SFB 559 — Teilprojekt M1
LS Informatik IV
Universität Dortmund
14. August 2002
Version 1.0

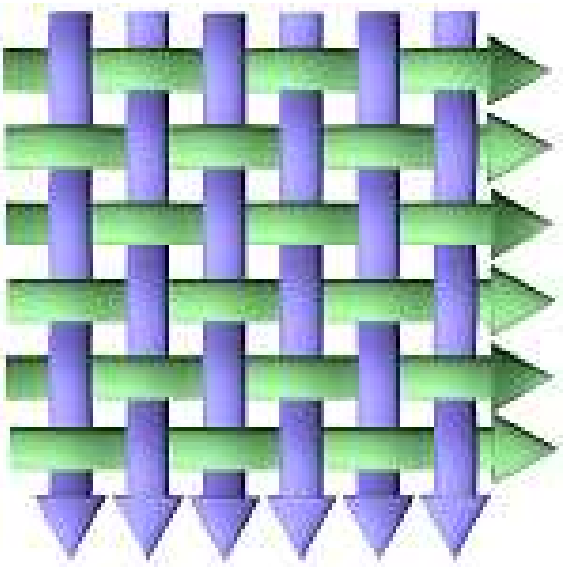
Sonderforschungsbereich 559

Modellierung großer
Netze in der Logistik

Technical Report 02004
ISSN 1612-1376

# Hands On B1-Tools

Eine beispielorientierte Einführung
in die Anwendung der B1-Tools



SFB 559 — Teilprojekt M1

Mirko Eickhoff
Michael Hierweck

# Vorwort

Dieses Dokument soll Ihnen helfen, einen Einstieg in die Verwendung der Tools zur Modellierung und Analyse logistischer Systeme zu finden. Es bietet einen Überblick über alle Tools und zwar aus der Sicht eines Anwenders, der erste Schritte mit diesen unternimmt, während er versucht ein Modell inkrementell umzusetzen und zu analysieren. Wir werden dabei natürlich nicht alle Funktionen jedes Tools verwenden, aber nach dem Durcharbeiten dieses Dokuments sollte es Ihnen leicht fallen, alles weitere durch Ausprobieren oder Studium der entsprechenden Programm-Dokumentationen herauszufinden.

Vorausgesetzt wird allerdings ein grundlegendes Verständnis des **B1-Paradigmas** [5] sowie Erfahrung im Einsatz von Software im Allgemeinen. Sollten sie noch keinen Überblick über die Ideen und Konzepte sowie die Bedeutung der einzelnen Modellelemente besitzen, so ist es empfehlenswert dieses nachzuholen, bevor Sie das Studium dieses Dokuments fortsetzen. Es wird nicht erwartet, dass Sie jedes Element inklusive der Attribute und deren Bedeutung exakt kennen. Viel mehr geht es darum, einen Überblick zu bekommen.

Im Folgenden wird nun ein Schnellimbiss mit einigen Sitzplätzen, Küche und Personal modelliert, da wir annehmen, dass wohl jeder eine intuitive Vorstellung der Abläufe einer solchen Einrichtung besitzt. Weiterhin lässt sich die Modellwelt - wie wir später noch sehen werden - leicht inkrementell modellieren, weiter präzisieren und erweitern.

Wir hoffen, dass dieses Dokument nicht nur trockene Theorie darstellt, sondern auch zum selbständigen Experimentieren mit den Tools einlädt. Ändern Sie einfach eine Einstellung in der Modellwelt und schauen Sie, welche Auswirkungen dies bei den Simulationsergebnissen mit sich bringt! Erweitern Sie das Modell um neue Speisen, fügen sie Küchengeräte hinzu und beobachten sie die Auswirkungen auf den Personalbedarf! Lassen Sie dabei die Verweildauer der Kunden nicht außer acht! Sie werden schnell erkennen, dass oft minimal erscheinende Änderungen, großen Einfluss haben werden.

Zu allerletzt bleibt Ihnen viel Freunde beim Lesen und Experimentieren sowie viel Erfolg bei der späteren Anwendung zu wünschen.

Dortmund, im Frühjahr 2002

# Inhaltsverzeichnis

# Kapitel 1

# Modell eines Schnellimbisses

Angenommen, Sie erhalten den Auftrag einen Schnellimbiss zu modellieren, um diesen anschließend im Sinne der **Leistungsbewertung** zu analysieren. Dies bedeutet, dass Sie versuchen sollen, einen konkreten Schnellimbiss mit den abstrakteren Elementen des **B1-Paradigmas** so darzustellen, dass die für die Anlayse relevanten Abläufe und Zusammenhänge erhalten bleiben. Bei der Analyse wird es Ihnen dann ermöglicht, durch **Simulation** verschiedene Informationen zu erhalten, beispielsweise die Verweildauer der Kunden im Schnellimbiss. Es besteht dann die Möglichkeit, durch Veränderungen am Modell die Auswirkungen auf die Messwerte zu untersuchen.

Bei dem zu modellierenden Schnellimbiss handelt es sich um einen kleinen Betrieb, der über Nacht schließt und welcher mit wenig Personal, welches keiner speziellen Aufgabenteilung unterliegt, seine Kunden bedient. Diese haben die Möglichkeit die Speisen mitzunehmen bzw. vor Ort zu verzehren. Existiert dieser Schnellimbiss bereits, so ist es sinnvoll statistische Erhebungen über relevante Daten, wie Kundenzahl, Bedienzeiten etc., durchzuführen, um der Realität möglichst nahe zu kommen, und die Werte im Modell entsprechend einzustellen.

Bei der Modellierung gehen wir nun top-down vor. Dies bedeutet, mit einem (komplexen) Gesamtsystem zu beginnen und dieses schrittweise in kleinere einfachere Systeme zu zerlegen, welche dann zusammen das Gesamtsystem darstellen. Auf einer höhereren Abstraktionsebene, muss man sich dann nicht mit den Details der kleineren Systeme beschäftigen. B1-Modelle sind hierarchisch gegliedert und unterstützen somit diesen Ansatz.

Es macht Sinn, zwar einerseits das Modell weitgehend flexibel zu halten, sich andererseits aber nicht zu sehr auf Details zu konzentrieren. Grundsätzlich sollte das Ziel der Analyse im Auge behalten werden. In unserem Fall interessiert es den Auftraggeber insbesondere, ob Kunden zügig bedient werden und ob das Personal sinnvoll ausgelastet ist. Dennoch sollen später vielleicht auch weitere Auswertungen vorgenommen werden.

## 1.1 Die Umgebung

Stellen wir uns nun einen Schnellimbiss als Blackbox vor. Was sieht der Beobachter von außen? In erster Linie wird er feststellen, dass Personen den Schnellimbiss betreten und wieder verlassen. Beobachtet er dies genauer, so wird er feststellen, dass manche länger im Schnellimbiss verweilen, andere ihn aber nach recht kurzer

Zeit mit Tüten beladen wieder verlassen. Es gibt also zwei verschiedene Typen von Kunden. Wir nennen die zuerst genannten *Sit-In-Gäste* und letztere *Take-Away-Gäste*. Beide Gast-Typen betreten und verlassen den Schnellimbiss. Also soll unser Schnellimbiss die Operation *besuchen* durch bestimmte Typen von Gästen unterstützen.

## 1.2 Prozesse

Ein Gast wird gemäß dem **B1-Modellformalismus** als **Prozess** dargestellt, welcher gestartet wird und den Imbiss durchläuft, bis er anschließend terminiert. Während er den Imbiss durchläuft, greift er auf Ressourcen zurück und nutzt damit den Imbiss bzw. seine Komponenten. Durchlauf und Nutzung orientieren sich dabei an der Handlungsabfolge, die ein Gast in einem Schnellimbiss üblicherweise vornimmt, beispielsweise: Betreten, Bestellen, Verzehren, Bezahlen, Verlassen. Dabei werden auch neue Prozesse initiiert, beispielsweise Kochen oder Kassieren, welches nicht durch den Gast vorgenommen wird, der Gast war jedoch Auslöser.

## 1.3 Ressourcen

Wie bereits angedeutet, benötigen die Prozesse entsprechende Ressourcen. Diese werden nun vorgestellt.

### 1.3.1 Der Schnellimbiss

Öffnen wir dazu die Blackbox und werfen wir einen Blick ins Innere. Wir stellen fest, dass in unserem Schnellimbiss zunächst einmal folgende für die logistischen Ressourcen relevante Objekte auftreten: Eine Theke, an der bedient wird, einige Tische mit Sitzplätzen für den Verzehr vor Ort, Personal und nicht zu vergessen, die Küche. All diese Objekte fassen wir zunächst wieder als Blackbox auf.

### 1.3.2 Das Personal

Das Personal zeichnet sich zunächst einmal dadurch aus, dass es aus einer festen Anzahl von Personen besteht. Diese können nun zu bestimmten Tätigkeiten herangezogen, also angefordert werden. Wir gehen davon aus, dass unser Personal universell ausgebildet ist und jede Aufgabe übernehmen kann.

### 1.3.3 Die Sitzplätze

Ähnlich dem Personal steht eine feste Anzahl von Sitzplätzen zur Verfügung. Für unsere Zwecke soll es genügen, mitzuzählen, wie viele Plätze belegt sind, und im Falle der Vollauslastung den Gast warten zu lassen.

### 1.3.4 Die Küche

In der Küche wird gekocht. Wir beschränken uns zu diesem Zeitpunkt darauf, dass die unterstützte Operation *Kochen* Zeit in Anspruch nimmt.

### 1.3.5 Die Theke

An der Theke werden die entscheidenden Dienstleistungen erbracht. Zunächst werden Bestellungen aufgenommen, anschließend die zubereiteten Speisen ausgegeben und je nach Typ des Gastes sofort oder nach dem Verzehr kassiert.

### 1.3.6 Die Kasse

Zum Kassieren benötigt man eine Kasse. Die Kasse sollte deshalb modelliert werden, da es nicht möglich erscheint, dass mehrere Mitglieder des Personals diese gleichzeitig verwenden. Wir werden später erkennen, dass dieser simpel erscheinende Umstand Folgen haben wird. Auch bei der Kasse handelt es sich um eine Ressource, die belegt und freigegeben werden kann.

# Kapitel 2

# Auswahl geeigneter Modellelemente

Nachdem wir uns überlegt haben, wie die verschiedenen Komponenten unseres zunächst stark vereinfachten Schnellimbisses zusammenspielen und welche relevanten Eigenschaften in das Modell einfließen sollen, stehen wir nun vor der Aufgabe, geeignete Komponenten aus dem B1-Modellformalismus zu wählen.

Zunächst einmal bietet es sich an, all das, was wir zunächst als Blackbox betrachtet haben, auch als solche umzusetzen. Das Modellelement **Funktionseinheit** eignet sich hierfür. Man kann es mit einer kleinen Programmbibliothek vergleichen, welche Funktionen zur Verfügung stellt. Den Funktionen können Parameter übergeben werden und sie können Rückgabewerte liefern. Die Funktionen heißen hier **Dienste**, welche beim Aufruf gestartet werden und im Innern der **Funktionseinheit** durch **Prozessketten** dargestellt werden. Sobald diese terminieren, ist der Aufruf beendet. Wie in der Programmierung kann man **Funktionseinheiten** intern ohne Auswirkungen auf die Nutzung der **Dienste** innerhalb des **Modells** verändern, sofern man nur die Schnittstelle unverändert lässt. Wir modellieren die Schnittstelle, indem wir die zur Verfügung stehenden **Prozesse** benennen und deren **Parameter** angeben.

**B1-Modelle** sind hierarchisch aufgebaut. Durch die **Funktionseinheiten** sind die Schnittstellen zwischen den Hierarchieebenen definiert. **Funktionseinheiten** bieten die Möglichkeit, ihre bereitgestellten **Dienste** auf der jeweils höheren Hierarchieebene zu nutzen.

**Externe Funktionseinheiten** können **Dienste**, die durch andere **Funktionseinheiten** bereitgestellt werden, sogar in entgegengesetzter Richtung importieren, d.h. Dienste, die auf der jeweils höheren Hierarchieebene bereit stehen, können innerhalb der **Externen Funktionseinheit** genutzt werden.

## 2.1 Die Umgebung

Die Umgebung stellt natürlich die oberste Ebene unserer Hierarchie dar. Wir verwenden zwei **Quellen**, welche die beiden Typen der Gäste mit bestimmter Häufigkeit auf die Reise schicken. Jeder Gast stellt somit einen **Prozess** dar, welcher den *Schnellimbiss* besucht und anschließend in einer **Senke** verschwindet.

## 2.2 Der Schnellimbiss

Der *Schnellimbiss* selbst wird durch eine **Funktionseinheit** repräsentiert. Innerhalb dieser wird der Besuch desselben modelliert. Ferner geben wir ihm eine selbständige **Prozesskette** mit der Bezeichnung *Aufraeumen*. Dieser regelmäßig startende **Prozess** soll Kräfte des Personals binden, da sich das Personal bekanntlich nicht ausschließlich den Gästen widmen kann, sondern noch Arbeiten, die unabhängig vom Erscheinen eines Gastes anfallen, erledigen muss. Diese Tätigkeiten benötigen zunächst einmal nur Zeit.

## 2.3 Das Personal

Bei der Wahl eines geeigneten Elements für das Personal entscheiden wir uns für einen **Server**. Ein **Server** stellt eine Ressource zur Verfügung, welche sich über den Aufruf *request* für einen Zeitraum anfordern lässt. Es sind mehrere Anforderungen zeitgleich möglich. Allerdings ist die Anzahl der zeitgleich möglichen Anforderungen beschränkt. Die Beschränkung ist einstellbar und entspricht in diesem Fall der Anzahl der zeitgleich eingesetzten Mitarbeiter des Schnellimbisses.

## 2.4 Die Sitzplätze

Bei den Sitzplätzen wählen wir das **Lager**. Dieses stellt einen mehrdimensionalen diskreten Raum dar. Im Fall der Sitzplätze benötigen wir nur eine Dimension. Es lassen sich die minimale und maximale Belegung vorgeben. Die minimale Belegung der Sitzplätze ist natürlich 0, die maximale Belegung ist die Anzahl der real verfügbaren Sitzplätze. Denkbar wäre es, eine weitere Dimension hinzuzufügen, und damit zwischen Sitz- und Stehplätzen zu unterscheiden. Um das Modell nicht unnötig zu komlizieren, verzichten wir darauf.

## 2.5 Die Küche

Für die Küche bietet sich wieder eine **Funktionseinheit** an. Die Vorgänge in der Küche sind aufwendig, und es müssen auch viele Ressourcen verwaltet werden. Die Küche stellt außerdem eine eigenständige logische Einheit dar. Man könnte sich sogar später einmal dazu entschließen, nicht selbst zu kochen, sondern ein Fremdunternehmen kochen zu lassen, die Küche auszulagern oder gar die Küche umzugestalten, ohne den Rest des Modells zu ändern. All dies wird von Funktionseinheiten leicht unterstützt. Zunächst einmal lassen wir die Küche sehr abstrakt und nehmen an, dass das Kochen lediglich Zeit erfordert und Personal des Schnellimbisses in Anspruch nimmt. Wir haben das Personal jedoch dem Schnellimbiss zugeordnet. Eine **Externe Funktionseinheit** ermöglicht es uns jedoch, auf diese Ressource über die aufrufende Hierarchieebene zuzugreifen.

## 2.6 Die Theke

Die Theke behandeln wir als zentralen Dienstleistungspunkt, der verschiedene Dienste anbietet und sich dabei, wie die Küche, auf die Ressource Personal abstützt. Allerdings sind die Abläufe nicht sonderlich komplex, weshalb wir sie schon in der ersten Fassung unseres Modells modellieren werden.

## 2.7 Die Kasse

Die Kasse ist - ähnlich den Sitzplätzen - eine weitere Ressource, welche geteilt werden muss. Wir verwenden einen **Counter**, der im Sinne eines Semaphors dafür sorgt, dass nicht zwei Mitglieder des Personals gleichzeitig buchen.

# Kapitel 3

# Modellierung mit dem Prozessketteneditor

Wir belassen es zunächst einmal bei diesem einfachen Modell und versuchen es mit Hilfe der B1-Elemente umzusetzen. Oftmals werden jetzt konzeptionelle Mängel deutlich, darum macht es Sinn, dies zu testen und auch probeweise Analysen durchzuführen.

## 3.1 Das B1-GUI

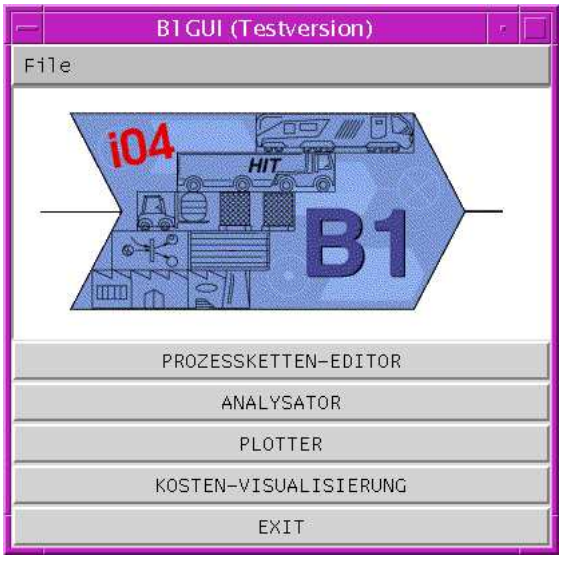



Abbildung 3.1: Hauptmenü des B1-GUIs nach dem Start

Starten wir also das B1-GUI, welches das zentrale Menü für alle weiteren Funktionen umfasst. Nach dem Start erscheint das in Abbildung 3.1 gezeigte Hauptmenü. Über die Schaltflächen lassen sich die verschiedenen Unterprogramme aufrufen.

## 3.2 Arbeiten mit dem Prozessketten-Editor

Wenden wir uns zunächst einmal dem Prozessketten-Editor zu. Für Details hinsichtlich der Verwendung des Prozessketteneditors verweisen wir auf das Handbuch [2], welches jeden Menüeintrag, jede Schaltfläche und Tastenkombination erläutert.

Bevor wir starten, müssen wir uns auf eine Zeiteinheit einigen, d.h. wir müssen die Beziehung zwischen der Modellzeit und der Echtzeit festlegen. Da alle Zeitangaben im Modell als Vielfache der Modellzeit-Einheit angegeben werden und in unserer Modellwelt viele nur kurze Zeit dauernde Aktivitäten vorliegen, bietet sich folgende Beziehung an: Eine Modellzeit-Einheit entspricht einer Sekunde der Realität.

### 3.2.1 Das Hauptfenster mit Hierachiebaum


Abbildung 3.2: Hauptfenster des Prozessketteneditors nach dem Start mit leerem Hierarchiebaum

Der Prozessketten-Editor startet mit einem neuen leeren Projekt. Abbildung 3.2 zeigt das Hauptfenster mit dem zunächst noch leeren Hierachiebaum. Über das Menü "Neues Fenster->Prozesskette" erhält man ein neues Arbeitsfenster zum Modellieren.

### 3.2.2 Verwendung des Arbeitsfensters

Nachdem sich das Arbeitsfenster geöffnet hat, erscheint eine Fläche, welche in mehrere Bereiche eingeteilt ist. Beim Modellieren verwenden wir den unteren Bereich für Funktionseinheiten, Server, Counter und Lager. Auf dem mittlereren Bereich können wir die restlichen Elemente plazieren. Dazu wählen wir ein Element aus der Buttonleiste unter dem Menü aus und klicken auf seine Zielposition. Attribute dieser Elemente werden über Masken editiert. Viele Attributwerte sind voreingestellt. Hat man die Werte vom Standardwert auf einen anderen geändert, so werden die Attribute in der Nähe der Elemente textuell angezeigt. Die Masken und zahlreiche

andere Funktionen, können über lokale Menüs an den Elementen aufgerufen werden. Verbindungen zwischen den Elementen werden ebenfalls mit der Maus gezogen bzw. gelöscht.

### 3.2.3 Die oberste Ebene: Der Imbiss

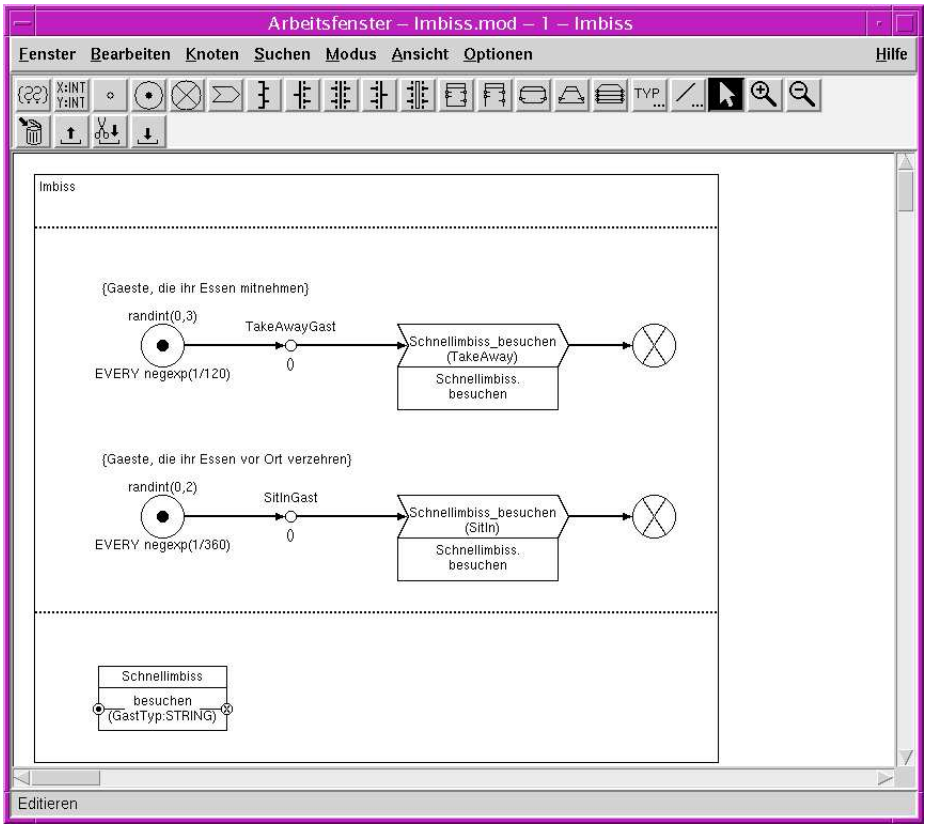



Abbildung 3.3: Editorfenster mit dem Modell der Außenansicht des Schnellimbisses

Die oberste Ebene enthält als wesentliche Elemente **Quellen**, **ProzessIDs** und **Senken**. Hier werden die **Prozesse** gestartet, welche ihren Weg durch unser Modell finden werden. Die Abbildung 3.3 zeigt das Modell der Außenansicht des Schnellimbisses, wie im Entwurf beschrieben.

Die **Quellen** werden mit verschiedener Intensität **Prozesse** starten. Beide **Prozessketten** rufen den Dienst *besuchen* der **Funktionseinheit** *Schnellimbiss* auf, wobei ein **Parameter** übergeben wird. Dieser dient zur Unterscheidung der Gästetypen auf der unteren Ebene. Die Kommentare sollen dem Betrachter helfen, das Modell schneller erfassen zu können. In unserem Fall erläutern sie die Variablenbelegung. Um die Übersichtlichkeit zu erhöhen, haben wir die beiden alternativ parallel ablaufenden Teilprozessketten auch parallel dargestellt, so dass sofort auffällt, dass hier ähnliches Verhalten modelliert wird.

### 3.2.4 Die Innenansicht: Der Schnellimbiss

Die Abbildung 3.4 zeigt den erwartungsgemäß aufwändigeren Schnellimbiss.

Zunächst einmal fällt die **Prozesskette** *besuchen* auf. Deutlich zu erkennen ist die fast lineare Abfolge der Prozesskettenelemente, deren Reihenfolge sich am Ablauf eines Schnellimbissbesuchs orientiert. Ein Besucher des Schnellimbisses führt nacheinander bestimmte Handlungen durch. Diese werden durch **Prozesskettenelemente** dargestellt. Allerdings tritt eine Fallunterscheidung auf, welche durch den **Oder-Konnektor** modelliert wird. Diese alternativen Abläufe sind optisch deutlich erkennbar. Es fällt durch die Anordnung der **Prozesskettenelemente** innerhalb dieser Fallunterscheidung auch auf, dass die Aktionen auf dem unteren Weg eine echte Teilmenge derer auf dem oberen Weg darstellen. Es ist jedoch nicht notwendig die Elemente so anzuordnen.

Küche und Theke wurden als **Funktionseinheiten** modelliert, Personal und Sitzplätze als **Server** bzw. **Lager**. Das **Lager**, welches die Sitzplätze repräsentiert, wird zweimal aufgerufen. Sobald ein Gast Platz nimmt, erhöhen wir die Anzahl der belegten Sitzplätze, nach dem Verzehr geben wir sie wieder frei. Wir zählen hiermit also die belegten Sitzplätze. Alternativ hätte man auch die noch freien Sitzplätze zählen können. Dann sollte das **Lager** allerdings auch entsprechend vorbelegt werden. Beim Aufruf des **Lagers** sollte man beachten, dass ein - hier nur eine Komponente enthaltender - Vektor übergeben wird, welcher durch eckige Klammern gekennzeichnet wird.

Auch die interne **Prozesskette** *aufraeumen* ist bereits modelliert worden. Sie steht nicht mit dem aufrufenden **Prozess** in Verbindung, teilt sich mit diesem aber die Ressource *Personal*.

Bei den **Funktionseinheiten** in der Abbildung 3.4 erkennt man deutlich, dass der **Dienst** *Personal.request* jeweils importiert wird.

### 3.2.5 Zunächst vereinfacht: Die Küche

Wir lassen die Küche zunächst einmal sehr abstrakt. Ihr einziger **Dienst** *kochen* nimmt lediglich Zeit in Anspruch. Arbeitet man mit mehreren Modellierern im Team, so wäre dies eine geeignete Schnittstelle zum Modellierer, welcher sich mit der Umsetzung der Küche befasst, da diese ein eigenständiges System darstellt. Damit das Modell lauffähig ist, verwenden wir vorübergehend ein **Prozesskettenelement**, welches für einen dynamischen Zeitraum Personal in Anspruch nimmt, welches für diesen Zeitraum nicht mehr für weitere Anforderungen verfügbar ist. Die Abbildung 3.5 zeigt das Editorfenster mit dem Modell der Küche.

### 3.2.6 Gemeinsam verwendet: Das Personal

Die Küche verwendet die Ressource Personal, welche jedoch dem Schnellimbiss zugeordnet wurde. Dies ist auch von der Idee her korrekt, da das Personal auch für die übrigen Aktivitäten verfügbar sein muss. Verfügt die Küche über exklusives Personal, z.B. dedizierte Köche, so würde man dieses der Küche zuordnen. Wir müssen uns das Personal aber teilen.

Dazu verwenden wir eine **Externe Funktionseinheit**. Über die Attributmaske der **Funktionseinheit** *Küche* werden die importierten **Dienste** editiert. Zunächst legt man auf der Seite Prozesszuordnung fest, welcher Dienst importiert wird. Dabei gibt man den Namen des importierten Dienstes, in diesem Fall *Personal.request*, und den Namen unter dem dieser innerhalb der Externen Funktionseinheit ansprechbar sein soll, in diesem Fall *Personal_request*, an. Anschließend legt man in den virtuellen Parameterlisten die Namen und Typen der Parameter fest, über die der Dienst verfügen soll, und verküpft diese mit den Parameter des importierten Dienstes.

Anschließend wird innerhalb der **Funktionseinheit** *Küche* im unteren Bereich eine **Externe Funktionseinheit** automatisch eingeblendet. Sie bietet den importierten **Dienst** an. Der Zugriff auf diesen **Dienst** erfolgt dann genauso, als ob er von einer gewöhnlichen **Funktionseinheit** dieser Hierachieebene bereitgestellt würde.

### 3.2.7 Vollständig modelliert: Die Theke

Das Aufnehmen der Bestellungen sowie die Ausgabe der Speisen benötigen offenbar Zeit und für die Dauer dieses Zeitraumes Personal. Gleiches gilt auch für den **Dienst** *kassieren*. Allerdings kann nur dann kassiert werden, wenn auch eine Kasse zur Verfügung steht. Wir verwenden einen **Counter** als Semaphor, allerdings werden sogar zwei gleichzeitige Verwendungen zugelassen; es existieren also zwei unabhängige Kassen. Personal wird aber erst dann angefordert, wenn auch eine Kasse zur Verfügung steht. Solange dies nicht der Fall ist, kann das Personal andere Tätigkeiten ausüben. Die Abbildung 3.6 zeigt das Editorfenster mit dem Modell der Theke.

### 3.2.8 Zusammenfassung

Wir haben eine erste Version des Modells unseres Schnellimbisses modelliert. Dabei kommen verschiedene Modellelemente zum Einsatz, welche über Attribute für unsere Zwecke angepasst wurden.

Das hier vorgestellte Modell ist nur eine Möglichkeit unter vielen einen Schnellimbiss zu modellieren. Vielleicht entwerfen Sie selbständig ein anderes Modell. In der Praxis wird man vielleicht mehrere Versuche starten, bis man zu einem Modell gefunden hat, welches den Anforderung entspricht.

Wir schließen nun die Editierfenster des Prozessketteneditors und speichern das Modell durch Auswählen des Menüpunktes “Speichern” aus dem “Datei”-Menü unter dem Namen “Imbiss.mod”.

Die Abbildung 3.7 zeigt, wie im Hauptfenster des Prozessketteneditors nun der Hierachiebaum dargestellt wird.

Bevor wir unser Modell nun analysieren können, müssen wir zunächst noch definieren, was wir auswerten möchten.

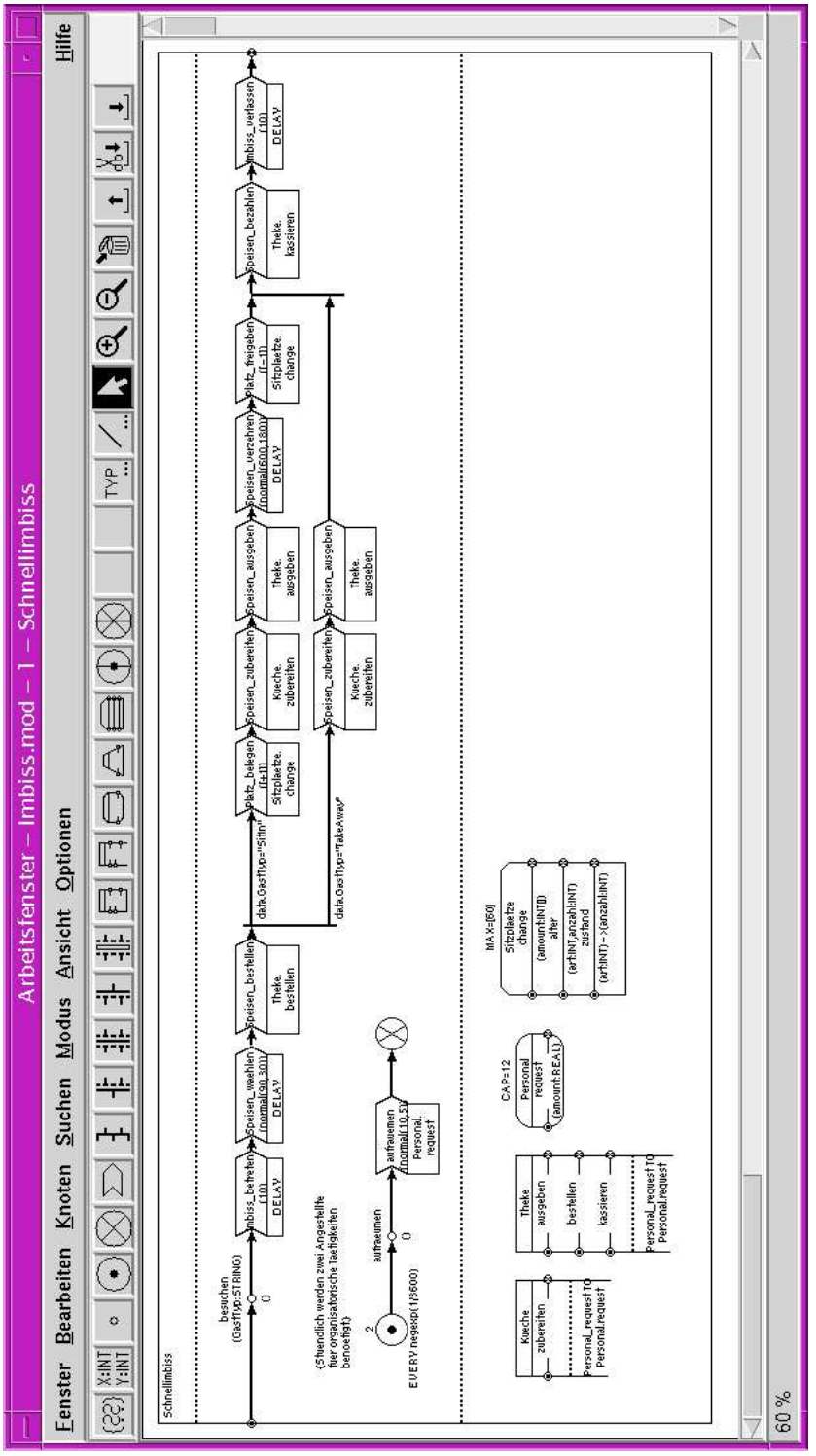



Abbildung 3.4: Editorfenster mit dem Modell der Innenansicht des Schnellimbisses

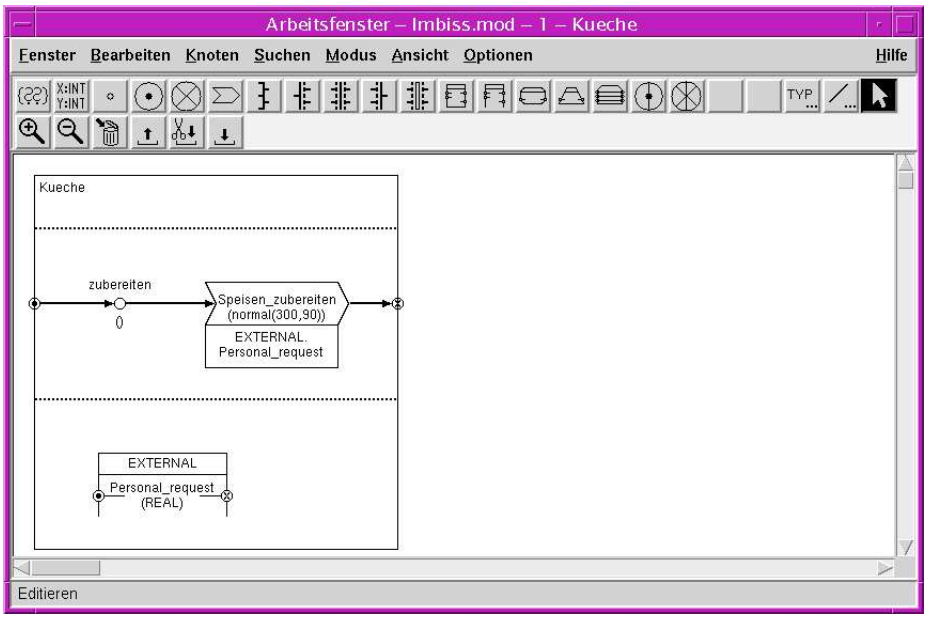



Abbildung 3.5: Editorfenster mit dem Modell der Küche

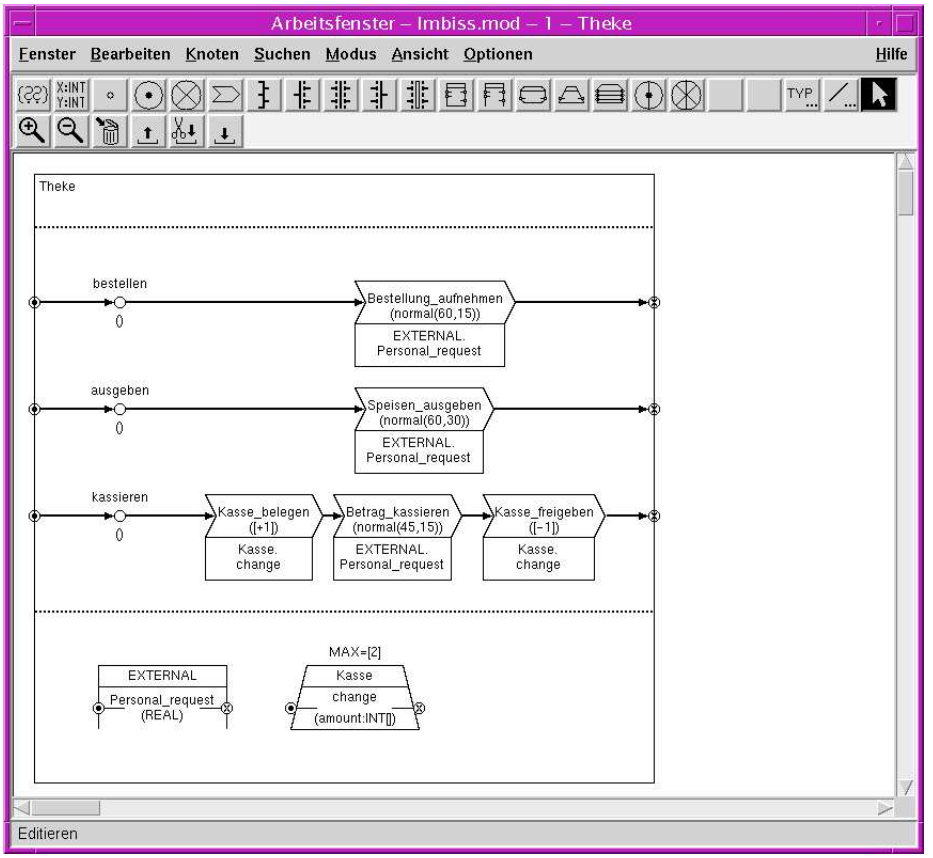



Abbildung 3.6: Editorfenster mit dem Modell der Theke

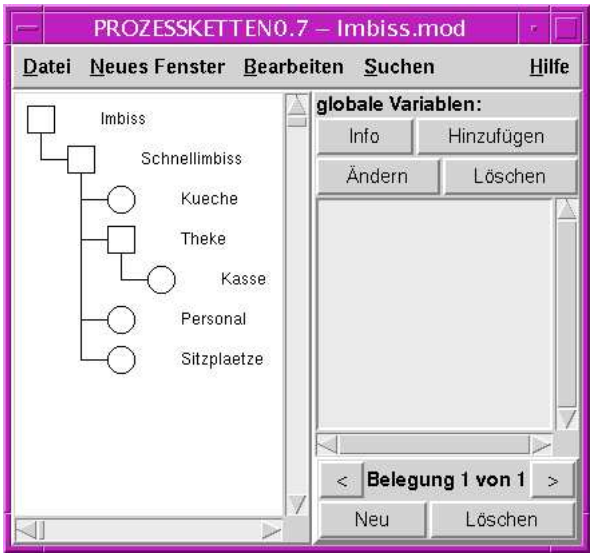



Abbildung 3.7: Hauptfenster des Prozessketteneditor mit Anzeige des Hierarchiebaums

# Kapitel 4

# Experimentdefinition mit dem Prozessketten-Editor

Nachdem wir nun eine erste Version des Modells konstruiert haben, möchten wir eine Analyse durchführen, um mögliche Fehler zu finden und einen Überblick über die Leistungsfähigkeit unseres Modells zu gewinnen.

## 4.1 Start der Experimentdefinition

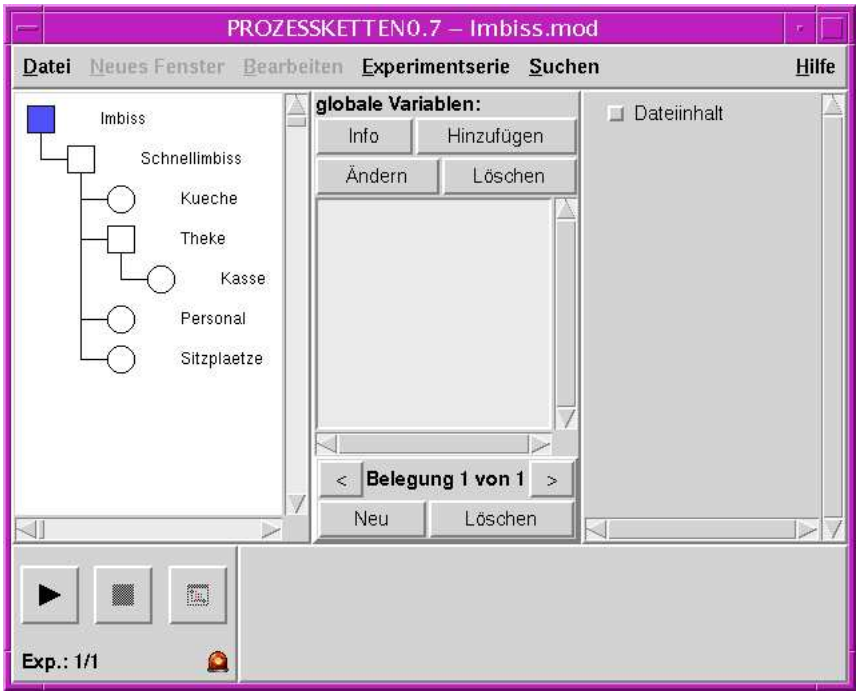



Abbildung 4.1: Experimentfenster mit Hierachiebaum

Zum Starten der Experimentdefinition wählen wir den obersten Knoten der zu analysierenden Hierarchie im Hierarchiebaum, der im Hauptfenster des Prozesskettene-

ditors zu sehen ist, aus. Im lokalen Menü des obersten Knotens starten wir das Experiment durch Auswahl der Menüpunkte "Experimentserie starten" - "B1-Analyse". Abbildung 4.1 zeigt das Experimentfenster nach dem Start der Definition des Experiments. Dabei wird der oberste Knoten der zu analysierenden Hierarchie blau dargestellt. Man hat durch Auswahl eines anderen Knotens die Möglichkeit, nur Teilhierarchien zu analysieren, was aber höchstens bei sehr umfangreichen Modellen sinnvoll erscheint und erst in einer späteren Version des Prozessketteneditors unterstützt werden wird.

## 4.2 Definition eines einfachen Messobjekts (DUE TO ALL)

Zunächst einmal messen wir die Auslastung unserer Sitzplätze. Dazu öffnen wir das Fenster, welches die Innenansicht des Schnellimbisses zeigt. Wir öffnen mit der Maus am **Lager** *Sitzplaetze* ein lokales Menü. Hier wählen wir die Menüpunkte "Neuer Messpunkt" - "UTILIZATION" - "DUE TO ALL". Das bedeutet, dass wir die Auslastung unabhängig vom Verursacher bestimmen wollen. Abbildung 4.2 zeigt das geöffnete Fenster. Das Fähnchen am **Lager** zeigt nach dem Abschluss der Definition, dass sich dort wenigstens ein Messobjekt befindet.

## 4.3 Definition eines Messobjekts mit Verursacherpfad (DUE TO Pfadname)

An der **Funktionseinheit** *Schnellimbiss* selbst, möchten wir die Verweilzeiten messen. Wir definieren nun drei Messpunkte an der **Funktionseinheit**, die uns Aussagen über die Verweildauer allgemein sowie die Verweildauer aufgeschlüsselt nach den Gasttypen liefern.

Dazu öffnen wir das Fenster, welches die Außenansicht des Schnellimbisses wie in Abbildung 4.3 zeigt. Wir definieren den Messpunkt für die allgemeine Verweildauer analog zum Messpunkt am **Lager**. Die Definition der Messpunkte für die aufgeschlüsselten Verweilzeiten ist ein wenig komplizierter.

Zunächst erzeugen wir über das lokale Menü an der **Funktionseinheit** einen neuen Verursacherpfad. Diesen nennen wir *PfadTakeAway*. Wir müssen nun nacheinander die Elemente angeben, welche den Verursacherpfad kennzeichnen. Dieses sind bei uns das **Prozesskettenelement** *Schnellimbiss.besuchen (TakeAway)* und die **Funktionseinheit** selbst. Wir wählen daher in dieser Reihenfolge aus den lokalen Menüs dieser beiden Elemente die Einträge "ergänzen mit...". Nachdem wir die **Funktionseinheit** hinzugefügt haben, bekommen wir die Meldung, dass der Verursacherpfad vollständig und eingefügt worden ist. Anschließend können wir im lokalen Menü nicht nur allgemein (DUE TO ALL), sondern auch speziell (DUE TO PfadTakeAway) messen.

Bei der Definition des Messpunkts für den Gasttyp Sit-In verfahren wir analog mit dem **Prozesskettenelement** *Schnellimbiss.besuchen (SitIn)* und der **Funktionseinheit** selbst.

Abbildung 4.4 zeigt das lokale Menü der **Funktionseinheit** *Schnellimbiss* nach dem Abschluss der Experimentdefinitionen. Alle Messpunkte werden zusammen mit den Verursacherpfaden angezeigt.

## 4.4 Speichern der Experimentdefinition

Bevor wir uns nun der Simulation zuwenden können, müssen wir das Experiment speichern. Dies ermöglicht es uns auch, das Experiment später wieder aufzurufen, zu kontrollieren oder abzuändern. Vielleicht editieren wir auch nur einige Parameter und schauen die Auswirkungen an.

Zum Speichern wählen wir den Eintrag "Experimentserie speichern" aus dem Menü "Datei". Vom Programm wird ein Name vorgeschlagen, wobei natürlich auch ein anderer gewählt werden kann. Es ist sinnvoll, eine Namenskonvention zu verwenden, die die Beziehung zwischen Modelldatei und Experiment verdeutlicht und das Experiment beschreibt. Da wir hauptsächlich **Verweilzeiten** messen, nennen wir das Experiment "Imbiss.TurnAround.exp".

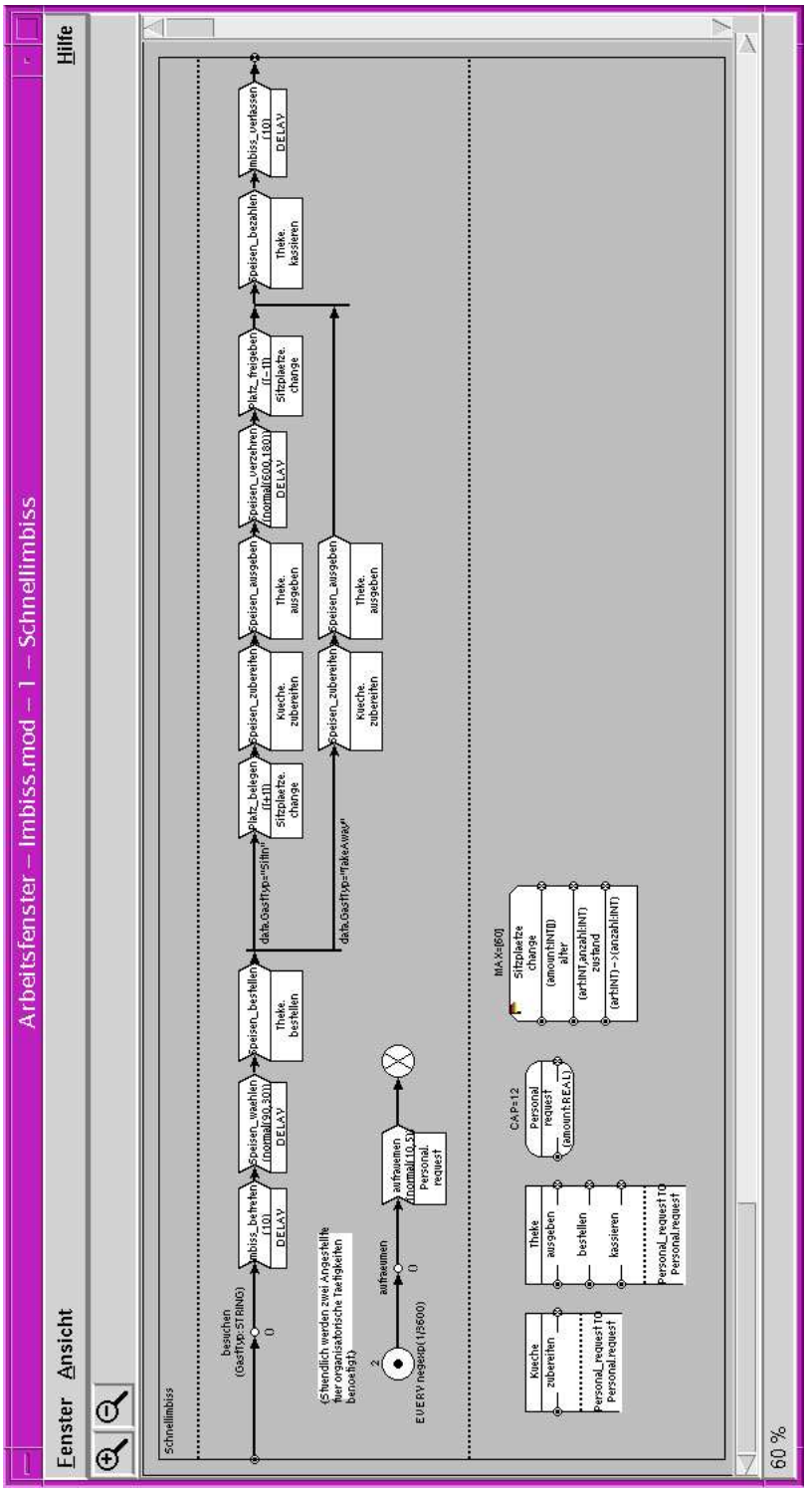



Abbildung 4.2: Experimentfenster mit der Innenansicht des Schnellimbisses

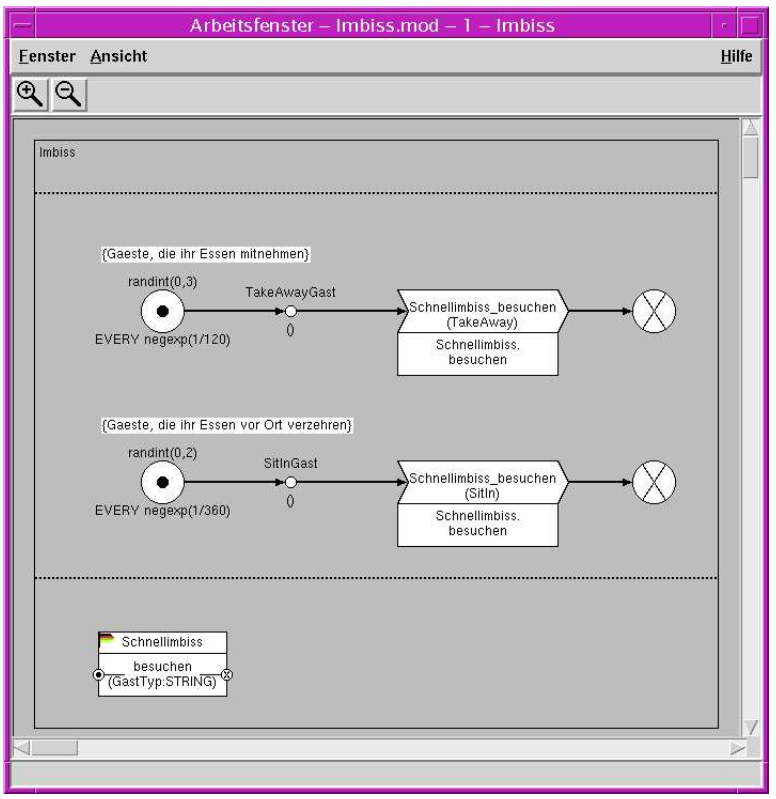



Abbildung 4.3: Experimentfenster mit der Außenansicht des Schnellimbisses

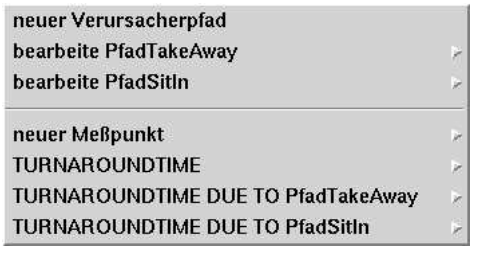



Abbildung 4.4: Lokales Menü der Funktioneinheit Schnellimbiss mit Anzeige der Messobjekte

# Kapitel 5

# Analyse und Ergebnisdarstellung

Nachdem wir uns bisher mit Vorüberlegungen und Modellierung beschäftigt haben, wenden wir uns nun der Analyse und der Ergebnisdarstellung zu und unsere bisherigen Bemühungen werden in gewisser Weise belohnt.

Wie die Modellierung werden auch Analyse und Ergebnisdarstellung durch die B1-Tools unterstützt.

Die Analyse erfolgt in Form einer Simulation durch einen Löser.

## 5.1 B1-Analysator

Aus dem Hauptmenü des B1-GUIs starten wir zunächst den B1-Analysator. Abbildung 5.1 zeigt das Hauptfenster des B1-Analysators unmittelbar nach dem Start. Dieser besitzt im oberen Bereich eine Eingabemaske, welche verschiedene Einstellungsoptionen anbietet. Einserseits können Pfade zu Programmen, welche für die Analyse benötigt werden eingestellt werden, andererseits wird das zu analysierende Modell festgelegt. Wir wählen natürlich unsere zuvor gespeicherte Modelldatei "Imbiss.mod" aus.

### 5.1.1 Einstellungen

Im mittleren Bereich des Fensters werden zahlreiche Einstellungsmöglichkeiten angeboten, welche Einfluss auf den Verlauf und die Ergebnisse der Analyse haben. Die Analyse ist deterministisch. Modell- und Experimentbeschreibung sowie die folgenden Parameter legen das Ergebnis eindeutig fest. Abbildung 5.2 zeigt das Fenster des B1-Analysators nach dem Eintragen der Einstellungen.

#### Modellzeit

Die Modellzeit gibt an, über wie viele Modellzeiteinheiten die Analyse durchgeführt werden soll. Wir hatten uns dazu entschieden, eine Modellzeiteinheit einer Sekunde entsprechen zu lassen. Da wir nicht wissen, ob und wann die stationäre Phase erreicht wird, wählen wir 20 Stunden, was 72000 Modellzeiteinheiten entspricht; da unser Schnellimbiss auch nicht länger als 20 Stunden geöffnet hat, stellt dies auch keine Einschränkung dar.

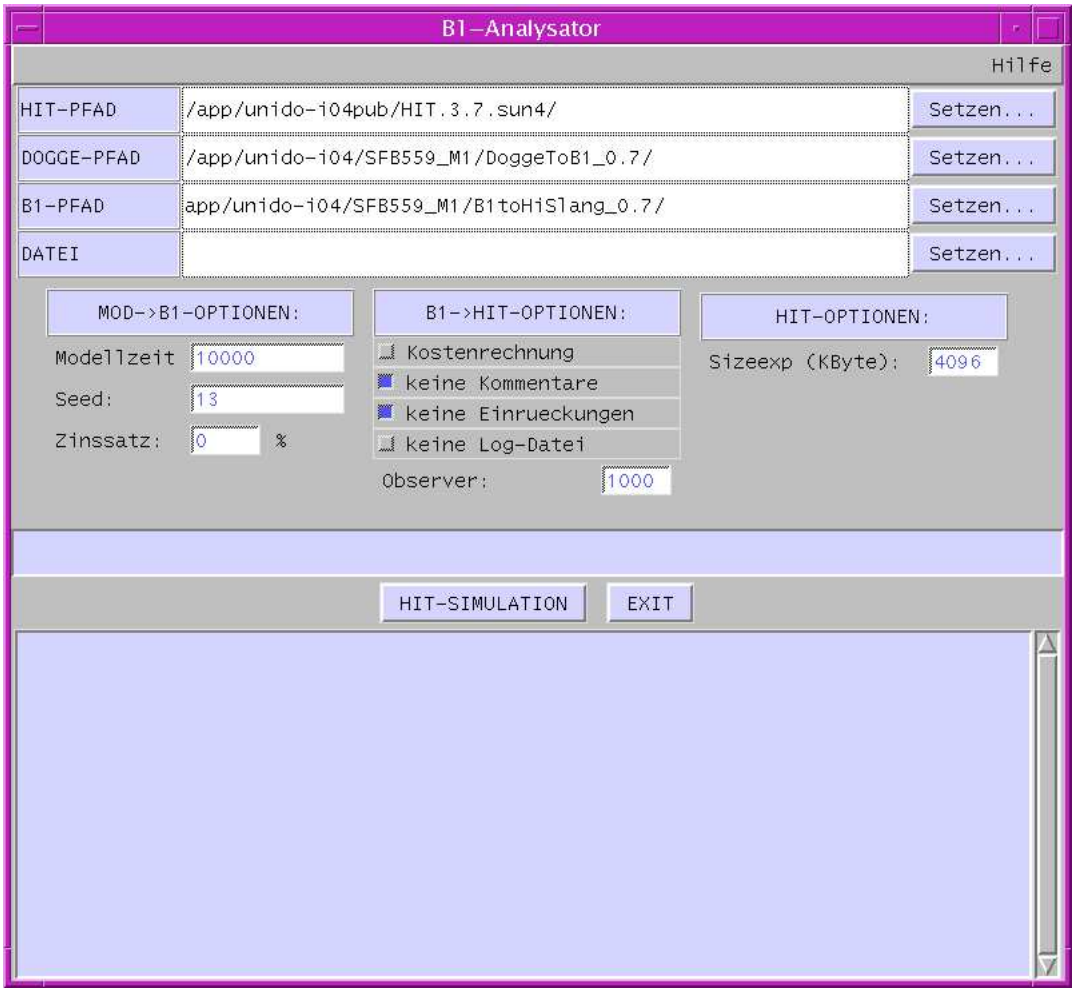



Abbildung 5.1: B1-Analysator nach dem Start

### Seed

Dies ist der Startwert für die Zufallszahlengenerierung des Analysetools. Wir belassen es beim Standardwert. Sinnvollerweise werden hier Primzahlen verwendet.

### Zinssatz

Der Zinssatz wird für die Kostenrechnung benötigt. Für die aktuelle Analyse ist er nicht relevant.

### Kostenrechung

Diese Option ist dann auszuwählen, wenn eine Analyse unter dem Gesichtspunkt der Kostenrechnung vorgenommen werden soll.

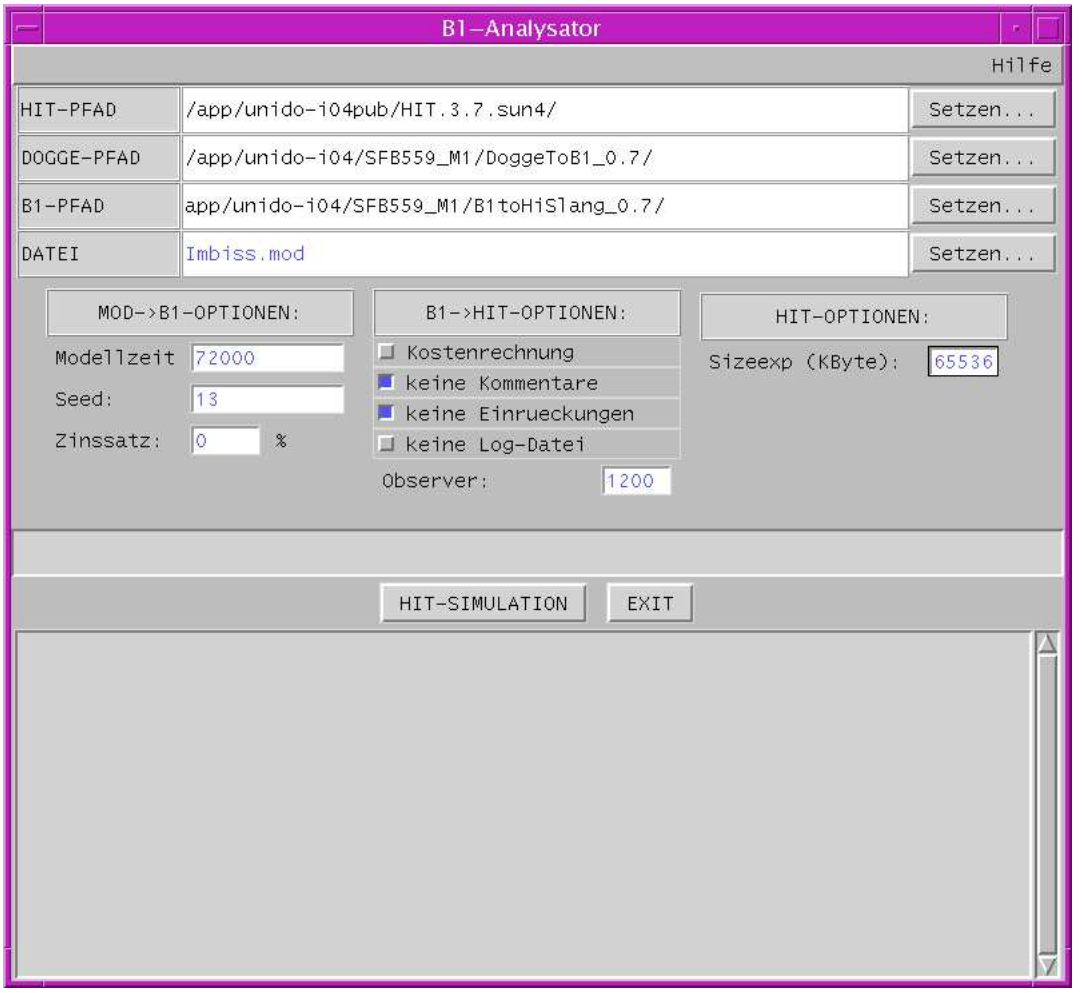



Abbildung 5.2: B1-Analysator mit Einstellungen für den Start der Simulation

**keine Kommentare, keine Einrückungen und keine Log-Datei**

Diese Optionen sind nur für Debuggingzwecke sinnvoll. Sie haben keinen Einfluss auf die Analyse und die Ergebnisse.

**Observer**

Die Ergebnisse werden in Form von Kurven in Diagrammen (Modellzeit -> Wert) dargestellt. Die Anzahl der Observerzeitpunkte gibt an, zu wie vielen Modellzeitpunkten Messwerte ermittelt werden sollen. Je mehr Observerzeitpunkte angegeben werden, desto länger dauert die Analyse. Wir wählen 1200 Observerzeitpunkte. Dies bedeutet, dass alle 60 Modellzeiteinheiten, also jede volle Minute, Messwerte ermittelt werden.

**Sizeexp**

Diese Einstellung gibt an, wie viel Arbeitsspeicher der Simulation zur Verfügung stehen soll. Grundsätzlich gilt: Wenig Speicher verlangsamt die Analyse drastisch.

Trotzdem kann man durch die Angabe von zu viel Arbeitsspeicher, den Rechner für andere Prozesse blockieren. Die sinnvolle Wahl hängt von vielen Faktoren, auch von der Modellgröße ab. Als Richtwert gilt, wenigstens 4 MB und nicht mehr als die Hälfte des verfügbaren Realspeichers anzugeben. 64 MB haben sich bei unserem Modell als ausreichend erwiesen.

### 5.1.2 Simulation

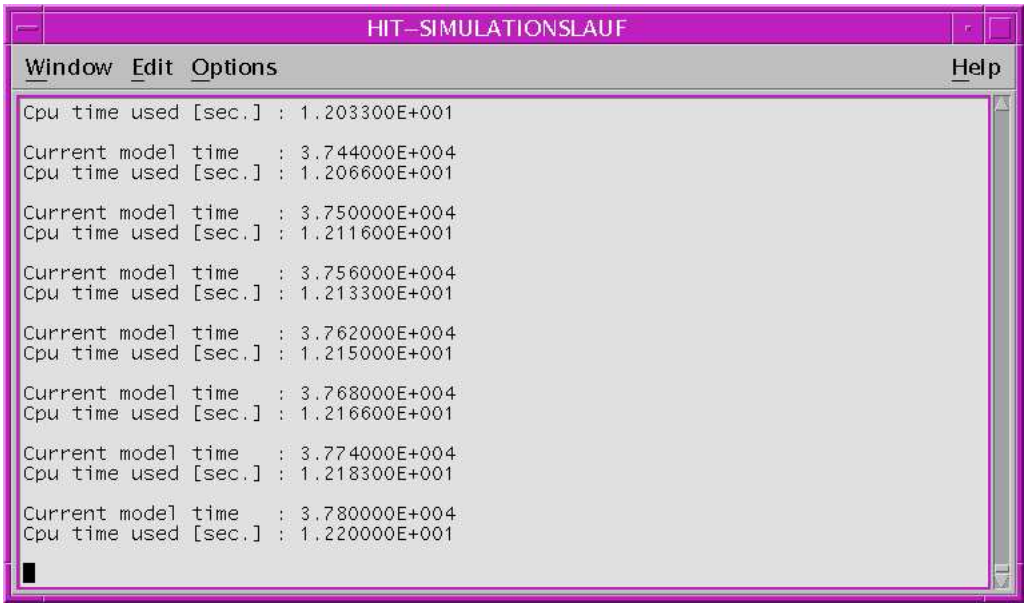



Abbildung 5.3: Ausgabe während der Simulation

Nachdem wir die Einstellungen vorgenommen haben, können wir die eigentliche Analyse, also die Simulation starten. Zunächst wird nach der Datei mit der Experimentbeschreibung gefragt. Wir wählen die Datei "Imbiss.TurnAround.mob" aus.

Während der Analyse werden Ausgabefenster geöffnet, die über den Fortschritt der Analyse informieren. Die Ausgabe entspricht der in Abbildung 5.3 gezeigten. Es werden die bisher abgearbeitete Modellzeit und verbrauchte CPU-Zeit angezeigt. Häufiges Auftreten der Garbage-Collection deutet daraufhin, dass der Speicher recht knapp bemessen ist, so dass man bei einer Wiederholung der Analyse des Modells doch mehr Speicher bewilligen sollte, sofern dies möglich ist.

Nach Abschluss der Simulation zeigt das Fenster des B1-Analysators im unteren Bereich die Zusammenfassung der Ausgaben des Lösers, ähnlich der Ausgabe, welche in Abbildung 5.4 zu sehen ist.

## 5.2 B1-Plotter

Nachdem wir die Simulation durchgeführt haben, können wir nun die Ergebnisse betrachten und auswerten. Dabei werden wir durch den B1-Plotter unterstützt, welcher sich, wie die anderen Tools, aus dem Hauptmenü des B1GUIs starten lässt. Abbildung 5.5 zeigt das Hauptfenster des B1-Plotters.

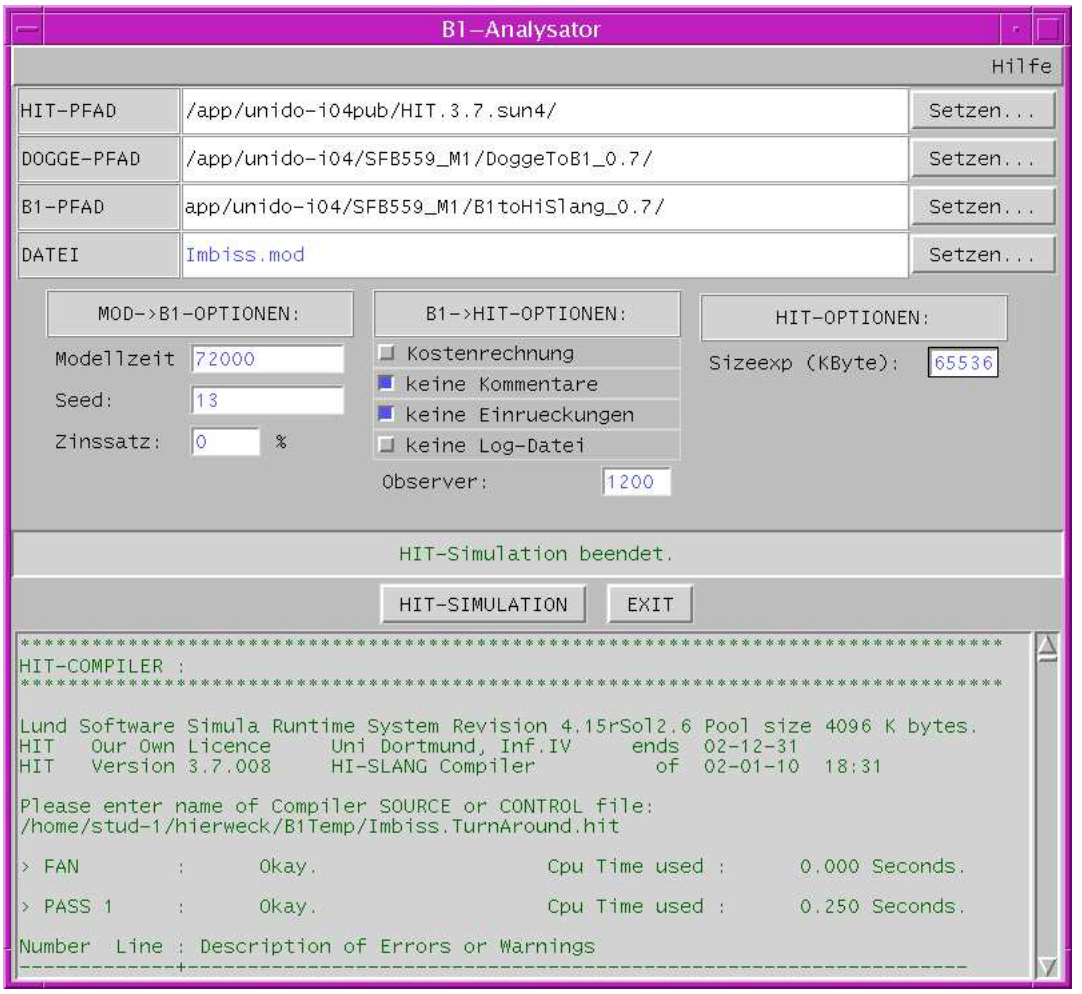



Abbildung 5.4: B1-Analysator mit der Zusammenfassung der Ausgaben des Lösers

Wir haben haben bei der Experimentdefinition zwei Auswertungen vorgesehen, die Verweildauer der Kunden innerhalb des Schnellimbisses und die Auslastung des Sitzplätze. Da es sich um Messergebnisse handelt, die nicht unmittelbar in Zusammenhang stehen, werden wir sie getrennt auswerten und darstellen.

Zunächst öffnen wir jedoch die Datei, welche unsere Ergebnisse enthält. Es handelt sich in Anlehnung an den Namen der Experimentbeschreibungsdatei um die Datei "t.Imbiss.TurnAround.dum".

Die Darstellung der Ergebnisse erfolgt in Form von Diagrammen. Dabei sind für die Kurven verschiedene Typen auswählbar, die aus den Ergebnissen berechnet werden. Diese sind: Mittelwert (MeanT), Mittelwert über dem Intervall zwischen zwei Observerzeitpunkten (MeanDeltaT), positive und negative Konfidenz (confplus und confminus) sowie die Standardabweichung (stddev).

### 5.2.1 Auswertung: Verweildauer im Schnellimbiss

Wenden wir uns zunächst der Verweildauer der Kunden im Schnellimbiss zu. Wir hatten drei Messungen in diesem Bereich definiert. Einerseits wollten wir die Ver-

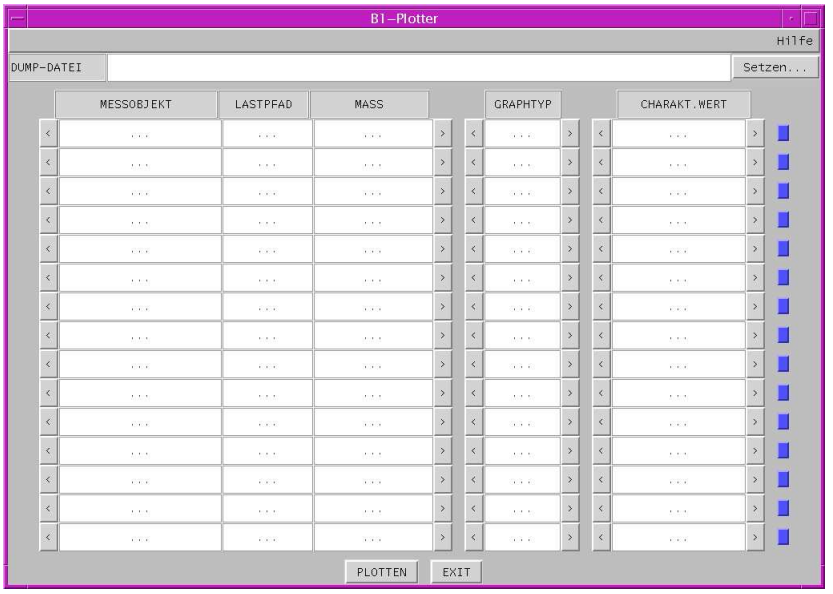



Abbildung 5.5: B1-Plotter nach dem Start

weildauer aller Gäste bestimmen, andererseits sollte dies nach GastTyp aufgeschlüsselt werden. Folglich werden wir drei Messkurven erhalten.

### Kurvenauswahl

Wir wählen die Messkurven, wie in Abbildung 5.6 dargestellt wird, aus der Liste der Messkurven aus. Dies geschieht per Mausklick auf die Pfeile neben den Zeilen, welche die Kurven bezeichnen, im linken Bereich des Fensters. Im rechten Bereich ist es per Mausklick möglich, den Typ der Kurven und die Anzeigefarbe auszuwählen.

### Diagramm

Nach dem Start des Plottvorgangs, wird das in Abbildung 5.7 dargestellte Diagramm angezeigt. Es zeigt die ausgewählten Messkurven und die Legende. Es besteht, die Möglichkeit das Diagramm als Postscriptdatei zu exportieren und weiterzuverwenden.

### Interpretationsversuch

Die Interpretation der Ergebnisse kann natürlich nicht durch die B1-Tools erfolgen, sie bleibt dem Benutzer überlassen.

Zunächst einmal erkennen wir, dass die Kurven einen recht ähnlichen Verlauf haben. Sie steigen zunächst steil an und erreichen bald einen vermutlich stationären Bereich, d.h. sie bleiben dann annähernd auf einem Niveau. Diesen Bereich, ab dem sich die Entwicklung der Mittelwerte stabilisiert hat, untersuchen wir nun.

Die drei Kurven können wir nun als parallele Gerade sehen. Dies bedeutet zunächst, dass die Verweildauer nicht weiter steigt. Im Sinne unseres Schnellimbisses heißt

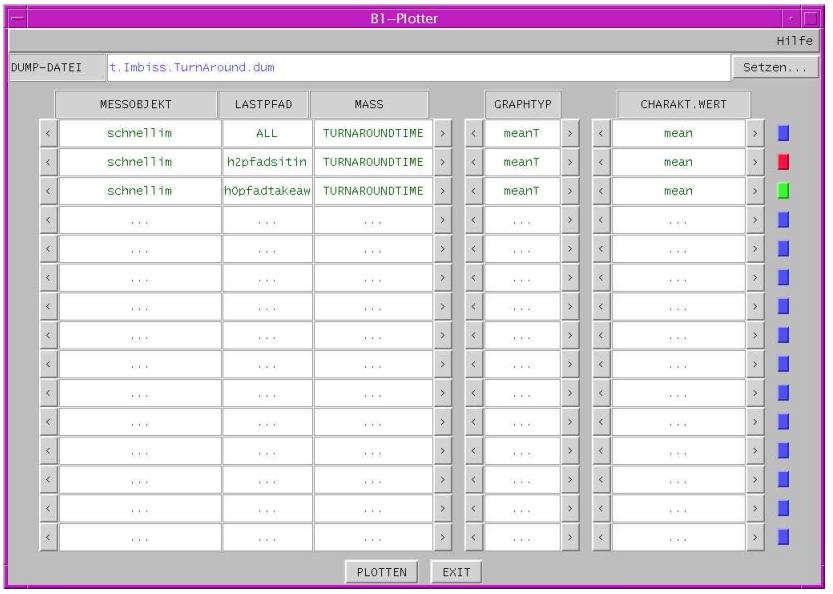



Abbildung 5.6: Auswahl der Kurven zur Auswertung der Verweildauer im Schnellimbiss

das, dass die Aufenthaltsdauer vermutlich beschränkt ist und sich - anschaulich betrachtet - keine immer länger werdende Schlange wartender Kunden bildet.

Aus der Legende bzw. der Farbcodierung geht hervor, dass sich die TakeAwayGäste im Mittel etwa 600 Zeiteinheiten, also 10 Minuten lang, im Schnellimbiss aufhalten. Die SitInGäste benötigen etwa doppelt so lange.

Die mittlere Verweildauer aller Gäste liegt nur minimal oberhalb der Verweildauer der TakeAwayGäste. Dies liegt daran, dass das der Schnellimbiss viel häufiger von TakeAwayGästen besucht wird, so dass die Verweildauer der SitInGäste bei der Mittelwertbildung einen geringeren Einfluss hat.

### 5.2.2 Auswertung: Auslastung der Sitzplätze

Nach der Verweildauer werten wir nun die Auslastung der Sitzplätze aus.

#### Kurvenauswahl

Wir haben hierfür nur ein Experiment definiert. Wir möchten uns aber nicht nur die Mittelwerte anschauen, sondern wählen, wie in Abbildung 5.8 gezeigt wird, auch die Kurventypen Standardabweichung sowie positive und negative Konfidenz aus.

#### Diagramm

Das Diagramm 5.9 zeigt die Kurvenschar an. Positive und Negative Konfidenz werden erst nach dem ersten Observerzeitpunkt dargestellt.

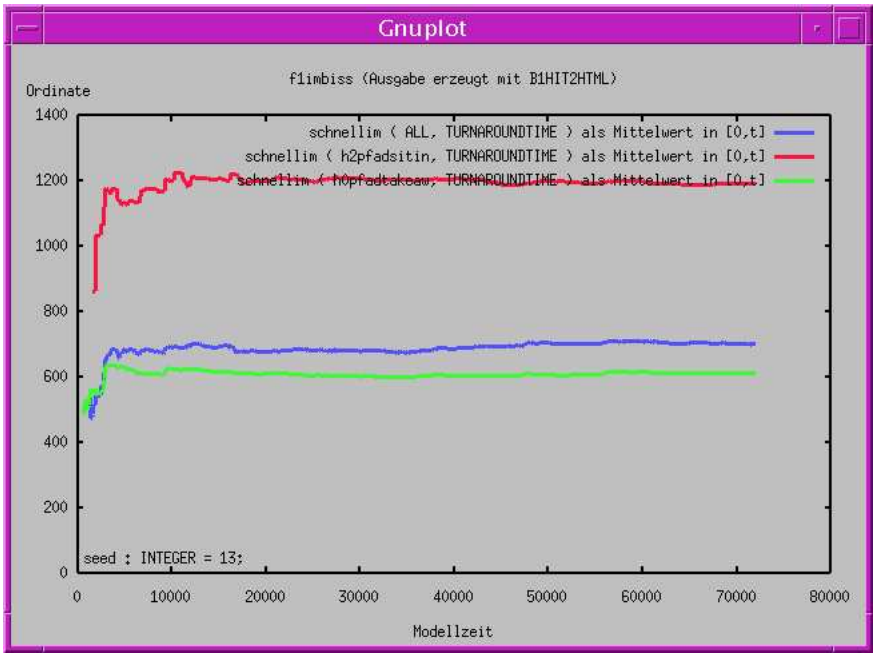



Abbildung 5.7: Diagramm mit Kurven zur Auswertung der Verweildauer im Schnellimbiss

#### Interpretationsversuch

Auch diese Ergebnisse lassen sich interpretieren.

Versuchen wir erneut die stationäre Phase zu identifizieren. Der Mittelwert scheint sich bei 2,5 einzupendeln. Positive und negative Konfidenz bilden einen Schlauch um den Mittelwert herum. Die Standardabweichung liegt offenbar bei 1,5. Die stationäre Phase wurde jedoch noch nicht erreicht, möchte man genauere Ergebnisse erhalten, so müsste man die Simulation mit längerer Simulationsdauer wiederholen.

Dennoch kann man versuchen Antworten auf Fragen nach der erforderlichen Sitzplatzkapazität zu finden. Der Mittelwert schwankte um 2.5 - benötigt der Schnellimbiss also nur 2,5 Sitzplätze?

Theoretisch ja. Da es sich aber um den Mittelwert der Auslastung während der bisherigen Experimentdauer handelt und die tatsächliche Auslastung schwankt, die Standardabweichung liegt ja bei 1,5, würde eine Beschränkung auf zwei Sitzplätze dafür sorgen, dass das Modell vermutlich nicht mehr stationär ist, da die Schlange wartender Kunden immer länger würde. Auch drei Sitzplätze werden vermutlich zu langen Wartezeiten führen.

## 5.3 Zusammenfassung

Wir haben eine Simulation unseres Modells und dabei verschiedene Auswertungen durchgeführt.

Insgesamt sehen wir, dass sich die Ergebnisse mit Hilfe des Modells und seiner Parameter erklären lassen. Ferner decken sich die Ergebnisse offenbar mit den Er-

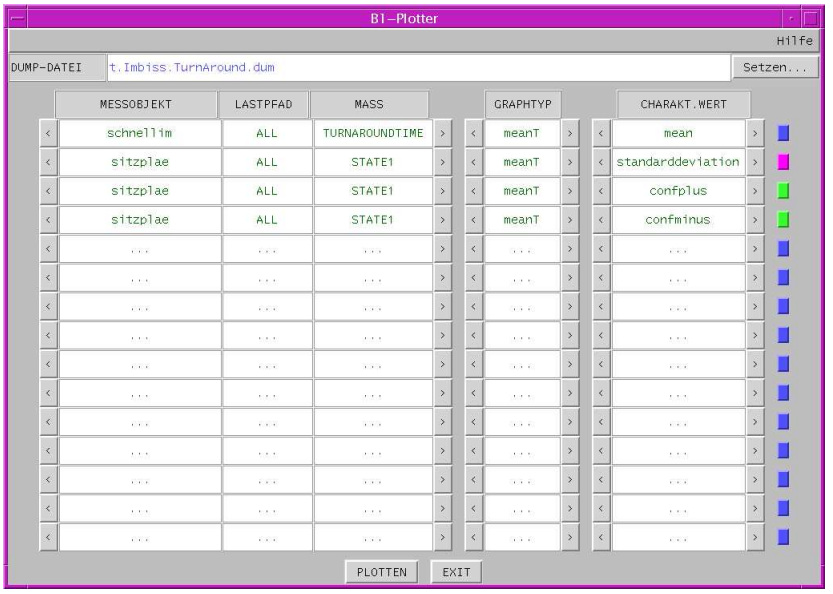



Abbildung 5.8: Auswahl der Kurven zur Auswertung der Auslastung der Sitzplätze

fahrungen, die wir beim Besuch eines Schnellimbisses gewonnen haben. Dies spricht dafür, dass das Modell eine gute Annäherung der Realität darstellt.

Nun können wir das Modell verfeinern oder abändern und die Auswirkungen beobachten. Interessant wäre beispielsweise, mit wie vielen oder besser wie wenig Sitzplätzen man noch vertretbare Verweilzeiten erhält. Auch der Personalbestand lässt sich vermutlich reduzieren... Was geschieht, wenn man eine Kasse schließt?

Wir werden diese Untersuchungen nicht vorführen, sondern überlassen sie Ihnen. Sie können sich dabei einfach an unserer Vorgehensweise orientieren und dabei ggf. auf die Online-Hilfe der Tools zurückgreifen.

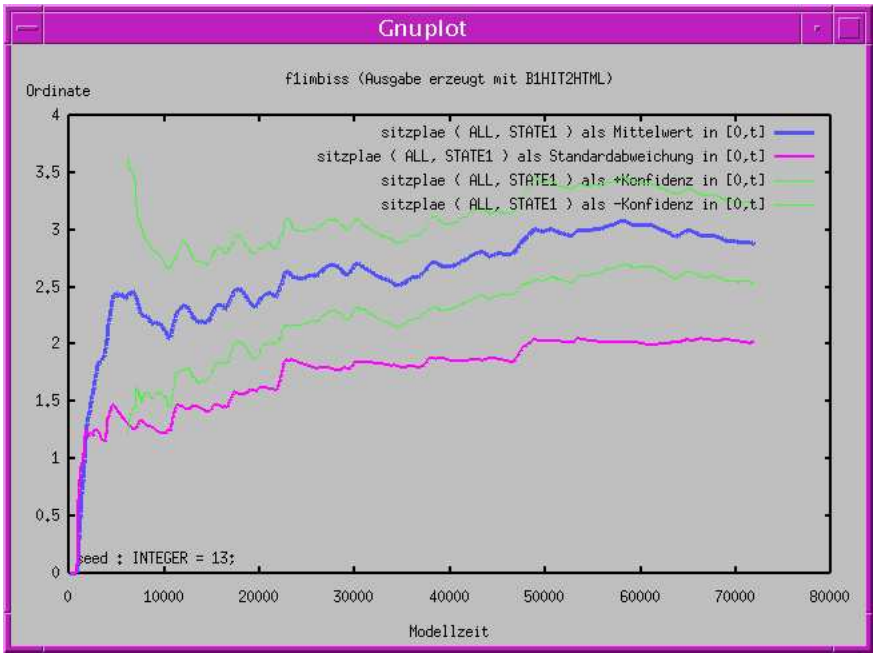



Abbildung 5.9: Diagramm mit Kurven zur Auswertung der Auslastung der Sitzplätze

# Kapitel 6

# Kostenrechnung

Bisher haben wir uns mit der Modellierung eines Schnellimbisses und dessen Auswertung unter Gesichtspunkten der Leistungsbewertung beschäftigt. Es ist jedoch auch möglich, Auswertungen hinsichtlich der Ökonomie durchzuführen. Wir werden im Folgenden das Modell so anpassen, dass diese Auswertungen möglich sind, eine Analyse durchführen und die Ergebnisse auswerten.

## 6.1 Einführung

Die Leistungsbewertung ermöglicht es, ein Modell hinsichtlich Auslastung, Durchsatz, Verweilzeiten und ähnlicher Aspekte auszuwerten. Ergebnisse lauten in unserem Fall beispielsweise, es sind ausreichend viele Sitzplätze vorhanden oder das Schließen einer Kasse verursacht eine immer länger werdende Schlange wartender Kunden.

Interessant sind aber auch Antworten auf Fragen nach den Kosten, die beispielsweise durch das Öffnen einer zweiten Kasse entstehen oder durch die Bereitstellung weiterer Sitzplätze. Geschäftsleute stellen sich natürlich auch die Frage, ob es evtl. sinnvoll wäre, den Schnellimbiss zu schließen oder gar nicht erst zu eröffnen, weil der zu erwartende Gewinn unterhalb des Gewinns liegt, den man z.B. durch eine andere Kapitalanlage, z.B. Zinsen, erwirtschaften könnte.

Die Kostenrechnung soll dabei helfen, Antworten auf diese Fragen zu finden.

### 6.1.1 Unterstützung

Die Kostenrechnung ist ähnlich der Leistungsbewertung in die B1-Tools integriert. Modellierung und Analyse werden auf die gleiche Weise durchgeführt, für die Ergebnisdarstellung existiert ein weiteres Tool. Leistungsbewertung und Kostenrechnung schließen sich nicht gegenseitig aus, sie können mit demselben Modell sogar zeitgleich durchgeführt werden. Die durch Simulation ermittelten Kosten können nach dem Verursacher aufgeschlüsselt ausgewertet werden.

### 6.1.2 Beschränkungen

Die Kostenrechnung unterstützt derzeit keine Externen Funktionseinheiten. Dies bedeutet, dass wir unser Modell des Schnellimbisses anpassen müssen, so dass es

keine Externen Funktionseinheiten benötigt. In Zukunft soll diese Einschränkung entfallen.

## 6.2 Prozessketteneditor

Im Prozessketteneditor besteht die Möglichkeit, in den Attributmasken der Funktionseinheiten festzulegen, welche Fixkosten, welche Kosten pro Aufruf eines Dienstes bzw. Zeiteinheit, die der Aufruf des Dienstes in Anspruch nimmt, etc. anfallen. Dazu werden die entsprechenden Größen in den Attributmasken auf der Karteikarte "Kostenrechnung" eingetragen.

### 6.2.1 Modell

Zunächst müssen wir einen Weg finden, der es uns ermöglicht, auf die Externen Funktionseinheiten zu verzichten. Wir haben die Ressource *Personal* gemeinsam genutzt. Eine mögliche Lösung besteht darin, diese Ressource aufzuspalten und den Funktionseinheiten, welche Personal benötigen, jeweils eigenständiges Personal zuzuordnen. Wir werden diese Möglichkeit nun umsetzen.

Die gemeinsam verwendete Ressource *Personal* befindet sich innerhalb der Funktionseinheit *Schnellimbiss*. Wir öffnen diese im Prozessketteneditor.

In der Attributmaske des Servers *Personal* setzen wir den Wert Kapazität auf 1. Damit steht der Prozesskette *Aufraeumen* nur noch ein Mitarbeiter zur Verfügung. An den Funktioneinseinheiten *Kueche* und *Theke* lösen wir nun die Verbindung zum Server *Personal*, in dem wir in der Attributmaske, die Prozesszuordnung und den jeweiligen virtuellen Parameter löschen.

Die Abbildung 6.1 zeigt die Innenansicht des Schnellimbisses nach der Änderung.

Nun müssen wir aber den Funktionseinheiten *Kueche* und *Theke* noch eigenes Personal zuordnen. Hierzu werden wir in beiden Funktionseinheiten lokal Server anlegen, welche das nun nicht mehr gemeinsam verwendete Personal repräsentieren.

Die Abbildungen 6.2 und 6.3 zeigen die Funktionseinheiten *Kueche* bzw. *Theke* nach den Änderungen.

Damit haben wir nun die notwendigen Vorbereitungen getroffen und das Modell so angepasst, dass es sich zum Einsatz mit der Kostenrechungsfunktion eignet.

### 6.2.2 Attribute

Für die Kostenrechnung müssen einige Attribute an den Modellelementen festgelegt werden. Die für die Kostenrechung relevanten Elemente, insbesondere die Funktionseinheiten, besitzen in der Attributmaske die Karteikarte "Kostenrechung". Dort lassen sich die Attribute, welche für die Kostenrechnung benötigt werden, einstellen.

Wir analysieren nun die Kosten, welche im Zusammenhang mit den Sitzplätzen entstehen. Wir wissen, dass die Sitzplätze dadurch Kosten verursachen, dass sie existieren, bei der Anschaffung oder durch die Miete des Raumes, die sie in Anspruch nehmen. Abhängig von der tatsächlichen Nutzung der Sitzplätze fallen weitere Kosten an, z.B. Verschleiß oder Reinigung. Gäste, die den Schnellimbiss besuchen, setzen einen bestimmten Betrag um, welcher bei Gästen, die ihre Speisen vor Ort

verzehren, erfahrungsgemäß höher liegt. Ferner besteht die Möglichkeit, einen Zinssatz festzulegen, so dass man feststellen kann, ob die Investition überhaupt sinnvoll war.

Wir legen folgende Werte fest: Fixkosten in Höhe von 10000 und Betriebskosten (State) in Höhe von [3] bei den Sitzplätzen – die eckigen Klammer sind erfordlich, da es sich im einen Vektor handelt – sowie einen Umsatz von 5 bei den SitIn-Gästen und 3 bei den TakeAway-Gaesten.

Wir speichern das modifizierte Modell unter dem Namen "KoRe.mod".

## 6.3 Experimentdefinition

Wie bei der Leistungsbewertung definieren wir ein Experiment. In diesem müssen aber keine Messobjekte definiert werden, wenn man ausschließlich die Kosten analysieren möchte, da das Kostenrechnungsmodul alle notwendigen Messobjekte automatisch anlegt, d.h. weitere Schritte zur Experimentdefinition sind nicht erforderlich. Grundsätzlich ist es aber möglich parallel zur Kostenrechung auch eine Leistungsbewertung durchzuführen.

Das Experiment speichern wir unter dem Namen "KoRe.Experiment.exp".

## 6.4 B1-Analysator

Nach dem Start des B1-Analysators, der auch in diesem Fall die Simulation starten wird, tragen wir die gleichen Parameter ein, wie beim vorhergegehenden Experiment. Allerdings heißt die zu analysierende Modelldatei nun "KoRe.mod". Wir schalten nun die Option Kostenrechung ein und legen als Zinssatz 7 % fest.

Die Abbildung 6.4 zeigt den B1-Analysator mit den Optionen der Kostenrechnung.

### 6.4.1 Auswahl der Experimentdatei und der Messobjekte

Nach dem Start der Simulation wird wieder nach der Experimentdatei gefragt. Diese heißt "KoRe.Experiment.mob". Anschließend werden zahlreiche Pfade zur Analyse vorgeschlagen. Es ist sinnvoll, alle auszuwählen und alle Fragen mit ja zu beantworten.

### 6.4.2 Simulation

Der Ablauf der Simulation entspricht dem aus Kapitel 5 bekannten Ablauf. Nachdem diese beendet wurde, wird wieder eine Zusammenfassung der Ergebnisse angezeigt.

## 6.5 Ergebnisdarstellung

Da wir nun eine Auswertung in Form der Visualisierung der Ergebnisse vornehmen möchten, starten wir die Kostenvisualisierung aus dem Hauptmenü des B1GUIs.

Zunächst erscheint das in Abbildung 6.5 gezeigte Fenster, in welchem die zur Durchführung der Auswertung benötigten Parameter eingegeben werden können. Notwendig ist es, die entsprechene Datei auszuwählen. Ferner besteht die Möglichkeit den Maßstab der Grafiken anzupassen.

Anschließend werden aus den Simulationsergebnissen HTML-Dateien erzeugt, welche über einen Webbrowser, der automatisch gestartet wird, angezeigt und ausgedruckt werden können.

### 6.5.1 Darstellung im Webbrowser

Der gestartete Webbrowser zeigt zunächst die in Abbildung 6.6 dargestellte Webseite mit dem Hierachiebaum des Modells an.

Im oberen Bereich der Seite werden vier Menüpunkte angeboten. Der Menüpunkt ¨Hierarchie" ermöglicht die Rückkehr zur Startseite, der Menüpunkt "Messwerte" bietet Zugang zur Visualisierung der Ergebnisse der Leistungsbewertung. Die eigentliche Kostenvisualisierung wird über den Menüpunkt "Kostenrechnung" gestartet. Hinter dem Menüpunkt "Hilfe" verbirgt sich die Online-Hilfe der Kostenvisualisierung, in der die Handhandung des Tools, aber auch Bedeutung der Symbole und Abkürzungen erklärt werden.

Viele der Beschriftungen oder Grafiken sind mit Hyperlinks hinterlegt, so dass man auf weitere Seiten gelangen kann. Beispielsweise ist der Text "Schnellimbiss" mit der Anzeige der Ergebnisse der Leistungsbewertung jener Funktionseinheit verknüpft.

### 6.5.2 Auswertung der Ergebnisse

Durch Auswahl des Menüpunkts "Kostenrechnung" gelangen wir auf die Webseite "Prozesskostenrechnung für Imbiss", an deren unterem Ende sich der in Abbildung 6.7 gezeigte Abschnitt mit Ergebnissen der Funktionseinheit *Sitzplaetze* befindet.

Die Ergebnisse werden in einer Tabelle dargestellt, in welcher auch die Formeln zur Berechnung angezeigt werden. Die Teilergebnisse sowie das Gesamtergebnis ergeben sich folgendermaßen:

Die Leistungskosten betragen 0 Geldeinheiten pro Zeiteinheit, da wir im Modell keine Leistungskosten angegeben hatten. Die Betriebskosten ergeben sich als Produkt der Kosten der Belegung der Sitzplätze von 3, die wir im Modell angegeben hatten, multipliziert mit der Belegung der Sitzplätze von 3,165, welche sich aus der Simulation ergibt. Somit betragen die Betriebskosten 9,495 Geldeinheiten pro Zeiteinheit. Die Fixkosten hatten wir im Modell mit 10.000 Geldeinheiten pro Zeiteinheit vorgegeben. Die Höhe der Gesamtkosten ergibt sich aus der Summe der Leistungskosten, Betriebskosten und Fixkosten in Höhe von 10.009,495 Geldeinheiten pro Zeiteinheit an, welche in der Exponentialdarstellung als 1.001e+04 angezeigt wird.

Die Formelzeichen werden in der Online-Hilfe erklärt. Weitergehende Informationen findet man in [7].

Abbildung 6.1: Innenansicht des Schnellimbisses nach der Änderung

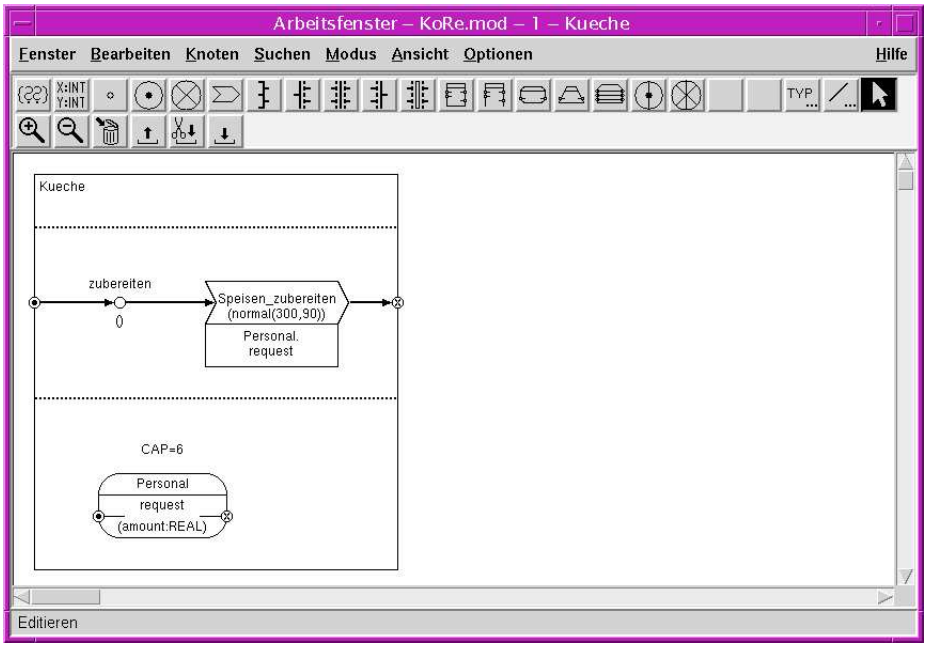



Abbildung 6.2: Ansicht der Küche nach der Änderung

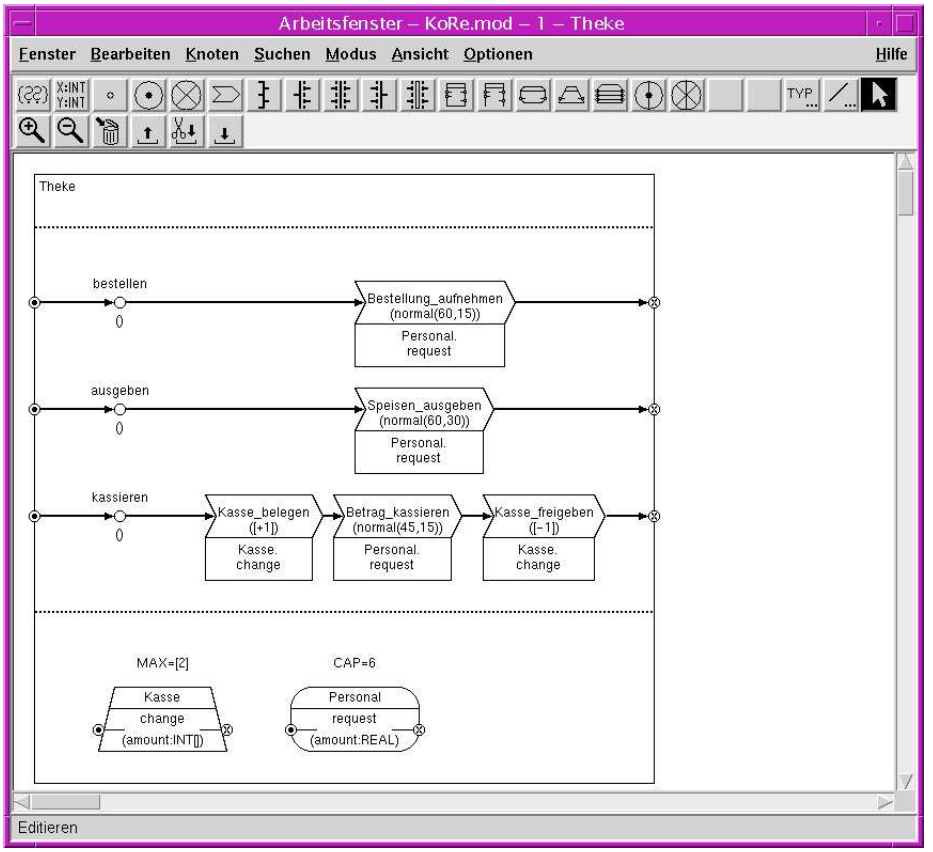



Abbildung 6.3: Ansicht der Theke nach der Änderung

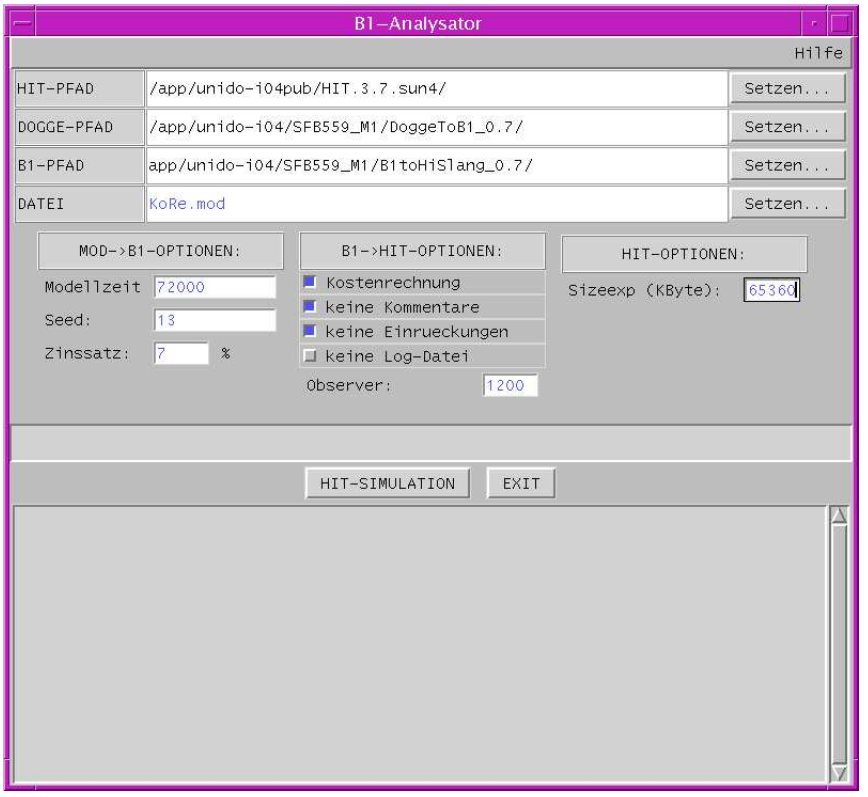



Abbildung 6.4: B1-Analysator mit den Optionen der Kostenrechnung


Abbildung 6.5: Einstellungen der Kostenvisualisierung

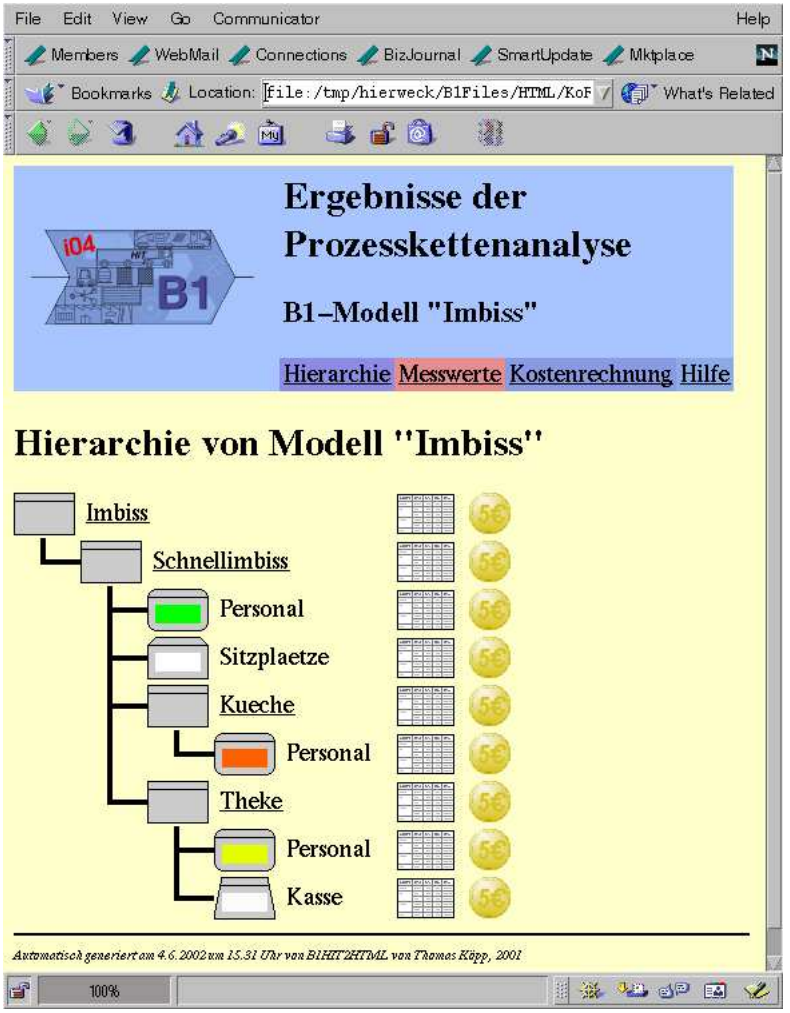



Abbildung 6.6: Webbrowser mit Überblick über die Hierarchie

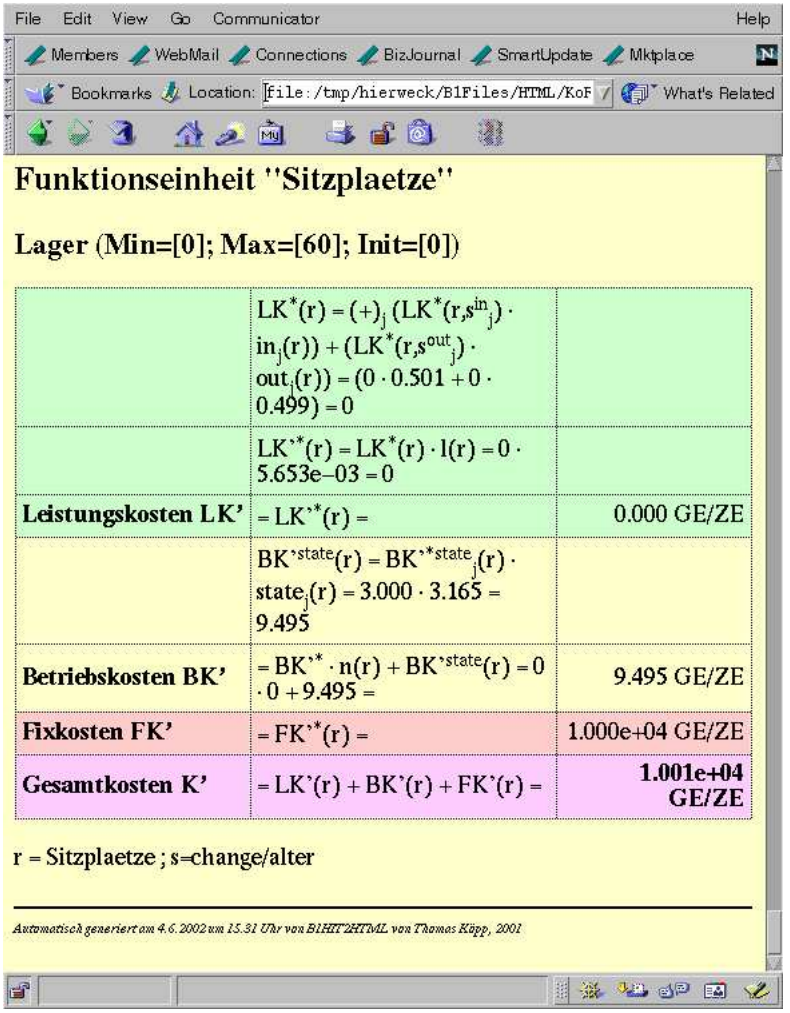



Abbildung 6.7: Webbrowser mit Kostenauswertung der Sitzplätze

# Anhang A

# Steady-State-Analyse

Viele Standard-Analyseverfahren für simulative Messreihen basieren auf der Berechnung des Mittelwertes und seinem Konfidenzintervall. Es existieren jedoch Simulationsmodelle, bei denen dieses Vorgehen zu grundsätzlich falschen Annahmen und Ergebnissen führt.

In den Software-Tools zur Steady-State-Analyse wird ein erweitertes Analyseverfahren durchgeführt. Es wird zunächst geprüft, ob ein stabiles Modell vorliegt. Durch diverse Methoden wird der Einfluss von dem Initialisierungszustand des Simulationsmodells auf die Ergebnisse eliminiert. Anschließend erfolgt die eigentliche statistische Auswertung und Ergebnisdarstellung.

Die Software-Tools wurde in der Programmiersprache C++ entwickelt und für das Betriebssystem UNIX optimiert. In den folgenden Abschnitten wird die Funktionsweise der Software-Tools zur Steady-State-Analyse näher erläutert. Es wird speziell darauf eingegangen, welche Voraussetzungen für eine effiziente Anwendung geschaffen werden müssen. Auf die konkrete Umsetzung wird nur kurz eingegangen. Im Vordergrund steht hier die Erläuterung der Modulstruktur der einzelnen Programme.

Die in diesem Kapitel beschriebenen Software-Tools können alle separat benutzt werden. Für den Benutzer ist es allerdings einfacher das Programm *ssa* ("steady-state" Analyse) zu verwenden. *ssa* ruft alle benötigten Dienstprogramme automatisch in geeigneter Reihenfolge auf und nimmt dem Benutzer hierdurch viel Arbeit ab. Um *ssa* aufzurufen, muss das zu analysierende HISLANG-Modell vorliegen und die im Abschnitt A.1.1 genannten Voraussetzungen müssen erfüllt sein.

## A.1 ssa: "steady-state" Analyse

Das Programm *ssa* ("steady-state" Analyse) ist ein Software-Tool zur Analyse von Simulationsdaten. Es wurde unter besonderer Berücksichtigung der Probleme entwickelt, die durch einen "untypischen" Initialisierungszustand entstehen. Die Analysealgorithmen sollen die Auswertungsschwierigkeiten umgehen, die in [6] im Kapitel 1.1 am Beispiel des Güterverkehrszentrums und in [6] im Kapitel 4.4 am Beispiel eines M/M/1-Servers dargestellt werden. Um dieses Ziel zu erreichen, werden einige der in [6] im Kapitel 6 vorgestellten Verfahren implementiert und geeignet miteinander kombiniert. Die Ideen zur Betrachtung der Zufallsverteilungen, die in [6] Kapitel 6.3.2 geschildert wurden, werden weiter verfolgt.

Bei der Auswahl der implementierten Analysemethoden wurde Wert darauf gelegt, dass sowohl ein langer Simulationslauf, als auch mehrere kurze Simulationsläufe analysiert werden können. Aus jeder Strukturgruppe der Analysemethoden (siehe Struktur in [6] Kapitel 6) wird mindestens ein Vertreter implementiert.

### A.1.1 Voraussetzungen

Das Programm *ssa* ist ein komplexes Software-Tool, da es parallele Simulationen durchführen kann und auf anderen Programmen basiert. Aus diesem Grund müssen einige Voraussetzungen geschaffen werden, damit ein effizienter Programmablauf gewährleistet ist.

Um *ssa* mitzuteilen, welche Rechner für die parallelen Simulationen zur Verfügung stehen, muss eine Datei mit diesen Informationen angelegt werden. Eine Zeile dieser Konstellationsdatei besteht aus drei Einträgen, die durch ein Leerzeichen voneinander getrennt sind. Der erste Eintrag ist der Rechnername. Dieser Name muss so gewählt sein, dass mit dem Standard-Tool "rsh" auf ihn zugegriffen werden kann. Der zweite Eintrag steht für die Geschwindigkeit des Rechners. Hier sollte die Taktfrequenz in MHz angegeben werden. Prinzipiell können hier auch beliebige andere Werte angegeben werden, entscheidend ist nur, dass das Verhältnis dieser Werte die Geschwindigkeitsunterschiede deutlich macht. Der dritte Eintrag steht für die Menge an Hauptspeicher in MB, die für die Simulationen auf dem Rechner zur Verfügung stehen soll. Als gutes Maß hat sich die Hälfte des gesamten Hauptspeichers erwiesen, denn es muss eingeplant werden, dass andere Benutzer den Rechner ebenfalls benutzen. Hier das Beispiel einer Konstellationsdatei, welche die Rechnerkonstellation beschreibt:

```
bruno 800 1024
gerd 400 64
christel 400 64
roland 333 64
astrid 300 64
bernd 143 64
elke 143 64
```

Der Zugriff auf die Dienstprogramme *steady*, *hit2steady* und *steady_hit* muss *ssa* ermöglicht werden. Auch dies geschieht über eine Datei. Diese Steuerdatei hat folgendes Format: In der ersten Zeile wird der Pfad und die Datei angegeben, welche die Rechnerkonstellation beinhaltet. In der zweiten, dritten und vierten Zeile wird die Position der Dienstprogramme *steady*, *hit2steady* und *steady_hit* angegeben. Das Beispiel einer Steuerdatei:

```
/app/unido-i04/SFB559_M1/SteadyStateAnalyse/rechner.txt
/app/unido-i04/SFB559_M1/SteadyStateAnalyse/steady
/app/unido-i04/SFB559_M1/SteadyStateAnalyse/hit2steady
/app/unido-i04/SFB559_M1/SteadyStateAnalyse/steady_hit
```

Diese Steuerdatei ermöglicht einen flexiblen Umgang mit den Dienstprogrammen. Das Verschieben der Dienstprogramme oder das Anbinden anderer Dienstprogramme mit ähnlicher Funktion wird hierdurch möglich. Eine einmal erstellte Steuerdatei muss nicht mehr geändert werden, solange keine Systemveränderungen vorliegen. Die Angabe der Rechnerkonstellation in einer gesonderten Datei ermöglicht es, dass jeder Benutzer seine eigene Systemkonfiguration benutzt.

Die externen Tools *xv* (Version 3.10a), *gnuplot* (Unix Version 3.7) und *octave* (Version 2.0.16) werden benötigt. Der Benutzer von *ssa* muss einen automatisierten Zugriff ermöglichen. Bei der Systemkonfiguration am Lehrstuhl IV des Fachbereichs Informatik der Universität Dortmund kann der Zugriff über die Module "extra/xv/3.10a", "extra/gnuplot/3.7" und "extra/octave/2.0.16" hergestellt werden.

Die parallelen Simulationen werden über die "Remote Shell" *rsh* realisiert. Es muss sichergestellt sein, dass der Rechner, auf dem *ssa* gestartet wurde, über *rsh* mit den Rechnern der Konstellationsdatei verbunden werden kann (siehe "man rsh"). Bei jedem internen Aufruf von *rsh* wird die "Loginshell" ausgeführt. Die Befehle der "Loginshell" können sehr zeitaufwendig sein, obwohl ihre Durchführung keine Auswirkungen hat. Daher sollte an einer geeigneten Stelle in der "Loginshell" folgende Zeile eingefügt werden:

```
if ($?prompt == 0) exit
```

Diese Anweisung verhindert die weitere Ausführung der "Loginshell", wenn durch *rsh* ein Befehl auf einem externen Rechner durchgeführt wird. In der Praxis hat sich gezeigt, dass dieses Vorgehen erhebliche Geschwindigkeitsvorteile mit sich bringt.

Da jede Verbindung über *rsh* zu einem anderen Rechner einen Teil des Hauptspeichers belegt, ist die Anzahl der parallelen Simulationen begrenzt. Erfahrungswerte haben gezeigt, dass 100 parallele Simulationen von einem Rechner mit 256MB Hauptspeicher (zum Beispiel "siegmar") durchgeführt werden können. Da die graphischen Benutzeroberflächen (zum Beispiel "Solaris") ebenfalls Hauptspeicher belegen, ist es sinnvoll, aber nicht notwendig, dass der Rechner, von dem die Simulationen gestartet werden, per "Remote" benutzt wird.

Wenn der Rechner, auf dem *ssa* gestartet wird, per "Remote" benutzt wird, ist zu beachten, dass die Bildschirmausgabe umgeleitet werden muss. Die Umleitung kann durch die üblichen Befehle erfolgen. Wenn der Arbeitsplatzrechner "astrid" heißt, und der "Remote"-Rechner "siegmar", muss auf "astrid" der Befehl

```
xhost +siegmar
```

und danach auf "siegmar" der Befehl

```
setenv DISPLAY astrid:0
```

eingegeben werden.

Eine weitere wichtige Voraussetzung ist, dass das Programm *ssa* in dem Verzeichnis gestartet werden muss, in dem sich die zu analysierende HISLANG-Datei befindet. Da im Laufe der Analyse ein Ergebnisverzeichnis und mehrere temporäre Dateien angelegt werden, ist es sinnvoll, ein gesondertes Arbeitsverzeichnis zu erzeugen. In dieses Arbeitsverzeichnis muss die zu analysierende HISLANG-Datei kopiert werden. Ein Aufruf von *ssa* kann nun innerhalb dieses Arbeitsverzeichnisses erfolgen.

### A.1.2 Aufruf und Ablauf

Wenn alle Voraussetzungen beachtet wurden, ist der Aufruf von *ssa* einfach:

```
PFAD/ssa/ STEUERDATEI
```

Zu Beginn des Programms *ssa* muss der Benutzer einige Experimentparameter festlegen. Als Erstes kann er den Startseed festlegen. Werden parallele Simulationen durchgeführt, ist der Startseed der Beginn der Zufallsperiode. Als Nächstes muss das zu analysierende HISLANG-Modell festgelegt werden. Hierbei ist zu beachten, dass nur die HISLANG-Modelle gewählt werden können, die sich im aktuellen Arbeitsverzeichnis befinden. Ist das HISLANG-Modell gewählt, macht *ssa* automatisch Vorschläge, welche Modellkomponenten betrachtet werden können. Durch "(j)a" und "(n)ein" kann hier eine Auswahl getroffen werden. Als Nächstes legt der Benutzer den Modellzeitabstand fest, an dem Simulationsdaten aus den Simulationsereignissen berechnet werden sollen. Dieser Modellzeitabstand entspricht der Intervallgröße des Filterprogramms *steady*.

Nun müssen die Auswertungsalgorithmen festgelegt werden. Zunächst wird ein Verfahren festgelegt, mit dem der Anfang der stationären Phase bestimmt werden soll. Je nach gewähltem Verfahren, müssen unterschiedliche Auswertungsparameter angegeben werden. Als Nächstes wird festgelegt, mit welchem statistischen Verfahren die stationäre Phase ausgewertet werden soll. Auch hierbei sind für unterschiedliche Verfahren unterschiedliche Auswertungsparameter anzugeben.

Während des Programmablaufes wird im aktuellen Arbeitsverzeichnis ein Ergebnisverzeichnis angelegt. In diesem Verzeichnis werden die Abbildungen gespeichert, die während der Analyse erzeugt werden. Des Weiteren wird der Benutzer über jedes Ergebnis bei jedem durchgeführten Test über eine Bildschirmausgabe informiert. Wird eine stationäre Phase erkannt, wird ihr Beginn (Modellzeit) auf dem Bildschirm ausgegeben und es werden die statistischen Auswertungsmethoden der stationären Phase angewendet. Auch hier wird der Benutzer jedesmal informiert, wenn sich durch weitere Messwerte ein neuer Mittelwert und ein neues Konfidenzintervall ergibt. So steht am Ende der Programmausführung das Analyseergebnis in der Bildschirmausgabe.

Bei der Analyse der gewählten HISLANG-Datei wird die Strategie aus dem Kapitel Vorgehen verfolgt. Ausgewählte Verfahren wurden implementiert. Bei der Auswahl der Verfahren wurde darauf geachtet, dass mindestens ein Vertreter jeder Gruppe (siehe Struktur in [6] Kapitel 6) realisiert wurde.

Die Abbildung A.1 zeigt das Ablaufdiagramm von *ssa*. Das Diagramm zeigt, an welcher Stelle des Programmablaufs die einzelnen Dienstprogramme automatisch ausgeführt werden. Der Benutzer muss nur das Programm *ssa* aufrufen, alle weiteren Aufrufe werden automatisch durch *ssa* durchgeführt. Des Weiteren ist zu erkennen, in welcher Reihenfolge die Analysemethoden angewendet werden. Die aufgeführten Kapitelnummern verweisen auf die entsprechenden Kapitel in dieser Diplomarbeit. Es ist noch anzumerken, dass zusätzlich zur abschließenden Ergebnisdarstellung auch während der Erkennung und Auswertung der stationären Phase die bisherigen Ergebnisse durch geeignete Diagramme dargestellt werden.

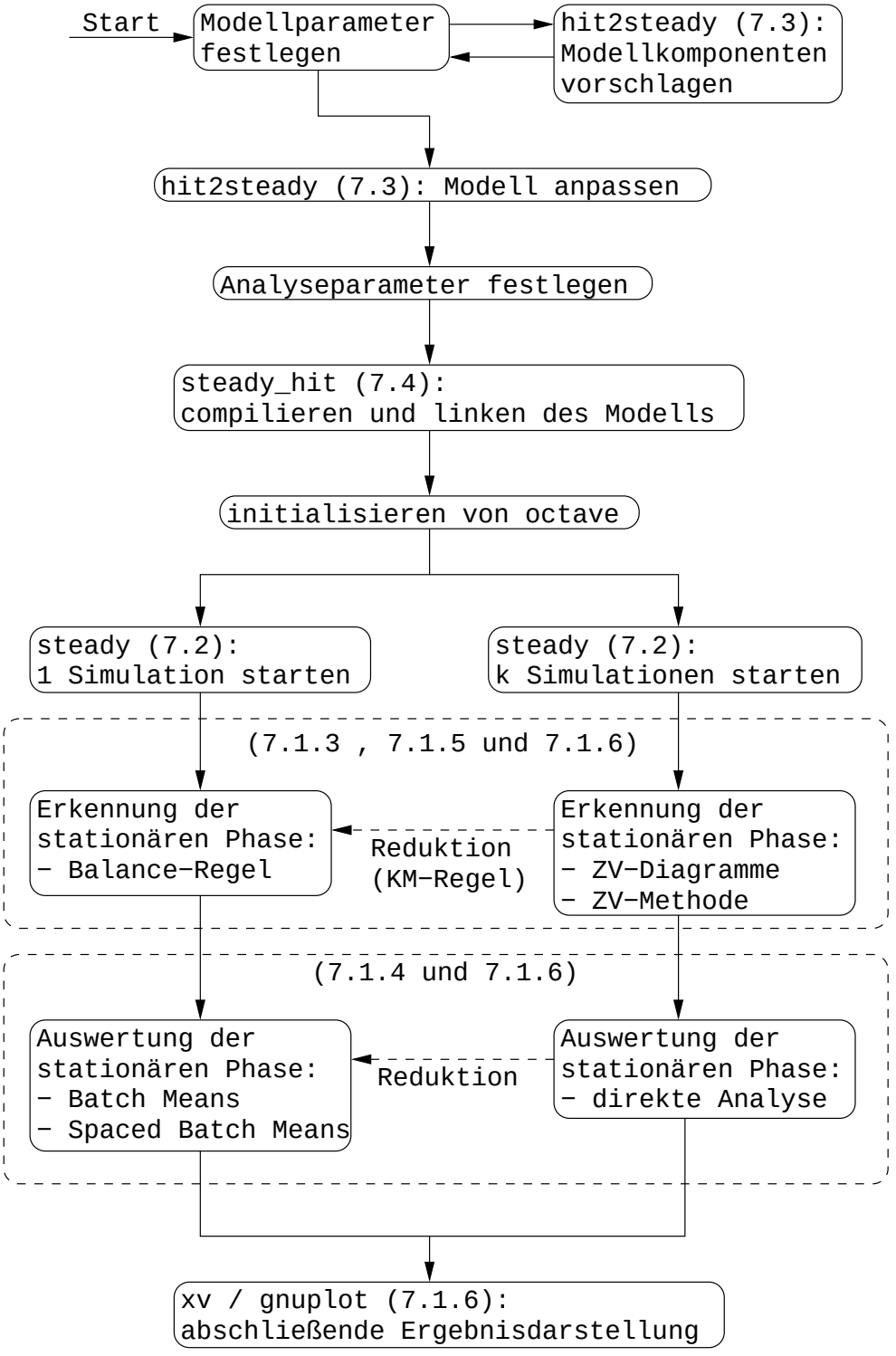



Abbildung A.1: Das Ablaufdiagramm des Analyse-Tools *ssa* mit den automatisierten Aufrufen der Dienstprogramme *hit2steady*, *octave*, *steady_ hit*, *steady*, *xv* und *gnuplot*.

### A.1.3 Erkennung der stationären Phase

Zur Erkennung der stationären Phase wurde die "Balance-Regel" (siehe [6] Kapitel 6.2.1), die Zufallsverteilungsdiagramme (siehe [6] Kapitel 6.3.2), die Zufallsverteilungsmethode (siehe [6] Kapitel 6.3.2) und die "kumulativer-Mittelwert-Regel" siehe [6] Kapitel(6.3.1) realisiert. Die folgenden Abschnitte beschreiben die genaue Umsetzung und welche Auswertungsparameter der Benutzer wählen kann.

#### Zufallsverteilungsdiagramme

Der Benutzer kann die horizontale Auflösung der Zufallsverteilungsdiagramme wählen. Hierbei ist zu beachten, dass pro Bildzeile ungefähr fünf Simulationen benötigt werden, um ein aussagekräftiges Diagramm zu erhalten. Da hundert parallele Simulationen eine realistische Größenordnung sind, entstehen recht kleine Diagramme mit einer horizontalen Auflösung von 20 Pixeln. Das verwendete Darstellungsprogramm "xv" ermöglicht jedoch eine vergrößerte Darstellung. Die Zufallsverteilungsdiagramme, die durch *ssa* erzeugt werden, wirken hierdurch im Vergleich zu den Zufallsverteilungsdiagrammen, die durch *visual* erzeugt werden, relativ grob. Sie ermöglichen allerdings trotz dieser Schwierigkeit die Erkennung der stationären Phase.

Nach dem Start der parallelen Simulationen werden die übermittelten Daten sequenziell durch die Zufallsverteilungsmethode analysiert. Hiervon unabhängig kann der Benutzer jederzeit die Analyse durch einfachen Tastendruck unterbrechen. Nach dem Tastendruck werden die Verteilungsdiagramme der vier Leistungsmaße Ankunftsrate, Abgangsrate, Population und Verweilzeit erzeugt und mittels "xv" dargestellt. Die Ausgabe erfolgt sortiert nach Modellkomponenten und Diensten. Eine direkte Bearbeitung der Diagramme ist möglich. Der Benutzer kann die Darstellungsgrenzen der relativen Häufigkeit festlegen und dadurch bestimmte Darstellungsbereiche hervorheben. Die veränderte Darstellung wird automatisch im Ergebnisverzeichnis gespeichert. Zu beachten ist noch, dass zu jedem Leistungsmaß eines Dienstes nur jeweils das letzte Diagramm gespeichert wird, da sonst die große Datenmenge schnell an die Systemgrenze stoßen könnte.

#### Zufallsverteilungsmethode

Zunächst legt der Benutzer die Anzahl der parallelen Simulationen fest. Das Verhältnis zwischen verworfenen Probanden und Beobachtungsintervall (siehe [6] Kapitel 6.3.2) kann der Benutzer ebenfalls festlegen. Der dritte Parameter, den der Benutzer festlegt, bezieht sich auf den Vergleich zweier Stichproben. Hier kann der Benutzer durch einen Grenzwert festlegen, bei wieviel "Unterschied" zwei Stichproben noch als "gleichverteilt" gelten.

Nach dem Start der parallelen Simulationen werden schrittweise neue Simulationsdaten ermittelt. Der Proband wird mit dem Beobachtungsintervall verglichen, wie es in [6] im Kapitel 6.3.2 beschrieben ist. Zum Vergleich zweier Stichproben können mehrere statistische Verfahren benutzt werden. Diese Verfahren sind im Kapitel Gleichheit beschrieben. Bei der Implementierung von *ssa* wurden der $\chi^2$-Test, der Kolmogoroff-Smirnoff-Test, der Mann-Whitney U-Test und der Wilcoxon Signed-Rank-Test getestet. Für einfache Modelle lieferten sie alle befriedigende Ergebnisse. Bei größeren Modellen war die Laufzeit des Algorithmus jedoch nicht mehr akzeptabel. Dies liegt nicht an einer ineffizienten Realisierung der Tests in dem Statistik-Tool *octave*, sondern an der Kommunikation zwischen *octave* und *ssa*. Sehr viele

Daten müssen zwischen den eigenständigen Programmen hin und her transportiert werden. Auf Grund dieser Schwierigkeiten wurde in *ssa* der einfache aber schnelle Differenz-Test (siehe [6] Kapitel 3.6.5) implementiert.

#### Balance-Regel

Die "Balance-Regel" basiert auf einem einzelnen Simulationslauf, daher sind mehrere parallele Simulationen nicht nötig. Im Gegensatz zu dem Vorgehen in [6] Kapitel 6.2.1 hat bei dieser Implementierung die Datenmenge zur Bestimmung des Mittelwertes nicht dieselbe Länge wie die Datenmenge, die auf "Balance" getestet wird. Dieses Vorgehen verspricht einen "genaueren" Mittelwert und ist ein Kompromiss aus der "Balance-Regel" und der "Balance-$\chi^2$-Regel". Der Benutzer kann also beide Parameter getrennt wählen. Die Anzahl der zu verwerfenden Stichproben ist von dem Benutzer ebenfalls frei wählbar. Hierdurch kann eine beliebige Genauigkeit des Verfahrens erreicht werden. Es ist jedoch zu beachten, dass eine hohe Genauigkeit mit einer langen Laufzeit verbunden ist. Durch einen weiteren Parameter kann der Benutzer bestimmen, ab wann eine Stichprobenmenge als "balanciert" gilt. Durch einen Grenzwert wird der erlaubte Bereich der Abweichung der beiden Balance-Werte festgelegt.

#### Kumulativer-Mittelwert-Regel

Die Reduktion von mehreren Simulationsläufen auf einen stellvertretenden Simulationslauf wird ebenfalls berücksichtigt. Bei dieser Reduktionsmethode muss die Anzahl der parallelen Simulationen durch den Benutzer festgelegt werden. Um den Programmieraufwand gering zuhalten, wird auf den stellvertretenden Simulationslauf die "Balance-Regel" angewendet. Der Benutzer wählt die entsprechenden Auswertungsparameter für die "Balance-Regel", wie zuvor beschrieben.

### A.1.4 Auswertung der stationären Phase

Zur statistischen Auswertung der stationären Phase wurden in dem Programm *ssa* drei Verfahren implementiert. Zum einen wurde die "direkte Analyse" (siehe [6], Kapitel 6.3.3) umgesetzt. Dieses Verfahren entspricht dem "Replication/Deletion"-Vorgehen, das von Law und Kelton in [4] vorgeschlagen wird. Zum anderen wurde das Verfahren "Batch Means" und die Variante "Spaced Batch Means" (siehe [6], Kapitel 6.2.2) implementiert. Die Reduktion von mehreren Simulationen auf einen stellvertretenden Simulationslauf wurde ebenfalls berücksichtigt.

#### Direkte Analyse

Bei der "direkten Analyse" der parallelen Simulationen kann der Benutzer einen Grenzwert für den Quotienten aus dem Mittelwert und dem 95%-Konfidenzintervall wählen, bei dem die Simulation abgebrochen werden soll. Als Konfidenzintervall wurde das Standardkonfidenzintervall (siehe [6], Kapitel 3.3.8) gewählt. Sollte der Quotient aus dem Mittelwert und dem 95%-Konfidenzintervall den Grenzwert überschreiten, so wird die Simulation fortgesetzt.

#### Batch Means

Bei dieser Implementierung des Verfahrens "Batch Means" wird mit der Gruppengröße eins gestartet. Diese Gruppengröße wird solange um eins erhöht, bis der "Run-Test" des Statistik-Tools "octave" die Unabhängigkeit der Daten erkennt. Mit diesen Gruppen wird dann so verfahren, wie es in [6] Kapitel 6.2.2 beschrieben ist. Der Benutzer hat auch hier die Möglichkeit, einen Grenzwert für den Quotienten aus dem Mittelwert und dem 95%-Konfidenzintervall zu wählen. Außerdem kann der Benutzer den Erfolg des "Run-Test" durch einen weiteren Grenzwert festlegen.

#### Spaced Batch Means

Die Implementierung von "Spaced Batch Means" ist an "Batch Means" angelehnt. Eine Gruppe besteht immer nur aus einem Wert. Es wird lediglich die Anzahl an nicht betrachteten Werten zwischen den Gruppen variiert. Auch hierbei testet der "Run-Test" des Statistik-Tools "octave" die Daten auf Unabhängigkeit. Daher legt der Benutzer dieselben Parameter fest, wie bei der Auswertung durch "Batch Means".

#### Reduktion

Bei der Reduktion auf eine stellvertretende Datenmenge werden keine gesonderten Parameter benötigt. Auf die stellvertretende Datenmenge kann entweder das "Batch Means"-Verfahren oder das "Spaced Batch Means"-Verfahren angewendet werden.

### A.1.5 Verkehrsflussgleichgewicht

Da das Dienstprogramm *steady* die Leistungsmaße Ankunftsrate und Abgangsrate liefert, kann überprüft werden, ob Verkehrsflussgleichgewicht (siehe [1], Kapitel 4) besteht. Das Prinzip des Verkehrsflussgleichgewichts ist eine notwendige Bedingung für Stationarität. Wenn sich die Ankunftsrate und die Abgangsrate mit steigender Simulationszeit nicht annähern, kann das zu analysierende Modell keine stationäre Phase haben. Leider lässt sich hierdurch noch keine Abbruchbedingung für die Simulation ableiten. Wenn eine bestimmte Simulationszeit betrachtet wurde, in der sich kein Verkehrsflussgleichgewicht eingestellt hat, kann noch nicht angenommen werden, dass das Modell keine stationäre Phase besitzt. Die stationäre Phase könnte in der weiteren Simulation noch eintreten. Das Prinzip des Verkehrsflussgleichgewichts kann also nur, wie schon erwähnt, als eine notwendige Bedingung für Stationarität betrachtet werden.

### A.1.6 Ergebnisdarstellung

Damit der Benutzer von *ssa* über den Verlauf der Tests und Analysen informiert ist, werden die gewonnenen Ergebnisse graphisch dargestellt. Anhand dieser Darstellungen kann der Auswertungsverlauf verfolgt werden. Der Benutzer erkennt, ob die Auswertungsparameter geeignet gesetzt wurden.

#### Ankunftsrate versus Abgangsrate

Die Abgangsrate und die Ankunftsrate werden von *ssa* miteinander verglichen. Die Abbildung A.2 zeigt den zeitlichen Verlauf der Vergleiche. Aufgetragen ist sowohl

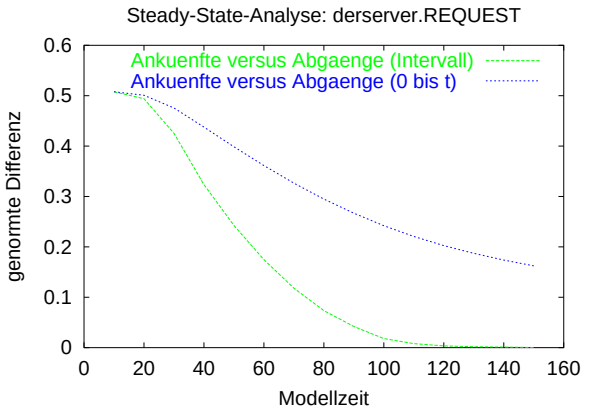



Abbildung A.2: Genormte Differenz der Ankunftsrate und Abgangsrate.

der Vergleich der jeweiligen Intervalle, als auch der Mittelwert aller betrachteten Vergleiche.

Ein Vergleich zwischen der Ankunftsrate $a$ und der Abgangsrate $c$ findet durch Differenzbildung statt:

$$e = \frac{a - c}{c} \tag{A.1}$$

$e$ sei die genormte Differenz. Sollte für die Abgangsrate $c = 0$ gelten, wird durch die Ankunftsrate $a$ dividiert. Sollte auch die Ankunftsrate $a = 0$ sein, so ist $e = 0$. Durch diese etwas umständliche Berechnung wird eine Division durch Null vermieden.

Eine stationäre Phase kann nur erreicht werden, wenn die genormte Differenz $e$ klein ist. Nur in diesem Fall kann das Verkehrsflussgleichgewicht eintreten. Ist die genormte Differenz $e$ konstant auf einem hohen Niveau, ist das betrachtete Modell nicht stabil.

### Vergleichsergebnisse der Zufallsverteilungsmethode

Bei der Zufallsverteilungsmethode wird eine ausgewählte Stichprobe mit ihren Nachfolgern verglichen. Der zeitliche Verlauf dieser Vergleiche ist in Abbildung A.3 aufgetragen. Die eine Kurve repräsentiert die Vergleiche der aktuellen Stichprobe (Proband) mit ihren nachfolgenden Stichproben. Die andere Kurve zeigt den Mittelwert aller Vergleiche der bisherigen Probanden. Fällt die erste Kurve auf ein niedriges Niveau, kann angenommen werden, dass eine stationäre Phase vorliegt. Bleibt die Kurve konstant auf einem hohen Niveau, sind die aufeinander folgenden Zufallsverteilungen unterschiedlich und es liegt keine stationäre Phase vor.

### Stationärer Mittelwert verglichen mit Gesamt-Mittelwert

Die Abbildung A.4 besteht aus drei Kurven. Die erste Kurve ist das eigentliche Leistungsmaß der einzelnen Intervalle. Die beiden anderen Kurven sind Mittelwerte dieser Leistungsmaße. Die eine Kurve beschreibt den Mittelwert von der Modellzeit Null bis zur aktuellen Modellzeit (Gesamt-Mittelwert). Die andere Kurve zeigt den Mittelwert beginnend mit dem Anfang der stationären Phase bis zum aktuellen Modellzeitpunkt (stationärer Mittelwert). Da diese drei Kurven in einem Diagramm aufgetragen werden, lässt sich der Vorteil, bedingt durch das Verwerfen der Daten der transienten Phase, direkt ablesen.

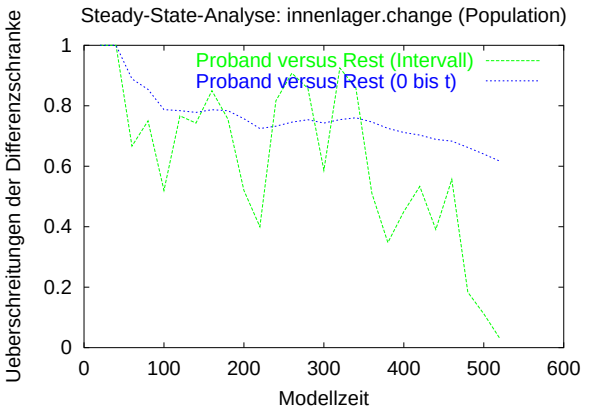



Abbildung A.3: Der zeitliche Verlauf der Vergleiche der aktuellen Stichprobe mit den nachfolgenden Stichproben.

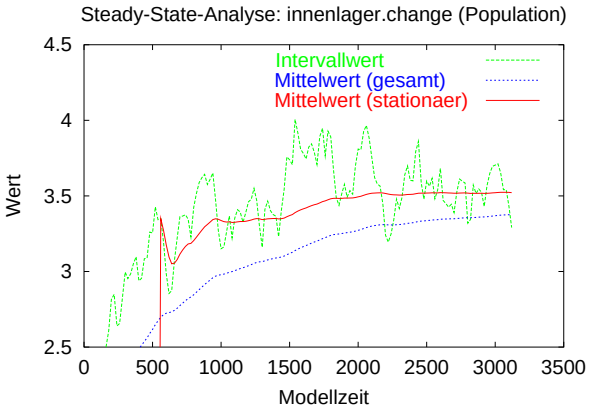



Abbildung A.4: Stationärer Mittelwert versus gesamter Mittelwert.

#### Stationäre Phase

Die übliche Darstellung von Leistungsmaßen, die durch Simulation bestimmt werden, wird auch durch *ssa* unterstützt. Hier wird jedoch die transiente Phase komplett außer Acht gelassen, so dass die Darstellung mit dem Beginn der stationären Phase beginnt. Aufgetragen ist der Mittelwert mit seinem Konfidenzintervall und der Standardabweichung. Der horizontale Darstellungsbereich lässt sich beliebig wählen. Die Abbildung A.5 ist ein Beispiel dieser Darstellungsform.

### A.1.7 Parallelisierung

Die parallelen Simulationsläufe werden entsprechend der Rechnerkonstellation und der Rechnergeschwindigkeiten auf die Rechner verteilt. So entsteht eine sternförmige Struktur der Simulationen mit dem Programm *ssa* im Mittelpunkt.

Die technische Umsetzung erfolgte mit Hilfe der Bibliothek “unistd”. Der Aufruf der parallelen Simulationen erfolgt durch Abspalten eines Prozesses durch die Funkti-

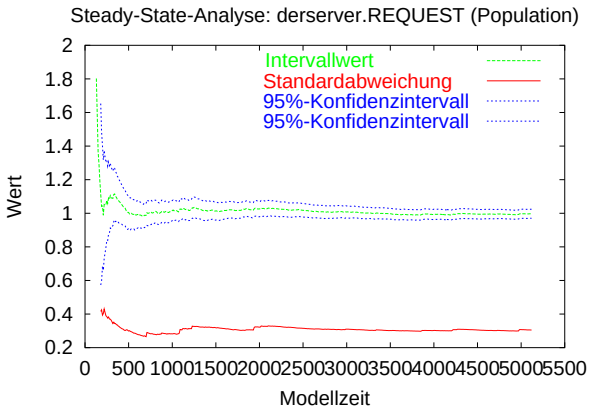



Abbildung A.5: Mittelwert und Konfidenzintervall.

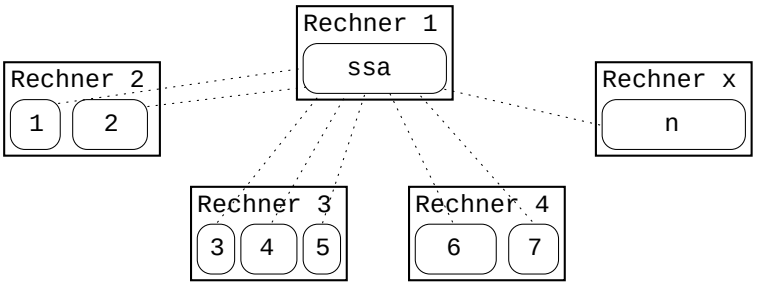



Abbildung A.6: Die $n$ Simulationen werden auf $x$ Rechner verteilt.

on "vfork" und anschließendem Systemaufruf durch "execlp". Die Einbindung von "Remote"-Rechnern erfolgt durch die "Remote Shell" *rsh*.

Für die Kommunikation zwischen den Simulationen und dem Programm *ssa* wurden sogenannte "Pipes" angelegt. Die Simulationen benutzen das eine Ende der "Pipe" zum Schreiben ihrer Simulationsdaten. Dadurch kann das Programm *ssa* die Daten am anderen Ende der "Pipe" empfangen. Da das Filterprogramm *steady* die Simulationsdaten normalerweise an den Standardausgabekanal leitet, muss ein kleiner Trick angewendet werden. Mit der Funktion "dub2" kann die Ausgabe an den Standardausgabekanal in die vorgesehene "Pipe" umgeleitet werden.

An jede "Pipe" ist ein Datenpuffer angeschlossen. Dieser Datenpuffer sorgt dafür, dass die geschriebenen Daten nicht direkt gelesen werden müssen. Wenn dieser Datenpuffer jedoch an seine Kapazitätsgrenze stößt, wird der Prozess blockiert, der die Daten in die "Pipe" schreiben will. Ist der Datenpuffer hingegen leer, wird der Prozess blockiert, der aus der "Pipe" lesen will. Daher hat die Verwendung der "Pipes" den Vorteil, dass die Synchronisation der Simulationen mit dem Programm *ssa* automatisch geschieht. Der schnellere Prozess wartet auf den langsameren Prozess.

Neben dieser automatischen Prozesssteuerung kann *ssa* die Simulationen über Signale steuern. Dazu muss *ssa* lediglich die Prozess-ID des Filterprogramms *steady* und der zugehörigen Simulation bekannt sein. Die Prozess-IDs können durch die Funktion "getid" erfragt werden. Bei dem Senden der Signale hilft die Funktion "kill" aus der Bibliothek "signal". Die nützlichsten Signale sind

**SIGKILL:** zum Beenden eines Prozesses,

**SIGSTOP:** zum Anhalten eines Prozesses,

**SIGCONT:** zum Fortsetzen eines Prozesses.

Sollte das Programm *ssa* aus irgendeinem Grund vorzeitig abgebrochen werden, werden zuerst alle gestarteten Simulationen beendet. Dies verhindert, dass nach Beendigung von *ssa* Simulationsprozesse weiterarbeiten, obwohl sie nicht benötigt werden. "Signal-Handler" stellen diese Funktion sicher.

### A.1.8 Implementierung

Das Programm *ssa* ist durch "Prototyping" entstanden. Das bedeutet, dass zuerst eine Version von *ssa* entwickelt wurde, die nur wenige Funktionen besaß. In weiteren Versionen wurde die Funktionalität erweitert. So ergaben sich Methoden, die semantisch zusammen gehören. Diese Methoden wurden nach und nach ausgelagert. So entstand die Modulstruktur von *ssa*. Die Module haben folgende Bedeutung:

**main:** Steuerung

**transient:** Erkennung der stationären Phase

**analyse:** Statistischen Auswertung der stationären Phase

**daten:** Datenstruktur für Simulationsdaten

**rechner:** Datenstruktur für die Rechnerkonstellation

**job:** Datenstruktur für Simulationsläufe

**darstellung:** Erzeugung von Abbildungen

**bmp:** Erzeugung von Dateien im BMP-Format

**octave:** Start und Kommunikation mit "octave"

**tools:** Sonstiges

Ein Schwerpunkt der Implementierung ist die Datenstruktur für die Simulationsdaten. Sie ist sortiert nach Komponenten, Diensten und Leistungsgrößen. Eine Leistungsgröße besteht wiederum aus vielen Modellzeitpunkten, die eine Stichprobe repräsentieren. Um eine hohe Effizienz zu garantieren, wurden doppelt verkettete Listen (Bibliothek "list") und binäre Suchbäume (Bibliothek "map") verwendet. Bei langen Simulationen wird die Datenstruktur sehr groß, sodass Effizienz für eine kurze Laufzeit sehr wichtig ist.

Eine große Anzahl an weiteren Standardbibliotheken wurde verwendet, auf eine Aufzählung und eine Beschreibung wird hier jedoch verzichtet.

## A.2 steady

Das Programm *steady* kann die Ereignisse eines Simulationslaufes interpretieren und die Leistungsgrößen Ankunftsrate, Abgangsrate, Verweilzeit und Population berechnen. Die Ausgabe der Ereignisse der Simulation muss in dem Format erfolgen, das in [3] Anhang G.5.1.2 beschrieben ist. Das Programm *steady* ist ein Filter, der aus den Ereignissen einer Simulation geeignete Leistungsmaße bestimmt. Der Programmablauf von *steady* findet parallel zur Simulation statt und die Leistungsmaße werden daher ebenfalls parallel zur Simulationszeit berechnet und ausgegeben.

### A.2.1 Aufruf und Ergebnisse

Der direkte Aufruf von *steady* ist kompliziert, da das Programm *steady* nur als ein Dienstprogramm für *ssa* gedacht ist. Der Benutzer muss zum einen sicherstellen, dass die zu analysierende HISLANG-Datei mit dem Programm *hit2steady* bearbeitet wurde. Des Weiteren muss die Messobjektdatei (siehe A.3) vorhanden sein und den Namen der zu analysierenden HISLANG-Datei besitzen, ergänzt um den Anhang ".steady.obj". Die durch *hit2steady* erzeugte HISLANG-Datei muss mit *steady_hit* in eine ausführbare Datei umgewandelt werden. Alle Dateien, die erzeugt wurden, müssen sich in einem Verzeichnis befinden. Sind diese Voraussetzungen geschaffen, kann *steady* manuell mit dieser Aufrufsyntax gestartet werden:

```
PFAD/steady HISLANGDatei Intervall Ausgaben Speicherbereich
```

Mit "HISLANGDatei" ist der Dateiname der ursprünglichen HISLANG-Datei gemeint. "Intervall" ist die Schrittweite in der Modellzeit, nach der eine Ausgabe der Leistungsmaße erfolgt. Durch "Ausgaben" kann die Anzahl der Ausgaben der Leistungsmaße festgelegt werden. Wird dieser Übergabeparameter auf "$-1$" gesetzt, erfolgt eine unbegrenzte Ausgabe. Der Parameter "Speicherbereich" legt die Größe des Hauptspeichers fest, die der Simulation zur Verfügung gestellt wird (siehe A.4).

Zu Beginn der Ausführung werden die Prozess-IDs des "steady"-Prozesses und des "steady.hitcode"-Prozesses an den Standardausgabekanal geliefert. Der "steady.hitcode"-Prozess entsteht durch den Simulationsstart (siehe A.4). Mit Hilfe dieser Prozess-IDs kann das Programm *ssa* die Ausführung anhalten, fortsetzen oder beenden. Alle berechneten Leistungsmaße werden ebenfalls an den Standardausgabekanal geliefert. Die Ausgabe erfolgt in einer festen Form:

- Eine Ausgabezeile besteht aus der aktuellen Modellzeit, dem Komponentennamen, dem Dienstnamen und den vier Leistungsmaßen Ankunftsrate, Abgangsrate, Verweilzeit und Population. Alle Werte sind durch ein Leerzeichen voneinander getrennt.
- Für jeden Dienst einer Komponente wird eine dieser Zeilen ausgegeben.
- Durch den Übergabeparameter "Intervall" wird festgelegt, zu welchen Modellzeiten eine Ausgabe erfolgt.
- Die Ausgaben der unterschiedlichen Modellzeiten werden durch eine Leerzeile getrennt.

Das nachfolgende Beispiel ist die Ausgabe einer Simulation, bei der nur eine Komponente betrachtet wurde:

```
5 derserver REQUEST 21 2.2 93.5785 98.6922
10 derserver REQUEST 1.2 1.2 91.4399 94.7961
15 derserver REQUEST 1 1.8 88.7344 92.6599
20 derserver REQUEST 1.2 2 82.4731 88.6372
25 derserver REQUEST 0.8 2 78.1499 81.923
30 derserver REQUEST 1.4 1.2 77.8613 80.5881
35 derserver REQUEST 1.2 2.2 73.2195 77.3992
40 derserver REQUEST 1.4 1.4 72.6554 75.0551
45 derserver REQUEST 0.2 2.4 65.5841 69.4173
50 derserver REQUEST 1.2 1 63.8301 65.6739
55 derserver REQUEST 0.6 1.6 60.142 63.7609
60 derserver REQUEST 1.2 2.6 52.0813 57.5621
```

```
65 derserver REQUEST 1.2 1.6 49.4571 51.6168
70 derserver REQUEST 0.6 2.2 42.8613 47.0197
75 derserver REQUEST 1.2 1.2 41.0098 43.8736
80 derserver REQUEST 0.4 2 35.6201 39.5753
85 derserver REQUEST 1.2 2.4 27.4399 32.0767
90 derserver REQUEST 1 2.6 20.6134 25.7083
95 derserver REQUEST 0.8 3.8 5.98514 13.7242
100 derserver REQUEST 0.8 2 2.02523 4.18755
```

Das Format ist sehr einfach und es müssen keine großen Änderungen vorgenommen werden, wenn eine weitere Leistungsgröße ergänzt werden soll.

### A.2.2 Leistungsgrößen

Auf die Bestimmung von Leistungsmaßen wurde schon in [6] Kapitel 4.1 eingegangen. In diesem Abschnitt soll die konkrete Realisierung in dem Programm *steady* beschrieben werden.

In dem Programm *steady* basiert die Berechnung der Leistungsmaße auf disjunkten Intervallen. Diese Intervalle werden herangezogen, um die Ankunftsrate, die Abgangsrate, die Verweilzeit und die Population zu berechnen. Diese vier Leistungsmaße sind klar definierte Begriffe (siehe [1]). Ihre Berechnung aus den Ereignissen der jeweiligen Intervalle wird in den folgenden Abschnitten geschildert.

Ein HISLANG-Modell besteht aus hierarchisch strukturierten Komponenten. Jede dieser Komponenten besteht aus einer "Entry-Area", einer "Service-Area" und einer "Exit-Area" (siehe [3]). Diese Bereiche haben eine unterschiedliche Intention. In dieser Diplomarbeit werden diese Bereiche jedoch außer Acht gelassen. Es werden nur die Ereignisse "Entry" und "Exit" berücksichtigt.

#### Ankunftsrate

Die Ankunftsrate $a$ beschreibt, wieviele "jobs" (Definition siehe [1]) pro Zeiteinheit eine Modellkomponente betreten. Das Betreten einer Modellkomponente geschieht durch das Ereignis "Entry". Zur Bestimmung der Ankunftsrate $a$ wird ein Zähler $A$ benötigt, der zu Beginn eines Intervalls auf Null gesetzt wird

$$A = 0 \quad \text{(zum Intervallbeginn)} \tag{A.2}$$

und jedes "Entry"-Ereignis des Intervalls zählt:

$$A := A + 1 \quad \text{für Ereignis "Entry".} \tag{A.3}$$

Ist das Ende des Intervalls erreicht, ergibt die Division dieses Zählers durch die Modellzeitlänge $T$ des Intervalls die Ankunftsrate:

$$a = \frac{A}{T} \tag{A.4}$$

Da die Länge der Intervalle immer größer als Null ist, ist eine Division durch Null unmöglich.

### Abgangsrate

Die Abgangsrate $c$ wird analog zur Ankunftsrate $a$ berechnet. Das grundlegende Ereignis ist in diesem Fall ein "Exit".

### Verweilzeit

Mit der Verweilzeit ist hier die kumulative Verweilzeit $J$ gemeint. Eine genaue Definition ist in [1] nachzulesen. Zur Berechnung der kumulativen Verweilzeit $J$ eines

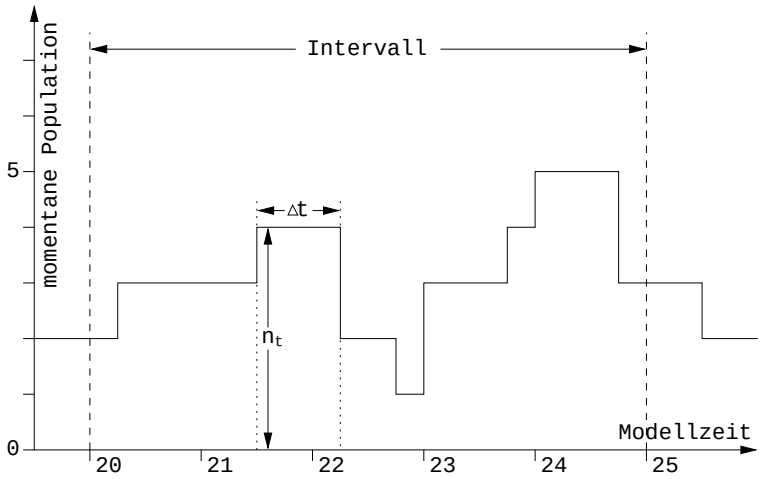



Abbildung A.7: Berechnung der kumulativen Verweilzeit eines Intervalls aus den konstanten Teilintervallen.

Intervalles muss seine Zustandstrajektorie (siehe [1]) betrachtet werden. Aus der Zustandstrajektorie ist ersichtlich, wie lange wie viele "jobs" in der Modellkomponente waren. Die kumulative Verweilzeit $J$ lässt sich durch die momentane Population $n_t$ bestimmen. Die momentane Population $n_t$ ist die Anzahl der "jobs" zum Modellzeitpunkt $t$ in der Modellkomponente. Durch eine Ankunft oder einen Abgang ändert sich die momentane Population. Die Zeit, in der $n_t$ konstant ist, sei $\delta t$. Die Multiplikation $n_t \cdot \delta t$ ergibt die kumulative Verweilzeit $J_{\delta t}$ des Teilintervalls $\delta t$. Die Summe aller $J_{\delta t}$ ergibt die kumulative Verweilzeit $J$ des gesamten Intervalls.

### Population

Die durchschnittliche Population $n$ eines Intervalls wird von der kumulativen Verweilzeit abgeleitet:

$$n = \frac{J}{T} \quad . \tag{A.5}$$

$T$ sei die Länge des betrachteten Intervalls.

### A.2.3 Implementierung

Das Programm *steady* ist in die Module "main", "datenstruktur", "kenngrössen" und "tools" aufgeteilt. Das Modul "main" steuert den Programmablauf. In dem Modul "datenstruktur" befinden sich alle Funktionen, die benötigt werden, um eine Datenstruktur aufzubauen und zu verwalten. Diese Datenstruktur beinhaltet, welche Modellkomponenten und Dienste es gibt. Ebenso verwaltet sie den Weg eines Prozesses (oder "jobs") durch das Modell. Das Modul "kenngrössen" ist für die Interpretation der Ereignisse zuständig. Die Leistungsmaße werden in Abhängigkeit von den Ereignissen berechnet und in der Datenstruktur aktualisiert. Das Modul "tools" übernimmt sonstige Aufgaben.

Zusätzlich werden einige Standardbibliotheken benutzt. Neben den üblichen Bibliotheken unistd.h, iostream, string, fstream, strstream, stdio.h und stdlib.h werden die Bibliotheken signal.h und sys/wait.h verwendet. Die beiden letztgenannten Bibliotheken ermöglichen eine effiziente Kommunikation zwischen *steady* und der Simulation. Des Weiteren ermöglichen sie eine Steuerung der Simulation durch das Filterprogramm *steady*. Die Datenstruktur setzt sich zusammen aus einigen Standard-"Dictonarys". Da bei einer Simulation sehr viele Ereignisse entstehen, ist die Wahl dieser "Dictonarys" von großer Bedeutung. Je nach Anforderung wurden entweder eine doppelt verkettete Liste (Bibliothek: list), ein Hashing-Verfahren (Bibliothek: hash_map) oder ein binärer Suchbaum (Bibliothek: map) verwendet.

Anzumerken ist noch, dass die Ausgabereihenfolge der Ereignisse durch die Simulation willkürlich ist, wenn Ereignisse zur selben Modellzeit stattfinden. Dies bedeutet, dass nicht immer ein "Exit" auf ein "Entry" folgen muss. Diese Besonderheit tritt häufig dann auf, wenn ein HISLANG-Modell aus Komponenten besteht, die keine Modellzeit verbrauchen. Bei der Berechnung der Leistungsmaße muss dies berücksichtigt werden, denn eine Auswertung, die in der Trace-Ausgabe der Simulation Zeile für Zeile vorgeht, kann zu Problemen führen.

## A.3 hit2steady

Das Programm *hit2steady* erweitert ein gültiges HISLANG-Modell um einige Eigenschaften. Eine Eigenschaft ist, dass der Startseed der Simulation über eine externe Datei festgelegt werden kann. Eine weitere Eigenschaft ist, dass das Modell alle Simulationsereignisse an den Standardausgabekanal liefert. Das Format dieser Ausgabe und die Definition der Ereignisse lassen sich in [1] Anhang G.5.1.2 nachlesen. Der ursprüngliche Experimentteil wird komplett ersetzt.

### A.3.1 Aufruf und Ergebnisse

*hit2steady* wird mit folgender Syntax aufgerufen:

```
PFAD/hit2steady [OPTION] [OPTION] ... HITDATEI
```

Beispiele:

```
hit2steady -info HitDatei.hit
hit2steady -interact HitDatei.hit
hit2steady -interact -messobjekte objekte.txt HitDatei.hit
```

In den drei Beispielen sind schon alle wichtigen Optionen aufgeführt. Das Programm hat noch mehr Optionen, diese werden an dieser Stelle jedoch nicht erwähnt, da ihre Funktionsweise für diese Diplomarbeit nicht von Interesse ist.

```
-info
   Diese Option unterdrückt die Erzeugung einer über-
   arbeiteten HISLANG-Datei. Es wird lediglich die Datei
   HITDATEI.cmp und die Datei HITDATEI.typ erzeugt.
-interact
   Diese Option erzeugt eine überarbeitete HISLANG-Datei,
   die keine Abbruchbedingung besitzt.
-messobjekte DATEINAME
   Wenn diese Option nicht gewählt ist, werden die
   Ereignisse aller Modellkomponenten protokolliert.
   Wenn über diese Option eine Datei angegeben wird,
   dann werden nur die Modellkomponenten protokolliert,
   die in der Datei vermerkt sind. Diese Datei muss das-
   selbe Format wie die Datei HITDATEI.cmp besitzen.
```

Da die Erstellung der Messobjektdatei, welche die zu analysierenden Messobjekte benennt, für den Benutzer recht aufwendig sein kann, erzeugt die Option "-info" die Datei "HITDATEI.cmp". Diese Datei beinhaltet alle automatisch erkannten Modellkomponenten. Durch Löschen der überflüssigen Modellkomponenten aus dieser Datei kann sich der Benutzer seine eigene Messobjektdatei erzeugen. Das Beispiel einer "HITDATEI.cmp"-Datei:

```
SSM
SSM.DerServer
```

"SSM" ist hierbei die oberste Modellkomponente und "SSM.DerServer" ist eine untergeordnete Komponente. Formal besteht die Datei aus den Namen der Modellkomponenten, die durch einen Zeilenumbruch voneinander getrennt sind. Die Hierarchiestufen werden durch einen "." deutlich gemacht.

Da der Experimentteil komplett ersetzt wird, muss auch ein neuer Modellname festgelegt werden. Es wird der Name "SSM" (Steady-State-Modell) benutzt. Der ursprüngliche Typname des Modells wird in der Datei "HITDATEI.typ" festgehalten.

Der Startseed der Simulation wird über eine externe Datei mit dem Namen "seed.txt" festgelegt. Diese Datei muss sich in demselben Dateiverzeichnis wie die überarbeitete HISLANG-Datei befinden. Die Seeddatei besteht aus einem Seedwert, der durch

einen Zeilenumbruch abgeschlossen wird. Der Seedwert muss eine Integerzahl aus dem Bereich $\{-2^{31};2^{31}-1\}$ sein.

Das neue HISLANG-Modell hat den Dateinamen des urprünglichen HISLANG-Modells mit angefügtem ".steady". Die Ergänzungen durch *hit2steady* an den HISLANG-Modellen werden nun an einem kleinen Beispiel demonstriert. Das ursprüngliche HISLANG-Modell hat folgende Form:

```
%COMMON
%PARM = WARN
%END
%COPY "counter"
%COPY "semaphor"
%COPY "observer"
TYPE voidRecordName RECORD; END TYPE voidRecordName;
TYPE initServer_Typ MODEL;
  COMPONENT DerServer : server
    (LET ACCEPT   := ALWAYS,
     LET SCHEDULE := FCFS  (LET CAPACITY := 1),
     LET DISPATCH := EQUAL (LET SPEED    := 1.0),
     LET OFFER    := ALL);
  TYPE Aufruf SERVICE;
    USE
      SERVICE Anfrage(B2H___amount : REAL DEFAULT 1.0);
    END USE;
  BEGIN
    Anfrage(negexp(2));
  END {SERVICE} TYPE Aufruf;
  TYPE Quellen SERVICE;
  BEGIN
    LOOP
      hold(negexp(1));
      CREATE 1 PROCESS Aufruf;
    END LOOP;
  END {SERVICE} TYPE Quellen;
  TYPE InitQuellen SERVICE;
  BEGIN
      CREATE 100 PROCESS Aufruf;
  END {SERVICE} TYPE InitQuellen;
 REFER
    Aufruf
  TO
    DerServer
  EQUATING
    Aufruf.Anfrage WITH DerServer.request;
  END REFER;
BEGIN
  CREATE 1 PROCESS InitQuellen;
  CREATE 1 PROCESS Quellen;
END {MODEL} TYPE initServer_Typ;
{--------------------------------------------------------------------}
EXPERIMENT versuch METHOD SIMULATIVE;
BEGIN
  EVALUATE MODEL initServer : initServer_Typ(LET SEED := 13);
    EVALUATIONOBJECT
    messgeraet VIA initServer.DerServer;

  BEGIN
    MEASURE POPULATION AT messgeraet;
    MEASURE UTILIZATION AT messgeraet;
    MEASURE THROUGHPUT AT messgeraet;
    MEASURE TURNAROUNDTIME AT messgeraet;
    CONTROL AT messgeraet STOP
    CONFIDENCE LEVEL 95 WIDTH 5.0 MEASURE POPULATION AND
    CONFIDENCE LEVEL 95 WIDTH 5.0 MEASURE UTILIZATION AND
    CONFIDENCE LEVEL 95 WIDTH 5.0 MEASURE THROUGHPUT AND
    CONFIDENCE LEVEL 95 WIDTH 5.0 MEASURE TURNAROUNDTIME;
  END EVALUATE;
END EXPERIMENT versuch;
```

Das überarbeitete Modell hat die Form:

```
%COMMON
%BIND "seedDateiName" TO seed.txt {*1*}
%BIND "TRACE" TO SYSOUT {*2*}
%PARM = TRACEFORMAT = 2 {*2*}
```

```
%PARM = WARN
%END
%COPY "counter"
%COPY "semaphor"
%COPY "observer"
TYPE voidRecordName RECORD; END TYPE voidRecordName;
TYPE initServer_Typ MODEL;
  COMPONENT DerServer : server
    (LET ACCEPT   := ALWAYS,
     LET SCHEDULE := FCFS  (LET CAPACITY := 1),
     LET DISPATCH := EQUAL (LET SPEED    := 1.0),
     LET OFFER    := ALL);
  TYPE Aufruf SERVICE;
    USE
      SERVICE Anfrage(B2H___amount : REAL DEFAULT 1.0);
    END USE;
  BEGIN
    Anfrage(negexp(2));
  END  TYPE Aufruf;
  TYPE Quellen SERVICE;
  BEGIN
    LOOP
      hold(negexp(1));
      CREATE 1 PROCESS Aufruf;
    END LOOP;
  END  TYPE Quellen;
  TYPE InitQuellen SERVICE;
  BEGIN
      CREATE 100 PROCESS Aufruf;
  END  TYPE InitQuellen;
 REFER
    Aufruf
  TO
    DerServer
  EQUATING
    Aufruf.Anfrage WITH DerServer.request;
  END REFER;
BEGIN
  CREATE 1 PROCESS InitQuellen;
  CREATE 1 PROCESS Quellen;
END  TYPE initServer_Typ;

EXPERIMENT SteadyState METHOD SIMULATIVE;
VARIABLE meinSeed  : INTEGER;
         seedDatei : INFILE;
BEGIN
  OPEN seedDatei, "seedDateiName" LENGTH 80; {*1*}
  READ FILE seedDatei, meinSeed; {*1*}
  CLOSE seedDatei; {*1*}
  EVALUATE MODEL SSM : initServer_Typ(LET SEED := meinSeed);
    EVALUATIONOBJECT messobjekt VIA SSM;
  BEGIN
    MEASURE POPULATION AT messobjekt;
    CONTROL TRACEALL {*2*}
            AT messobjekt; {*2*}
  END EVALUATE;
END EXPERIMENT SteadyState;
```

Die mit "{*1*}" markierten Zeilen sorgen dafür, dass der Startseed aus der externen Datei "seed.txt" gelesen wird. Die mit "{*2*}" markierten Zeilen erzeugen eine Ausgabe der Ereignisse. In diesem Fall werden die Ereignisse aller Modellkomponenten ausgegeben. Sollen jedoch nicht alle Modellkomponenten betrachtet werden, so hat der Experimentteil eine etwas unterschiedliche Form:

```
EXPERIMENT SteadyState METHOD SIMULATIVE;
VARIABLE meinSeed  : INTEGER;
         seedDatei : INFILE;
BEGIN
  OPEN seedDatei, "seedDateiName" LENGTH 80; {*1*}
  READ FILE seedDatei, meinSeed; {*1*}
  CLOSE seedDatei; {*1*}
  EVALUATE MODEL SSM : initServer_Typ(LET SEED := meinSeed);
    EVALUATIONOBJECT
       messobjekt VIA SSM;
       messobjekt0 VIA SSM.DerServer;
  BEGIN
    MEASURE POPULATION AT messobjekt;
    CONTROL {*2*}
            AT messobjekt0 TRACE; {*2*}
```

```
  END EVALUATE;
END EXPERIMENT SteadyState;
```

Mit diesem Experimentteil wird nur die Komponente "SSM.DerServer" betrachtet. Da die Ausgabe der Ereignisse eine sehr große Datenmenge zur Folge hat, ist es sehr wichtig, dass nur ausgewählte Komponenten betrachtet werden können.

```
0.000000E+000       3 BIRTH          initserver_t       aufruf
0.000000E+000       3 ANNOUNCE                                    ...
0.000000E+000       3 >ENTRY         initserver_t       aufruf
0.000000E+000       3 ENTRY>SERVICE  initserver_t       aufruf
0.000000E+000       3 ANNOUNCE       initserver_t       aufruf     ...
0.000000E+000       3 >ENTRY         derserver          REQUEST
0.000000E+000       3 ENTRY>SERVICE  derserver          REQUEST    ...
3.083724E-001       3 SERVICE>EXIT   derserver          REQUEST
3.083724E-001       3 EXIT>          derserver          REQUEST
3.083724E-001       3 SERVICE>EXIT   initserver_t       aufruf
3.083724E-001       3 EXIT>          initserver_t       aufruf
3.083724E-001       3 DEATH          initserver_t       aufruf
```

Diese Ereignisse sind nur ein willkürlicher Auszug aus der Ausgabe einer Simulation. Das genaue Ausgabeformat ist in [1] Anhang G.5.1.2 beschrieben. Dort wird ebenfalls die Interpretation der Ereignisse angegeben.

### A.3.2 Implementierung

Einen Hauptteil der Implementierung macht das Erkennen der Struktur des HISLANG-Modells aus. Die hierarchische Struktur muss in einer geeigneten Datenstruktur erfasst werden können, um dem Benutzer eine Hilfestellung in Bezug auf die Wahl der zu analysierenden Modellkomponenten geben zu können. Hierbei muss zwischen dem Typ und der Instanz einer Komponente unterschieden werden.

Die ursprüngliche HISLANG-Datei wird zeilenweise eingelesen. Ergänzungen, die nicht von der hierarchischen Struktur abhängen, können direkt eingefügt werden. Der ursprüngliche Experimentteil wird gelöscht und ersetzt. Der neue Experimentteil kann erst erzeugt werden, wenn die zu analysierenden Modellkomponenten bekannt sind.

Die Programmstruktur von *hit2steady* ist modular aufgebaut. Die Module sind so gewählt, dass semantisch ähnliche Funktionen zusammengefasst wurden:

**einlesen:**
: Einlesen der ursprünglichen HISLANG-Datei und der Messobjekte.

**erzeugen:**
: Schreiben der veränderten HISLANG-Datei.

**interact:**
: Erzeugt den Experimentteil für eine Simulation ohne Abbruchbedingung.

**main:**
: Erkennen der Aufrufparameter und Ablaufsteuerung.

**struktur:**
: Datenstruktur für ein hierarchisches HISLANG-Modell.

**utils:**
Sonstige Funktionen

Neben den schon erwähnten Modulen wird auf einige Standardbibliotheken zurückgegriffen: iostream, string, fstream, strstream, list. Diese Standardbibliotheken stellen eine Vielzahl von Standardoperationen zur Verfügung.

## A.4 steady_hit

*steady_hit* ist eine Befehlsabfolge ("Script"), die eine HISLANG-Datei in ein ausführbares Programm verwandelt. Zunächst wird die Syntax des HISLANG-Modells überprüft, bei einem Fehler wird die Ausführung abgebrochen. Im nächsten Schritt erfolgt eine "Compilation" mit anschließendem "Linken". Bei diesen Schritten werden temporäre Links und Dateien erzeugt. Ebenfalls wird eine ausführbare Datei mit dem Namen *steady.hitcode* erzeugt. Bei der Ausführung dieser Datei wird eine Simulation des ursprünglichen HISLANG-Modells durchgeführt.

Das "Script" *steady_hit* ist von dem bekannten "Script" *hit* abgeleitet. Während *hit* jedoch auf die "Compilation" und das "Linken" eine Simulation folgen lässt, wird bei der Ausführung von *steady_hit* nur eine ausführbare Datei erzeugt, ohne die Simulation zu starten.

Zu beachten ist noch, dass das Programm *steady.hitcode* einige der temporären Dateien zur Ausführung benötigt. Daher werden nicht alle temporären Dateien durch das Script *steady_hit* gelöscht. Diese temporären Dateien können von dem Benutzer nach der Simulationsausführung gelöscht werden.

Die Syntax zum Aufruf von *steady_hit* besteht aus dem Befehlsnamen und der HISLANG-Datei:

```
PFAD/steady_hit Dateiname
```

Der Aufruf von *steady.hitcode* hat folgende Syntax:

```
PFAD/steady.hitcode -p -g -k=Speicherbereich
```

Die Optionen "-p" und "-k" sind Standardoptionen, die immer angegeben werden sollten. Durch "-k=Speicherbereich" kann die Größe des Speicherbereiches festgelegt werden. Ist dieser Wert zu groß gewählt, werden die überschüssigen Daten aus dem Hauptspeicher des Computers auf die Festplatte ausgelagert und die Simulationsdauer wird massiv verzögert. Wird dieser Wert allerdings zu klein gewählt, muss die Simulation ihren Speicherbereich sehr oft reorganisieren, um nicht mehr benötigte Daten zu löschen. Dies führt ebenfalls zu einer sehr langsamen Simulation. Die Wahl der Größe des Speicherbereichs ist also entscheidend. Die Angabe erfolgt in "Kilobyte". Beispiel:

```
steady.hitcode -p -g -k=65536
```

In diesem Beispiel wird eine Simulation gestartet, die 64MB (oder 65536KB) Speicher benutzt.

*steady_hit* wird im weiteren Verlauf dieser Diplomarbeit benutzt, um eine ausführbare Simulationsdatei zu erzeugen. Diese Datei wird mehrfach kopiert und mit unterschiedlich großem Speicherbereich auf unterschiedlichen Computern gestartet. So können mehrere Simulationen parallel gestartet werden.

# Literaturverzeichnis


[1] H. Beilner:
Leistungsbewertung von Rechen- und Kommunikationssystemen,
Vorlesungsskript, Universität Dortmund, 1999.

[2] Prozessketteneditorhandbuch.
lokales File auf dem Server des Lehrstuhls für Informatik:
/app/unido-i04/SFB559_M1/
Prozessketten_<*aktuelle Version*>/Handbuch.pdf
oder extern als Handbuch.pdf-File im Installationsverzeichnis des Prozessketteneditors

[3] H. Beilner:
HI-SLANG Referenzhandbuch,
Universität Dortmund, 1999.

[4] A. M. Law, W. D. Kelton:
Simulation Modelling and Analysis,
McGraw-Hill Higher Education, 2000

[5] H. Beilner, F. Bause, H. Tatlitürk, A. van Almsick, M. Völker:
Zum B-Modellformalismus - Version B1, SFB-Bericht 99002,
Universität Dortmund,1999.
lokales File auf dem Server des Lehrstuhls für Informatik:
/app/unido-i04/SFB559_M1/ERGEBNIS_DOCS/
DOKUMENTE/B1-Bericht.pdf

[6] M. Eickhoff:
Statistische Auswertung und Erkennung der stationären Phase in der Simulation zustands-diskreter Systeme,
Diplomarbeit, Universität Dortmund, 2002.

[7] Th. Köpp:
Visualization of Performance and Cost Measures for Logistic Systems Derived from Process Chain Paradigms,
Diplomarbeit, Universität Dortmund, 2001.

# Abbildungsverzeichnis