitfehlerwahrscheinlichkeit, die in Abbildung 53 zu sehen ist.

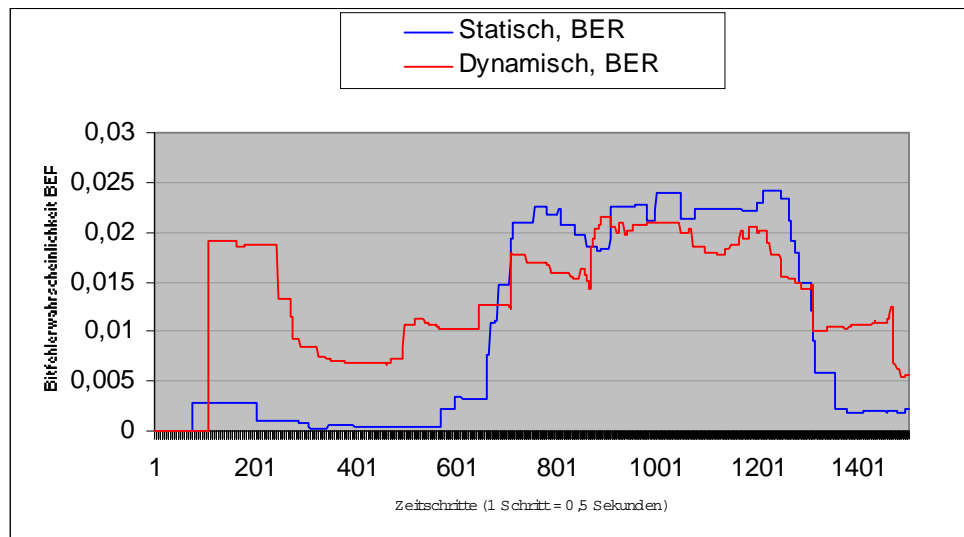


Abb. 53: Vergleich der Bitfehlerwahrscheinlichkeit im Sektor 3 der BS 4 (statisch/dynamisch)

Wie erwartet, hat die Ressourcenverteilung die Lage der Interferenzen im Mittel leicht verschlechtert. Wegen der großen Dichte der wiederholten Frequenzen, ist die Qualität der Verbindungen im dynamischen Fall phasenweise schlechter als im statischen, obwohl das marktbasierende Verfahren immer die besseren Frequenzen auswählt. Man kann allerdings die Empfindlichkeit des Verfahrens bezüglich der Interferenzen verstärken, indem bei der Kostenfunktion eine höhere Gewichtung des Störgliedes gewählt wird (siehe Abbildung 54). Bei der zweiten Konfiguration (siehe Kapitel 8.2.2) wird dieser Nachteil wegen der realistischeren Dichte der wiederholten Frequenzen unabhängig von der Gewichtung abgemildert.

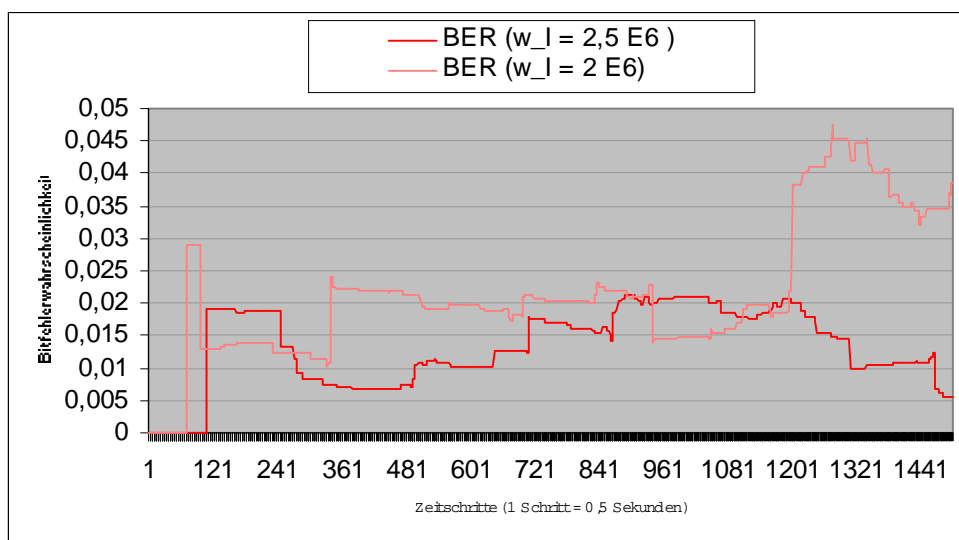


Abb. 54: Wirkung der Gewichtung auf die Bitfehlerwahrscheinlichkeit (BS 4, Sektor 3)

Im folgenden wird der zweite Sektor (Sektor 3 der Basisstation 16), der sich eher als Verkäufer verhält, untersucht. Wegen des relativ niedrigen Angebotes und des daraus, relativ zu den Sektoren in der Netzmitte, resultierenden niedrigen Erlang'schen Verlustes, sinkt die Anzahl der Frequenzen in diesem Sektor im Laufe der Simulation (Abbildung 55). Einzelne Schwankungen sind zwar zu sehen, aber das Verhalten des Sektors als Verkäufer ist eindeutig zu erkennen.

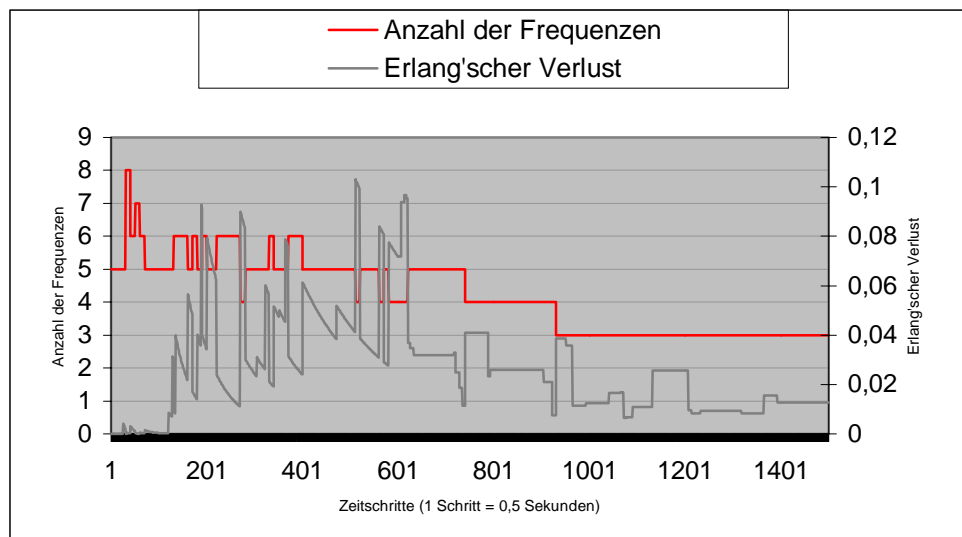


Abb. 55: Variation der Anzahl der Frequenzen in Abhängigkeit von dem Erlang'schen Verlust im Sektor 3 der BS 16

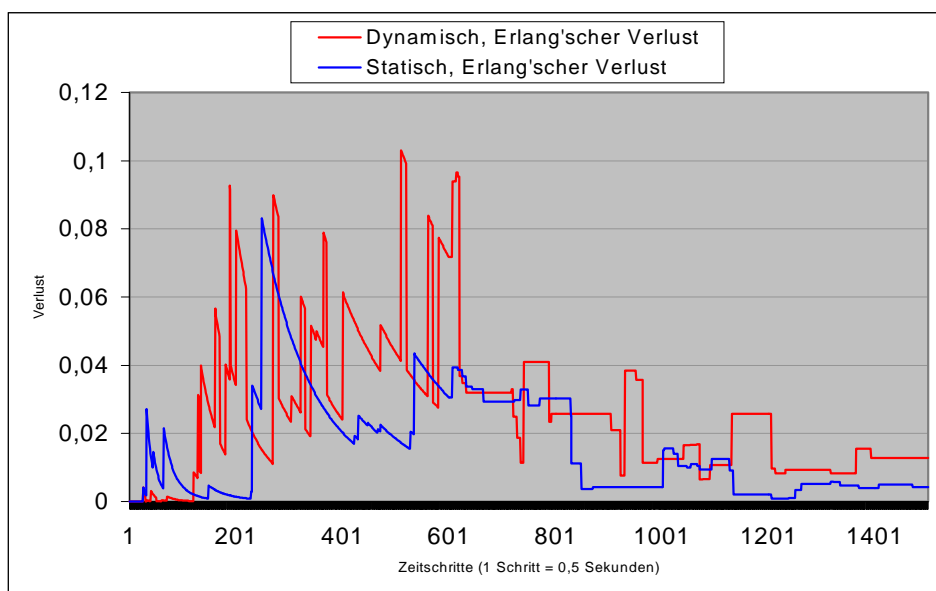


Abb. 56: Realer und Erlang'scher Verlust des Sektors 3 der BS 16 (statisch/dynamisch)

Abbildung 56 zeigt die Verluste des Sektors bei einer statischen und einer dynamischen Ressourcenverteilung. Der reale Verlust bleibt in beiden Fällen null. Der Erlang'sche Verlust hingegen steigt zwar bei dem dynamischen Fall, bleibt aber in einem akzeptablen Bereich. Eine solche geringe Erhöhung ist bei einem Verkäufersektor zu erwarten, weil er immerhin Ressourcen abgibt und dadurch die Wahrscheinlichkeit einer Blockierung erhöht. Wenn man dabei bedenkt, daß diese verkauften Ressourcen von einem Nachbarsektor, der in „Not“ ist, besser genutzt werden, ist eine durch die ausgeführten Transaktionen Benachteiligung eines Verkäufersektors akzeptabel.

Nachdem zwei Sektoren, die jeweils typisch für das Verhalten einer Klasse von Sektoren (Käufer, Verkäufer) sind, beschrieben wurden, ist es von Interesse geworden, anhand einer einheitlichen Darstellung das Verhalten des gesamten Netzes zu beobachten.

Abbildung 57 zeigt die über das gesamte Netz gemittelten Verluste. Es ist zu erkennen, daß die von den Käufersektoren erzielten Gewinne (Verminderung der Verluste) die oben erwähnte Erhöhung der Verluste der Verkäufersektoren bei weitem kompensieren, so daß der gesamte Gewinn des Netzes durch eine Verminderung der gesamten Verluste erhöht wird.

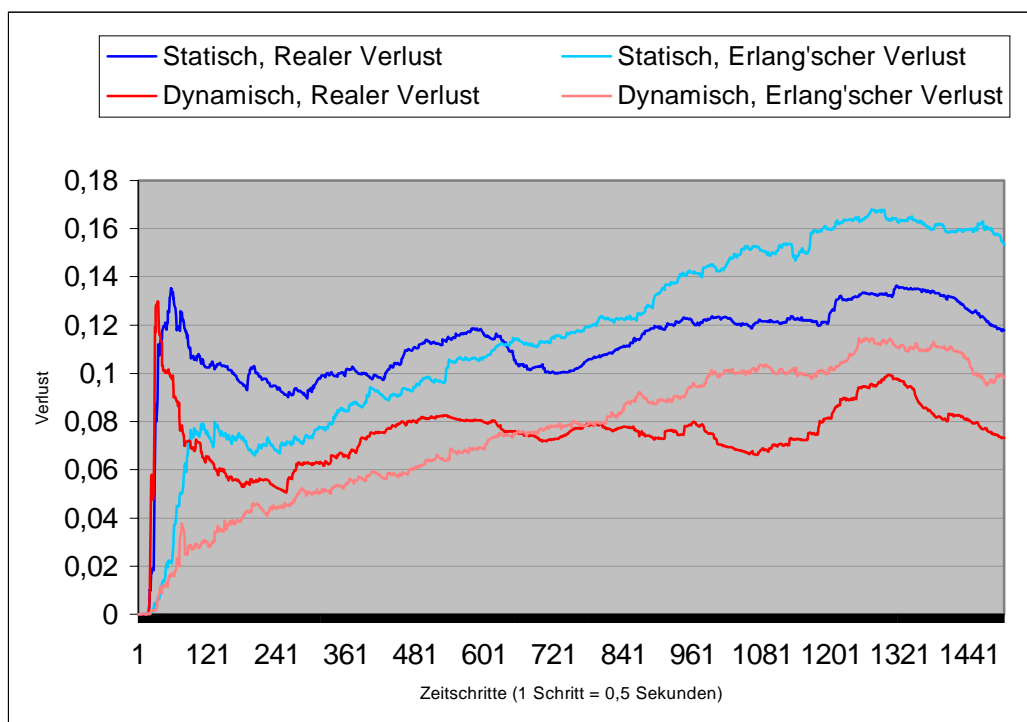


Abb. 57: Realer und Erlang'scher Verlust des gesamten Netzes (statisch/dynamisch)

8.2.2 Zweite Konfiguration: 25 Basisstationen, Clustergröße 7

Die mit der ersten Konfiguration durchgeführten Simulationen haben in Bezug auf die Verluste gute Ergebnisse geliefert. Die Qualität der Verbindungen hat sich aber bei der dynamischen Ressourcenverteilung wegen der relativ hohen Interferenzen verschlechtert. Diese Verschlechterung lag nicht direkt an dem marktbasierten Verfahren, sondern an der Konfiguration, auf deren Grundlage die Frequenzen dicht aneinander wiederholt wurden. Um eine faire Aussage über den Einfluß des implementierten Verfahrens auf die Qualität der Verbindungen zu machen, wurde mit einer zweiten Konfiguration simuliert, die einen höheren Frequenzwiederholabstand festlegt. So können anhand des marktbasierten Verfahrens Frequenzen verschoben werden, ohne die Qualität der Verbindungen stark zu beeinträchtigen. Abbildung 58 zeigt die Topologie des Netzes bei der zweiten Konfiguration. Es geht um ein Netz mit 28 Zellen mit jeweils 3 Sektoren (analog zur ersten Konfiguration). Die Clustergröße ist 7, was bedeutet, daß die Sektoren mit den gleichen Frequenzen, verglichen mit der ersten Konfiguration, in größeren Abständen verteilt sind.

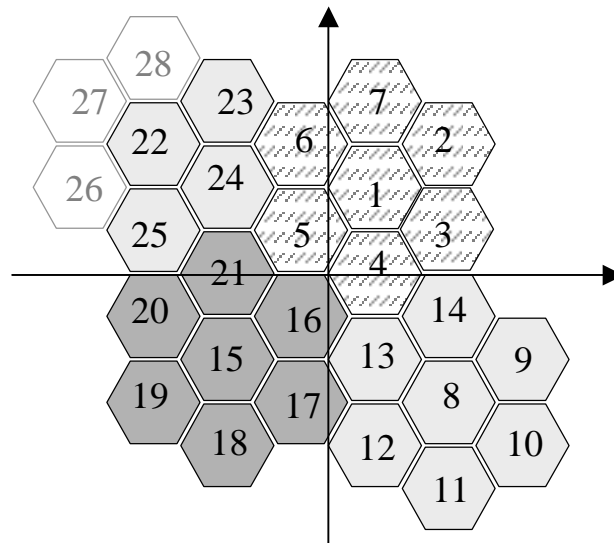


Abb. 58: Topologie des Netzes bei der zweiten Konfiguration

Die in grau abgebildeten Zellen dienen nur der Vollständigkeit des Netzbildes. Sie wurden bei den Simulationen nicht in Betracht gezogen, weil sie sich außerhalb des Netzzadius befinden. Die Verhandlungen des marktbasierten Verfahrens und die Ergebnisse der Simulationen beziehen sich also nur auf die Zellen 1-25.

Mit dieser Konfiguration wurde eine Reihe von Simulationen mit 200 Mobilstationen durchgeführt. Diese Simulationen werden in der folgenden Tabelle beschrieben.

Simulation	Angebot	Bewegungsprofil	Ressourcenverteilung	Randbedingungen
1	200	2 Züge + R (134)	statisch	1
2	200	2 Züge + R (134)	dynamisch	1
3	150	2 Züge + R (134)	statisch	1
4	150	2 Züge + R (134)	dynamisch	1
5	200	2 Züge (40)	statisch	1
6	200	2 Züge (40)	dynamisch	1
7	200	2 Züge + R (134)	dynamisch	2
8	200	2 Züge + R (134)	dynamisch	3

Randbedingungen 1: Die Nachbarsektoren eines Sektors X einer Zelle Y sind die Sektoren der Nachbarzellen der Zelle Y, die die gleiche Ausrichtung aufweisen wie der Sektor X. Zwei benachbarte Zellen dürfen nicht die gleiche Frequenz besitzen.

Randbedingungen 2: Die Nachbarsektoren eines Sektors X einer Zelle Y sind alle Sektoren der Nachbarzellen der Zelle Y. Zwei benachbarte Zellen dürfen nicht die gleiche Frequenz besitzen.

Randbedingungen 3: Die Nachbarsektoren eines Sektors X einer Zelle Y sind alle Sektoren der Nachbarzellen der Zelle Y. Zwei benachbarte Zellen dürfen die gleiche Frequenz besitzen.

Die Simulationen 1-4 dienen zur Simulation eines Hotspots bei zwei verschiedenen Angeboten, um das Verfahren bei einer inhomogen verteilten Last in Abhängigkeit des Angebotes zu untersuchen. 134 Teilnehmer der insgesamt 200 befinden sich in den zwei Zügen und auf den Ringen, was zu einer Konzentration der Last in der Netzmitte führt. Bei den Simulationen 5-6 wurde hingegen ein Szenario simuliert, bei dem die Last relativ homogen auf das ganze Netz verteilt ist. Nur 40 Teilnehmer befinden sich in den Zügen. Der Rest ist auf der gesamten Netzfläche zufällig verteilt. Die Simulationen 7-8 dienen schließlich zur Untersuchung des Einflusses der Verhandlungsrandbedingungen, u.a. der Nachbarschaftsdefinition, auf die Effizienz des Verfahrens.

Weil die Kurven der Angebote, der Verluste und der Bitfehlerwahrscheinlichkeiten bei diesen Simulationen von der Form her analog zu denen der ersten Konfiguration verlaufen (siehe Kapitel 8.2.1), werden die Ergebnisse der Simulationen in diesem Kapitel hauptsächlich anhand von Werten und Tabellen ausgewertet.

8.2.2.1 Simulation 1-4: inhomogen verteilter Verkehr (Hotspot)

Jeweils mit einer statischen und einer dynamischen Ressourcenverteilung wurden zwei Simulationen durchgeführt. Die erste mit einem Verkehrsangebot von 200 Erlang, die zweite mit einem von 150 Erlang. Die Ergebnisse werden in der folgenden Tabelle dargestellt.

Angebot	Verteilung	Verbindungen				Verlust [%]	BER
		Reg.	Block.	Abgeb.	Gesamtdauer [s]		
200 Erl	statisch	3268	2087	914	77854	38,97	0,0033
	dynamisch	3570	1346	1675	84300	27,37	0,0019
150 Erl	statisch	2575	1476	796	69312	36,43	0,0064
	dynamisch	2824	844	1193	75223	23,00	0,0095

Reg.: Anzahl der erfolgreich durchgeführten Verbindungen.

Block.: Anzahl der blockierten Verbindungen (abgewiesene Belegungswünsche).

Abgeb.: Anzahl der unterbrochenen Verbindungen (wegen schlechter Qualität oder fehlendem Verkehrskanal bei einem Handover).

Aus der Tabelle ist zu erkennen, daß der reale Verlust bei beiden Verkehrsangeboten durch die dynamische marktbasierete Ressourcenverteilung deutlich geringer geworden ist, dies bei einer akzeptablen Qualität der Verbindungen (BER). Die Verbesserung in Bezug auf den Verlust beträgt 29,76 % bzw. 36,86 % (bei 200 Erlang bzw. 150 Erlang). Die gesamte Dauer der durchgeführten Verbindungen ist in beiden Fällen höher geworden, was sich für den Netzbetreiber als purer Gewinn herausstellt. Durch den Interferenzadaptiven Aspekt der Kostenfunktion des marktbasiereten Verfahrens ist sogar eine Verbesserung der Qualität erreicht worden (bei 200 Erlang). Die in Kapitel 8.2.1 angesprochene Verschlechterung der Qualität tritt also mit der neuen Konfiguration nicht mehr auf. Es liegt daran, daß das Netz groß genug ist, um die gleichen Frequenzen so weit wie möglich voneinander entfernen zu können. Der Anstieg der Anzahl der abgebrochenen Verbindungen läßt sich folgendermaßen erklären: Es ist zuerst aus den niedrigen Werten der Bitfehlerwahrscheinlichkeit auszuschließen, daß diese Verbindungen wegen einer Verschlechterung der Qualität abgebrochen worden sind. Der Grund für die Unterbrechungen läßt sich durch die Tatsache erklären, daß die telefonierende Mobilstationen beim Verlassen eines Sektors versucht, sich bei dem Nachbarsektor einzubuchen und einen Kanal für die Fortsetzung der Verbindung zu belegen. Bedenkt man, daß die meisten Mobilstationen sich gleichzeitig von einem Ort zum

nächsten in die gleiche Richtung bewegen und dazu die Last im Netz generell relativ hoch ist, läßt sich daraus folgern, daß die meisten Frequenzen des Nachbarsektors entweder belegt oder im vorigen Zeitschritt wegen des dynamischen Aspektes des Verfahrens abgegeben worden sind. Der Grund für die Unterbrechungen ist also auf die relativ zum hohen Verkehrsangebot niedrige Kapazität des Netzes zurückzuführen. Diese Aussage läßt sich durch die zweite Simulation (150 Erlang) bestätigen. Wegen des realistischeren Verkehrsangebotes ist die Quote der unterbrochenen Verbindungen (dynamisch/statisch) um 22,27 % niedriger geworden als bei der ersten Simulation (200 Erlang).

8.2.2.2 Simulation 5-6: Homogen verteilter Verkehr

Bei diesen Simulationen wurde ein Szenario mit einer beinahe homogenen Verteilung der Last simuliert. Nur 40 der insgesamt 200 Teilnehmer befinden sich in dem sich in Richtung Netzmitte bewegenden Zug. Der Rest ist fast gleichmäßig auf der ganzen Netzfläche verteilt. Der Verlust bei einer statischen Ressourcenverteilung, verglichen mit dem oben beschriebenen Szenario (inhomogene Lastverteilung), ist halb so hoch (21,95% statt 38,97%). Bei diesem Szenario ist also nicht viel zu verbessern. Eine statische Ressourcenverteilung scheint gewissermaßen zu der homogenen Lastverteilung zu passen. Eine Umverteilung der Ressourcen bringt dann keine großen Vorteile. Immerhin ist aber eine Verbesserung des gesamten Verlustes und der Qualität der Verbindungen zu vermerken (siehe folgende Tabelle).

Angebot	Verteilung	Verbindungen				Verlust [%]	BER
		Reg.	Block.	Abgeb.	Gesamtdauer [s]		
200 Erl	statisch	4610	1297	1879	82709	21,95	0,0190
	dynamisch	4346	1070	2071	85328	19,75	0,0184

8.2.2.3 Simulation 7-8: Einfluß der Verhandlungsrandbedingungen

Anhand dieser Simulationen wird der Einfluß der Verhandlungsrandbedingungen auf die Effizienz des marktbasiereten Verfahrens untersucht. Dafür wurde jeweils mit einer statischen Ressourcenverteilung, einer dynamischen mit den Randbedingungen 1 (Sim1), einer dynamischen mit den Randbedingungen 2 (Sim2) und einer dynamischen mit den Randbedingungen 3 (Sim3) simuliert.

Im folgenden wird anhand der folgenden Tabelle kurz an die Randbedingungen erinnert.

Simulation	Randbedingungen
Sim1	In jeder Nachbarzelle befindet sich genau <u>ein</u> Nachbarsektor (derjenige, der in die gleiche Richtung ausgerichtet ist). Frequenzen <u>dürfen nicht</u> in einer Nachbarzelle wiederholt werden.
Sim2	In jeder Nachbarzelle befinden sich <u>drei</u> Nachbarsektoren (alle Sektoren unabhängig von der Richtung). Frequenzen <u>dürfen nicht</u> in einer Nachbarzelle wiederholt werden.
Sim3	In jeder Nachbarzelle befinden sich <u>drei</u> Nachbarsektoren (alle Sektoren unabhängig von der Richtung). Frequenzen <u>dürfen</u> in einer Nachbarzelle wiederholt werden.

Bei allen Simulationen mit einer dynamischen Ressourcenverteilung ist eine Verbesserung bezüglich des Verlustes zu erkennen. Dabei wird die Qualität der Verbindungen unterschiedlich beeinträchtigt. Die Ergebnisse aller Simulationen sind in der folgenden Tabelle dargestellt.

Randbe.	Verteilung	Verbindungen				Verlust [%]	BER
		Reg.	Block.	Abgeb.	Gesamtdauer [s]		
---	statisch	3223	2341	763	71101	42,07	0,0035
1	dynamisch	3653	1658	1565	86519	31,21	0,0074
2	dynamisch	3905	1542	1653	84004	28,30	0,0151
3	dynamisch	3948	1679	1814	85172	29,83	0,0173

Bei der Simulation mit den wenigsten Freiheitsgraden (Sim1) ist zwar die geringste Verbesserung des Verlustes erreicht worden (31,21%), die Qualität der Gespräche hat sich aber nur gering verschlechtert. Es liegt daran, daß bei diesen Randbedingungen mehr auf die Interferenzen geachtet wird. Erstens wird nämlich ausgeschlossen, daß zwei Sektoren mit derselben Frequenz einander praktisch gegenüberstehen und sich somit gegenseitig stören, zweitens, daß zwei benachbarte Zellen dieselbe Frequenz besitzen.

Simulation Sim2 profitiert von einem Freiheitsgrad mehr als Sim1. Hier wurde ein niedriger Verlust auf Kosten der Qualität erreicht. Grund dafür ist, daß die Käufersektoren über eine größere Auswahl an Verkäufern verfügen. Die potentiellen Ressourcen, die sie kaufen können, sind damit mehr. Auf der anderen Seite vermeidet diese Annahme nicht den Fall, bei dem sich Sektoren wegen ihrer Ausrichtung gegenseitig stören.

Bei der dritten Simulation Sim3 wurde zusätzlich zu diesem erläuterten Freiheitsgrad noch einer eingesetzt. Diese Ansatz hat sich als ungeeignet erwiesen. Das Verfahren erlaubt es, benachbarten Zellen sogar dieselben Frequenzen zu besitzen, was zu einer stärkeren Verschlechterung der Verbindungsqualität geführt hat. Der Verlust ist auch nicht besser geworden. Diese dritte Simulation zeigt, daß ein Kompromiß zwischen der Verbesserung des Verlustes und der Qualität gefunden werden muß.

Es hat sich also durch diese Simulationen erwiesen, daß nicht immer eine Verstärkung des dynamischen Aspektes des Verfahrens die besseren Ergebnisse liefert. Es gibt eine Grenze, bei der die Qualität der Verbindungen wegen der eingesetzten Freiheitsgrade der Ressourcenverteilung so stark verschlechtert wird, daß das gesamte Verhalten des Netzes negativ beeinträchtigt wird.

8.2.2.4 Häufigkeit der ausgeführten Transaktionen

Die folgende Abbildung wurde von den Ergebnissen einer Simulation mit einer dynamischen Ressourcenverteilung abgeleitet, und dient zur Untersuchung der Häufigkeit der ausgeführten Transaktionen im Laufe der Simulation.

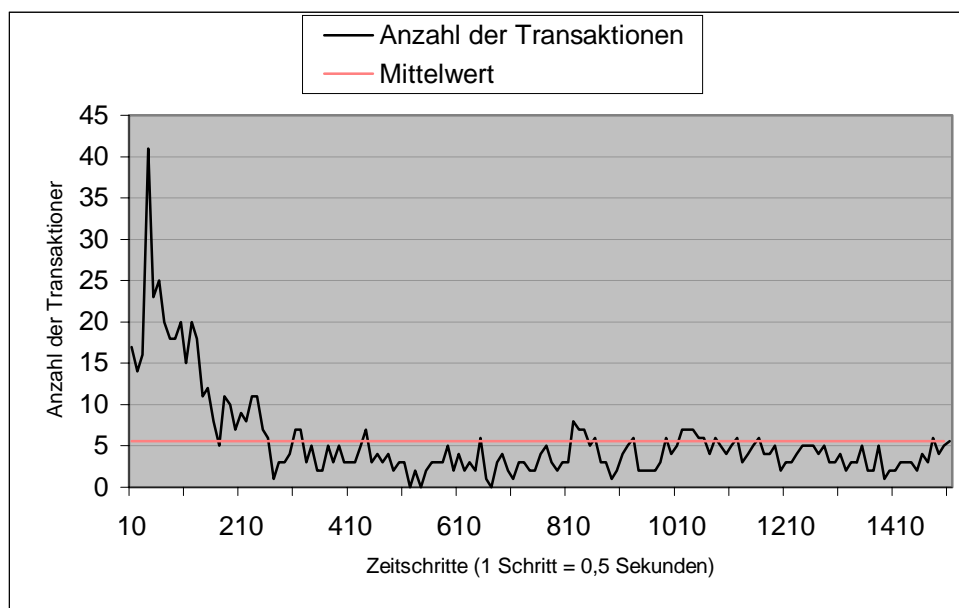


Abb. 59: Anzahl der Transaktionen über die Simulationszeit

Aus der Kurve ist ersichtlich, daß das Netz in der Anfangsphase der Simulation versucht, sich an die neuen Gegebenheiten anzupassen. Weil das Verkehrsangebot plötzlich von Null auf einen bestimmten hohen mittleren Wert springt, versuchen alle Sektoren des Netzes ihren Bedarf zu decken, was zu einer relativ hohen Anfangsdynamik führt. Die Anzahl der

Transaktionen erreicht anfangs Spitzenwerte bis zu 41. Im Laufe der Simulation werden die Ressourcen immer besser auf diejenigen Sektoren verteilt, die sie mehr brauchen und damit besser ausnutzen können, bis die Lage sich praktisch beruhigt. Die Anzahl der Transaktionen sinkt also im Mittel stetig, um sich dann bei einem realistischen Wert zu stabilisieren, der praktisch dem Mittelwert aller Transaktionen über die ganze Simulation entspricht (5,56 Transaktionen). Die Schwankungen der Anzahl der Transaktionen um diesen Mittelwert sind dann bis zum Ende der Simulation minimal (von 0 bis 8 Transaktionen).

Den Mittelwert der Anzahl der Transaktionen kann man in eine Richtung regulieren, indem die Schwelle des marktbasierten Verfahrens für die auszuführenden Transaktionen variiert wird (siehe Kapitel 5.3). Diese Schwelle ist ein Maß für die Tendenz des Verfahrens viele oder wenige Transaktionen auszuführen.

9 Zusammenfassung und Ausblick

Die Motivation zu der vorliegenden Diplomarbeit war die immer kritischer werdende Effizienz der Ressourcenausnutzung in Mobilfunknetzen. Durch die stetig wachsende Teilnehmeranzahl müssen neue Verfahren zur Ressourcenverteilung entwickelt werden, die zu einer besseren Ausnutzung der einem Netzbetreiber zur Verfügung stehenden Ressourcen der Luftschnittstelle führt.

Im Rahmen dieser Arbeit wurde der im IANT entworfene Mobilfunksimulator MOSIT um ein dynamisches Ressourcenverteilungsverfahren erweitert, das eine marktbasierete Regelung anwendet. Wie aus den Eigenschaften des Verfahrens zu erkennen ist, umfaßt diese Arbeit zwei unterschiedliche Gebiete: Mobilfunknetze und marktbasierete Regelung. Das Verfahren sollte nämlich reale Marktgesetze und -verfahren aus der Wirtschaft an einem Mobilfunknetz anwenden.

Zu Beginn der Arbeit wurde ein Überblick über die für das Verständnis des implementierten Verfahrens vorausgesetzten Hintergrundkenntnisse aus dem Gebiet der Mobilfunknetze gegeben, gefolgt von einer Einführung in die Theorie der marktbasiereten Regelung und der Auktionen. Darüber hinaus wurden nach einer Literaturrecherche bestehende Ressourcenverteilungsverfahren vorgestellt, auf die einige Aspekte des entwickelten marktbasiereten Verfahrens zurückzuführen sind.

Nachdem die erwähnten theoretischen Grundlagen für die Arbeit erfaßt wurden, wurden die Möglichkeiten, eine dezentrale marktbasierete Regelung in einem Mobilfunknetz einzusetzen, untersucht. In Folge dessen wurde ein Modell für das Mobilfunknetz so konzipiert, daß Marktgesetze abgebildet werden können. Das Mobilfunknetz wurde als Markt modelliert, auf dem die Frequenzen die Güter oder die Ware darstellen und die Verhandlungen zwischen den Zellagenten, Verkäufer und Käufer, stattfinden. Darüber hinaus wurde ein Protokoll festgelegt, das die Handelsbeziehungen und die verschiedenen Aktionen und Abläufe innerhalb des Markts abbildet. Wichtig für die Effizienz des Handelsverlaufs war eine gut einstudierte Preisgestaltung. Diesem Zweck entsprechend wurde eine Kostenfunktion entworfen, auf deren Grundlage die Frequenzen, unter Berücksichtigung von Randbedingungen, bewertet werden. Bei der Entwicklung der Kostenfunktion wurde nicht nur auf eine Verteilung der Frequenzen aufgrund des Verkehrs geachtet, sondern es wurde auch

auf die Qualität der angebotenen Dienste Wert gelegt. Die letztendlich entwickelte Kostenfunktion integriert verschiedene Aspekte, so daß die Ressourcen von unterschiedlichen Gesichtswinkeln betrachtet werden. Hierzu wurden die Begriffe Verkehrsglied, Reichtumsglied und Störglied eingeführt.

Ergebnis der Überlegungen war ein marktbasierendes Ressourcenverteilungsverfahren, das einen verkehrs- und interferenzadaptiven Aspekt aufweist. Das Verfahren basiert zusätzlich auf einem hybriden Ansatz (statisch/dynamisch), so daß eine minimale Kanalanzahl in einer Zelle garantiert ist und eine maximale nicht überschritten wird.

Weil das in Java zu implementierende Verfahren Daten aus MOSIT empfangen sollte, auf deren Grundlage es Entscheidungen trifft und zurücksendet, mußte eine Einarbeitung in MOSIT durchgeführt und eine Schnittstelle SDL-Java festgelegt werden, über die die Kommunikation zwischen dem Simulationstool und dem das Verfahren einbettenden externen Programm stattfinden soll. Zu diesem Zweck wurden verschiedene Möglichkeiten untersucht und bewertet. Die Entscheidung ist schließlich auf die günstigste verfügbare Alternative gefallen: Environment-Funktionen in Zusammenarbeit mit JNI (Java Native Interface) wurden für die Einbettung des Verfahrens in MOSIT eingesetzt. Mit Hilfe dieser Schnittstelle wurde eine 3-tier Architektur für die Implementierung konzipiert. Die Informationen fließen dabei von der SDL-Seite über die C-Environment-Funktionen zu der Java-Seite und umgekehrt.

Bei der Implementierung wurde auf Wiederverwendbarkeit und Erweiterbarkeit geachtet, um bei späteren Arbeiten zum Thema Ressourcenverteilung in Mobilfunksystemen eine aufwendige Einarbeitung in MOSIT zu vermeiden. Resultat der Implementierung ist ein Rahmenwerk, das alle benötigten Daten zur Umverteilung von Ressourcen bereitstellt und die Entwicklung weiterer Verfahren stark vereinfacht.

Schließlich wurden verschiedene Simulationen durchgeführt, um den marktbasierenden Ansatz hinsichtlich seiner dynamischen Eigenschaften und seiner Auswirkungen auf das Verhalten des Netzes zu untersuchen. Dabei wurden die Gewichte der Kostenfunktion und die simulierten Szenarien variiert, um aussagekräftige Schlüsse über die Effizienz des Verfahrens ziehen zu können.

Die Auswertung der Simulationsergebnisse hat gezeigt, daß das realisierte dynamische marktbasierende Ressourcenverteilungsverfahren den Verlust in einem Mobilfunksystem reduzieren kann. Bei einigen Simulationen wurde sogar eine Qualitätssteigerung erreicht. Das war zwar nicht der Regelfall, aber es spricht für das Potential, das durch eine Erweiterung und Optimierung des implementierten Verfahrens entwickelt werden kann.

Allerdings zeigte sich bei der Auswertung der Simulationen, daß diese Arbeit noch in manchen Punkten erweitert und fortgesetzt werden kann, was durchaus als Schwerpunkt späterer Arbeiten denkbar wäre. Diese Punkte werden im folgenden erläutert:

- Aus statistischen Gründen wären Simulationen über längere Zeiträume mit mehr Mobilstationen, mehr Verkehr und verschiedenen Szenarien aussagekräftiger. Wegen des hohen Zeitaufwandes wurden die in dieser Arbeit durchgeführten Simulationen über relativ kurze Zeiträume durchgeführt.
- Um den Einfluß des Verfahrens auf das Verhalten des Netzes genauer zu untersuchen, könnten weitere realistischere Szenarien simulieren werden, die unterschiedliche Arten von Hotspots und Lastverteilungen abbilden.
- Die in der Kostenfunktion eingesetzten Glieder könnten genauer untersucht werden. Es wäre auch durchaus vorstellbar, zusätzliche Glieder hinzuzufügen, die neue Aspekte einführen, was zu einer besseren Bewertung der Ressourcen führt.
- Das verwendete Störglied, das ein Maß für die eventuellen Interferenzen darstellt, könnte genauer berechnet werden, indem die Ausrichtung und die Dämpfung der Antennen innerhalb der Sektoren berücksichtigt werden. Bisher wurden diese Effekte nicht einbezogen und es wurde immer streng vom „worst case“ ausgegangen, was die Einbuße eines Freiheitsgrads bedeutet.
- Der Einfluß der Preisgestaltung auf die Effizienz des Verfahrens hinsichtlich der Netzperformance und der Qualität der Dienste (Quality of Service) könnte genauer untersucht werden. Die Auswahl der bei der Preisgestaltung eingesetzten Gewichte der Kostenfunktion könnte optimiert werden, indem diese Gewichte dynamisch zur Laufzeit der Simulation und abhängig von der aktuellen Lage geändert werden.
- In der entworfenen Kostenfunktion wurde der Erlang'sche Verlust eingesetzt, der eine Blockierungswahrscheinlichkeit darstellt. Das ist sozusagen ein Vorhersagewert. Man könnte statt dessen den über die Vergangenheit gemittelten realen Verlust verwenden. In dem Fall werden auf vergangene Situationen bezogene Entscheidungen getroffen. Es

bleibt aber zu untersuchen, inwieweit diese vergangenen Situationen eine aussagekräftige Prognose des Verlustes erstellen können.

- Der implementierte Algorithmus versucht zwar, bevor er die Ressourcenverteilung durchführt, diese in mehreren Iterationen zu optimieren, indem nach jedem Ausführen der besten Transaktion der Netzzustand upgedated wird, garantiert aber nicht die absolut beste Verteilung. Es wäre folglich sinnvoll, Optimierungsverfahren zu untersuchen, die auf mathematische Basen von einer aktuellen Lage aus die optimale Lösung berechnen.
- Die meisten in der Literatur beschriebenen Verfahren versuchen den Erlang'schen Verlust zu optimieren. Interessant zu untersuchen wäre, inwieweit die höchsten Gewinne für den Netzbetreiber dadurch erreicht werden. Es drängt sich also die Frage auf, ob eine Optimierung des Verlustes eine Optimierung des Gewinns reflektiert. Der Verlust (entgangene Anrufe) muß nämlich relativ zum Einkommen (eingegangene Anrufe) betrachtet werden.
- Bei dem implementierten Verfahren wurde von einer konstanten mittleren Belegungsdauer ausgegangen. Darüber hinaus wurde auf einen Tarif für die eingegangenen Anrufe nicht geachtet. Eine interessante Erweiterung des marktbasierenden Verfahrens wäre die Einführung der Belegungsdauer und –tarif in die Preisgestaltung. So können die durch die Anrufe tatsächlich eingegangenen Gebühren ein sich der Realität sehr annäherndes Kriterium für die Ressourcenverteilung bilden.

Abschließend hat sich durch die vorliegende Arbeit erwiesen, daß der marktbasierende Ansatz eine gute Möglichkeit bietet, die knappen Ressourcen eines GSM-Netzes zu verteilen. Mit der Einführung von UMTS (Universal Mobile Telecommunication System) als Mobilfunknetz der 3. Generation erweitert sich das Dienstangebot von einfacher Telefonie hin zu Bildtelefonie, Datenübertragung, Videokonferenzen, mobilem Internetzugriff usw., was zu einem noch höheren Bedarf an Ressourcen führt. Infolgedessen könnte der Einsatz eines marktbasierenden dynamischen Ressourcenverteilungsverfahrens in der Zukunft an größerer Bedeutung gewinnen.

10 Anhang

10.1 Literaturverzeichnis

- [AGO00] Agorics; A Survey of Auctions: www.agorics.com/new.html
- [BEN96] T. Benkner: „Kapazitätsteigernde Maßnahmen für digitale Mobilfunksystemen der dritten Generation“, Schriften zur Nachrichtenübermittlungstechnik der Universität Siegen, Shaker Verlag 1996
- [BKR97] J.Bredin, D.Kotz, D.Rus: “Market-based Resource Control for Mobile Agents“; Department of Computer Science, Dartmouth College Hanover 1997
- [BOR97] Rick Borovoy, Rob Guttman: “Market-based Control“ (MAS.738 - Modeling Adaptive Behavior); <http://guttman.www.media.mit.edu/people/guttman/research/commerce/talk3/slide1.html>
- [CLE96] Scott H.Clearwater: “Market-based Control: A Paradigm for distributed Resource Allocation“; World Scientific Publishing Co.Pte. Ltd. 1996
- [DAV96] K. David, T.Benkner: “Digitale Mobilfunksysteme“ ; Teubner Verlag, Stuttgart 1996
- [DEL97] F.Delli, N.Magnani, V.Palestini, F.Sestini: “Application of dynamic channel allocation strategies to the GSM cellular network“; IEEE-Journal. Vol.15, no.8; p.1558-67, Oct. 1997
- [DES99] M.Descher: “Marktbasierte regelung der Ressourcenvergabe in Netzwerken“; Diplomarbeit, Institut für Rechnergestützte Wissensverarbeitung, Universität Hannover, Juni 1999
- [EV99] J. Eberspächer, H.-J. Vögel: “GSM Global System for Mobile Communication “; 2. Auflage Teubner Verlag Stuttgart 1999
- [FET00] H.Fette: “Untersuchung und Implementierung von automatisch regelnden Agenten zur Optimierung der Ressourcenvergabe in einem zellvermittelnden Mobilkommunikationssystem“; Diplomarbeit, Institut für Allgemeine Nachrichtentechnik, Universität Hannover, Januar 2000
- [GER98] Ralf Geerdsen: “Implementierung und Bewertung verschiedener Kanalvergabeverfahren für Mobilfunkluftschnittstellen der 3. Generation“; Diplomarbeit, Institut für Allgemeine Nachrichtentechnik, Universität Hannover, November 1998

- [KAT96] I.Katzela, M. Naghshineh: "Channel Assignment Schemes for Cellular Mobile Telecommunication Systems: A comprehensive Survey"; IEEE Personal Communications, pp.10 – 31, June 1996
- [SCH00] B.Schmelz: "Untersuchung und Implementierung eines Netzmanagementmodells für zukünftige zellvermittelnde Mobilfunksysteme unter Verwendung von UML und SDL"; Studienarbeit, Institut für Allgemeine Nachrichtentechnik, Universität Hannover, April 2000
- [SUN00] Sun microsystems: The Java Tutorial, Trail: Java Native Interface; <http://java.sun.com/docs/books/tutorial/native1.1/>
- [TEL] Telelogic Tau On-Line Help: Hilfe zum SDT-Tool der Firma Telelogic, verfügbar unter den internen Web-Seiten des IANT's.
- [TIN98] Tin-Ho Chan: "Seminar Java-Technologien", CORBA, IDL, Java-ORB; FH München, FB 07 Informatik/Mathematik 1998
- [WAL00] B. Walke: "Mobilfunknetze und ihre Protokolle"; Band 1 Teubner Verlag Stuttgart 2000

10.2 Abbildungsverzeichnis

Abb. 1: Beispiel eines zellularen Netzes mit Frequenzwiederholung	8
Abb. 2: Beispiel eines 7-Zellen-Cluster zellularen Netzes.....	9
Abb. 3: Kanäle in einem FDMA-System.....	10
Abb. 4: Zeitvielfachzugriff TDMA	11
Abb. 5: Zusammenhang zwischen angebotem Verkehr und Blockierungswahrscheinlichkeit bei FCA und DCA [BEN96]	14
Abb. 6: Prinzip der Kanal-Leih-Verfahren	15
Abb. 7: Reuse Partitioning [BEN96].....	17
Abb. 8: Regelkreis CMRA-Verfahren [FET00]	19
Abb. 9: Regelkreis STBR-Verfahren [FET00]	19
Abb. 10: Beispiel einer Verteilung	32
Abb. 11: Transaktionstabelle.....	32
Abb. 12: Transaktionstabelle nach der ersten Transaktion und dem Filtern	33
Abb. 13: Transaktionstabelle nach dem Update	34
Abb. 14: Die während der Auktion ausgeführten Transaktionen	35
Abb. 15: Abbildung der während der Auktion ausgeführten Transaktionen	35
Abb. 16: Ressourcenverteilung vor und nach einer Auktion.....	36
Abb. 17: Aufbau von MOSIT	37
Abb. 18: Struktur des Blocks Netzmanagement.....	39
Abb. 19: Architektur eines SDL Systems	42
Abb. 20: Kommunikation zwischen Prozessen über Signale	43
Abb. 21: CORBA Architektur	44
Abb. 22: Postmaster	46
Abb. 23: Schnittstelle über Environment-Funktionen	47
Abb. 24: Die Funktionen zum Umgang mit globalen Referenzen	50
Abb. 25: Die Funktionen zur Synchronisation von Threads	51
Abb. 26: HelloWorld.java.....	52
Abb. 27: Die Erzeugung einer eigenen virtuellen Maschine	53
Abb. 28: Der Aufbau der Struktur JDK1_1InitArgs	53
Abb. 29: Architektur der implementierten Lösung.....	56
Abb. 30: Ablaufdiagramm der Prozedur <i>RsrcDistribution</i>	58
Abb. 31: Die Initialisierungsdatei <i>Marked_Based.ini</i>	59
Abb. 32: Parameter des Output-Signals <i>Out_ClusterState</i>	60
Abb. 33: Parameter des Input-Signals <i>In_Result</i>	60
Abb. 34: Datentypen der Transaktionen.....	61
Abb. 35: Die Prozeduren <i>TakeSlotFrom</i> und <i>GiveSlotTo</i>	62
Abb. 36: Architektur der Schnittstelle über die Environment-Funktionen und JNI	63
Abb. 37: Die grundlegenden MOSIT-Datenstrukturen	64
Abb. 38: Die Datenstruktur <i>BSListT</i>	64

Abb. 39: Die Datenstruktur ClusterStateT	65
Abb. 40: Die Datenstruktur TF_Zuordnung_Feld	65
Abb. 41: Die Datenstruktur Kanal_Bel_Feld_Up	65
Abb. 42: Die Datenstruktur Kanal_Bel_Feld_Down	65
Abb. 43: Die wichtigsten JNI Abläufe	67
Abb. 44: Ablaufdiagramm der Methode <i>Process_Network</i>	75
Abb. 45: Ablaufdiagramm der Methode <i>Process_Tabelle</i>	75
Abb. 46: Ablaufdiagramm der Methode <i>Update_Tabelle</i>	76
Abb. 47: Abbild des simulierten Netzes, 1. Konfiguration (16 Basisstationen)	81
Abb. 48: Beispiel einer Simulation mit zwei Zügen und Ringen	82
Abb. 49: Wirkung der Clustergröße auf die Interferenzlage nach einer Transaktion	84
Abb. 50: Vergleich des Angebotes in einem Sektor in der Stadtmitte und außerhalb	85
Abb. 51: Realer und Erlang'scher Verlust des Sektors 3 der BS 4 (statisch/dynamisch)	85
Abb. 52: Variation der Anzahl der Frequenzen in Abhängigkeit von dem Erlang'schen Verlust im Sektor 3 der BS 4	86
Abb. 53: Vergleich der Bitfehlerwahrscheinlichkeit im Sektor 3 der BS 4 (statisch/dynamisch)	87
Abb. 54: Wirkung der Gewichtung auf die Bitfehlerwahrscheinlichkeit (BS 4, Sektor 3)	87
Abb. 55: Variation der Anzahl der Frequenzen in Abhängigkeit von dem Erlang'schen Verlust im Sektor 3 der BS 16	88
Abb. 56: Realer und Erlang'scher Verlust des Sektors 3 der BS 16 (statisch/dynamisch)	88
Abb. 57: Realer und Erlang'scher Verlust des gesamten Netzes (statisch/dynamisch)	89
Abb. 58: Topologie des Netzes bei der zweiten Konfiguration	90
Abb. 59: Anzahl der Transaktionen über die Simulationszeit	95

10.3 Abkürzungsverzeichnis

API	Application Programming Interface
BFA	Borrowing First Available
BCO	Borrowing with Channel Ordering
BDCL	Borrowing with Directional Channel Locking
BP	Bewegungsprofil
BS	Basisstation
CDMA	Code Division Multiple Access
CIR	Carrier to Interference Ratio
CORBA	Common Object Request Broker Architecture
CSEG	Channel Segregation
CMRA	Control Method based Resource Allocation algorithm
DCA	Dynamic Channel Allocation
DS-CDMA	Direct Sequence-CDMA
FA	First Available
FCA	Fixed Channel Allocation
FBCA	Forced Borrowing Channel Assignment
FDMA	Frequency Division Multiple Access
GSM	Global System for Mobile Communication
HCA	Hybrid Channel Allocation
IDL	Interface Definition Language
JDK	Java Development Kit
JNI	Java Native Interface
JVM	Java Virtual Machine
MOSIT	Mobilfunk Simulationstool
MSIR	Minimum Signal to Noise
OMG	Object Management Group
ORB	Object Request Broker
P-STBR	Traffic Prediction STBR
SB	Simple Borrowing
SBR	Simple Borrowing from the Richest
SCS	Sequential Channel Search

SDL	Specification and Description Language
SDT	SDL Development Tool
STBR	Simple Threshold Based Resource Allocation Algorithm
TDMA	Time Division Multiple Access
UMTS	Universal Mobile Telecommunication System