UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

PG 476

Bottleneckanalyse des Informatikstudiums der Universität Dortmund

Zwischenbericht

Erstellt 12. Juni 2006

INTERNE BERICHTE
INTERNAL REPORTS

Lehrstuhl IV (Modellierung und Simulation)
Fachbereich Informatik
Universität Dortmund

Veranstalter:
Jürgen Mäter

GERMANY · D-44221 DORTMUND

Inhaltsverzeichnis

1 Einleitung **1**

2 Seminar **2**

2.1 Leistungsbewertung im Überblick . . . 2
2.2 Grundlagen der Simulation . . . 4
2.3 Grundlagen der Wahrscheinlichkeitstheorie und Statistik . . . 5
2.4 Erstellung gültiger und glaubwürdiger Simulationsmodelle; Auswahl von Eingangs-Wahrscheinlichkeitsverteilungen . . . 7
2.5 Zufallszahlengenerator und Erzeugung von Zufallszahlen . . . 8
2.6 Hierarchisches Leistungsbewertungs - Tool HIT . . . 9
2.7 Projektmanagement . . . 9
2.8 Datenanalyse für Einzelsysteme sowie Vergleich alternativer Systeme . . . 11
2.9 Experiment-Design . . . 14
2.10 Prozesskettenparadigma . . . 14
2.11 Das ProC/B-Toolset . . . 15
2.12 Bottleneckanalyse . . . 16

3 Modellierungspraktikum **18**

3.1 Aufgabe 2 Gruppe 1 . . . 19
3.2 Aufgabe 2 Gruppe 2 . . . 22
3.3 Aufgabe 2 Gruppe 3 . . . 25
3.4 Aufgabe 4 Gruppe 3 . . . 28
3.5 Aufgabe 4 Gruppe 1 . . . 35
3.6 Aufgabe 4 Gruppe 2 . . . 37
3.7 Aufgabe 5 Gruppe 1 . . . 42
3.7.1 Vorgehensweise . . . 42
3.7.2 Anpassung des HI-SLANG-Quellcodes in HIT . . . 42
3.7.3 Anpassung des HI-SLANG-Quellcodes in ProC/B . . . 44
3.7.4 Datenanalyse mit dem R Statistikpaket . . . 44
3.7.5 Codelistings . . . 46
3.8 Aufgabe 5 Gruppe 2 . . . 49
3.8.1 Das ursprüngliche Modell in ProC/B . . . 49
3.8.2 Das Modell in HIT . . . 51
3.8.3 Die Simulationsergebnisse . . . 52
3.8.4 Validierung der Modelle . . . 52
3.9 Aufgabe 5 Gruppe 3 . . . 55
3.9.1 Das ursprüngliche Modell in HIT . . . 55
3.9.2 Das Modell in ProC/B . . . 56

3.9.3 Die Simulationsergebnisse . . . 58
3.9.4 Validierung der Modelle . . . 59

4 Projektmanagement 65

5 Simulationsstudie 67
5.1 Problemstellung und Planung der Studie . . . 67
5.2 Modellentwurf und Datenerhebung . . . 67
5.2.1 Struktur- und Ablaufdiagramme . . . 67
5.2.2 Datenerhebung/-analyse . . . 77
5.3 Validierung und Annahmen . . . 80
5.3.1 Bestimmung des Intervalls für die diskrete Gleichverteilung . . . 81
5.3.2 Annahmen für den Bestcase . . . 84
5.3.3 Annahmen für den Worstcase . . . 85
5.3.4 Zusammenfassung der Annahmen . . . 85
5.4 Implementierung und Verifikation des HIT-Modells für den Bestcase . . . 87
5.4.1 Bestcase . . . 88
5.4.2 uni . . . 89
5.4.3 fbi . . . 89
5.4.4 ls . . . 90
5.4.5 Dummies . . . 94
5.4.6 kerninfo . . . 94
5.4.7 grundstudiumki und hauptstudiumki . . . 95
5.4.8 semester1 bis semester4 . . . 96
5.4.9 semester5 bis semester9 . . . 97
5.4.10 gti . . . 98
5.4.11 dap1 . . . 101
5.4.12 PG . . . 101
5.4.13 sonstige Komponententypen . . . 102
5.4.14 Verifikation . . . 102
5.5 Implementierung des HIT-Modells für den Worstcase . . . 102
5.5.1 Vorlesungen . . . 102
5.5.2 PG und SoPra . . . 103
5.6 Implementierung und Verifikation des ProC/B-Modells für den Bestcase . . . 104
5.6.1 Aufbau und Struktur der ProC/B Modellierung . . . 104
5.6.2 Hinweise zur Dokumentation der einzelnen Funktionseinheiten . . . 106
5.6.3 Bestcase . . . 107
5.6.4 Uni . . . 108
5.6.5 FBI . . . 108
5.6.6 ExterneFBs . . . 113
5.6.7 Angebot (im FBI) . . . 113
5.6.8 Angebot (in ExterneFBs) . . . 114
5.6.9 KI . . . 117
5.6.10 muendl_Pruefung . . . 117
5.6.11 Vorlesung (im FBI) . . . 121
5.6.12 Vorlesung (in ExterneFBs) . . . 123
5.6.13 Lehrstuhl2 . . . 127

5.6.14 GS . . . 128
5.6.15 HS . . . 128
5.6.16 Sem1 . . . 130
5.6.17 Sem2 . . . 131
5.6.18 Sem3 . . . 132
5.6.19 Sem4 . . . 133
5.6.20 Sem5 . . . 134
5.6.21 Sem6 . . . 134
5.6.22 Sem7 . . . 135
5.6.23 Sem8 . . . 136
5.6.24 Semester9 . . . 136
5.6.25 Verifikation und Abbruch der ProC/B Analyse . . . 137
5.7 Modellvalidierung . . . 138
5.8 Produktionsläufe, Analyse und Interpretation der Daten (HIT) . . . 144
5.8.1 Bestcase . . . 144
5.8.2 Worstcase . . . 147
5.9 Probleme bei der Modellierung . . . 149

6 Weitere Vorgehensweise **151**

Literatur **152**

1 Einleitung

Die Projektgruppe 476 setzt sich aus den folgenden Studenten zusammen: Manuel Beer, Eren Boncuklu, Menderes Deniz, Viktor Faber, Rainer Fels, Genadi Glasirin, Dominic Ketteltasche, Darius Lohmeier, Yongfeng Mai, Oktay Özdemir, Akansel Sereflioglu und Michael Thüner. Leiter der PG ist Herr Jürgen Mäter, wissenschaftlicher Mitarbeiter am Lehrstuhl IV der Universität Dortmund.
Die Aufgabe der PG ist es herauszufinden, ob die Universität Dortmund mit den vorhandenen Studienangeboten, den Ressourcen, wie Lehrende, Hör- und Seminarräume, eine Regelstudienzeit garantieren kann. Aufgrund der Größe und des modularen Aufbaus der Universität beschränkt sich das prinzipielle Vorgehen der Projektgruppe 476 nur auf den Fachbereich Informatik (FBI).
Am Anfang der Projektarbeit steht die Seminarphase (Kapitel 2), in der sich die Teilnehmer die für eine Bottleneck-Analyse notwendigen Grundlagen aneignen. Dazu gehören beispielsweise Grundlagen der Simulation, der Wahrscheinlichkeitstheorie und Statistik, der verwendeten Werkzeuge und der Bottleneck-Analyse selbst. Zusätzlich werden für die zeitliche Planung der Projektgruppe auch Grundlagen des Projektmanagements (Kapitel 4) benötigt, um sicherzustellen, dass Fristen eingehalten und die zur Verfügung stehenden Mittel nicht überschritten werden.
Die vertiefende Einarbeitung in die verwendeten Simulationswerkzeuge HIT und ProC/B soll durch das Modellierungspraktikum (MoPra) erreicht werden (Kapitel 3).
Daran schließt sich die Simulationsstudie (Kapitel 5), die eigentliche Arbeit der Projektgruppe, an. Dazu gehören ein initialer Modellentwurf, sowie die Erhebung der benötigten Daten und deren Analyse. Die daraus gewonnenen Erkenntnisse werden bei der Implementierung mit den beiden Werkzeugen umgesetzt. Zeitgleich mit der Implementierung wird die Modellvalidierung vorgenommen, um die Konsistenz der beiden entwickelten Modelle untereinander und mit den zuvor angefertigten Struktur- und Ablaufdiagrammen zu sichern. Im Anschluss an die Simulationsläufe der fertigen Modelle folgt noch die Analyse und Interpretation der Ergebnisdaten.
An dieser Stelle beschränken wir uns auf Modelle zum Best- und Worstcase. Die weitere Vorgehensweise (Kapitel 6), den Realcase betreffend, wird im letzten Abschnitt beschrieben.
Wir möchten uns an dieser Stelle bei allen bedanken, die uns bei unserer Arbeit unterstützt haben. Besonders bei Frau Tanja Heinrich vom Zentrum für Studienangelegenheiten, Frau Iris Zerweck und Herrn Gerd Konnegen von der Universitätsverwaltung der Universität Dortmund, Dezernat 2.2 Abteilung Statistik sowie Kosten- und Leistungsrechnung und Herrn Hans Decker, Geschäftsleitung des Dekanats des Fachbereichs Informatik an der Universität Dortmund, die uns schnell und zuverlässig die von uns benötigten Grunddaten zur Verfügung gestellt haben. Des Weiteren danken wir Herrn Marcus Völker, Wissenschaftlicher Mitarbeiter, dafür, dass er uns mit Rat und Tat bei Modellierungsfragen und -problemen zur Seite gestanden hat, sowie Daniel Arndt, Jan Kriege und Mathias Schwenke, studentische Hilfskräfte, alle vier am Lehrstuhl 4 des Fachbereichs Informatik der Universität Dortmund, die uns immer nach Kräften bei Problemen mit dem ProC/B-Editor und Fehlern des ProC/B-Editor unterstützt haben.

2 Seminar

Die Arbeit der Projektgruppe begann mit einem extern abgehaltenen Seminar. Die Themen des Seminars dienen teils der Auffrischung von im Studium bereits erlangtem Wissen, welches in der PG gebraucht wird, teils der spezifischen Vorbereitung im Umgang mit Modellierungswerkzeugen. Das folgende Kapitel setzt sich aus den extended abstracts der einzelnen Vorträge zusammen, wobei es sich bei einem extended abstract um eine Art Zusammenfassung eines entsprechenden Vortrags handelt. Dabei werden die einzelnen extended abstracts in der chronologischen Reihenfolge abgedruckt, in der die einzelnen Vorträge gehalten wurden.

2.1 Leistungsbewertung im Überblick

Darius Lohmeier

Einleitung

Als Leistungsbewertung bezeichnet man die Aufgabe, die Leistung eines Systems auf reproduzierbarem Wege zu messen und in Bezug auf andere Systeme zu bewerten. Damit ein System überhaupt eine messbare Leistung erbringen kann, benötigt es eine zu erledigende Aufgabe, eine sog. Arbeitslast. Ist man an dem vom System gelieferten Ergebnis bezüglich einer bestimmten Arbeitslast interessiert, so simuliert man diese Arbeitslast, indem man sie dem System kontrolliert übergibt. In einem Optimierungsszenario möchte man die Eingabe finden, die zu einem gewünschten Ergebnis führt. Soll die Simulation an einem realen System vermieden werden, benötigt man ein Modell, d.i. ein Bild des Systems, das hinreichend genau zu sein hat, um die Ergebnisse der Simulation auf das reale System übertragen zu können. Der Prozess der Messung, Aufzeichnung und eventuellen Analyse von Bewertungskriterien selbst wird von Monitoren durchgeführt.
Häufig auftretende Fehler sind die Abwesenheit von Zielen, die Voreingenommenheit gegenüber den Zielen, ein unsystematischer Ansatz, eine Analyse ohne Verständnis des Problems, die Auswahl der falschen Bewertungskriterien, eine nichtrepräsentative Arbeitslast, die Wahl einer falschen Auswertungstechnik, das außer Acht lassen ausschlaggebender Parameter, das Ignorieren signifikanter Faktoren, ein unangemessenes Experiment-Design, eine unangemessene Detailgenauigkeit, keine oder eine fehlerhafte Analyse, keine Empfindlichkeitsanalyse, das Ignorieren von Eingabefehlern, eine unsachgemäße Behandlung von Ausreißern, das Voraussetzten einer statischen Zukunft, das Ignorieren von Schwankungen, eine zu komplexe Analyse, die schlechte Präsentation der Resultate, das außer Acht lassen von sozialen Aspekten und das Aussparen von Annahmen.
Ein systematischer Ansatz zur Leistungsbewertung berücksichtigt vorher gesetzte Ziele und Systemgrenzen, listet angebotene Dienste und Ergebnisse auf, wählt Bewertungskriterien aus, listet Parameter auf, wählt zu betrachtende Faktoren, Auswertungstechniken und Arbeitslasten aus und legt Wert auf den Entwurf der Experimente, die Analyse und Interpretation der Daten sowie die Präsentation der Resultate.
Bei der Auswahl hinreichender Evaluierungstechniken gilt es zwischen einem analytischen Modell, einer Simulation und einer Messung am Realsystem zu unterscheiden. Die Entscheidung hängt z.B.

von der bisherigen Realisierung des Systems, der verfügbaren Zeit, dem Aufwand, der benötigten Präzision, den Kosten und dem Grad der Akzeptanz ab. Generell gilt es, nie nur einer einzigen Technik zu vertrauen. Geläufige Bewertungskriterien sind die Antwortzeit, die Reaktionszeit eines Systems, die Ausführungszeit, der Durchsatz, die Kapazität, die Effizienz, die Auslastung einzelner Systemkomponenten, die Verlässlichkeit und die Verfügbarkeit, gemessen an der durchschnittlichen Zeit, in welcher das System auf Anfragen antwortet.

Arbeitslasten

Eine Arbeitslast ist eine Eingabe, die dem System zu Testzwecken übergeben wird und die die realen Eingaben so gut wie möglich simulieren soll, um konstruktive Messergebnisse zu erhalten. Reale Arbeitslasten sind diejenigen, die bei einem nicht beeinflussten, alltäglichen Umgang mit dem System auftreten. Synthetische Arbeitslasten sind eigens für die Leistungsbewertung konzipiert und zeichnen sich durch gute Portierbarkeit und Flexibilität aus. Klassische Testarbeitslasten, also solche, die in einer Leistungsbewertung benutzt werden, bestehen z.B. aus einer Reihe von Additionsbefehlen, einem Kernel, der eine komplexe Aufgabe (Matrixinversionen etc.) simuliert, einem künstlichem Programm, das auch I/O-Operationen berücksichtigt oder einer konkreten Anwendung.

Monitore

Ein Monitor ist ein Werkzeug, das die Aktivitäten eines Systems verfolgt, aufzeichnet und ggf. zu statistischen Analysen auswertet. Grundlegende Begriffe aus dem Bereich des Monitoring sind das Ereignis, das Ablaufprotokoll, der Overhead, die Domäne, die Eingaberate, das Auflösungsvermögen und die Eingabebreite. Für das Protokollieren höherer Systemaktivitäten benutzt man Softwaremonitore. Diese sind zwar günstig und leicht zu entwickeln, haben aber einen großen Overhead, da sie auf die Ressourcen des Systems zugreifen müssen. Ein Hardwaremonitor steht außerhalb des zu untersuchenden Systems und bezieht seine Daten via Messfühler. Dadurch ist der durch ihn produzierte Overhead, wenn überhaupt vorhanden, sehr gering und er bildet für das Gesamtsystem eine kleinere Fehlerquelle als Softwaremonitore. Welche Art von Monitor in einer speziellen Situation einzusetzen ist, kann oft an Hand der Domäne identifiziert werden.

Simulation

Wenn ein reales System noch nicht verfügbar ist oder die Nutzung eines solchen vermieden werden soll, ist eine Simulation der praktikabelste Weg alternative Parametereinstellungen zu probieren. Fehler, die schon zu Beginn der Simulation vermieden werden können, sind eine unzulängliche Detailgenauigkeit, eine unpassende Sprache, die Nutzung nichtverifizierter oder ungültiger Modelle, die Annahme falscher Anfangsbedingungen, eine kurze Simulationszeit, schlechte Zufallszahlengeneratoren und eine schlechte Wahl der Generatorstartwerte. Ebenso können eine falsche Fehleinschätzung des Zeitaufwandes, ein schlecht gewähltes Ziel, unzureichende Fähigkeiten, eine schlechte Zusammenarbeit mit dem Nutzer, eine unzulängliche Dokumentation, die Unfähigkeit zur Projektverwaltung oder unerwartete Resultate die Simulation scheitern lassen.

Warteschlangentheorie

Die verbreitetste Methode der analytischen Modellierung ist die der Warteschlangentheorie. Da in einem Computersystem fast immer mehrere Jobs auf begrenzte Ressourcen, z. B. die einzige CPU,

zugreifen wollen, die Ressourcen aber nur eine begrenzte Anzahl an Aufträgen verarbeiten können, müssen sich die übrigen Jobs in eine Warteschlange einreihen. Die Warteschlangentheorie bietet nun Möglichkeiten, die durchschnittliche Wartezeit innerhalb einer solchen Schlange zu berechnen.
Warteschlangen werden üblicherweise in der Kendall-Notation beschrieben:

$$A/S/m/B/K/SD$$

Der Zusammenhang zwischen der durchschnittlichen Anzahl der Jobs in einem System, der Rate der das System betretenden Jobs und der Verarbeitungszeit wird durch Littles Gesetz beschrieben.

$$Durchschnittliche\,Anzahl\,im\,System = Ankunftsrate * Verarbeitungszeit$$

Wenn ein Modell mehrere Warteschlangen beinhaltet, die von den Jobs nacheinander durchlaufen werden können, so nennt man es ein Warteschlangen-Netzwerk. Besonders einfach fällt der Umgang mit Netzwerken, die aus der Hintereinanderschaltung von einfachen Warteschlangen entstehen, da jede Schlange unabhängig von den anderen betrachtet werden kann, und sich die Wahrscheinlichkeit der Längen aller Warteschlangen aus dem Produkt der Wahrscheinlichkeiten für eine jede einzelne Schlange berechnen lässt.
Grundlage dieses Vortrages ist [Jain 91].

2.2 Grundlagen der Simulation

Manuel Beer

Simulationen bieten eine kostengünstige Möglichkeit zur Studie und Analyse sehr unterschiedlicher, realer und vor allem sehr komplexer Systeme, im Hinblick auf wünschenswerte, geforderte und erwartete Eigenschaften. Diese gesuchten Informationen sind ebenso vielfältig wie die Systeme selbst und erstrecken sich von Aussagen über Systemleistung und Kapazitätsengpässe, über die Kalkulation von Änderungen am System selbst bis hin zu Neuentwicklungen kompletter Systeme. Zum Zecke der Informationsgewinnung werden so genannte Systemmodelle erstellt, die das reale System entsprechend der gesuchten Informationen ausreichend genau repräsentieren. So verschieden wie die Systeme sind auch die zugehörigen Modelltypen. Charakterisiert werden diese nach drei Gesichtspunkten: dynamisch oder statisch, deterministisch oder stochastisch und diskret oder kontinuierlich. Die Analyse dieser Modelle erfolgt dann mit Mitteln der Simulation. Eine sehr gebräuchliche Art der Simulation stellt die diskrete ereignisorientierte Simulation dar, wie sie zum Beispiel bei der Simulation von Warteschlangensystemen eingesetzt wird. Hierbei handelt es sich um einen dynamischen Simulationstyp. Grundlegend für die Dynamik der Simulation ist die Art und Weise des Simulationsablaufes. Bei diskreten ereignisorientierten Simulationen wird die zeitliche Steuerung innerhalb der Simulation von speziellen Steuerungsroutinen übernommen. Hierbei verfolgt man zwei verschiedene Ansätze. Eine Möglichkeit ist es die Simulationszeit direkt an den Zeitpunkt des nächsten zukünftigen Ereignisses zu koppeln, so können die inaktiven Zeitperioden während der Simulation, in denen keine Ereignisse eintreten, und somit auch keine Veränderungen am System resultieren, übersprungen werden. Eine weitere Möglichkeit ist die Erhöhung der Simulationszeit um einen vordefinierten Festwert. Dieser Mechanismus spielt allerdings eine eher untergeordnete Rolle. Das Zusammenspiel dieser Zeitroutinen mit der Ereignisliste, in welcher die Zeitpunkte der zukünftigen Ereignisse gespeichert werden sichert die dynamische Struktur der Simulation. Die Zeitpunkte der zukünftigen Ereignisse werden anhand festgelegter Wahrscheinlichkeitsverteilungen zufallsgeneriert. Die Modellierung dieser Verteilungen als korrektes Abbild der realen Gegebenheiten für das System stellen einen weiteren wichtigen

Aspekt bei der Durchführung einer Simulationsstudie dar. Falsche oder nicht korrekt modellierte Eingabedaten führen unweigerlich zu völlig unbrauchbaren Simulationsergebnissen, aufgrund derer keine verlässlichen Entscheidungen und Rückschlüsse zu treffen sind. Zu den Ausgaben und Ergebnissen einer Simulation muss grundlegend gesagt werden, dass diese nie als absolute Werte zu interpretieren sind. Vielmehr sind die Ergebnisse lediglich Schätzungen der realen Werte, deren Brauchbarkeit und Genauigkeit sehr stark von der Korrektheit des Simulationsmodells und der jeweiligen Daten abhängig sind. Sehr wichtig für die Korrektheit sind auch die inneren Abläufe während der einzelnen Ereignisse. So ist peinlich darauf zu achten, dass die Berechnungen und Veränderungen am System und für die Sammlung der Ergebnisse in der richtigen Reihenfolge stattfinden, da es sonst ebenfalls zu fehlerhaften Werten kommt. Es versteht sich von selbst, dass auch mögliche Sonderfälle, die außerhalb des wünschenswerten Ablaufes liegen, nicht vernachlässigt werden dürfen, sondern ebenso korrekt in die Modellierung eingebunden werden müssen. Sind in einfachen Systemen, wie beispielsweise Einzelserver-Warteschlangensystemen, die Auswahl der Ereignisse und der statistischen Zähler zur Sammlung der Ergebnisdaten noch relativ einfach, es lässt sich leicht nachvollziehen, dass als mögliche Ereignisse ankommende und gehende Kunden in Frage kommen und statistische Werte, wie die durchschnittliche Wartezeit, die Gesamtwartezeit aller Kunden und die Auslastung des Servers, sinnvolle Betrachtungen bieten, so wird dies bei komplexeren System zu einer durchaus schwierigen und umfangreichen Aufgabe. Bei der Durchführung einer kompletten Simulationsstudie von der ersten Planung bis hin zur Auswertung und Präsentation der Ergebnisse handelt es sich, nicht nur begründet durch die hohe Komplexität der zu untersuchenden Systeme, letztlich nicht um einen schlichten seriellen Ablauf. Es wird nämlich zwischenzeitlich immer wieder notwendig sein die Einblicke und Resultate, die man aus fortgeschrittener Betrachtung gewonnen hat, zur Korrektur und zur Verbesserung der Genauigkeit in vorherige Schritte einzubinden.
Grundlage des Vortrages ist das 1. Kapitel aus [Law 91].

2.3 Grundlagen der Wahrscheinlichkeitstheorie und Statistik

Genadi Glasirin

Der Gebrauch von Wahrscheinlichkeiten und Statistiken ist ein wesentlicher Bestandteil einer Simulationsstudie. Insbesondere sind Wahrscheinlichkeit und Statistiken erforderlich, um zu verstehen wie man ein Wahrscheinlichkeitssystem modelliert, das Simulationsmodell validiert, die Eingangswahrscheinlichkeitsverteilungen wählt, Zufallsstichproben von diesen Verteilungen erzeugt, statistische Analysen der Simulationsausgangsdaten durchführt und Simulationsexperimente entwirft.
Die Simulationen werden durch verschiedene Prozesse beschrieben, deren Ergebnisse als Stichproben bezeichnet werden und bilden einen Stichprobenraum. Stichproben der Simulation werden durch die Zufallsvariablen mit reellen Zahlen verbunden. Mit welcher Wahrscheinlichkeit eine Stichprobe der Zufallsvariable zugewiesen wird, wird durch die Verteilungsfunktion bestimmt. Man unterscheidet zwischen diskreten und kontinuierlichen Zufallsvariablen. Diskrete Zufallsvariablen können höchstens eine abzählbare Anzahl, kontinuierlichen entgegen unabzählbare endlose Anzahl von Werten annehmen. Alle Wahrscheinlichkeitsaussagen über beide Arten von Zufallsvariablen werden entsprechend von Wahrscheinlichkeitsmaßfunktionen bzw. Wahrscheinlichkeitsdichtefunktionen berechnet.
Bei einer Simulation muss man sich normalerweise mit n (positive Ganzzahl) Zufallsvariablen beschäftigen. Da die meisten Simulationsmodelle Zufallsvariablen als Eingangsdaten verwenden, sind die Simulationsausgangsdaten selbst zufällig und es muss große Aufmerksamkeit beim Ziehen der Schlussfolgerungen über die zutreffenden Eigenschaften des Modells gewährleistet werden.
Aus langjährigen Beobachtungen und Erfahrungen kann man mit großer Sicherheit sagen, dass die

Simulationsdaten immer korreliert (auf einander bezogen) sind. Deswegen ist es wichtig bei einer Simulation zu wissen, ob die Eingangsdaten als Zufallsvariablen entweder voneinander abhängig sind oder nicht, wobei die Abhängigkeit durch die Kovarianz bestimmt wird. Weitere Eigenschaften der Zufallsvariablen wie Erwartungswert, Median, Abweichungen von dem Erwartungswert spielen auch eine sehr wichtige Rolle während der Simulation, da man durch die Berechnung dieser Eigenschaften den Simulationsprozess in die nötige Richtung „steuern“ und in manchen Fällen auch die Ausgangsdaten vorhersagen kann.
Prozesse, die Simulationsmodelle beschreiben und durch die Kollektionen der zeitlich geordneten und auf einem Stichprobenraum definierten Zufallsvariablen dargestellt sind, werden stochastisch genannt. Man unterscheidet auch hier zwischen zeitdiskreten und zeitkontinuierlichen Prozessen, was entsprechend von der Kollektion der Zufallsvariablen abhängt.
Um Folgerungen über einen zugrunde liegenden stochastischen Prozess aus einer Menge Ausgabedaten der Simulation zu ziehen, muss man manchmal Annahmen über den stochastischen Prozess bilden, die möglicherweise nicht zutreffend in der Praxis sein können. (Jedoch ohne solchen Annahmen kann eine statistische Analyse der Ausgangsdaten oft nicht möglich sein.) Ein Beispiel von so einer Annahme ist ein stochastischer Prozess stationärer Kovarianz, also ein Prozess, wo der Mittelwert und die Abweichung stationär über die Zeit sind und die Kovarianz zwischen zwei Zufallsvariablen X_i und X_{i+j} nur von dem Intervall j und nicht von der tatsächlichen Zeitwerten i und $i + j$ abhängt.
Da es sehr oft in der Realität unmöglich ist, den Erwartungswert und die Abweichung der Zufallsvariablen zu bestimmen, muss man sie schätzen, wobei als Primärzielsetzung der Mittelwert (Erwartungswert) zu bestimmen ist und die Schätzung von der Abweichung sekundärer Bedeutung hat. Bei der Schätzung sind Stichprobenmittelwert (erwartungstreuer Punktschätzer von Erwartungswert) und die Stichprobenabweichung (erwartungstreuer Punktschätzer der Abweichung) von großer Bedeutung. Ein üblicher Weg, die Präzision des Stichprobenmittelwertes als Schätzer des Erwartungswertes festzusetzen, ist ein Vertrauensintervall für Erwartungswert zu konstruieren.
Als erster Schritt dabei ist, die Abweichung des Stichprobenmittelwertes zu schätzen. Da die Eingangszufallsvariablen unabhängig sind, kommt man zu der Schlussfolgerung, die auch intuitiv nachzuvollziehen ist, dass je größer die Stichprobengröße ist, desto näher der Stichprobenmittelwert zu dem Erwartungswert ist. Allgemein kann man feststellen, dass Simulationsausgangsdaten korreliert sind und die klassischen Formeln, die auf unabhängigen und identisch verteilten Zufallsvariablen basieren, können nicht direkt für das Schätzen von Abweichungen verwendet werden. Jedoch ist es häufig möglich, Simulationsausgangsdaten in neue „Beobachtungen“ zu gruppieren und die Formeln dann, auf diese „Beobachtungen“, anzuwenden.
Um unmittelbar das Vertrauensintervall zu konstruieren, wird der zentrale Grenzwertsatz herangezogen, der besagt, dass wenn die Anzahl von Stichproben „groß genug“ ist, sind diese Stichproben ungefähr als standard normalverteilte Zufallsvariablen verteilt. Da die Anwendung dieses Satzes in der Praxis schwierig ist, werden die Zufallsvariablen ein bisschen geändert und an die Grenzen für normale Standardvariablen angepasst, dadurch wird ein ungefähres Vertrauensintervall für den Erwartungswert gegeben. Die Schwierigkeit beim Konstruieren, liegt am meistens beim Festlegen der Anzahl der Zufallsvariablen, da man nicht eindeutig sagen kann, was „groß genug“ bedeutet. Im Allgemeinen gilt, dass je mehr die Verteilung der Zufallsvariablen streut, desto größer sollte die Anzahl der Zufallsvariablen sein, um höhere Annäherung zu dem Erwartungswert zu erreichen.
Das zweitwichtigste Resultat in der Wahrscheinlichkeitstheorie (nach dem zentralen Grenzwertsatz) ist das starke Gesetz der großen Zahlen, dass uns sagt, wenn wir eine Menge der Zufallsvariablen mit dem bestimmten Mittelwert (Erwartungswert) haben, und eine endlose Anzahl von Experimenten durchführen, wobei jedes in einem erwartungstreuen Punktschätzer von Erwartungswert resultiert und Anzahl der Variablen groß genug ist, dann ist der Stichprobenmittelwert willkürlich nah an den

gegebenen Mittelwert für fast alle Experimente.
Da Simulationsmodelle fast immer mit zufälligen Daten operieren, muss man gewaltig aufpassen, um falsche Ausgangsdaten oder unerwartete Ergebnisse zu vermeiden. Dabei sollte man in erste Linie darauf achten, dass man die vorgegebene Information richtig einordnet und erst dann die Rechenwege auswählt.
Grundlage des Vortrages ist das 4. Kapitel aus [Law 91].

2.4 Erstellung gültiger und glaubwürdiger Simulationsmodelle; Auswahl von Eingangs-Wahrscheinlichkeitsverteilungen

Yongfeng Mai

Unser Ziel ist es, solche Modelle zu entwickeln, die sowohl theoretisch richtig sind, als auch vom Projektmanagement als richtig akzeptiert werden. Man soll schon am Anfang der Modellentwicklung die Gültigkeit des Modells berücksichtigen und so viele Informationen über das System wie möglich sammeln. Besonders zu beachten ist die Überprüfung der Annahmen des Modells, welche auch richtig sein müssen. Nach dem Aufbau des Simulationsmodells muss es noch durch statistische Methoden überprüft werden, ob das entworfene Modell das originale System gut repräsentiert.

Die meisten Simulationsmodelle enthalten mehr als eine Zufallsquelle, die von irgendeiner Wahrscheinlichkeitsverteilung repräsentiert wird. Die Auswahl einer richtigen Verteilungsfunktion für die Zufallsquelle ist auch ein Teil der Aufgabe, die richtigen Annahmen des Simulationsmodells zu bestimmen. Im Allgemeinen schätzt man eine Wahrscheinlichkeitsverteilung für eine Zufallsquelle auf der Basis von historischen Eingaben. Je nach Abhängigkeit der Eingabedaten verwenden wir heuristische Techniken oder statistische Methoden. Zur Bestimmung einer Verteilungsfunktion machen wir drei Schritte:

Wir schätzen zuerst den Typ der Wahrscheinlichkeitsverteilung. Bei den heuristischen Techniken benutzt man oft Histogramm- und Linien-Graph-Verfahren, die jeweils bei kontinuierlicher und diskreter Situation angewendet werden. Und bei den statistischen Techniken steht das Quantil-Verfahren zur Verfügung, mit dem man den Charakter der Form der Verteilungsfunktion schätzt, um eine passende Verteilungsfunktion zu wählen.

Nach der Wahl eines Verteilungsfunktionskandidaten brauchen wir noch die genauen Werte der Zufallsparameter der Funktion abzuschätzen, so dass wir die Funktion vervollständigen. Hier ist eine ganz oft verwendete Methode die Maximum-likelihood-Methode.

Am Ende müssen wir noch überprüfen, ob die geschätzte Wahrscheinlichkeitsverteilung die gegebenen Daten gut anpasst. Bei den heuristischen Techniken kann man einen Histogramm-Vergleich oder Quantile-Quantile-Plot (Q-Q-Plot) und Probability-Probability-Plot (P-P-Plot) verwenden. Die Grundidee dieser Verfahren ist, die Form der geschätzten Funktion mit der graphischen Darstellung der Eingabedaten zu vergleichen und zu bestimmen, ob große Abweichungen dazwischen liegen. Der Q-Q-Plot und P-P-Plot dienen dazu, die Abweichung der beiden Funktionen darzustellen. Für statistische Techniken werden oft der Chi-Quadrat- oder Kolmogorov-Smirnov-Test verwendet. Diese Methoden rechnen nach gewissen Regeln die statistische Abweichung zwischen der geschätzten Verteilungsfunktion und der empirischen Funktion (direkt aus Eingabedaten) und bilden schließlich ein Konfidenzintervall dafür, um zu überprüfen, unter welchem Vertrauensniveau man die Anpassung der

beiden Funktionen akzeptieren kann oder nicht.
Grundlagen des Vortrages sind die Kapitel 5 und 6 aus [Law 91].

2.5 Zufallszahlengenerator und Erzeugung von Zufallszahlen

Oktay Özdemir

Als Zufallszahlengenerator oder kurz Generator verstehen wir ein Verfahren, dass eine Folge von Zufallszahlen erzeugt. Der Bereich, aus dem die Zufallszahlen erzeugt werden, hängt dabei vom speziellen Zufallszahlengenerator ab. Es kann beispielsweise die Menge aller 32-Bit Zahlen oder auch die Menge der reellen Zahlen im Intervall $[0, 1]$ sein. Meistens ist von einem Zufallszahlengenerator erwünscht, dass er gleichverteilte Werte aus dem jeweiligen Bereich erzeugt. Für gewisse statistische Simulationen sind auch solche Zufallszahlengeneratoren interessant, die eine vorgegebene Verteilung (z.B. Exponentialverteilung oder Binomialverteilung) erzeugen. Man unterscheidet grundsätzlich zwischen nicht-deterministischen und deterministischen Zufallszahlengeneratoren. Nicht-deterministisch ist ein Zufallszahlengenerator dann, wenn er auch bei gleichen Ausgangsbedingungen unterschiedliche Werte liefert. Dies wollen wir aber nicht weiter vertiefen. Ein deterministischer Zufallszahlengenerator liefert bei gleichen Ausgangsbedingungen dagegen immer die gleiche Folge von Zufallszahlen (Pseudozufallszahlen). Die Erzeugung sagt allerdings noch nichts über die Güte der Zufallszahlen aus, also inwieweit sie statistischen Tests genügen. Dabei wird zum Beispiel überprüft, ob alle Werte in einem Intervall gleichmäßig auftauchen, ob gerade und ungerade Zahlen in etwa je zur 50 % erzeugt werden, und so weiter. Bevor ein Generator benutzt wird, ist es dringend empfehlenswert zu sehen, welche Art von Generator es ist und welche numerischen Parameter er hat. Es sei denn dieser Generator wurde geprüft und ist für ”gut” erklärt. Da sich eine Pseudozufallszahlenfolge nach einiger Zeit wiederholt (man spricht von einer Periode), ist es erwünscht, dass man eine möglichst lange Folge von Zahlen erhält, bevor die Wiederholung einsetzt. Einen Generator der Gestalt $Z_i = (a*Z_i-1+c)(\ mod\, m)$, $c > 0$, wobei a, c und Z_i ganze Zahlen aus $[0, m)$ sind, wollen wir einen gemischten Generator nennen. Mit geeigneter Wahl der Parameter a und c kann man einen Generator mit voller Periode, d.h. mit Periode m finden, sogar im Fall $m = 2b$, wo b die Nummer der einzelnen Bits ist. Um Zufallsvariablen zu erzeugen, müssen wir die gewünschten Verteilungen benutzen. Diese Verteilungen sind oft vorgeschrieben, wie z.B. die Exponential-Verteilung, Gamma-Verteilung oder Poisson-Verteilung. Dabei sind Zufallszahlen zunächst als Ergebnis stochastisch physikalischer Vorgänge aufzufassen. Ein Verfahren zur Erzeugung von Zufallszahlen wird Zufallszahlengenerator genannt. Man nehme irgendeine Verteilungsfunktion und führt damit die inverse Transformationsmethode durch. Diese Transformation benutzen wir, um Zufallszahlen zu erzeugen. Dies ist zwar eine simple Methode und nutzt die ganzen Vorteile der inversen Transformation, aber sie ist kostengünstig und schnell. Der Basis-Bestandteil für jede Methode zur Erzeugung von Zufallszahlen von irgendeiner Verteilung oder eines Zufallsprozesses ist die Quelle von $U(0, 1)$ Zufallsverteilungen. Aus diesem Grund, ist es wesentlich, dass ein statistisch zuverlässiges Zufallszahlengenerator erhältlich ist. Ohne einen akzeptablen Zufallszahlengenerator ist es unmöglich eine Zufallsvariable korrekt von irgendeiner Verteilung zu erzeugen.
Grundlagen des Vortrages sind die Kapitel 7 und 8 aus [Law 91].

2.6 Hierarchisches Leistungsbewertungs - Tool HIT

Menderes Deniz

Die Entwicklung von Computersystemen ist eng verbunden mit Problemen, weil die Leistungsanforderungen und die technologischen Prozesse rapide ansteigen. Es wird ein Tool benötigt, dass die existierenden Computersysteme qualitativ und quantitativ analysieren kann. Ein solches Tool, ist das Software-Tool HIT. HIT ist ein Werkzeug, das modellbasierte Auswertungen der Rechnerleistung unterstützt. Die in HIT konstruierten Modelle zeigen eine hoch strukturierte Sicht der zu beurteilenden Systeme.
Zu diesem Zweck wurde, im Jahre 1983, am Lehrstuhl IV der Universität Dortmund das Objektorientierte Leistungsbewertungs-Tool HIT entwickelt.
Die wichtigen Elemente von HIT sind die Last und die Maschine. Unter einer Last kann man sich einen Prozess, einen Dienst, ein Job, etc. vorstellen. Die Maschine, die aus wechselseitig unabhängigen Komponenten besteht, hat die Aufgabe einen Dienst zur Nutzung anzubieten. Demnach belastet die Last das System mit Anforderungen, die von angebotenen Diensten der Maschine zu erfüllen sind.
Eng verbunden mit HIT sind HITGRAPHIC und HI-SLANG. HITGRAPHIC ist ein TOOL das die bildliche Darstellung von der Modellierung ermöglicht. Demnach ist HITGRAPHIC eine graphische Benutzeroberfläche zu HIT, einschließlich einer Datenhaltung. HI-SLANG ist die Spezifikationssprache von HIT, d.h. die von HIT modellierten Systeme werden in HI-SLANG formuliert.
In HIT ist der Einsatz unterschiedlicher Analysetechniken, ohne jeweilige Neuspezifikationen zugreifbar. Man benötigt in diesem Sinne kein Expertenwissen. Im Wesentlichen werden zwei Methoden angeboten, die analytische und die simulative Methode. Je nach Entscheidung überführt das HIT-System die HI-SLANG Repräsentation in einen analytischen Algorithmus oder in einen Simulationsprogramm. HIT verfügt über eine Reihe von Lösungsverfahren, wie exakte Produktformalgorithmen für separable Netzwerke, Approximationstechniken für große separable Netze, numerische Verfahren zur Lösung allgemeiner Modelle sowie stochastische ereignisdiskrete Simulation mit entsprechenden statistischen Verfahren zur Schätzung der Leistungskenngrößen.
Die praktische Vorgehensweise bei allen der zu spezifizierenden Modellierungen ist ähnlich:

- Spezifikation eines Modelltyps mit Last- und Maschinenbeschreibung
- Spezifikation der Auswertungsbeschreibung
- Spezifikation der Experimentbeschreibung
- Experimentlauf

Grundlagen des Vortrages sind die folgende Quellen [Beil 89], [Leng 00], [Mäte 04], [Bütt 96], [Bütt 00].

2.7 Projektmanagement

Dominic Ketteltasche

Projekte sind Aufgabenstellungen von besonderer Einmaligkeit und Komplexität, deren Ziele in einem Team in einer bestimmten Zeit mit festgelegten Ressourcen erreicht werden sollen. Das Projekt teilt sich auf in vier Hauptphasen. Während der Projektdefinition werden die Ziele und die Vorgehensweisen festgelegt. Danach folgen die Planung, während des Projekts die Kontrolle und am Ende der Projektabschluss. Am Anfang der Planung wird das Projekt in Arbeitspakete unterteilt, die mittels

des Projektstrukturplans hierarchisch gegliedert werden. Die Gliederung ist entweder objektorientiert, funktionsorientiert oder ablauforientiert. Beim nachfolgenden Ablaufmanagement werden die Arbeitspakete in Vorgänge unterteilt und in eine Reihenfolge gebracht. Diese Ausarbeitung geschieht meist mit einem Netzplan, bei dem der Ereignisknoten-, der Vorgangspfeil- und der Vorgangsknoten-Netzplan die gängigsten Varianten darstellen. Mit einem Vorgangsknoten-Netzplan kann man ebenfalls die Terminplanung durchführen, indem man die Vorwärts- und Rückwärtsrechnung durchführt und die zeitlichen Spielräume berechnet. Das Endergebnis lässt sich anschaulich mit einem Balkendiagramm darstellen. Nach der Terminplanung wird die Einsatzmittelplanung durchgeführt, an die sich logisch die Aufwandsschätzung anschließt. Diese verwendet wiederum analytische Schätzmethoden, welche einen funktionalen Zusammenhang zwischen der Produktgröße und dem zu schätzenden Aufwand herstellen, oder die Expertenbefragung, bei der auf das Fachwissen von Experten zurückgegriffen wird. Unter die analytischen Aufwandsschätzmethoden fallen die algorithmischen Methoden, die Vergleichsmethoden und die Kennzahlmethoden. Bei der Expertenbefragung gibt es die Formen der Einzelschätzung, der Mehrfachbefragung, der Delphi-Methode und der Schätzklausur. Nachdem das Projekt zu Ende geplant wurde und sich in der Ausführung befindet, werden ständig Plan/Ist-Vergleiche durchgeführt, um mittels der Aufwandskontrolle Abweichungen zu erkennen und bei Bedarf steuernde Maßnahmen einzuleiten. Neben der Aufwandskontrolle gibt es noch die Aufwandstrendanalyse, mit der Prognosen angestellt werden, ob die Aufwandsschätzungen am Ende zutreffen. Zu dem Überbegriff der Kontrolle zählt auch der Projektfortschritt mit der Berechnung des Fertigstellungswertes, sprich dem Wert der erbrachten Leistungen zu einem Stichtag. Auch die integrierte Projektsteuerung ist unter diesem Punkt zu nennen. Sie betrachtet die Teilaspekte Planung, Überwachung und Steuerung als kontinuierlichen Prozess und hat als Hauptziel frühzeitig Abweichungen zu erkennen und Steuerungsmaßnahmen einzuleiten, wozu sie sich drei verschiedener Trendanalysen bedient. Die Meilenstein-Trendanalyse definiert Meilensteine und überprüft diese periodisch, um die jeweiligen Schätzungen in ein Diagramm einzutragen. Mit diesem sind Terminabweichungen und Abstimmungsdefizite auf einen Blick erkennbar, wodurch das Terminbewusstsein geschärft wird. Bei der Kosten-Trendanalyse wird regelmäßig der Schätzwert der Gesamtkosten berechnet, welcher als Frühwarnindikator dient und das Kostenbewusstsein schärft. Die Earned-Value-Analyse analysiert in drei Schritten verstrichene Zeit, verrichtete Arbeit und verursachte Kosten. Der dadurch entstehende Aufwand wird durch die hohe Aussagekraft der Analyse wieder ausgeglichen. Der letzte Punkt im Bereich Projektkontrolle ist das Risikomanagement. Die Risikoanalyse teilt sich auf in Identifikation, Bewertung und Dokumentation. Nachdem das Projekt beendet ist und der Projektabschluss durchgeführt wurde (alle Restarbeiten wurden erledigt, das Team wurde aufgelöst), muss das Projekt ausgewertet und dokumentiert werden, um daraus für zukünftige Projekte zu lernen. Das Fazit, das wir für die Projektgruppe ziehen können, ist die Wichtigkeit, im Vorfeld über die Ziele, die Vorgehensweise und die damit verbundenen Risiken nachzudenken. Außerdem ist es nützlich, regelmäßig zu hinterfragen, ob man auf dem richtigen Weg ist. Zusätzlich kann die Projektgruppe einige Methoden des Projektmanagements verwenden, wie zum Beispiel den Projektstrukturplan oder das Ablauf- und Terminmanagement.
Grundlagen des Vortrages sind die folgende Quellen [Burg 00], [Kopp 05], [TNG 99], [Zing 04].

2.8 Datenanalyse für Einzelsysteme sowie Vergleich alternativer Systeme

Eren Boncuklu

Datenanalyse für Einzelsysteme

Mit einer Simulation versucht man die richtigen Antworten für das entsprechende Modell zu bekommen. Das Problem der Simulation ist, dass es sich um stochastische Simulationsmodelle handelt und daher die Ergebnisse dem Zufall untergeordnet sind, mit der Konsequenz, dass die Ergebnisse aus verschiedenen Simulationsdurchgängen variieren können.
Aus diesem Grund sollte man bei der Datenanalyse auf angemessene statistische Methoden zurückgreifen, um Fehlinterpretationen zu vermeiden.
Wenn man versucht mit den Ergebnissen aus einem einzigen Simulationsdurchgang die richtigen Antworten für das Modell abzuleiten, hat man das Problem, dass die einzelnen Beobachtungen nicht identisch und unabhängig verteilt sind, so dass man die klassischen statistischen Methoden nicht direkt anwenden kann.
Daher sollte man n unabhängige Simulationsdurchläufe(Replikationen) durchführen, da die einzelnen Beobachtungen aus den verschiedenen Replikationen zu einander IID sind und man somit dann die klassischen Analyse Methoden anwenden kann.
Die Unabhängigkeit zwischen den Läufen ist der Schlüssel für die Datenanalyse.

Transientes und stationäres Verhalten von stochastischen Prozessen

Wenn man einen stochastischen output Prozess betrachtet, stellt man fest, dass die Verteilungsfunktionen der einzelnen Zufallsvariablen nicht identisch zu einander verteilt sind. Das hat zur Konsequenz, dass die Verteilungsfunktionen ein unterschiedliches Aussehen besitzen werden.
Es gibt zwei Verhaltensmuster von stochastischen Prozessen. Wenn die Verteilungsfunktionen für einen Bereich der Indizes i ein unterschiedliches Aussehen haben, dann nennt man diesen Bereich die transiente Phase. Wenn die Verteilungen für große i gegen eine gemeinsame Verteilung konvergieren, nennt man diese Verteilung die stationäre Verteilung.

Simulationstypen bzgl. der Auswertung

Abhängig vom Simulationstyp, wird man die entsprechenden Analyse Methoden anwenden.
Es gibt zwei Haupttypen von Simulationen:

- terminierende Simulation:
  - es existiert ein natürliches Ereignis E, dass die Länge des Simulationsdurchlaufes bestimmt. Hierbei kann das Ereignis entweder ein konkretes Ereignis oder ein genau festgelegter Zeitpunkt sein
- nicht-termierende Simulation:
  - es existiert kein natürliches Ereignis E, dass die Länge jeder Replikation spezifiziert.
  - in diesen Fällen ist man im Langzeit-Verhalten des Laufes interessiert:
    - wenn die Erfolgsmessgröße für solch eine Simulation die Charakteristik einer stationären Verteilung aufweist, nennt man es stationäre Größe

Statistische Analyse von terminierenden Simulationen

Man geht folgenderweise dabei vor. Zunächst führt man n unabhängige Replikationen der terminierenden Simulation durch. Dabei ist zu beachten, dass jede Replikation durch das selbe Ereignis E terminiert und jede Replikation mit den gleichen Anfangsbedingungen startet.
Die resultierenden Zufallsvariablen sind dann IID, so dass man die klassischen statistischen Methoden anwenden kann.
Man berechnet dann den Punktschätzer, Varianz und anschließend ein Konfidenzintervall für den Erwartungswert μ.
Ein Problem bei der Analyse der terminierenden Simulation ist der, dass der Initial-Zustand das Leistungsmaß beeinflussen kann. Daher sollte man die Simulation angemessen initialisieren.
Es gibt zwei Möglichkeiten wie man den initialen Zustand bestimmen kann. Entweder man lässt das System früher in einem natürlichen Anfangszustand starten und simuliert dann bis zum Startzeitpunkt, oder man ermittelt per Messung mögliche Anfangszustände und deren Wahrscheinlichkeiten und zieht dann ein Anfangszustand aus der Verteilungsfunktion.

Statistische Analyse stationärer Simulationen

Die Analyse ist nicht so einfach wie bei terminierenden Simulationen, aufgrund des Startup- Problems (Problem der transienten Phase).
Das Problem was sich hinter dem Startup-Problem verbirgt ist, dass die ersten Beobachtungen aus der transienten Phase den Erwartungswert verfälschen. Eine mögliche Lösung für dieses Problem nennt sich warming up the Model oder auch inital-data Deletion genannt. Die Idee dabei ist, die ersten Beobachtungen aus der Anfangsphase zu löschen, bis die Werte der Beobachtungen nicht mehr so stark auseinander driften. Und mit den restlichen Beobachtungen dann den Erwartungswert zu ermitteln.
Um die warmup Periode l zu bestimmen, also den Zeitindex bis wohin man die Beobachtungen löscht, kann man die graphische Prozedur nach Welch benutzen.

Wenn man nun zur eigentlichen Analyse kommt, ist es das Ziel, einen Punktschätzer und ein Konfidenzintervall für den stationären Erwartungswert zu bestimmen.
Eine mögliche Methode dafür ist die Replikation/Lösch Methode. Hierbei ist die Analyse fast gleich wie bei der der terminierenden Simulationen, nur mit dem Unterschied, dass man nur die Beobachtungen nach der warmup Periode l zur Bestimmung des Erwartungswertes benutzt.
Es gibt weitere Methoden für die Bestimmungen des Erwartungswertes. Diese Methoden, im Gegensatz zu der Replikation/Lösch-Methode, führen nur noch einen einzigen langen Lauf durch. Eine mögliche Methode dazu nennt sich die Batch Means Methode. Bei dieser Methode führt man einen einzigen langen Lauf aus und unterteilt anschließend die Beobachtungen in n Gruppen der Größe k. Diese Gruppen werden dann batches genannt. Danach berechnet man für jeden Batch den Mittelwert, dann den Gesamtmittelwert, die Varianz über den Batchmittelwerten und zu guter letzt das Konfidenzintervall.

Vergleich alternativer Systeme

Die meisten Simulationsprojekte besitzen meisten mehr als ein Simulationsmodell. Daher ist es sinnvoll die einzelnen Modelle miteinander zu vergleichen, damit man die Leistungen der einzelnen Modelle besser beurteilen kann. Oder um die Korrektheit eines Modells nachzuweisen.

Konfidenzintervalle für den Unterschied zwischen Leistungsmaßen zweier Systeme

Damit man zwei Systeme miteinander vergleichen kann, bildet man ein Konfidenzintervall für den Unterschied der beiden Erwartungswerte. Und anschließend analysiert man das Konfidenzintervall folgendermaßen. Man schaut nach, ob das Intervall die Null enthält. Wenn ja, dann unterscheiden sich die Systeme nicht voneinander. Wenn das Intervall die Null verfehlt, dann unterscheiden sich die Systeme von einander.
Um das Konfidenzintervall zu bilden gibt es zwei Methoden:

- Paired-t Konfidenzintervall:
  - wenn die Anzahl der Beobachtungen/Replikationen der beiden Modelle gleich sind, paart man die Beobachtungen beider Modelle miteinander, indem man die Differenz bildet. Die dabei neu entstandenen Variablen sind IID Zufallsvariablen, so dass man die gleichen Methoden wie bei der terminierenden Simulation anwenden kann.
- Modifiziertes Two-Sample Konfidenzintervall:
  - paart die Beobachtungen nicht miteinander, unter der Vorraussetzung, dass die Beobachtungen der beiden Modelle unabhängig sind, so dass die Anzahl der Replikationen nun unterschiedlich sein kann. Man berechnet für jedes Modell den Mittelwert und Varianz, und bildet anschließend die Mittelwertdifferenz und die Standardabweichung davon und konstruiert dann daraus das Konfidenzintervall.

Konfidenzintervalle für den Vergleich mehrerer Systeme

Für den Vergleich mehrerer alternativer Systeme benutzt man die beiden vorgestellten Methoden zur Konfidenzintervallbildung.
Je nachdem, wie viele Systeme existieren, wird man dementsprechend mehrere Konfidenzintervalle erzeugen. Die Analyse dabei ist wie gehabt, man schaut nach ob die Intervalle die Null enthalten oder nicht.
Es gibt zwei Möglichkeiten wie man mehrere Systeme miteinander vergleichen kann:

- Vergleich mit dem Standard:
  - bei k Systemen ist ein System der Standard und die anderen $k-1$ Systeme die Alternativen. Man vergleicht dann das Standard System mit jedem alternativen System, so dass man hier für $k-1$ Konfidenzintervalle erzeugen muss.
- Paarweise Vergleich:
  - bei k Systemen ohne Standard, vergleicht man jedes System mit jedem, so dass man $k(k-1)/2$ Konfidenzintervalle erzeugen muss.
  - Bemerkung: wenn z.B. System 1 und 2 sich von System 3 sich nicht unterscheiden, kann es trotzdem sein, dass sich System 1 von System 2 unterscheidet.

Grundlagen des Vortrages sind die Kapitel 9 und 10 aus [Law 91].

2.9 Experiment-Design

Akansel Serefliogl u

Beim Experiment-Design geht es um Optimierungstechniken für das Ausführen von einem Experiment in einem Simulationsmodell. Es soll jenes Modell bestimmt werden, welches zu dem besten Ergebnis führt. Die eingehenden Parameter werden „Faktoren“ genannt, und die Ausgabe bezeichnet man als „Antwort“. Experiment-Design beschafft einen Weg zum Entscheiden von Konfigurationen vor dem Start, so dass die gewünschten Informationen mit der geringsten Anzahl an Simulationen erreicht werden können. Einige Faktoren und Informationen sind am Anfang eines Experiments von Nutzen, wenn man eine Vorstellung davon hat, welche Faktoren wichtig sind und wie sie die Antwort beeinflussen können. Je mehr man aus dem Verhalten des Modells lernen und Schlussfolgern kann, desto präziser wird die Festlegung des Ziels. Die so genannten Techniken, namens Metamodellierung und Antwortflächenverfahren, können für das Erzielen der erwähnten Ziele benutzt werden. Das Ziel einer Metamodellierung ist es, die Antwortfläche zu berechnen oder sich ihnen zu nähern. Zudem bestimmt man, ob die Faktoren untereinander interagieren, z.B. ob die Faktoren untereinander abhängig sind. Ein Weg zur Abschätzung des Effekts eines Faktors ist die Bestimmung der Level und dann entsprechende Simulationsabläufe für alle verschiedenen Level der Faktoren zu machen, um zu sehen, wie die Antwort den Faktor verändert. Ein ökonomischer Vorgang mit dem man auch Interaktionen abschätzen kann, wird als „$2k$ faktorieller Entwurf“ bezeichnet. Die dabei entstehenden Simulationsabläufe für die $2k$ möglichen Faktorstufen-Kombinationen werden auch Entwurfspunkte genannt. Es wird deutlich, dass das Aufschreiben einer solchen Aufzählung, welches auch Entwurfs-Matrix genannt wird, die Kalkulationen der Faktorwirkungen und die Interaktionen erleichtert. Ein $2k$ faktorieller Entwurf kann schnell unüberschaubar werden. Wenn man eine große Zahl von Faktoren vorliegen hat, wird geguckt, welche Faktoren wichtig zu sein scheinen. Die Faktoren, die irrelevant sind, brauchen nicht weiter betrachtet werden. Außerdem können Faktoren, die den gleich Effekt haben, was individuell eingeschätzt werden muss, in einer Gruppe zusammengefasst werden, so dass man nicht jeden Faktor einzeln betrachten und analysieren muss. Das erspart natürlich sehr viel Simulationszeit. Dadurch würde man auch eine Maßnahme für die Reduzierung der Anzahl der Faktoren finden. Eine Simulation ist nur eine Funktion, die durch Vektoren oder durch Wahrscheinlichkeiten dargestellt wird. Die Methode der Antwortflächenberechnung ist eine wichtige Methode, die einfache Formeln entwickelt und sich dieser Funktion annähert. Diese Formel kann dann als Stellvertreter für die ganze Simulation genutzt werden, um eine grobe Idee davon zu bekommen, was bei einer großen Anzahl von Kombinationen der Eingabe-Parameter passieren könnte. Ein anderes Ziel ist es, herauszufinden, wie Änderungen in den Einsatzparametern die Ausgabemaße der Leistung beeinflussen. Wenn die Parameter stetig variieren, stellt man der erwarteten Antwortfunktion in Bezug auf die Einsatzparameter im Grunde genommen eine Frage über die Teilableitungen. Der Vektor von diesen Teilableitungen wird als „Gradient“ bezeichnet. In der Gradientenberechnung wird eine Richtung oder ein Vektor angegeben, in der man sich in dem Antwortflächenmodell bewegen müsste, um die optimale Lösung zu bekommen.
Die folgenden Quellen wurden von mir für den Vortrag benutzt [Law 91], [Müll 01].

2.10 Prozesskettenparadigma

Rainer Fels

Das allgemeine Prozesskettenparadigma beschäftigt sich mit der prozessorientierten Darstellung und Analyse einer gegebenen Aufgabenstellung. Es unterstützt dabei ein Vorgehen „vom Groben ins De-

tail". Bei diesem ist es möglich, eine gegebenen Aufgabenstellung auf verschiedenen „Hierarchie-Ebenen" so zu modellieren, dass jede Ebene getrennt von den Anderen betrachtet werden kann. Es wird somit versucht eine bessere Verständlichkeit zu gewährleisten und die Analyse zu vereinfachen. Im Wesentlichen modelliert das allgemeine Prozesskettenparadigma eine konkrete Aufgabenstellung als eine zeitliche Abfolge von Prozessen, die, gemäß der Aufgabenstellung, bestimmte Aktivitäten durchführen. Eine Aktivität wird dabei durch ein so genanntes Prozesskettenelement dargestellt. Die Verkettung bzw. die Verbindung von den verschiedenen Prozesskettenelementen eines Prozesses führt zu einer Prozesskette, welche diesen Prozess schließlich modelliert. Da kompliziertere Aufgabenstellungen das Vorhandensein von mehreren Prozessen implizieren, ist es möglich, Prozessketten miteinander zu einem so genannten Prozesskettenplan zu verbinden, welcher dann die Beziehungen bzw. die Abhängigkeiten der einzelnen Prozessketten mit graphischen und textuellen Mitteln darstellt. Diese Darstellung wird entscheidend durch den ProC/B-Modellformalismus geprägt. Dieser stellt eine konkretisierte Teilmenge des allgemeinen Prozesskettenparadigma dar. D.h., der ProC/B-Formalismus unterstützt bzw. erfüllt alle Anforderungen, die das allgemeine Prozesskettenparadigma an eine prozessorientierte Beschreibungssprache stellt. Er beschreibt dabei die wesentlichen Modellierungskonstrukte und deren Semantik, ohne die eine maschinen gestützte Analyse eines entwickelnden Modells nicht möglich wäre. Das Ziel des ProC/B-Modellformalismus ist es nun eine Modellbeschreibung einer Aufgabenstellung zu konstruieren, die direkt analysierbar sein soll, ohne erst eine Übersetzung in eine andere Beschreibungssprache vornehmen zu müssen.
Grundlagen des Vortrages sind: [Beil 99], [Käpp 02].

2.11 Das ProC/B-Toolset

Michael Thüner

Einführung

Das ProC/B-Toolset wurde während einer Zusammenarbeit der Fachbereiche Maschinenbau, Betriebswirtschaftslehre, Statistik und Informatik der Universität Dortmund und dem Fraunhofer Institut für Materialfluss und Logistik am Fachbereit Informatik entwickelt. Das Toolset verbindet die bekannten Analysetechniken des IT-Bereichs mit dem Anwendungsgebiet logistischer Systeme. Als genereller Rückzugspunkt für die Analyse wird die Simulation angesehen, daneben existieren aber Möglichkeiten zur Analyse von Warteschlangensystemen und numerischen Markovketten. Außerdem bietet es eine speziell für das Anwendungsgebiet wichtige Möglichkeit der Kostenanalyse. Als Modellierungsparadigma werden Prozessketten zugrunde gelegt.

Prozessketten in ProC/B

Das ProC/B-Paradigma unterscheidet zwei Hierarchien, einmal die Strukturhierarchie und die Verhaltenshierarchie. Der Strukturteil beschreibt was im Modell existiert und woraus sich die einzelnen Teile des Modells zusammensetzen, während der Verhaltensteil mit Hilfe von Prozessketten die einzelnen Verhaltensweisen und Abläufe innerhalb des Systems darstellt. Die Verbindung dieser beiden Hierarchien besteht darin, das für jede potentiell Zeit verbrauchende Aktivität genau spezifiziert werden muss, durch welchen Dienst, welcher Funktionseinheit diese ausgeführt wird, also wo in der Struktur die Aktivität stattfindet.

Analyse der Prozessketten

Das ProC/B-Toolset besteht aus einer grafischen Oberfläche, in der in der oben beschriebenen Weise Modelle erstellt werden können und verschiedenen Wandlermodulen, die diese Modelle für die Analyse übersetzen und über eingebundene andere Toolboxen ausführen können. Momentan existieren Wandler für HIT und die APNN-Toolbox, wodurch mit ersterem die Simulation und die Analyse von Warteschlangensystemen und mit dem zweiten die Analyse von zeitkontinuierlichen numerischen Markovketten ermöglicht wird. Um ein mit ProC/B angefertigtes Modell zu simulieren, wird dieses Modell in HI-SLANG übersetzt und anschließend über HIT ausgeführt. Die Vorteile einer Analyse von Warteschlangensystemen liegen in den frühen Designphasen, bei denen die Einschränkungen, die Warteschlangennetze mit sich bringen noch nicht so wichtig sind und die vielen Parametervariationen in den noch groben Modellen schnell ausgewertet werden können. Die Übersetzung der Modelle in zeitkontinuierliche numerische Markovketten erfolgt über einen kleinen Umweg über generalisierte stochastische Petrinetze. Zwar bringt auch dieser Umweg einige zusätzliche Beschränkungen mit sich, dafür sind aber auch Analysetechniken für Petrinetze anwendbar, die bei einer direkteren Übersetzung wegfallen würden.

Kostenrechnung

Wie schon erwähnt, bietet das ProC/B-Toolset auch die Möglichkeit einer Kostenanalyse des Modells. Diese ist für logistische Systeme oft von großer Bedeutung. Diese Kostenrechnung basiert auf den folgenden Voraussetzungen:
Kostenwerte beeinflussen das dynamische Verhalten des Models/Systems nicht; Andernfalls müssten Kostenbegriffe in das Model selbst eingebunden werden. Die analysierten Modelle werden stationär und die benutzen Leistungsergebnisse beziehen sich auf die stationäre Phase; dadurch erhalten Maße wie Kosten pro Funktionszeiteinheit eine eindeutige Bedeutung; alle Kostenwerte sind im folgenden als Erwartungswerte zu verstehen. Jede berücksichtigte Kosteneinheit ergibt sich aus der Existenz und/oder Operationen einer bestimmten FU; dies ist eine pragmatische, jedoch realistische Annahme, da jede FU einen Teil der Systemstruktur darstellt.
Grundlage dieses Vortrages sind [Baus 02], [Eick 02], [Hier 05].

2.12 Bottleneckanalyse

Viktor Faber

Die Gesamtleistung eines Systems wird durch die Kapazität seiner Einzelmaschinen beschränkt. Abhängig von der Art des Systems haben einige Maschinen einen größeren Einfluss auf die Gesamtleistung als andere. Üblicherweise können die Beschränkungen des Gesamtsystems auf die Beschränkungen von einer oder zwei Maschinen zurückgeführt werden. Allgemein werden diese auch Engpässe genannt.
Wie ein enger Flaschenhals, der den Durchfluss massiv beeinträchtigt, wirkt sich ein Engpass in einem Unternehmen oder einer selbstständigen Existenz auf die erfolgreiche Entwicklung aus. Aber wie findet sich der zentrale Engpass, dessen Vorhandensein sehr viel Energie verschlingt und das Vorwärtskommen lähmt. Wie z.B. das Abstimmen der Leistungsparameter aller Stationen verketteter Fertigungs- und Montagesysteme, um eine geplante Kapazität sicherzustellen. Das Wachstum einer Pflanze richtet sich nach dem Nährstoff, den sie am wenigsten besitzt. Es gilt sehr weitgehend in gleicher Weise für wirtschaftliche Wachstums- und Entwicklungsprozesse. Es ist also entscheidend

wichtig, Engpässe zu finden und seine Kräfte auf diese zu konzentrieren. Dadurch eröffnen sich für ein Unternehmen Möglichkeiten die Treffsicherheit ihrer mittel- und langfristigen Planung zu verbessern, Zielschwankungen frühzeitig zu entdecken und den Erfolg durch Simulation verschiedener Szenarien zu optimieren. Es gibt zahlreiche Tools und Methoden zur Ermittlung des Engpasses.
Bei der Engpassanalyse von Produktionssystemen mit Kennlinien wird bei einem System das aus mehreren Maschinen besteht, zunächst der Einfluss der technischen Verfügbarkeit auf die Ausbringungsleistung des Gesamtsystems mathematisch abgebildet und visualisiert, um so die Basis für ein vereinfachtes Verfahren zur dynamischen Analyse von Produktionssystemen zu schaffen. Mittels einer Kennlinie wird dann der Verlauf der Nutzungsgraderhöhung in Abhängigkeit von der Verfügbarkeitsverbesserung des jeweiligen Engpasses dargestellt.
Eine andere, neuartigeren Methode zur Ermittlung des Engpasses in einem diskreten Ereignis-System bestimmt den Engpass anhand der durchschnittlichen aktiven Laufzeit einer Maschine. Bei einem System das aus mehreren Maschinen besteht wird die durchschnittliche aktive Laufzeit jeder einzelnen Maschine bestimmt. Die Maschine mit der längsten durchschnittlichen aktiven Laufzeit ist mit sehr hoher Wahrscheinlichkeit der Engpass des Gesamtsystems.
Zur Erkennung eines sog. *Moving Bottleneck* (Bsp. Engpass im Verkehr auf einer Schnellstrasse) werden hochauflösende grafische Darstellungen verwendet, die ganz genau anzeigen, wo und zu welchem Zeitpunkt ein Engpass entsteht. Diese entstehen aus zahlreichen experimentellen Beobachtungen.
Eine andere Diagnose und Behandlung von Engpässen in Computersystemen verwendet die Entscheidungstheorie als Technik. Die Techniken beruhen auf einem Modell der Wechselwirkung zwischen Anwendungsauslastungen, dem Betriebssystem und System-Hardware. Wichtig hierbei ist nicht nur die Ermittlung des Engpasses, sondern auch die Suche nach seinen Ursachen. Die wahrscheinlichkeitstheoretische Charakterisierung zusammen mit der Statistik eines Fehlermodels führt zu einer schnellen numerischen Suche nach der wahrscheinlichsten Ursache. Die Kenntnis über die Ursachen des Engpasses liefert eine Empfehlung eines kosteneffizienten Hardwareupgrades um den Engpass zu beseitigen.
Grundlage dieses Vortrages sind folgende Quellen [Blak 95], [Muno 02], [Pete 01], [Wien 02].

3 Modellierungspraktikum

Das Modellierungspraktikum dient der Einarbeitung in die Werkzeuge HIT und ProC/B. Es wurden fünf Aufgaben bearbeitet, wobei die erste und die dritte Aufgabe von jedem einzelnen anhand zweier Tutorials bearbeitet wurden. Aus diesem Grund wurden die beiden Aufgaben auch nicht dokumentiert. Für die weiteren drei Aufgaben des Modellierungspraktikums wurden drei Untergruppen gebildet, die sich wie folgt zusammensetzen:

Gruppe 1: Boncuklu, Faber, Ketteltasche, Lohmeier

Gruppe 2: Deniz, Mai , Özdemir, Serefliogl u

Gruppe 3: Beer, Fels, Glasirin, Thüner

Die Darstellung der Gruppenergebnisse in Aufgabe 4 erfolgt in geänderter Reihenfolge, da sich die Gruppen 1 und 2 auf Inhalte der Gruppe 3 beziehen.

Weiterhin beziehen sich alle drei Gruppen bei der Aufgabe 2 auf dasselbe Modell, wobei jede Gruppe mit anderen Parametern rechnet. Die Abbildung 3.1 zeigt dieses sehr einfache Modell der Universität Dortmund. Der Dienst *jahrgang* ruft mit der jeweiligen Bedien-Anforderung den used service *studieren* mit dem Befehl *studieren(negexp(..))* auf, der von dem Server *UNI* bereitgestellt wird. Die Bediengeschwindigkeit wird in der Kontrollprozedur des Servers durch Festlegen des Parameters *speed* eingestellt. Die Ankunftsrate der Studenten wird in dem CREATE-Statement des Activities-Blocks umgesetzt.

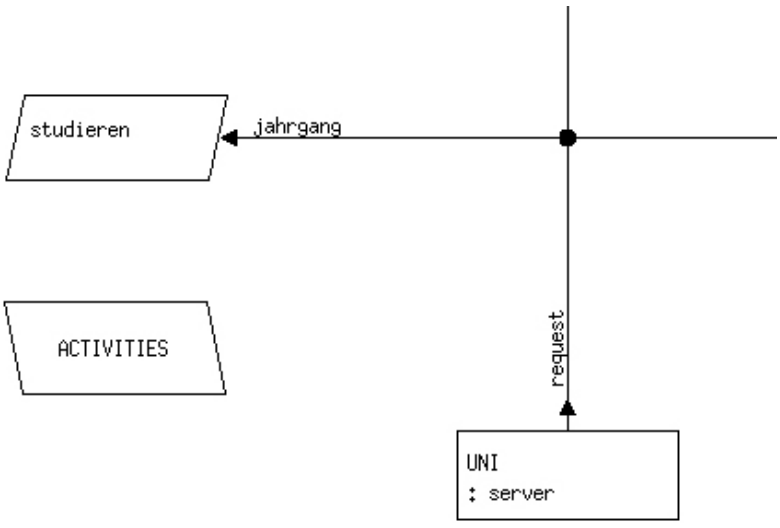


Abbildung 3.1: Einfaches Modell der Universität Dortmund

3.1 Aufgabe 2 Gruppe 1

Aufgabenstellung: An der Universität Dortmund kommen Studierende mit einer Ankunftsrate λ=1/2 Jahrgänge/Semester an. Jeder Jahrgang möchte das Studium nach 8 Semestern abgeschlossen haben. Die Bedien-Geschwindigkeit des Systems sei 9/2 Semester/Semester. Wir wollen das System als M/M/1-System modellieren.

1. Berechne die mittlere Auslastung, die mittlere Durchlaufzeit und den mittleren Durchsatz mittels Formeln für ein M/M/1-System mit einer Processorsharing-Bedienstrategie.

   Die Ankunftsrate :

   $$\lambda = \frac{1}{2} \quad \frac{Jahrgang}{Semester}$$

   Die gewünschte Dauer :

   $$8 \ Semester$$

   Die Bedien-Geschwindigkeit :

   $$\frac{9}{2} \quad \frac{Semester}{Semester}$$

   Die Bedienrate :

   $$\mu = \frac{9}{2} * \frac{1}{8} = \frac{9}{16} \quad \frac{Jahrgang}{Semester}$$

   Die mittlere Auslastung :

   $$\rho = \frac{\lambda}{\mu} = \frac{1/2}{9/16} = \frac{8}{9}$$

   Die mittlere Anzahl im System :

   $$E(n) = \frac{\rho}{1-\rho} = \frac{8/9}{1-(8/9)} = 8$$

   Die mittlere Durchlaufzeit :

   $$E(r) = \frac{1/\mu}{1-\rho} = \frac{\frac{1}{9/16}}{1-\frac{8}{9}} = 16\frac{Semester}{Jahrgang}$$

   Der mittlere Durchsatz :

   $$\frac{E(n)}{E(r)} = \frac{8}{16\frac{Semester}{Jahrgang}} = 0.5\frac{Jahrgang}{Semester}$$

Bereits anhand der viel zu hohen mittleren Auslastung kann man sehen, dass das Studium nicht in den gewünschten acht Semester zu bewerkstelligen ist. Dies wird durch die mittlere Durchlaufzeit belegt, welche 16 Semester pro Jahrgang beträgt. Da wir einen mittleren Durchsatz von einem halben Jahrgang pro Semester vorliegen haben, befinden sich durchschnittlich acht Jahrgänge im System.

2. Entwickle ein HIT-Modell des M/M/1-System und ermittle die o.a. Leistungsmasse mit Hilfe der Lösungsverfahren **DOQ4**, **NUMERICAL** und **SIMULATIVE**.
   Die Werte, die sich bei den drei angewendeten Lösungsverfahren ergeben, sind in den folgenden Tabellen zusammengefasst. Dabei fällt auf, dass die Werte von DOQ4 und NUMERICAL mit den theoretischen Lösungen übereinstimmen. Im Gegensatz dazu, zeigt die Simulation eine geringe Abweichung.

   Das Lösungsverfahren DOQ4:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	8.000000	0.500000	16.000000	0.888889

   Das Lösungsverfahren Markov:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	8.000000	0.500000	16.000000	0.888889

   Das Lösungsverfahren Simulative:

Model Parameters	Name	Type	Actual Value
	seed	INTEGER	13

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	7.430828	0.499960	14.862126	0.878092
	Stdev	7.946040	2.002345	23.804715	0.327179
	Con	7.430828±11.82%	0.49960±0.91%	14.862126±6.93%	0.878092±0.80%
	95 %	10 9	10 0	10 10	10 10

3. Entwickle ein HIT-Experiment, in dem für das „simulative“ M/M/1-Modell der Einfluss des Startwertes des verwendeten Zufallszahlengenerators für die Startwerte 17, 19 und 127 untersucht werden kann.
   An dieser Stelle wird erneut eine Simulation durchgeführt, wobei untersucht wird, welchen Einfluss unterschiedliche Startwerte des Zufallszahlengenerators auf das Ergebnis haben. Um diese Aufgabe zu realisieren, wird der Body-Code des Experiments wie folgt gestaltet:

```
FOR d := 17, 19, 127
    LOOP
        EVALUATE UniDo_eva;
    END LOOP;
```

Die Simulation mit dem Startwert 17:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	8.211495	0.499310	16.445982	0.887426
	Stdev	8.967201	2.003028	27.090717	0.316072
	Con	8.211495±13.50%	0.499310±0.78%	16.445982±7.55%	0.887426±0.74%
	95 %	10 10	10 5	10 10	10 10

Die Simulation mit dem Startwert 19:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	8.546829	0.495640	17.243325	0.886561
	Stdev	9.459321	2.011909	28.848706	0.317129
	Con	8.546829±14.69%	0.495640±0.90%	17.243325±7.27%	0.886561±0.81%
	95 %	10 10	10 0	10 10	10 10

Die Simulation mit dem Startwert 127:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	8.205089	0.496070	16.540436	0.878744
	Stdev	10.699408	2.038418	31.620750	0.326424
	Con	8.205089±19.55%	0.946070±0.89%	16.540436±9.99%	0.878744±0.80%
	95 %	10 8	10 3	10 10	10 10

	Konfidenzintervall			
	Untergrenze	Obergrenze	Theoretischer Wert	liegt im Intervall
seed-Wert=13				
Population	6,55250413	8,3091519	8	**ja**
Throughput	0,49505364	0,5041464	0,5	**ja**
Turnaroundtime	13,83218067	15,892071	16	**nein**
Utilization	0,871067264	0,8851167	0,888888889	**nein**
seed-Wert=17				
Population	7,102943175	9,3200468	8	**ja**
Throughput	0,496031538	0,5038305	0,5	**ja**
Turnaroundtime	15,20431036	17,687654	16	**ja**
Utilization	0,880859048	0,893993	0,888888889	**ja**
seed-Wert=19				
Population	7,29129982	9,8023582	8	**ja**
Throughput	0,49117924	0,5001008	0,5	**ja**
Turnaroundtime	15,98973527	18,496915	16	**ja**
Utilization	0,879379856	0,8937421	0,888888889	**ja**
seed-Wert=127				
Population	6,600994101	9,8091839	8	**ja**
Throughput	0,937649977	0,95449	0,5	**nein**
Turnaroundtime	14,88804644	18,192826	16	**ja**
Utilization	0,871714048	0,885774	0,888888889	**nein**

Abbildung 3.2: Auswertung der Konfidenzintervalle

Auswertung der Konfidenzintervalle
Die Auswertung der Konfidenzintervalle (siehe Abb. 3.2) zeigt, dass bis auf vier Werte alle theoretisch ermittelten Werte innerhalb der durch die Simulationen ermittelten Konfidenzintervalle liegen. Man sieht hier den Einfluss des Seed-Wertes auf die Ergebnisse einer Simulation. Die wenigen Werte außerhalb der Intervalle, liegen aber auch recht nahe an diesen, so dass die Ergebnisse der Simulationen allgemein als sehr ordentlich bewertet werden können.

3.2 Aufgabe 2 Gruppe 2

Aufgabenstellung: An der Universität Dortmund kommen Studierende mit einer Ankunftsrate λ=1/2 Jahrgänge/Semester an. Jeder Jahrgang möchte das Studium nach 8 Semestern abgeschlossen haben. Die Bedien-Geschwindigkeit des Fachbereichs sei 9/2 Semester/Semester. Wir wollen den Fachbereich als M/D/1-System modellieren.

1. Berechne die mittlere Auslastung, die mittlere Anzahl im System, die mittlere Durchlaufzeit und den mittleren Durchsatz mittels der Formeln für ein M/D/1-System mit einer FCFS- Bedienstrategie.

   Die Ankunftsrate:
   $$\lambda = \frac{1}{2} \ \frac{Jahrgang}{Semester}$$

   Die gewünschte Dauer :
   $$8 \ Semester$$

   Die Bedien-Geschwindigkeit :
   $$\frac{9}{2} \ \frac{Semester}{Semester}$$

   Die Bedienzeit per Jahrgang :
   $$E(s) = \frac{1}{Bedien - Geschwindigkeit} * 8 = \frac{16}{9} \ \frac{Semester}{Semester}$$

   Die mittlere Auslastung:
   $$\rho = \lambda E(s) = \frac{8}{9} \approx 0,889$$

   Die mittlere Anzahl im System :
   $$E(n) = \rho + \rho^2/[2(1-\rho)] \approx 4,449 \ \ Jahrgänge$$

   Die mittlere Durchlaufzeit :
   $$E(r) = E(s) + \rho E(s)/[2(1-\rho)] = \frac{80}{9} \approx 8,89 \ \ Semester$$

   Der mittlere Durchsatz :
   $$weil \ \ \rho < 1 \ \ ist, \ \ Durchsatz = \lambda = 0,5$$

2. Entwickle ein HIT-Modell des M/D/1-Systems und ermittle die o.a. Leistungsmasse mit Hilfe des Lösungsverfahrens **SIMULATIVE**. Bei der Simulation sind insbesondere die Konfidenzintervalle der Leistungsmasse interessant und zu diskutieren.

   Man sieht, dass je länger man die Simulation laufen lässt, die Annäherung an die theoretischen Ergebnisse immer größer wird. Wie gut die Ergebnisse sind, zeigt die Betrachtung der Konfidenzintervalle. Die starke Schwankung während der Simulation wird deutlich und man kann zu dem sehen, wie verlässlich die Simulationsergebnisse sind.

   Das Lösungsverfahren Simulative

Model Parameters	Name	Type	Actual Value
	seed	INTEGER	13

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	4.010246	0.507400	7.872595	0.902001
	Stdev	3.586341	0.819256	6.160541	0.297313
	Con	4.010246±18.81%	0.507400±1.82%	7.872595±18.37%	0.902001±1.83%
	95 %	10 10	10 7	10 3	10 10

   Die exakten Werte der theoretischen Ergebnisse liegen in den berechneten Konfidenzintervallen.

3. Entwickle ein HIT-Experiment, in dem für das M/D/1-Modell der Einfluss des Startwertes des verwendeten Zufallszahlengenerators für die Startwerte 17, 19, 127 untersucht werden kann.

   Das erstellte Experiment zeigt den Einfluss des Initialwertes für den Zufallsgenerator auf die Simulationsergebnisse. Dies macht deutlich, wie wichtig es ist, auch bei der Wahl des Seed-Wertes eine gewisse Sorgfalt walten zu lassen, bzw. diesen auch bei der Analyse der Ergebnisse einzubeziehen.

   Die Simulation mit dem Startwert 17:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	4.810028	0.522900	9.378218	0.911767
	Stdev	4.106552	0.793609	7.087650	0.283633
	Con	4.810028±20.46%	0.512900±2.00%	9.378218±19.63%	0.911767±1.94%
	95 %	10 10	10 8	10 8	10 10

   Bis auf den Durchsatz und die Auslastung, die sehr gering abweichen, liegen die Werte in den Konfidenzintervallen.

Die Simulation mit dem Startwert 19:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	4.422749	0.496100	8.911491	0.881816
	Stdev	4.218593	0.922000	7.344698	0.322826
	Con	4.422749±25.58%	0.496100±2.25%	8.911491±23.63%	0.881816±1.19%
	95 %	10 7	10 10	10 5	10 10

Die exakten Werte der theoretischen Ergebnisse liegen in den berechneten Konfidenzintervallen.

Die Simulation mit dem Startwert 127:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	5.310178	0.511500	10.380011	0.909329
	Stdev	4.755323	0.827899	8.269007	0.287153
	Con	5.310178±24.81%	0.511500±1.88%	10.380011±23.34%	0.9093320±1.82%
	95 %	10 8	10 9	10 3	10 9

Auswertung der Konfidenzintervalle:
Die Auswertung der Konfidenzintervalle liefert folgende Ergebnisse. Bis auf die Messwerte *THROUGHPUT* und *UTILIZATION* der Seed-Werte 17 und 127 liegen die theoretischen Werte in den Konfidenzintervallen. Bei den vier Ausnahmen weichen die theoretischen Werte sehr geringfügig von den Konfidenzintervallen ab. Es wird deutlich, dass die Auswahl des initialen Seed-Wertes einen erheblichen Einfluss auf die Simulationsergebnisse haben kann.

	Konfidenzintervall			
	Untergrenze	Obergrenze	Theoretischer Wert	liegt im Intervall
seed-Wert=13				
Population	3,255918727	4,764573	4,449	**ja**
Throughput	0,49816532	0,516635	0,5	**ja**
Turnaroundtime	6,426399299	9,318791	8,89	**ja**
Utilization	0,885494382	0,918508	0,888888889	**ja**
seed-Wert=17				
Population	3,825896271	5,79416	4,449	**ja**
Throughput	0,502642	0,523158	0,5	**nein**
Turnaroundtime	7,531646876	11,22479	8,89	**ja**
Utilization	0,89407872	0,929455	0,888888889	**nein**
seed-Wert=19				
Population	3,291409806	5,554088	4,449	**ja**
Throughput	0,48493775	0,507262	0,5	**ja**
Turnaroundtime	6,806596826	11,01639	8,89	**ja**
Utilization	0,87132239	0,89231	0,888888889	**ja**
seed-Wert=127				
Population	3,992722838	6,627633	4,449	**ja**
Throughput	0,5018838	0,521116	0,5	**nein**
Turnaroundtime	7,957316433	12,80271	8,89	**ja**
Utilization	0,892782158	0,925882	0,888888889	**nein**

Abbildung 3.3: Auswertung der Konfidenzintervalle

3.3 Aufgabe 2 Gruppe 3

Aufgabenstellung: Am Fachbereich Informatik der Universität Dortmund kommen Studierende mit einer Ankunftsrate λ=1/2 Jahrgänge/Semester an. Jeder Jahrgang möchte das Studium nach 8 Semestern abgeschlossen haben. Die Bedien-Geschwindigkeit des Fachbereichs sei 88/20 Semester/Semester. Wir wollen den Fachbereich als M/M/1-System modellieren.

1. Berechne die mittlere Auslastung, die mittlere Anzahl im System, die mittlere Durchlaufzeit und den mittleren Durchsatz mittels der Formeln für ein M/M/1-System mit einer Processorsharing-Bedienstrategie.

Die Ankunftsrate :

$$\lambda = \frac{1}{2} \quad \frac{Jahrgang}{Semester}$$

Die gewünschte Dauer :

$$8 \quad \frac{Jahrgang}{Semester}$$

Die Bedien-Geschwindigkeit :

$$\frac{88}{20} \quad \frac{Semester}{Semester}$$

Die Bedienrate :

$$\mu = \frac{88}{20} * \frac{1}{8} = 0.55 \quad \frac{Jahrgang}{Semester}$$

Die mittlere Auslastung :

$$\rho = \frac{\lambda}{\mu} = \frac{1}{2 * 0.55} = 0.90\overline{90}$$

Die mittlere Anzahl im System :

$$E(n) = \frac{\rho}{1-\rho} = \frac{0.9090\overline{90}}{1 - 0.9090\overline{90}} = 9.999890001$$

Die mittlere Servicezeit :

$$\overline{T_s} = \frac{1}{\mu} = \frac{1}{0.55} = 1.81\overline{81}$$

Die mittlere Wartezeit :

$$\overline{T_w} = \frac{\rho}{1-\rho} * \frac{\rho}{\lambda} = \frac{0.90\overline{90}}{1 - 0.90\overline{90}} * \frac{0.90\overline{90}}{0.5} = 18.18184$$

Die mittlere Durchlaufzeit :

$$\overline{T} = \overline{T_w} + \overline{T_s} = 1.81\overline{81} + 18.18184 = 20.0000$$

Der mittlere Durchsatz :

$$E(x) = \frac{E(n)}{\overline{T}} = \frac{9.999890001}{20.0000} = 0.4999945001$$

Die hier auf theoretischem Wege ermittelten Werte dienen dazu, in den folgenden Abschnitten die verschiedenen Lösungsverfahren zu bewerten und deren Ergebnisse zu verifizieren und zu beurteilen.

2. Entwickle ein HIT-Modell des M/M/1-System und ermittle die o.a. Leistungsmasse mit Hilfe der Lösungsverfahren **DOQ4**, **NUMERICAL** und **SIMULATIVE**.

   Im folgenden Abschnitt werden die Unterschiede der verschiedenen Lösungsverfahren näher beleuchtet. Man kann erkennen, dass die verschiedenen Lösungsverfahren aufgrund ihrer unterschiedlichen Ansätze doch zu leicht differenzierten Ergebnissen führen. Besonders deutlich wird dies beim simulativen Lösungsverfahren. Hier wird die größte Differenz zu den oben ermittelten theoretischen Ergebnisse deutlich. Je länger man allerdings die Simulation laufen lässt, desto größer wird die Annäherung an die theoretischen Ergebnisse. Wie gut die Ergebnisse der Simulation sind zeigt die Betrachtung der Konfidenzintervalle. Hier wird deutlich wie stark die Werte während der Simulation schwanken und wie verlässlich das Ergebnis der Simulation ist.

   Das Lösungsverfahren DOQ4:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	10.000000	0.500000	20.000000	0.909091

   Das Lösungsverfahren Markov:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	9.999998	0.500000	19.999995	0.909091

   Das Lösungsverfahren Simulative:

Model Parameters	Name	Type	Actual Value
	seed	INTEGER	13

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	10.356114	0.500631	20.686143	0.909395
	Stdev	10.891198	2.004065	33.934562	0.287047
	Con	10.356114±4.78%	0.500631±0.28%	20.686143±2.25%	0.909395±0.22%
	95 %	10 10	10 4	10 10	10 10

3. Entwickle ein HIT-Experiment, in dem für das ”simulative” M/M/1-Modell der Einfluss des Startwertes des verwendeten Zufallszahlengenerators für die Startwerte 17, 19, 127 untersucht

werden kann. Das erstellte Experiment zeigt sehr schön den Einfluss des Initialwertes für den Zufallsgenerator auf die Simulationsergebnisse. Dies macht deutlich wie wichtig es ist, auch bei der Wahl des Seed-Wertes eine gewisse Sorgfalt walten zu lassen, bzw. diesen auch bei der Analyse der Ergebnisse mit einzubeziehen.

Die Simulation mit dem Startwert 17:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	10.091478	0.498948	20.225193	0.907961
	Stdev	10.745351	2.007324	33.438439	0.289081
	Con	10.091478±4.78%	0.498948±0.27%	20.225193±2.32%	0.907961±0.21%
	95 %	10 10	10 6	10 10	10 10

Die Simulation mit dem Startwert 19:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	10.293618	0.500444	20.568797	0.909174
	Stdev	10.839358	2.000542	33.767223	0.287361
	Con	10.293618±4.90%	0.500444±0.27%	20.568797±2.26%	0.909174±0.23%
	95 %	10 10	10 0	10 10	10 10

Die Simulation mit dem Startwert 127:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	10.406247	0.499699	20.824652	0.909593
	Stdev	11.276449	2.007848	34.838896	0.286764
	Con	10.406247±4.71%	0.499699±0.25%	20.824652±2.20%	0.909593±0.20%
	95 %	10 10	10 0	10 10	10 10

	Konfidenzintervall			
	Untergrenze	Obergrenze	Theoretischer Wert	liegt im Intervall
seed-Wert=13				
Population	9,861091751	10,851136	9,999890001	ja
Throughput	0,499229233	0,5020328	0,4999945	ja
Turnaroundtime	20,22070478	21,151581	20	nein
Utilization	0,907394331	0,9113957	0,9090909	ja
seed-Wert=17				
Population	9,609105352	10,573851	9,999890001	ja
Throughput	0,49760084	0,5002952	0,4999945	ja
Turnaroundtime	19,75596852	20,694417	20	ja
Utilization	0,906054282	0,9098677	0,9090909	ja
seed-Wert=19				
Population	9,789230718	10,798005	9,999890001	ja
Throughput	0,499092801	0,5017952	0,4999945	ja
Turnaroundtime	20,10394219	21,033652	20	nein
Utilization	0,9070829	0,9112651	0,9090909	ja
seed-Wert=127				
Population	9,916112766	10,896381	9,999890001	ja
Throughput	0,498449753	0,5009482	0,4999945	ja
Turnaroundtime	20,36650966	21,282794	20	nein
Utilization	0,907773814	0,9114122	0,9090909	ja

Abbildung 3.4: Auswertung der Konfidenzintervalle

Auswertung der Konfidenzintervalle
Die Auswertung der Konfidenzintervalle (siehe Abb. 3.4) der einzelnen Simulationsläufe für die unterschiedlichen Seed-Werte offenbart die Qualität der Simulationsergebnisse. Es liegen bis auf drei Ausnahmen alle vorab ermittelten theoretischen Ergebniswerte in den durch die Simulationsläufe berechneten Konfidenzintervalle. Bei den drei Ausnahmen handelt es sich ausschließlich um drei der vier Werte für die zu erwartende Verweildauer, allerdings liegen hier die theoretischen Ergebnisse auch nur geringfügig außerhalb des entsprechenden Konfidenzintervalls. Hieran wird wiederum nochmals deutlich, welchen Einfluss die Auswahl des initialen Seed-Wertes auf die Simulationsergebnisse haben kann.

3.4 Aufgabe 4 Gruppe 3

Aufgabenstellung: Entwickle ein ProC/B-Simulationsmodell des M/M/1-Systems aus Aufgabe 2 (siehe Kapitel 3.3) und berechne die mittlere Auslastung, die mittlere Anzahl im System, die mittlere Durchlaufzeit und den mittleren Durchsatz. Bei einer Simulation sind insbesondere die Konfidenzintervalle der Leistungsmasse interessant und zu diskutieren. Vergleiche die Ergebnisse mit den Simulationsergebnissen aus Aufgabe 2.

Das Simulationsmodell in ProC/B

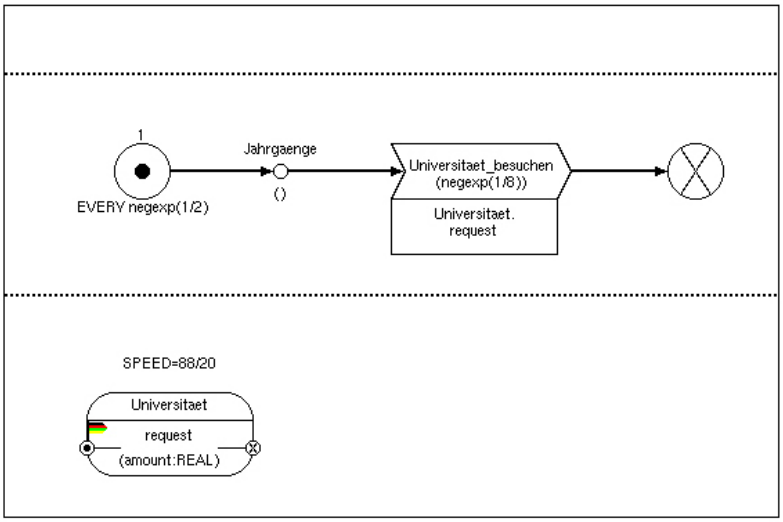


Abbildung 3.5: Modellumgebung in ProC/B

Das Modell (vgl. Abb. 3.5) besteht aus 5 Elementen: Quelle, ProzessID, Prozesskettenelement, Senke, Server.

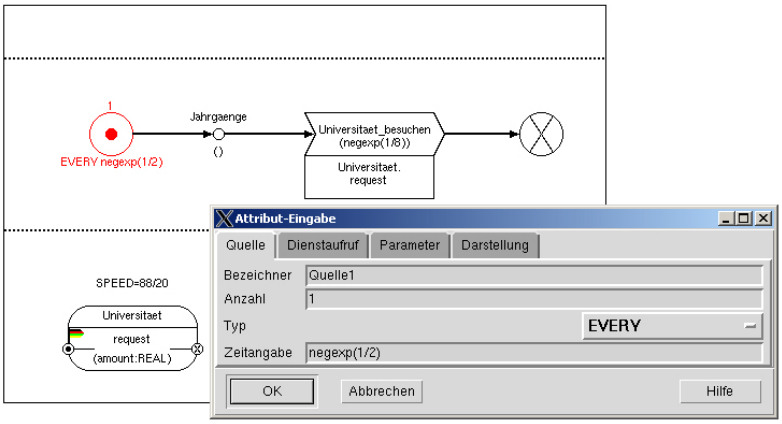


Abbildung 3.6: Quelle

Im Attributfenster der Quelle (Abb. 3.6) tragen wir die Verteilung der Zeitangabe *negexp(1/2)* ein, da die Ankunftsrate $\lambda = 1/2$ ist. Die Anzahl im Attributfenster setzen wir auf *1*, weil jedes Jahr immer ein Jahrgang eingeschrieben wird. Das ProzessID-Element nennen wir *Jahrgaenge*.

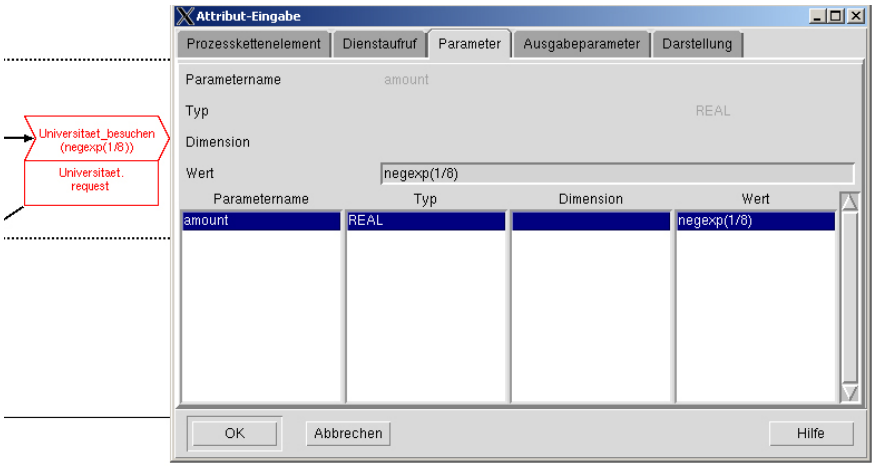


Abbildung 3.7: Prozesskettenelement

Im Attributfenster des Prozesskettenelementes (Abb. 3.7) tragen wir die Verteilung der Zeitangabe *negexp(1/8)* ein, da jeder Jahrgang das Studium nach 8 Semestern abgeschlossen haben möchte.

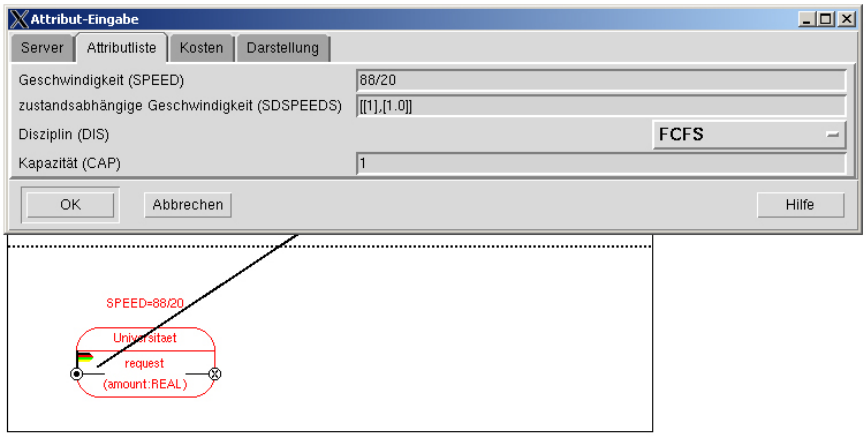


Abbildung 3.8: Server

Die Universität wird durch einen Server dargestellt, wo die Bediendisziplin auf FCFS und die Geschwindigkeit gleich *88/20* gesetzt werden (siehe Abb. 3.8).

Zuerst legen wir eine Experimentserie an, wo wir die mittlere Auslastung, die mittlere Anzahl im System, die mittlere Durchlaufzeit und den mittleren Durchsatz bestimmen wollen (siehe Abbildung 3.9).

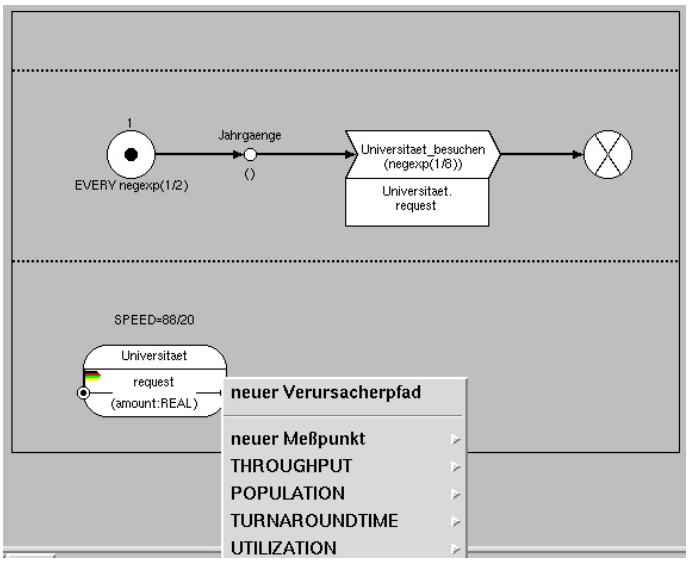


Abbildung 3.9: Experimentserie mit Messpunkten

Die Experimentserie

Die Experimentserie beginnt mit der Simulation des Modells mit dem Seed-Wert 13 und Modellzeit 10000000 Zeiteinheiten. Die Simulationsergebnisse mit diesem Seed-Wert werden in Abb. 3.10 graphisch dargestellt. An der Abbildung kann man deutlich erkennen das die Werte der Messströme *Turnaroundtime* und *Population* der Simulation zu Beginn stark schwanken. Dies ist durchaus üblich und normal. Erst im weiteren Verlauf pendeln sich die Ergebnisse der Messwerte auf einen bestimmten Wert ein. Da im Weiteren ein Vergleich mit den HIT-Ergebnissen gefordert war, werden die Ergebnisse der ProC/B-Simulation ebenfalls in der HIT-typischen Tabellenform (s.u.) präsentiert, dies soll einen Vergleich der Ergebnisse vereinfachen.

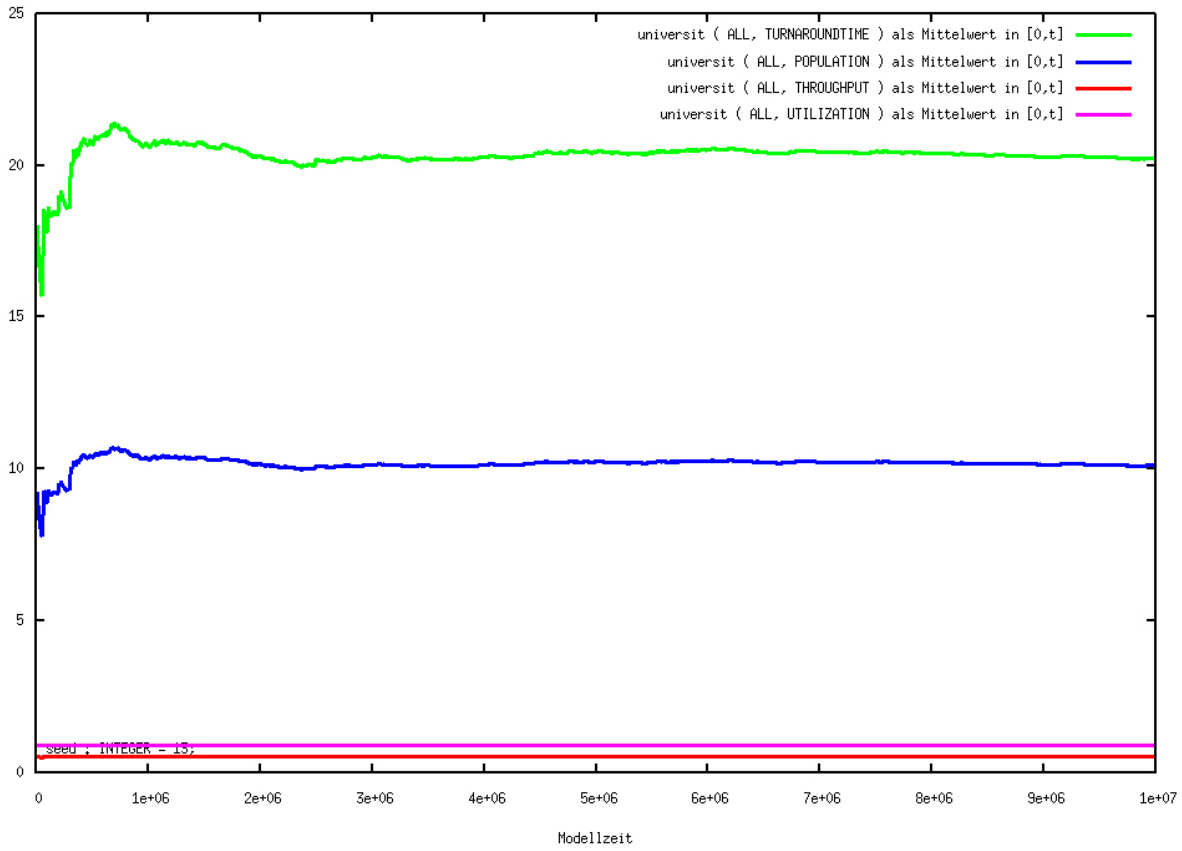


Abbildung 3.10: Die Simulationsergebnisse mit Seed-Wert 13

Die drei folgenden Simulationen werden mit den Seed-Werten 17, 19 und 127 durchgeführt, alle übrigen Einstellungen werden beibehalten. Auf eine graphische Darstellung der Ergebnisse wird verzichtet, da sich diese nur minimal von den bereits gezeigten unterscheiden. Die genauen Simulationsergebnisse aller Simulationsläufe finden sich in den folgenden Tabellen. Zusätzlich werden diese neu ermittelten Ergebnisse aus dem ProC/B-Modell den Ergebnissen aus dem HIT-Modell aus Aufgabe 2 gegenübergestellt, um die Modelle miteinander vergleichen zu können.

Die Simulation mit dem Startwert 13:

ProC/B Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	10.111044	0.500086	20.218599	0.909356
	Stdev	10.653765	1.999712	20.316907	0.287102
	Con	10.111044±1.27%	0.500086±0.07%	20.218599±1.26%	0.909356±0.07%
	90 %	10 10	10 0	10 10	10 10

HIT-Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	10.356114	0.500631	20.686143	0.909395
	Stdev	10.891198	2.004065	33.934562	0.287047
	Con	10.356114±4.78%	0.500631±0.28%	20.686143±2.25%	0.909395±0.22%
	95 %	10 10	10 4	10 10	10 10

Die Simulation mit dem Startwert 17:

ProC/B Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	9.937837	0.499670	19.888813	0.908681
	Stdev	10.448719	2.000621	19.930293	0.288062
	Con	9.937837±1.24%	0.499670±0.07%	19.888813±1.25%	0.908681±0.06%
	90 %	10 10	10 0	10 10	10 10

HIT-Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	10.091478	0.498948	20.225193	0.907961
	Stdev	10.745351	2.007324	33.438439	0.289081
	Con	10.091478±4.78%	0.498948±0.27%	20.225193±2.32%	0.907961±0.21%
	95 %	10 10	10 6	10 10	10 10

Die Simulation mit dem Startwert 19:

ProC/B Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	10.034332	0.499931	20.071438	0.908844
	Stdev	10.665022	2.001312	20.319490	0.287831
	Con	10.034332±1.31%	0.499931±0.07%	20.071438±1.28%	0.908844±0.06%
	90 %	10 10	10 9	10 10	10 10

HIT-Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	10.293618	0.500444	20.568797	0.909174
	Stdev	10.839358	2.000542	33.767223	0.287361
	Con	10.293618±4.90%	0.500444±0.27%	20.568797±2.26%	0.909174±0.23%
	95 %	10 10	10 0	10 10	10 10

Die Simulation mit dem Startwert 127:

ProC/B Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	10.028658	0.500069	20.05442	0.908693
	Stdev	10.672072	2.000072	20.374604	0.288045
	Con	10.028658±1.29%	0.500069±0.07%	20.05442±1.29%	0.908693±0.07%
	90 %	10 10	10 0	10 10	10 10

HIT-Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	10.406247	0.499699	20.824652	0.909593
	Stdev	11.276449	2.007848	34.838896	0.286764
	Con	10.406247±4.71%	0.499699±0.25%	20.824652±2.20%	0.909593±0.20%
	95 %	10 10	10 0	10 10	10 10

Bei den Ergebnissen aus den beiden Modellen stellt man fest, dass diese für alle Werte sehr nah beieinander liegen. Aus den ermittelten Übereinstimmungen kann man vermuten, dass die beiden dargestellten Modelle denselben Sachverhalt und dieselben Leistungsmasse widerspiegeln. Diese Vermutung verifizieren wir im folgendem durch Auswertung der Konfidenzintervalle.

Auswertung der Konfidenzintervalle des ProC/B-Modells
Die Auswertung der Konfidenzintervalle (siehe Abb. 3.11) der einzelnen Simulationsläufe für die unterschiedlichen Seed-Werte des ProC/B-Modells bietet eine weitere Möglichkeit die Simulationsergebnisse zu bewerten und vor allem eine gute Möglichkeit diese mit denen des HIT-Modells aus Aufgabe 2 zu vergleichen. Bemerkenswert ist, dass die Ergebnisse des ProC/B-Modell sogar noch ein klein wenig besser sind als die ursprünglichen des HIT-Modells. Für das ProC/B-Modell liegen alle theoretischen Werte (vgl. Kapitel 3.3) innerhalb der ermittelten Konfidenzintervalle. Die Betrachtung der Konfidenzintervalle des HIT-Modell erfolgte in Abbildung 3.4.

	Konfidenzintervall			
	Untergrenze	Obergrenze	Theoretischer Wert	liegt im Intervall
seed-Wert=13				
Population	9,982633741	10,239454	9,999890001	ja
Throughput	0,49973594	0,5004361	0,4999945	ja
Turnaroundtime	19,96384465	20,473353	20	ja
Utilization	0,908728445	0,9100016	0,9090909	ja
seed-Wert=17				
Population	9,814607821	10,061066	9,999890001	ja
Throughput	0,499320231	0,5000198	0,4999945	ja
Turnaroundtime	19,64020284	20,137423	20	ja
Utilization	0,908135791	0,9092262	0,9090909	ja
seed-Wert=19				
Population	9,902882251	10,165782	9,999890001	ja
Throughput	0,499581048	0,500281	0,4999945	ja
Turnaroundtime	19,81452359	20,328352	20	ja
Utilization	0,908298694	0,9093893	0,9090909	ja
seed-Wert=127				
Population	9,899288312	10,158028	9,999890001	ja
Throughput	0,499718952	0,500419	0,4999945	ja
Turnaroundtime	19,79571798	20,313122	20	ja
Utilization	0,908056915	0,9093291	0,9090909	ja

Abbildung 3.11: Auswertung der Konfidenzintervalle des ProC/B-Modells

3.5 Aufgabe 4 Gruppe 1

Aufgabenstellung: Entwickle ein ProC/B-Simulationsmodell des M/M/1-Systems aus Aufgabe 2 (siehe Kapitel 3.1) und berechne die mittlere Auslastung, die mittlere Anzahl im System, die mittlere Durchlaufzeit und den mittleren Durchsatz. Bei einer Simulation sind insbesondere die Konfidenzintervalle der Leistungsmasse interessant und zu diskutieren. Vergleiche die Ergebnisse mit den Simulationsergebnissen aus Aufgabe 2.

Das Simulationsmodell in ProC/B

Das Modell besteht aus 5 Elementen:
Quelle, ProzessID, Prozesskettenelement, Senke, Server.
Da sich das Modell von dem voranstehenden Modell (siehe Abbildung 3.5) nur in den Werten der Parameter unterscheidet, verzichten wir an dieser Stelle auf redundante Screenshots.

Das Attribut $Zeitangabe$ der Quelle bekommt den Wert *negexp(1/2)*, da die Ankunftsrate λ gleich $1/2$ ist. Das Attribut $Anzahl$ setzen wir auf *1*, weil jedes Jahr immer ein Jahrgang eingeschrieben wird. Das ProzessID-Element bezeichnen wir mit *Jahrgaenge*. Für den Wert des Parameters des Prozesskettenelementes *UniversitaetBesuchen* tragen wir die Verteilung der Zeitangabe *negexp(1/8)* ein, da jeder Jahrgang das Studium nach 8 Semestern abgeschlossen haben möchte. Die Universität wird als Server modelliert, die Bediendisziplin auf FCFS und die Geschwindigkeit gleich $9/2$ gesetzt. Zuerst legen wir eine Experimentserie ein, wo wir die mittlere Auslastung, die mittlere Anzahl im System, die mittlere Durchlaufzeit und den mittleren Durchsatz bestimmen wollen.

Die Experimentserie

Die Experimentserie beginnt mit der Simulation des Modells mit dem Seed-Wert 17 und einer Modellzeit von 10000000 Zeiteinheiten. Anschließend führen wir weitere Simulationen mit jeweils unterschiedlichen Seed-Werten (19, 127) und bei gleicher Modellzeit durch.
Die Simulationsergebnisse werden tabellarisch dargestellt.

Die Simulation mit dem Startwert 17:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	7.967143	0.499673	15.944722	0.888516
	Stdev	8.478132	2.000683	15.989398	0.314731
	Con	7.967143±1.03%	0.499673±0.07%	15.944722±1.03%	0.888516±0.06%
	90 %	10 10	10 4	10 10	10 10

Die Simulation mit dem Startwert 19:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	8.009291	0.499931	16.020794	0.888647
	Stdev	8.573535	2.001416	16.148359	0.314568
	Con	8.009291±1.08%	0.499931±0.07%	16.020794±1.05%	0.888647±0.07%
	90 %	10 10	10 9	10 10	10 10

Die Simulation mit dem Startwert 127:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	8.021809	0.500069	16.041399	0.88850
	Stdev	8.601395	2.000224	16.236796	0.314750
	Con	8.021809±1.06%	0.500069±0.07%	16.041399±1.06%	0.888500±0.07%
	90 %	10 10	10 0	10 10	10 10

Aus diesen drei Tabellen ist zu entnehmen, dass die verschiedenen Startwerte zu unterschiedlichen Ergebnissen führen.

	Konfidenzintervall			
	Untergrenze	Obergrenze	Theoretischer Wert	liegt im Intervall
seed-Wert=13				
Population	6,55250413	8,309152	8	**ja**
Throughput	0,49505364	0,504146	0,5	**ja**
Turnaroundtime	13,83218067	15,89207	16	**nein**
Utilization	0,871067264	0,885117	0,888888889	**nein**
seed-Wert=17				
Population	7,885081427	8,049205	8	**ja**
Throughput	0,499323229	0,500023	0,5	**ja**
Turnaroundtime	15,78049136	16,10895	16	**ja**
Utilization	0,88798289	0,889049	0,888888889	**ja**
seed-Wert=19				
Population	7,922790657	8,095791	8	**ja**
Throughput	0,499581048	0,500281	0,5	**ja**
Turnaroundtime	15,85257566	16,18901	16	**ja**
Utilization	0,888024947	0,889269	0,888888889	**ja**
seed-Wert=127				
Population	7,936777825	8,10684	8	**ja**
Throughput	0,499718952	0,500419	0,5	**ja**
Turnaroundtime	15,87136017	16,21144	16	**ja**
Utilization	0,88787805	0,889122	0,888888889	**ja**

Abbildung 3.12: Auswertung der Konfidenzintervalle

Auswertung der Konfidenzintervalle
Anhand der Konfidenzintervalle lassen sich die Ergebnisse und letztendlich auch die beiden Modelle hervorragend miteinander vergleichen. Die Auswertung der Konfidenzintervalle (siehe Abb. 3.12) zeigt, dass im neuen Modell bis auf zwei Werte alle theoretisch ermittelten Werte innerhalb der durch die Simulationen ermittelten Konfidenzintervalle liegen. Die Betrachtung der Konfidenzintervalle für Aufgabe 2 erfolgte in Abbildung 3.2. Die Auswertung der Konfidenzintervalle der ProC/B Ergebnisse stimmen mit denen der HIT Ergebnisse bis auf zwei Messströme für den Seed Wert 127 überein. Diese Messströme sind THROUGHPUT und UTILIZATION, welche nun, im Gegensatz zu vorher, im Konfidenzintervall liegen.

3.6 Aufgabe 4 Gruppe 2

Aufgabenstellung: Entwickle ein ProC/B-Simulationsmodell des M/D/1-Systems aus Aufgabe 2 (siehe Kapitel 3.2) und berechne die mittlere Auslastung, die mittlere Anzahl im System, die mittlere Durchlaufzeit und den mittleren Durchsatz. Bei einer Simulation sind insbesondere die Konfidenzintervalle der Leistungsmasse interessant und zu diskutieren. Vergleiche die Ergebnisse mit den Simulationsergebnissen aus Aufgabe 2.

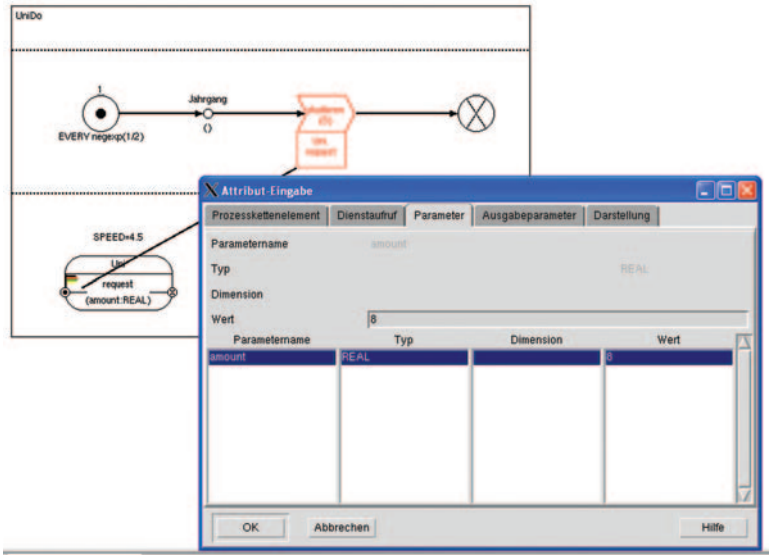


Abbildung 3.13: Parameter des Prozesskettenelementes „studieren“

Bei dem Prozesskettenelement „studieren“ benutzen wir die konstante Zahl “8“ Semester als Parameter.

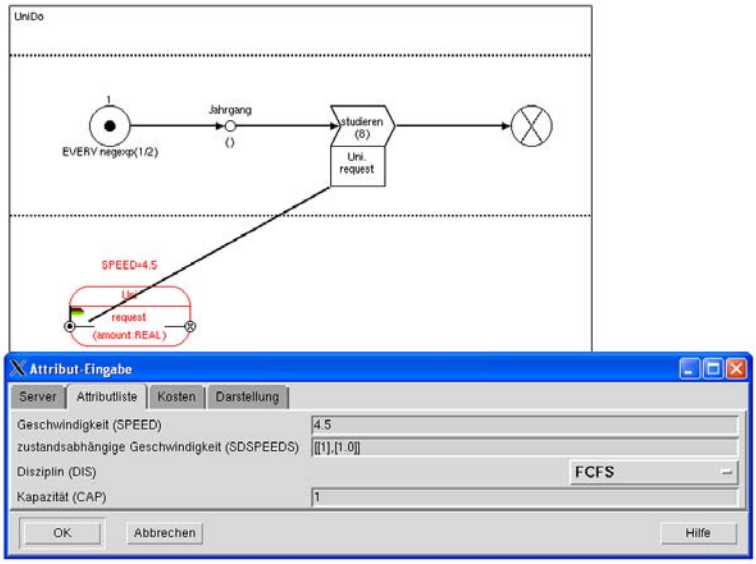


Abbildung 3.14: Serverparameter „Uni“

Die Universität wird durch den Server „Uni“ dargestellt, wo die Bediendisziplin auf FCFS und die Geschwindigkeit gleich 4.5 gesetzt wird.

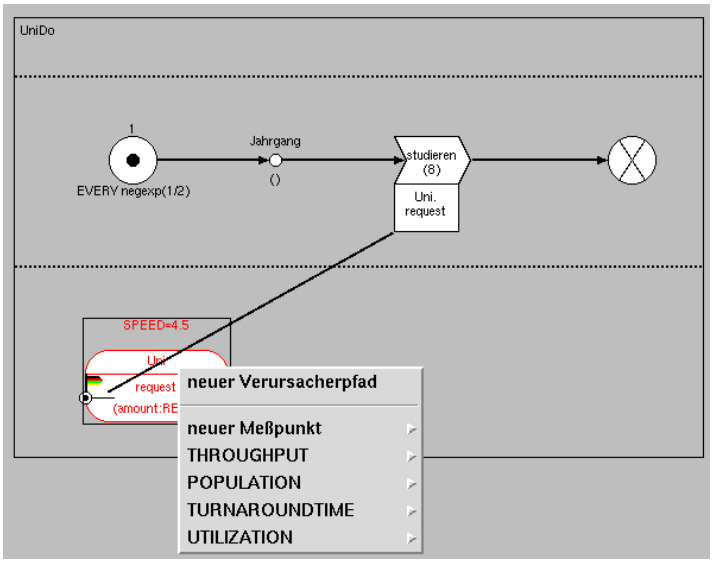


Abbildung 3.15: Messpunkte des Experiments

Zuerst legen wir eine Experimentserie an, wo wir die mittlere Auslastung, die mittlere Anzahl im System, die mittlere Durchlaufzeit und den mittleren Durchsatz bestimmen wollen.
Die Experimentserie beginnt mit der Simulation des Modells mit dem Seed-Wert 13 und einer Modellzeit von 100000 Zeiteinheiten.
Der Übersicht halber geben wir zuerst die exakten Werte an. Diese lauten wie folgt:

Population	Throughput	Turnaroundtime	Utilization
4.449	0.5	8.89	0.889

Zusätzlich wird die Simulation auch mit den Seed-Werten 17, 19 und 127 durchgeführt, wobei die Parameter unverändert bleiben. Ebenso wie die Gruppe 3, haben auch wir auf die Dokumentation der graphischen Repräsentation der Simulationsergebnisse verzichtet. Die neu ermittelten Ergebnisse aus dem ProC/B-Modell werden im folgenden den Ergebnissen aus dem HIT-Modell aus Aufgabe 2 (vgl. Kapitel 3.2) gegenübergestellt um die Modelle miteinander vergleichen zu können.

Die Simulation mit dem Startwert 13:

ProC/B Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	4.792529	0.502569	9.533404	0.893445
	Stdev	5.570249	0.883433	9.980342	0.308547
	Con	4.792529±19.89%	0.502569±1.01%	9.533404±20.92%	0.893445±0.99%
	90 %	10 10	10 10	10 9	10 10

HIT-Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	4.010246	0.507400	7.872595	0.902001
	Stdev	3.586341	0.819256	6.160541	0.297313
	Con	4.010246±18.81%	0.507400±1.82%	7.872595±18.37%	0.902001±1.83%
	95 %	10 10	10 7	10 3	10 10

Die Simulation mit dem Startwert 17:

ProC/B Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	4.305023	0.501701	8.578968	0.891882
	Stdev	3.963011	0.925988	6.872304	0.310529
	Con	4.305023±10.95%	0.501701±1.02%	8.578968±10.49%	0.891882±1.03%
	90 %	10 10	10 9	10 8	10 10

HIT-Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	4.810028	0.522900	9.378218	0.911767
	Stdev	4.106552	0.793609	7.087650	0.283633
	Con	4.810028±20.46%	0.512900±2.00%	9.378218±19.63%	0.911767±1.94%
	95 %	10 10	10 8	10 8	10 10

Die Simulation mit dem Startwert 19:

ProC/B Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	4.293870	0.493021	8.709922	0.876420
	Stdev	4.245793	0.994542	7.427710	0.329102
	Con	4.293870±12.82%	0.493021±1.11%	8.709922±12.04%	0.876420±1.1%
	90 %	10 6	10 8	10 5	10 10

HIT-Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	4.422749	0.496100	8.911491	0.881816
	Stdev	4.218593	0.922000	7.344698	0.322826
	Con	4.422749±25.58%	0.496100±2.25%	8.911491±23.63%	0.881816±1.19%
	95 %	10 7	10 10	10 5	10 10

Die Simulation mit dem Startwert 127:

ProC/B Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	4.681687	0.502118	9.324119	0.892651
	Stdev	4.358822	0.922854	7.580928	0.309556
	Con	4.681687±11.96%	0.8502118±1.03%	9.324119±11.39%	0.892651±1.00%
	90 %	10 10	10 9	10 6	10 10

HIT-Ergebnisse

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	5.310178	0.511500	10.380011	0.909329
	Stdev	4.755323	0.827899	8.269007	0.287153
	Con	5.310178±24.81%	0.511500±1.88%	10.380011±23.34%	0.9093320±1.82%
	95 %	10 8	10 9	10 3	10 9

Aus den Ergebnissen der beiden Modelle stellt man fest, dass diese für alle Werte sehr nah beieinander liegen. Dies ist ein erstes Anzeichen dafür, dass die beiden Modelle vermutlich denselben Sachverhalt modellieren. Allerdings muss diese Annahme im folgenden mit Hilfe der Konfidenzintervalle der theoretischen Werte weiter verifiziert werden.

	Konfidenzintervall			
	Untergrenze	Obergrenze	Theoretischer Wert	liegt im Intervall
seed-Wert=13				
Population	3,839294982	5,745763	4,449	**ja**
Throughput	0,497493053	0,507645	0,5	**ja**
Turnaroundtime	7,539015883	11,52779	8,89	**ja**
Utilization	0,884599895	0,90229	0,888888889	**ja**
seed-Wert=17				
Population	3,833622982	4,776423	4,449	**ja**
Throughput	0,496672732	0,506909	0,5	**ja**
Turnaroundtime	7,679034257	9,478902	8,89	**ja**
Utilization	0,882695615	0,901068	0,888888889	**ja**
seed-Wert=19				
Population	3,743395866	4,844344	4,449	**ja**
Throughput	0,487548467	0,498494	0,5	**nein**
Turnaroundtime	7,661247391	9,758597	8,89	**ja**
Utilization	8,28427938	8,468561	0,888888889	**nein**
seed-Wert=127				
Population	4,121757235	5,241617	4,449	**ja**
Throughput	0,841453827	0,858968	0,5	**nein**
Turnaroundtime	8,262101846	10,38614	8,89	**ja**
Utilization	0,88372449	0,901578	0,888888889	**ja**

Abbildung 3.16: Auswertung der Konfidenzintervalle

Auswertung der Konfidenzintervalle des ProC/B-Modells
Man beachte, dass die Ergebnisse des ProC/B-Modells sogar noch ein klein wenig besser sind als die ursprünglichen des HIT-Modells, da bei ProC/B nur 3 theoretische Werte anstatt 4 bei HIT außerhalb der Konfidenzintervalle liegen. Die Betrachtung der Konfidenzintervalle des HIT-Modells erfolgte in Abbildung 3.3.

3.7 Aufgabe 5 Gruppe 1

Aufgabenstellung: Weil wir im Rahmen der PG-Arbeit unsere Modelle zum Zwecke der Validierung einerseits als ProC/B-Modelle und andererseits als HIT-Modelle erstellen werden, erfolgt nun eine erste Spezialisierung der Untergruppen.

Modellvalidierung In Aufgabe 2 (siehe Kapitel 3.1) wurde ein M/M/1-System mittels HIT u.a. auch simulativ gelöst, während dasselbe System in Aufgabe 4 (siehe Kapitel 3.5) mit dem ProC/B-Werkzeug gelöst wurde. Abgesehen davon, dass wir bei M/M/1-Systemen die theoretischen Ergebnisse exakt berechnen können, stellst sich aber auch hier die Frage, ob die beiden Modelle wirklich dasselbe System modellieren? Da sich diese Fragestellung in unserer weiteren Projektarbeit ständig wiederholen wird, soll ein „Kochrezept“ entwickelt werden, wie HIT - und ProC/B-Modelle verglichen, oder besser noch, validiert werden können.

Vergleich von alternativen Simulationsmodellen

- Arbeiten Sie sich in das Statistikpaket R [Vena 05] ein.
- Entwerfen Sie unter Verwendung der u.a. Lösungshinweise ein Kochrezept zur Durchführung einer HIT-, ProC/B-Modellvalidation mittels R.
- Führen Sie eine einfache HIT-, ProC/B-Modellvalidation am M/M/1-System durch.

Das folgende *Kochrezept* dient der Validierung alternativer Modelle, welche sowohl mit dem Tool **HITGRAPHIC** als auch mit **ProC/B** erstellt worden sind.

3.7.1 Vorgehensweise

Während der Ausführung der Experimentserie, wird eine Dump-Datei angelegt aus der man die gewünschten Messströme, die HIT an den jeweiligen Auswertungsobjekten bereitstellt(z.B. *TURNAROUNDTIME, POPULATION*...), ausliest und diese in tabellarischer Form in einer Datei abspeichert. Dies macht man sowohl für HITGRAPHIC als auch für ProC/B, da beide Tools auf HIT basieren. Im weiteren Verlauf werden die Datensätze der beiden Tabellen mit der Statistikumgebung R ausgewertet und validiert. Dazu wird der in Abschnitt 2.8 beschriebene two-sample Konfidenzintervalltest eingesetzt. Es wird nun erst der angepasste HI-SLANG Code und dann die Auswertung der Daten mittels R an einem Beispiel illustriert. Als Beispiel wählen wir das bereits in Aufgabe 2 dokumentierte Universitätsmodell. (siehe Kapitel 3.1)

3.7.2 Anpassung des HI-SLANG-Quellcodes in HIT

In dem HITGRAPHIC Tool lässt sich der Quellcode bis auf wenige Ausnahmen bequem in der grafischen Umgebung anpassen.

Die Abbildung 3.17 zeigt:

- das Anlegen einer Experimentserie im Code des Experimentbodys mit dem direkten Abgreifen der Messströme (hier im Beispiel: *POPULATION, UTILIZATION, OCCUPATION, THROUGHPUT und TURNAROUNDTIME*) durch den Befehl *get_result*
- die Variablendeklaration
- und das Anlegen der Ausgabe Datei im Controlfenster

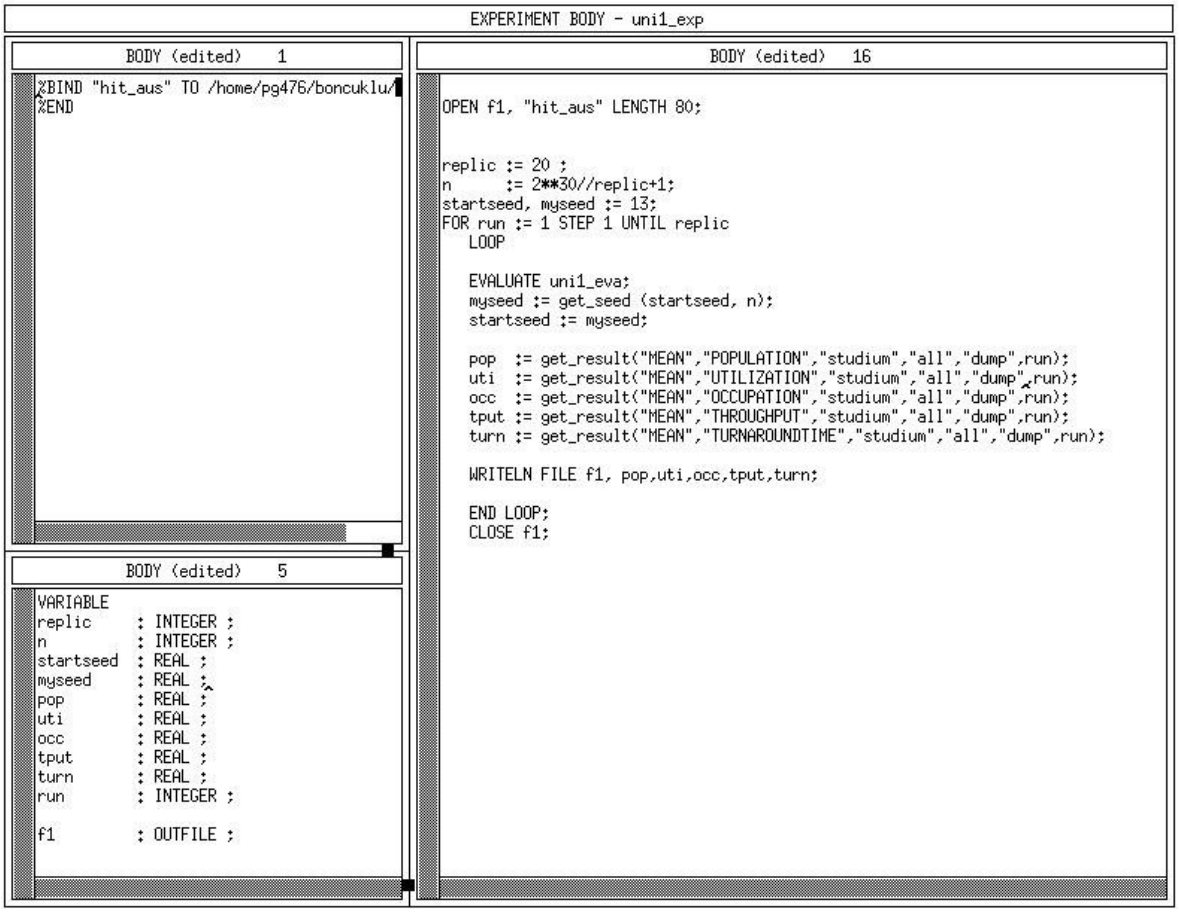


Abbildung 3.17: Der angepasste Code in HITGRAPHIC

Die Experimentserie wird mittels einer For-Schleife realisiert, deren Iterationenanzahl durch den Wert der Variable *replic* gegeben ist. In jedem Schleifendurchlauf wird eine neue Zufallszahl *myseed* generiert, die die Unabhängigkeit der Simulationsdurchläufe gewährleistet. Die Zeile *n := 2**30//replic+1* sorgt dafür, dass replic viele gleichgroße Intervalle bereitgestellt werden, so dass jede Replikation auf vorher noch nicht benutzte Zufallszahlen zugreifen kann, vorausgesetzt, dass die Anzahl der benötigten Zufallszahlen nicht die Größe des Gesamtintervalls übersteigen. Die Zuweisung des Rückgabewertes der getseed-Methode *myseed := getseed (startseed, n)* trägt sorge dafür, dass die als Startseed

für die nächste Replikation gewählte Zahl, in keinem bereits verwendetem Intervall liegt und so die Unabhängigkeit der Zufallszahlen gewährleistet.
Im *get_result* Befehl muss der gewünschte Schätzer, der Name des gewünschten Messstromes, das Evaluationsobjekt (im Beispiel „studium“), die Hierarchie, der Linkname, also der Name der Dump-Datei, und der Abszisse spezifiziert werden. Die Übergabe des Abszissen Parameters *run* sorgt dafür, dass jeweils die Ergebnisse der aktuellen Iteration aus der richtigen Position in der Dump-Datei gelesen werden. Die gelesenen Werte werden mittels *WRITELN FILE* in die im Control-Block spezifizierte Datei *.data geschrieben.
In der erzeugten HI-SLANG Datei *.hit lassen sich die oben beschriebenen Codesegmente mühelos wiederfinden. Die einzige manuell vorzunehmende Änderung betrifft das Hinzufügen der Anweisung *Abscissa run*, welche das Auslesen der Ergebnisse an der richtigen Stelle ermöglicht. Der HI-SLANG-Code befindet sich in Kapitel 3.7.5 in den Abbildungen 3.20, 3.21, 3.22.

3.7.3 Anpassung des HI-SLANG-Quellcodes in ProC/B

Da das Anlegen einer Experimentserie von der ProC/B-GUI nicht unterstützt wird, muss der entsprechende Code in der HI-SLANG Datei per Hand eingefügt werden. Die Anpassungen sind am einfachsten vorzunehmen, indem man sich an der gerade vorgestellten HI-SLANG Datei orientiert. In der Measurezeile eines jeden einzelnen Messwertes muss folgender Code ergänzt werden:

```
MEASURE THROUGHPUT AT e23unido DUE TO ALL ABSCISSA run
ESTIMATOR MEAN,STANDARDDEVIATION,CONFIDENCE LEVEL 95
GLOBALSTOP WIDTH 5;
```

Der HI-SLANG-Code befindet sich in Kapitel 3.7.5 in den Abbildungen 3.23, 3.24, 3.25.
Nach der Simulation der beiden Modelle durch den Aufruf *hit <dateiname.hit>* liegen die zwei Tabellen hit.data und procb.data vor, die die Messwerte der einzelnen Ströme spaltenweise enthalten.

3.7.4 Datenanalyse mit dem R Statistikpaket

Mit den gewonnen Ergebnissen kann man nun durch wenige R-Befehle (Abb. 3.18) die Gleichheit der Modelle testen.

```
hit <- scan("hit.data", list(0,0,0,0,0));
procb <- scan("procb.data", list(0,0,0,0,0));
pop_h <- hit[[1]];
uti_h <- hit[[2]];
occ_h <- hit[[3]];
tput_h <- hit[[4]];
turn_h <- hit[[5]];
pop_p <- proc[[1]];
uti_p <- proc[[2]];
occ_p <- proc[[3]];
tput_p <- proc[[4]];
turn_p <- proc[[5]];
t.test(pop_h,pop_p);
t.test(uti_h,uti_p);
t.test(occ_h,occ_p);
t.test(tput_h,tput_p);
t.test(turn_h,turn_p);
```

Abbildung 3.18: Die Befehle der R-Umgebung

Nach dem Einlesen der Tabellen durch die *Scan*-Befehle werden die Spalten selektiert und den entsprechenden Variablen zugewiesen, so dass für jedes Leistungsmaß ein two-sample Konfidenzintervalltest durchgeführt werden kann. Die Bedingungen für diesen Test sind gegeben, da die beiden Modelle unabhängig voneinander sind, genauso gut könnte man auch den gepaarten Test durchführen, weil auch dafür die Bedingungen erfüllt sind - im gepaarten Test muss der zusätzliche optionale Parameter *paired=True* gesetzt werden. Zusätzlich könnte man bei dem Aufruf das Konfidenzintervall bestimmen, welches per default bei 95% liegt.

```
> t.test(turn_h,turn_p);

        Welch Two Sample t-test

data:  turn_h and turn_p
t = 0, df = 38, p-value = 1
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.6732309  0.6732309
sample estimates:
mean of x mean of y
 15.65862  15.65862
```

Abbildung 3.19: Die Ergebnisse des t-Testes

Wie aus dem Ergebnis des Testes in Abbildung 3.19 zu erkennen ist, stimmen die simulierten Leistungen der beiden Modelle für das Leistungsmaß *Turnaroundtime* überein, da das ermittelte Konfidenzintervall die Null enthält. Die beiden Modelle gelten als validiert, wenn auch die anderen t.tests für die Leistungsmaße die Null in ihren Konfidenzintervallen enthalten.

3.7.5 Codelistings

```
%COMMON
%PARM = INDENT = 0
%BIND "hit_aus" TO /home/pg476/boncuklu/hit.data
%END

%TITLE MODEL TYPE uni1
TYPE uni1 MODEL;

   {DECLARATION OF SERVICE jahrgang}
   {WITHIN COMPONENT/MODEL TYPE uni1}

   TYPE jahrgang SERVICE;

      USE {of service/procedure jahrgang}
         SERVICE   studieren (
            amount : REAL
            )
;
      END USE;  {of service/procedure jahrgang}



      {START OF BODY OF SERVICE/PROCEDURE jahrgang}
      {WITHIN COMPONENT/MODEL TYPE uni1}

      BEGIN
         studieren (negexp (1/8));
;

   END {SERVICE} TYPE jahrgang;

   {END OF DECLARATION OF SERVICE  jahrgang}
   {WITHIN COMPONENT/MODEL TYPE uni1}


%TITLE MODEL/COMPONENT TYPE uni1
   COMPONENT unido : server (
      LET  DISPATCH  :=  SHARED  (
         LET speed := 4.5

         ) ) ;

   REFER {of component/model type uni1}
      jahrgang
   TO
      unido
   EQUATING
      jahrgang.studieren WITH unido.request;
   END REFER;  {of component/model type uni1}


%TITLE ACTIVITIES OF MODEL TYPE uni1
   {START OF ACTIVITIES (BODY) OF MODEL TYPE uni1}

   BEGIN
      CREATE 1 PROCESS Jahrgang LIMIT 200 EVERY negexp (0.5);
;

END {MODEL} TYPE uni1;
```

Abbildung 3.20: Der HI-SLANG-Code I

```
%TITLE EXPERIMENT uni1_exp
{HI-SLANG code for experiment}
{              uni1_exp}
{generated by     HITGRAPHIC V. 3.7.16}
{         at      30.11.05 11:29}
{    for user     boncuklu}
{         on      koenig sun4u SunOS 5.9}

EXPERIMENT uni1_exp METHOD SIMULATIVE;
VARIABLE
replic    : INTEGER ;
n         : INTEGER ;
startseed : REAL ;
myseed    : REAL ;
pop       : REAL ;
uti       : REAL ;
occ       : REAL ;
tput      : REAL ;
turn      : REAL ;
run       : INTEGER ;

f1        : OUTFILE ;


%TITLE BODY OF EXPERIMENT uni1_exp
{START OF BODY OF EXPERIMENT uni1_exp}

BEGIN

   OPEN f1, "hit_aus" LENGTH 80;

   replic := 2 ;
   n      := 2**30//replic+1;
   startseed, myseed := 13;
   FOR run := 1 STEP 1 UNTIL replic
      LOOP


%TITLE EVALUATION uni1_eva
   {START OF EVALUATION OF MODEL TYPE uni1}

   EVALUATE

     MODEL uni1_eva : uni1
    (
     LET seed := myseed
     )   ;
```

Abbildung 3.21: Der HI-SLANG-Code II

```
EVALUATIONOBJECT
{for model uni1_eva}
   studium VIA uni1_eva.
      unido
   DEFAULT
   OUTPUT      TABLE "TABLE",
   DUMPFILE "DUMP";

{START OF MEASURE STATEMENTS OF EVALUATION uni1_eva}

BEGIN

MEASURE THROUGHPUT DEGREE 10,
   TURNAROUNDTIME DEGREE 10,
   POPULATION DEGREE 10,
   UTILIZATION DEGREE 10,
   OCCUPATION DEGREE 10
   AT studium
   DUE TO
      ALL
   ABSCISSA run
   ESTIMATOR
      MEAN,
      STANDARDDEVIATION,
      CONFIDENCE LEVEL 95
   GLOBALSTOP
      WIDTH 5
   ;

CONTROL

   AT studium
     STOP MODELTIME 10000

   AT studium
     STOP CPUTIME 100000

;
END EVALUATE;

{END OF EVALUATION  uni1_eva FOR MODEL TYPE uni1}



   myseed := get_seed (startseed, n);
   startseed := myseed;

   pop  := get_result("MEAN","POPULATION","studium","all","dump",run);
   uti  := get_result("MEAN","UTILIZATION","studium","all","dump",run);
   occ  := get_result("MEAN","OCCUPATION","studium","all","dump",run);
   tput := get_result("MEAN","THROUGHPUT","studium","all","dump",run);
   turn := get_result("MEAN","TURNAROUNDTIME","studium","all","dump",run);

   WRITELN FILE f1, pop,uti,occ,tput,turn;

   END LOOP;
   CLOSE f1;
;


END EXPERIMENT uni1_exp;

{END OF EXPERIMENT  uni1_exp}
```

Abbildung 3.22: Der HI-SLANG-Code III

```
%COMMON
%PARM = WARN
%BIND "PROC_AUS" TO /home/pg476/boncuk1u/procb.data
%END
%COPY "counter"
%COPY "semaphor"
%COPY "observer"
%COPY "prioserver"
TYPE voidRecordName RECORD; END TYPE voidRecordName;

TYPE F1_Typ MODEL;

COMPONENT modellobserver : observer(10000/100);

COMPONENT F23UniDo : server
(LET ACCEPT   := ALWAYS,
 LET SCHEDULE := FCFS  (LET CAPACITY := 1),
 LET DISPATCH := EQUAL (LET SPEED    := 4.5),
 LET OFFER    := ALL);

TYPE REC5_Jahrgang RECORD;
END {RECORD} TYPE REC5_Jahrgang;

TYPE P5Jahrgang SERVICE(data : POINTER FOR REC5_Jahrgang);
USE
SERVICE L9UniDo(B2H__amount : REAL DEFAULT
1.0);
END USE;

BEGIN
NEW REC5_Jahrgang POINTER data;

L9UniDo(
negexp(0.125));

END {SERVICE} TYPE P5Jahrgang;

TYPE UnbedQuellen SERVICE;
BEGIN
LOOP
BLOCK
VARIABLE
hitLoopSchleifenZaehler      : INTEGER;
hitLoopSchleifenProzessanzahl : INTEGER;
BEGIN
hitLoopSchleifenZaehler := 0;
hitLoopSchleifenProzessanzahl := 1;
WHILE hitLoopSchleifenZaehler < hitLoopSchleifenProzessanzahl
LOOP
CREATE 1 PROCESS P5Jahrgang(NONE);
hitLoopSchleifenZaehler := hitLoopSchleifenZaehler +1;
END LOOP;
END BLOCK;
hold(negexp(0.5));
END LOOP;
END {SERVICE} TYPE UnbedQuellen;
```

Abbildung 3.23: Der ProC/B-Code I

```
TYPE StatusAusgabe SERVICE;
  VARIABLE prozent:INTEGER DEFAULT 0;
           schritt:REAL DEFAULT 0;
  BEGIN
      WRITELN "Die Simulation startet jetzt...";
      {*** Status-Ausgabe in Prozent alle 5% ***}
      schritt := 10000*0.05;
      LOOP
        prozent := prozent + 5;
        hold(schritt-0.00000001);
        WRITELN prozent::3, "%";
      END LOOP;
  END {SERVICE} TYPE StatusAusgabe;

REFER
P5Jahrgang
TO
F23UniDo
EQUATING
P5Jahrgang.L9UniDo
WITH F23UniDo.request;
END REFER;

BEGIN
WRITELN("Die Simulation startet jetzt...");
CREATE 1 PROCESS UnbedQuellen;
{*** Aktivierung der Statusaugabe, bei Observer deaktiviert ***}
{*** CREATE 1 PROCESS StatusAusgabe; ***}
END {MODEL} TYPE F1_Typ;
```

```
EXPERIMENT versuch METHOD SIMULATIVE;

VARIABLE
replic    : INTEGER;
startseed : REAL;
myseed    : REAL;
pop       : REAL;
uti       : REAL;
occ       : REAL;
tput      : REAL;
turn      : REAL;
n         : INTEGER;
run       : INTEGER;

f1        : OUTFILE ;

BEGIN
OPEN f1, "PROC_AUS" LENGTH 80;
 replic := 20;
   n     := 2**30//replic+1;
   startseed, myseed := 13;
   FOR run := 1 STEP 1 UNTIL replic
      LOOP
```

Abbildung 3.24: Der ProC/B-Code II

```
EVALUATE MODEL F1 : F1_Typ(LET SEED := myseed);
EVALUATIONOBJECT
messgeraet VIA F1;
e23unido VIA
F1.F23UniDo
DEFAULT ESTIMATOR
  CONFIDENCE LEVEL 95
OUTPUT TABLE "Table", DUMPFILE "DUMP";

BEGIN
MEASURE THROUGHPUT AT messgeraet;

MEASURE THROUGHPUT AT e23unido DUE TO ALL ABSCISSA run
ESTIMATOR MEAN,STANDARDDEVIATION,CONFIDENCE LEVEL 95 GLOBALSTOP WIDTH 5;

MEASURE OCCUPATION AT e23unido DUE TO ALL ABSCISSA run
ESTIMATOR MEAN,STANDARDDEVIATION,CONFIDENCE LEVEL 95 GLOBALSTOP WIDTH 5;

MEASURE TURNAROUNDTIME AT e23unido DUE TO ALL ABSCISSA run
 ESTIMATOR MEAN,STANDARDDEVIATION,CONFIDENCE LEVEL 95 GLOBALSTOP WIDTH 5;

MEASURE POPULATION AT e23unido DUE TO ALL ABSCISSA run
 ESTIMATOR MEAN,STANDARDDEVIATION,CONFIDENCE LEVEL 95 GLOBALSTOP WIDTH 5;

MEASURE UTILIZATION AT e23unido DUE TO ALL ABSCISSA run
 ESTIMATOR MEAN,STANDARDDEVIATION,CONFIDENCE LEVEL 95 GLOBALSTOP WIDTH 5;


CONTROL AT messgeraet STOP MODELTIME 10000;
END EVALUATE;
   myseed    := get_seed (startseed, n);
     startseed := myseed;


     pop       := get_result("MEAN","POPULATION","e23unido","all","dump",run);
     uti       := get_result("MEAN","UTILIZATION","e23unido","all","dump",run);
     occ       := get_result("MEAN","OCCUPATION","e23unido","all","dump",run);
     tput      := get_result("MEAN","THROUGHPUT","e23unido","all","dump",run);
     turn      := get_result("MEAN","TURNAROUNDTIME","e23unido","all","dump",run);

     WRITELN FILE f1, pop,uti,occ,tput,turn;



   END LOOP;
   CLOSE f1;
;
END EXPERIMENT versuch;
```

Abbildung 3.25: Der ProC/B-Code III

3.8 Aufgabe 5 Gruppe 2

Aufgabenstellung: Zur Vertiefung in die HIT-Modellwelt soll ein etwas umfangreicheres Modell erstellt und validiert werden. Zuerst soll der Schnellimbiss in der reduzierten Version des ProC/B-Modells mittels HITGRAPHIC modelliert werden.

3.8.1 Das ursprüngliche Modell in ProC/B

Im folgenden wird das ProC/B-Modell in Abb. 3.26 dargestellt.

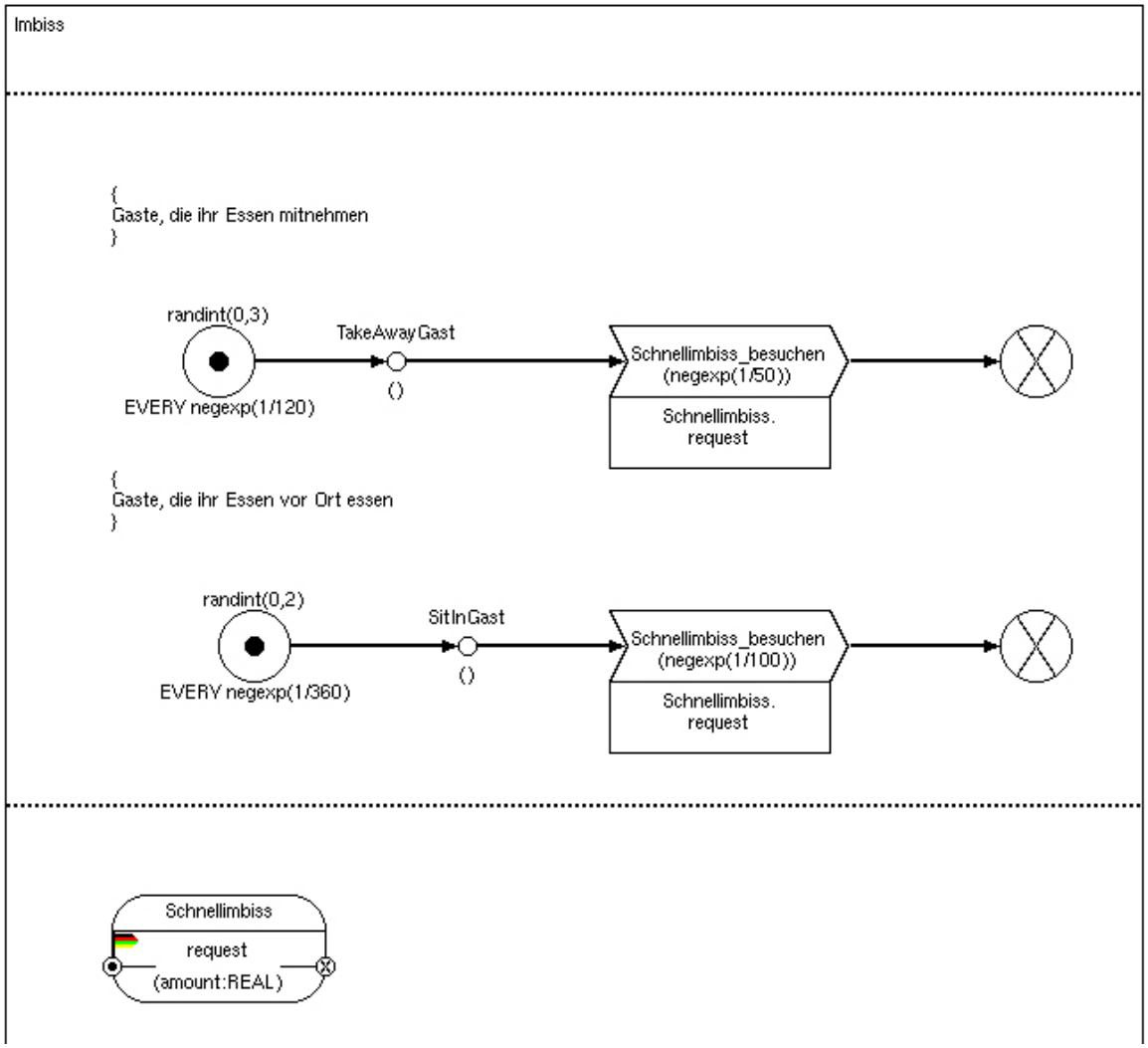


Abbildung 3.26: „Schnellimbiss“ in ProC/B

Bei der Betrachtung des Modells in ProC/B, fällt auf, dass zwei parallel ablaufende Prozesse stattfinden. Diese Prozesse werden *TakeAwayGast* und *SitInGast* genannt. Aus Analogiegründen betrachten wir nur *SitInGast*. Der Prozess *SitInGast* wird von der Quelle erzeugt. Die Parameter der Quelle werden in Abb. 3.27 wie folgt eingetragen.

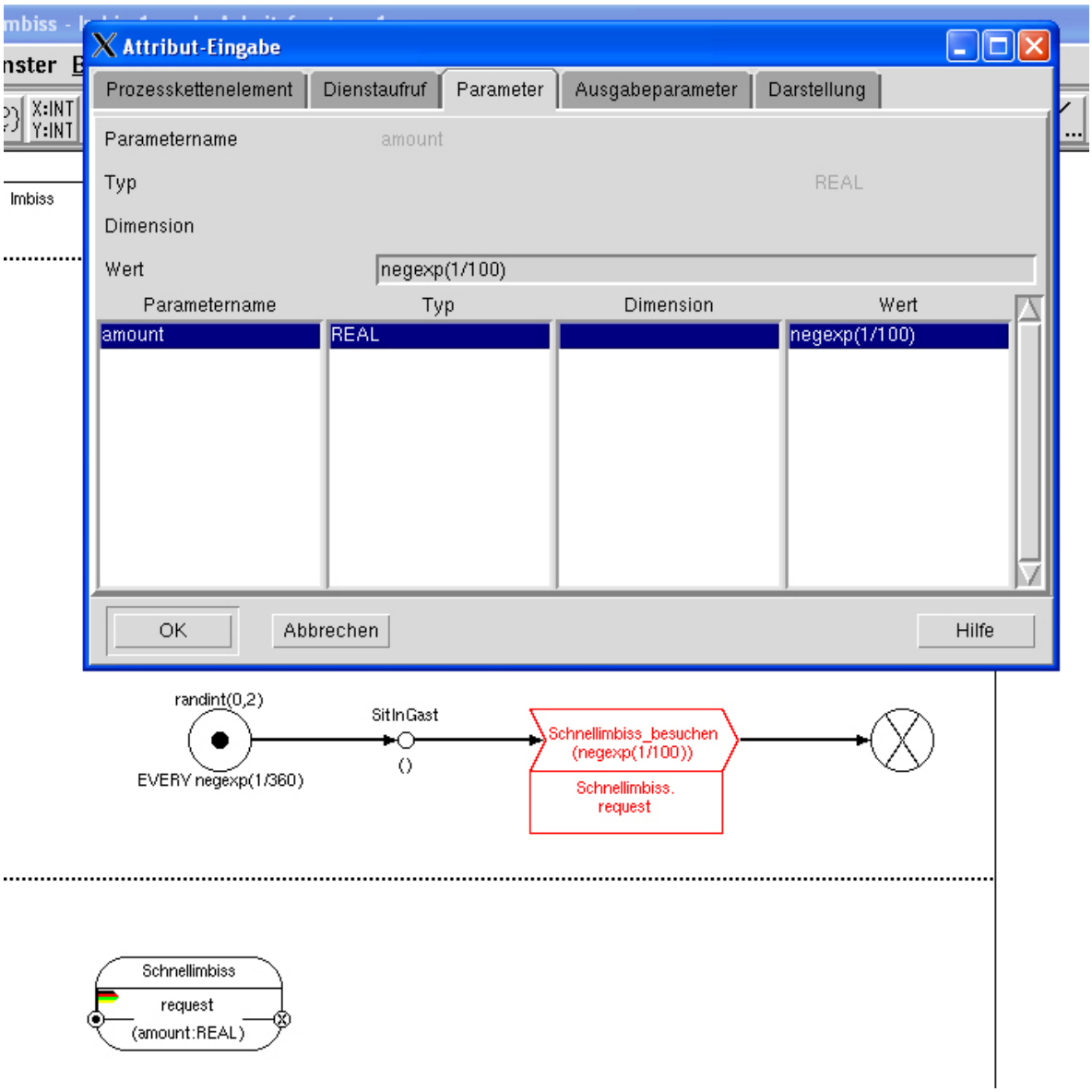


Abbildung 3.27: Parameter der Komponente SitInGast

In dem Feld Bezeichner bekommt die Quelle einen Namen. Die Anzahl gibt an, wie viele Prozesse zufällig stattfinden können. In diesem Fall ist es *randint(0,2)*. Für den Typ und die Zeitangabe haben wir *EVERY* bzw. *negexp(1/360)* gewählt. Das bedeutet, dass im Durchschnitt alle 6 Minuten ein Kunde *SitInGast* den Imbiss besucht. Die Prozesskette *Schnellimbiss_besuchen* hat als einzigen Parameter den Wert *negexp(1/100)*, welcher die Bedienanforderung eines SitIn Gastes angibt. Der Schnellimbiss wird als Server modelliert.

3.8.2 Das Modell in HIT

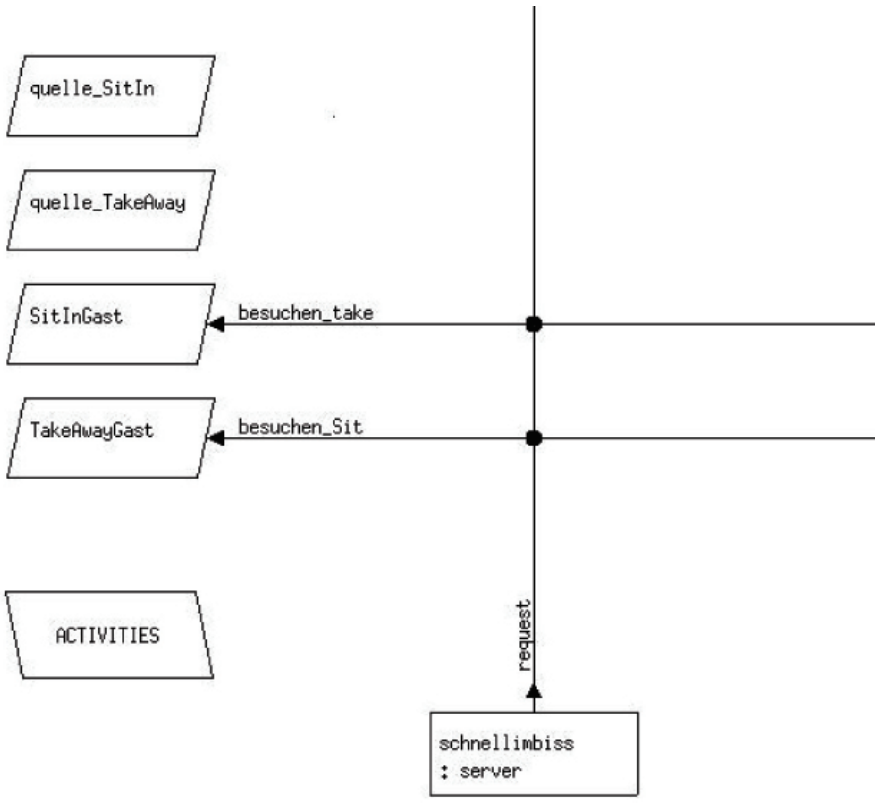


Abbildung 3.28: Der „schnellimbiss“ in HIT

Das entwickelte HIT-Modell ist in Abb. 3.28 zu sehen. Hierbei werden vier Services und eine Komponente vom Typ Server benutzt. Auch bei diesem Modell wird aus Analogiegründen nur der SitIn Teil betrachtet. Das unten stehende Codesegment zeigt den Body vom Service *SitInGast*, welcher das zugehörige *use service* mit der jeweiligen Bedienanforderung *negexp(1/100)* beinhaltet.
In den Body des *SitInGast* wird nur die Zeile *besuchen_take (negexp(1/100))* geschrieben. Um die Quellen der Prozesse nun in HITGRAPHIC darstellen zu können, benötigen wir die beiden Services *quelle_SitIn* und *quelle_TakeAway*. Diese beiden Services werden in Activities aufgerufen.
In das Feld *ACTIVITIES* von *imbiss* wird folgendes eingetragen:

```
CREATE 1 PROCESS quelle_TakeAway;
CREATE 1 PROCESS quelle_SitIn;
```

Um die zufällige Erzeugung der Prozesse *SitInGast* in HITGRAPHIC modellieren zu können, bedarf es der folgenden Schleife:
Im Body-Fenster:

```
LOOP
  schleifenzaehler := 0;
  schleifenprozessanzahl:= randint(0,2);
  WHILE schleifenzaehler < schleifenprozessanzahl
```

```
    LOOP
      CREATE 1 PROCESS SitInGast;
      schleifenzaehler := schleifenzaehler + 1;
    END LOOP;
  hold (negexp(1/360));
END LOOP;
```

Es werden zufällige Zahlen zwischen 0 und 2 gezogen, die die Anzahl der Prozesse angeben. *hold(negexp(1/360))* bedeutet, dass durchschnittlich alle 360 Zeiteinheiten 0 bis 2 Prozesse von *SitInGast* erzeugt werden.

Im Locals-Fenster:

```
VARIABLE
schleifenzaehler : INTEGER;
schleifenprozessanzahl  : INTEGER;
```

3.8.3 Die Simulationsergebnisse

Die Ergebnisse der beiden Modellierungsarten sehen wie folgt aus:
In HITGRAPHIC erhalten wir die folgenden Werte:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	2.456313	0.0134600	184.685179	0.714112
	Stdev	2.567477	95.861560	145.096069	0.451836
	Con	Undefined	0.013400±19.63%	184.685179±39.53%	0.714112±17.19%
	95 %		10 0	10 3	10 4

In ProC/B erhalten wir folgende Werte:

Hierarchy	Esti	POPULATION	THROUGHPUT	TURNAROUNDTIME	UTILIZATION
ALL	Mean	3.002408	0.013090	229.623314	0.748284
	Stdev	2.901087	90.1388410	176.722960	0.433999
	Con	3.002408±25.93%	0.01390±9.63%	229.623314±23.28%	0.748284±9.24%
	95 %	10 4	10 3	10 1	10 5

Da die Ergebnisse für beide Modelle nah beieinander liegen, müssen wir nun die Ähnlichkeit mit statistischen Tests nachweisen.

3.8.4 Validierung der Modelle

Um das HIT-Modell und ProC/B-Modell vergleichen zu können, werden die Ergebnisse der beiden Simulationen mit Hilfe des *R-Tools* validiert. Abb. 3.29 und Abb. 3.30 zeigen jeweils die Ergebnisse aus dem HIT-Modell bzw. aus dem ProC/B-Modell. Jede der zwei Tabellen haben vier Spalten, die von links nach rechts die Ergebnisse für *POPULATION*, *UTILIZATION*, *THROUGHPUT* und *TURNAROUNDTIME* zeigen. Da die 20 Replikationen mit verschiedenen Seed-Werten laufen, zeigen die Tabellen die Ergebnisse jeder Replikation. Man sieht, dass die beiden Tabellen unterschiedlich sind.

Deswegen muss ein Hypothesentest durchgeführt werden, die wir im Abschnitt 3.8.4 durchgeführt haben.
Ein Experiment mit 20 Replikationen für das HIT-Modell:

```
3.550280E+001 9.471610E-001 1.515300E-002 2.341188E+003
1.915562E+001 9.480064E-001 1.521100E-002 1.258179E+003
4.130235E+001 9.696546E-001 1.527600E-002 2.703435E+003
2.330943E+001 9.580763E-001 1.536900E-002 1.515143E+003
1.938878E+001 9.458732E-001 1.514500E-002 1.280043E+003
2.227336E+001 9.522667E-001 1.517700E-002 1.465353E+003
1.889886E+001 9.415911E-001 1.502700E-002 1.256841E+003
3.380055E+001 9.677692E-001 1.537700E-002 2.195746E+003
2.076229E+001 9.523094E-001 1.516900E-002 1.365384E+003
3.702057E+001 9.700388E-001 1.543200E-002 2.399104E+003
1.625880E+001 9.375883E-001 1.499500E-002 1.083003E+003
2.557018E+001 9.481352E-001 1.509700E-002 1.693838E+003
3.484901E+001 9.663783E-001 1.537400E-002 2.265049E+003
2.822271E+001 9.718470E-001 1.543500E-002 1.828189E+003
2.096804E+001 9.634217E-001 1.536000E-002 1.358438E+003
4.244354E+001 9.747553E-001 1.548600E-002 2.719077E+003
1.727500E+001 9.448200E-001 1.499900E-002 1.151136E+003
2.244287E+001 9.481623E-001 1.533100E-002 1.462874E+003
2.079566E+001 9.418191E-001 1.506600E-002 1.379950E+003
2.328819E+001 9.493495E-001 1.507000E-002 1.545247E+003
```

Abbildung 3.29: Simulationsergebnisse des HIT-Modells

Ein Experiment mit 20 Replikationen für das ProC/B-Modell:

```
5.652646E+001 9.789781E-001 1.554100E-002 3.637469E+003
2.081701E+001 9.529524E-001 1.524100E-002 1.364959E+003
2.694792E+001 9.545710E-001 1.537900E-002 1.751372E+003
2.612141E+001 9.666245E-001 1.542500E-002 1.692871E+003
2.324933E+001 9.523116E-001 1.528700E-002 1.520956E+003
1.846402E+001 9.432413E-001 1.494800E-002 1.235155E+003
2.835141E+001 9.700771E-001 1.534800E-002 1.839919E+003
1.866618E+001 9.497155E-001 1.530300E-002 1.219797E+003
2.056067E+001 9.612096E-001 1.524700E-002 1.348490E+003
1.941981E+001 9.480604E-001 1.524000E-002 1.271617E+003
2.210120E+001 9.358915E-001 1.511000E-002 1.462736E+003
2.804578E+001 9.632780E-001 1.533000E-002 1.828480E+003
4.019478E+001 9.692005E-001 1.542200E-002 2.605468E+003
2.513256E+001 9.530214E-001 1.526400E-002 1.645984E+003
2.894002E+001 9.605336E-001 1.534900E-002 1.872310E+003
6.012850E+001 9.807466E-001 1.550800E-002 3.876346E+003
3.117713E+001 9.630818E-001 1.533700E-002 2.032927E+003
2.485926E+001 9.639947E-001 1.538400E-002 1.612317E+003
2.040586E+001 9.572040E-001 1.518400E-002 1.343982E+003
1.925414E+001 9.601175E-001 1.522700E-002 1.264465E+003
```

Abbildung 3.30: Simulationsergebnisse des ProC/B-Modells

Hypothesentest mittels R: Der folgende Test gibt die vier Ergebnisse des Vergleiches für *POPULATION*, *UTILIZATION*, *THROUGHPUT* und *TURNAROUNDTIME* (von oben nach unten), die jeweils aus dem HIT-Modell und ProC/B-Modell sind.

```
> t.test(pop_h, pop_p)

        Welch Two Sample t-test

data:  pop_h and pop_p
t = -0.5613, df = 34.225, p-value = 0.5782
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -8.276945  4.693461
sample estimates:
mean of x mean of y
 26.17643  27.96817


> t.test(uti_h, uti_p)

        Welch Two Sample t-test

data:  uti_h and uti_p
t = -1.1903, df = 37.918, p-value = 0.2413
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.011585014  0.003006244
sample estimates:
mean of x mean of y
0.9549512 0.9592406


> t.test(tput_h, tput_p)

        Welch Two Sample t-test

data:  tput_h and tput_p
t = -1.6403, df = 36.969, p-value = 0.1094
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.704415e-04  1.794151e-05
sample estimates:
 mean of x  mean of y
0.01522745 0.01530370


> t.test(turn_h, turn_p)
```

```
        Welch Two Sample t-test

data:  turn_h and turn_p
t = -0.5306, df = 34.261, p-value = 0.5991
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -521.6378  305.5975
sample estimates:
mean of x mean of y
 1713.361  1821.381
```

(Die Durchführung und Einsetzung der Variablen und Werte für die Validierung wurden von Gruppe 1 erklärt, siehe auch Kapitel 3.7)

Man kann an den oben aufgeführten Ergebnissen erkennen, dass die jeweiligen Konfidenzintervalle die Null enthalten und somit das ProC/B-Modell und HITGRAPHIC-Modell das gleiche modellieren.

3.9 Aufgabe 5 Gruppe 3

Aufgabenstellung: Zur Vertiefung in die ProC/B Modellwelt soll ein etwas umfangreicheres Modell erstellt und validiert werden.

Literaturrecherche als ProC/B-Modell
Das Experiment *bib1_vert_exp* aus [Beil 04] ist mittels ProC/B zu modellieren.

Validierung des ProC/B-Modells mit Gruppe 1
Das in Aufgabe 5 von der Gruppe 1 (siehe Kapitel 3.7) vorbereitete Validierungskonzept ist mit Unterstützung von Gruppe 1 zur Validierung des ProC/B-Modell einzusetzen. Hierzu ist

- ein Experiment mit n Replikationen für das ProC/B-Modell zu realisieren.
- ein Experiment mit n Replikationen für das HIT-Modell zu realisieren.
- ein Hypothesentest mittels R auszuführen und zu diskutieren.

3.9.1 Das ursprüngliche Modell in HIT

Das ursprüngliche Modell für diese Aufgabe modelliert eine Literaturrecherche in der Uni-Bibliothek. Die Nutzer der Dienstleistung Literaturrecherche stellen Anfragen an ein Rechensystem, welches die entsprechenden Antworten zurückgibt. Dem ursprünglich in HIT modellierten Modell wurden folgende Parameter und Eingabewerte zugrunde gelegt:

1. Die Ankünfte der Benutzer finden in unregelmäßigen Zeitabständen statt. Diese Unregelmäßigkeiten lassen sich mit einem Variationskoeffizienten der Zwischenankunftszeit von 1,5 ausdrücken.
2. Für die Anzahl der Anfrageoptionen pro Sitzung wird eine diskrete Verteilung gefordert. Es wird eine Gleichverteilung über den ganzen Zahlen von 1 bis 19 angenommen.

3. Die maximale Anzahl der gleichzeitig geöffneten Sitzungen liegt bei n=10.

4. Die Bediengeschwindigkeit des Rechensystems liegt bei 8 Operationen pro Sekunde.

5. Die einzelnen Anfragen an das Rechensystem können parallel abgearbeitet werden.

Die Abbildung 3.31 zeigt das in HIT erstellte Modell.
Für die Auswertungen waren die Werte der Ströme *THROUGHPUT*, *TURNAROUNDTIME* und *UTILIZATION* von Interesse. Diese Werte werden auch die Ausgabe im ProC/B Modell bestimmen und Basis für die Validierung des ProC/B Modells sein.

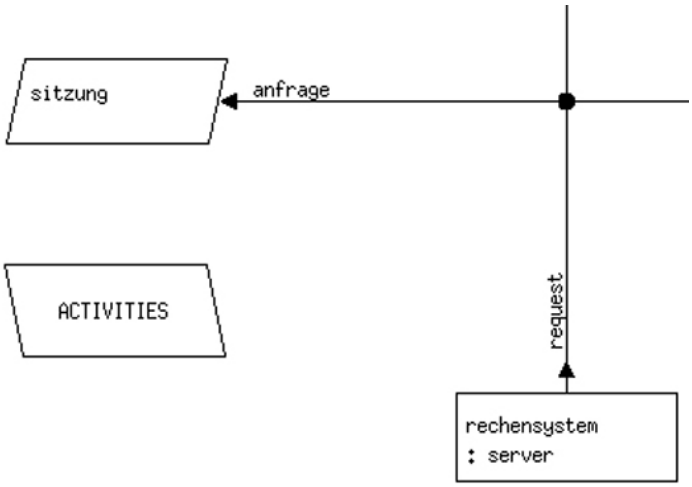


Abbildung 3.31: Das ursprüngliche in HIT

3.9.2 Das Modell in ProC/B

Im folgenden wird die Umsetzung des ursprünglichen HIT-Modells in ProC/B erläutert. Die Abbildung 3.32 zeigt die oberste Hierarchieebene des ProC/B-Modells. Hier werden ausgehend von der Quelle, gemäß den Vorgaben, mit der Verteilung *Cox(1.0,1.5)* neue Prozesse generiert. Die Aufteilung mittels des ODER-Konnektor war nötig, um die geforderte Limitierung der offenen Sitzungen einbinden zu können. In HIT erledigte dies das LIMIT 10 Statement des *CREATE*-Befehls. Die Vorgehensweise mit dem ODER-Konnektor ist deswegen nötig, weil in HIT der Parameter LIMIT 10 dafür sorgt, dass zuviel erzeugte Prozesse nicht den Weg ins Modell finden, sondern direkt wieder verworfen werden. Dieses war in ProC/B nur über den kleinen Umweg möglich. Es werden also am ODER-Konnektor solange neue Prozesse in den eigentlichen Systempfad aufgenommen, bis die maximalen 10 Sitzungen erreicht sind. Die Abfrage und Überwachung des Status wurde dabei über eine globale Variable *Prozessanzahl* realisiert.
Die Modellierung der eigentlichen Anfragen an das System werden auf der nächsten Hierarchieebene modelliert. Die zugehörige Funktionseinheit *Literaturrecherche* wird in Abbildung 3.33 gezeigt. Hier wird die Anzahl der Anfrageoptionen modelliert, nach den Anforderungen der ursprünglichen Modellierung, als Verteilung über die ganzen Zahlen 1 bis 19 *(randint(1,19))*. Das Rechensystem wird als Server modelliert, die Bediengeschwindigkeit des Servers wird auf *8* gesetzt, die Bediendisziplin auf *Prozessor Sharing*. Beide Parameter ergeben sich aus den obigen Anforderungen.

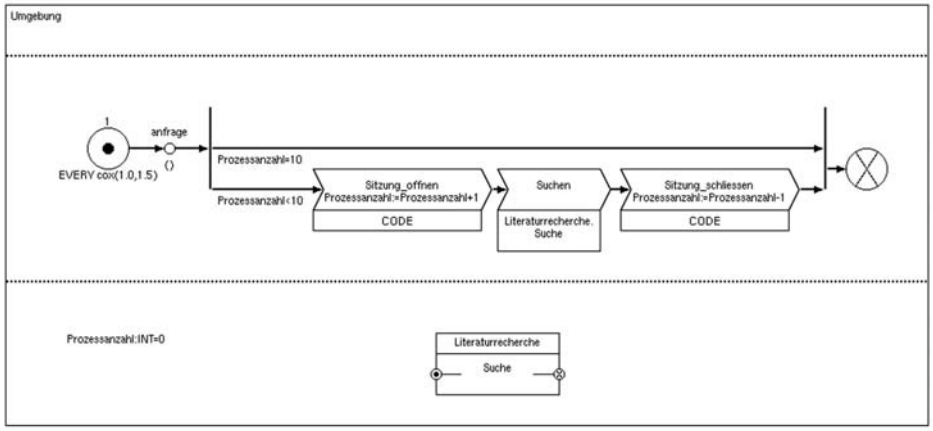


Abbildung 3.32: Modellumgebung in ProC/B

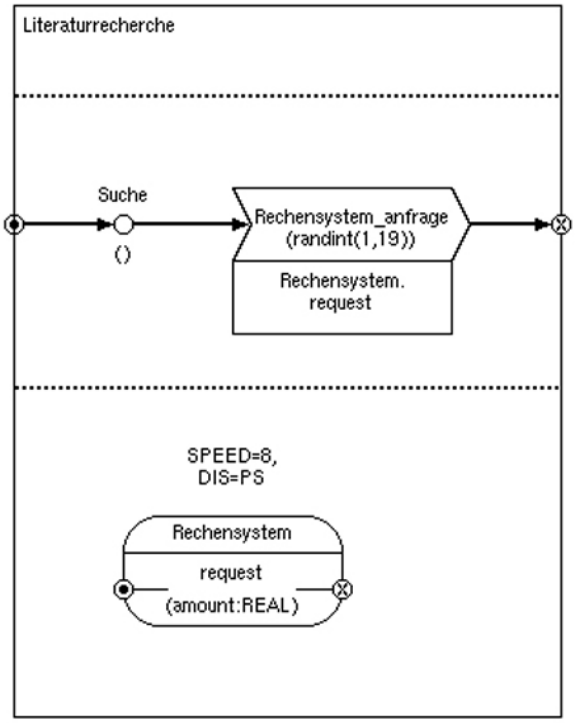


Abbildung 3.33: Literaturrecherche

Die Abbildung 3.34 zeigt die Einstellungen und die Konfiguration der Simulationsserie und Messpunkte, die dann zur Validierung und Überprüfung des Modells zugrunde gelegt werden.

Wie im ursprünglichen Modell werden auch hier wieder die Ströme *THROUGHPUT*, *TURNAROUND-TIME* und *UTILIZATION* ausgewertet. Diese werden am Server gemessen, da die Bearbeitung der Anfragen die Systemleistung bestimmt.

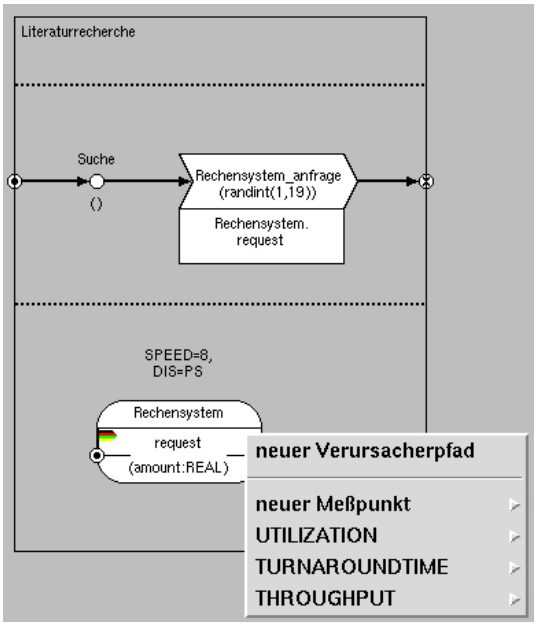


Abbildung 3.34: Messpunkte

Die Simulation wird mit den folgenden Einstellungen durchgeführt:

- Modellzeit: 28000
- Seed-Wert: 13
- Observer: 1200

3.9.3 Die Simulationsergebnisse

Die folgenden Abbildungen zeigen die ermittelten Simulationsergebnisse. Vergleicht man die Ergebnisse des ProC/B-Modells mit denen aus der HIT-Simulation (siehe folgende Tabellen), so stellt man fest, dass diese sehr nahe beieinander liegen. Die Vermutung das die Modelle die gleichen Systemleistungen modellieren liegt nahe, muss jedoch zusätzlich noch überprüft und validiert werden.

Ergebnisse der ProC/B-Simulation	Mean Value	Konf.Int 90%
Throughput	0.74	±0.90%
Turnaroundtime	8.81	± 1.59%
Utilization	0.96	±0.66%

Ergebnisse der HIT-Simulation	Mean Value	Konf.Int 95%
Throughput	0.75	±2.50%
Turnaroundtime	8.68	± 4.80%
Utilization	0.93	±1.86%

3.9.4 Validierung der Modelle

Zur genauen Analyse, ob die beiden Modelle tatsächlich wie gefordert dieselben Daten und Verhältnisse modellieren, genügt ein einfacher Vergleich der Simulationsergebnisse jedoch keineswegs. Hierfür bedarf es genauerer statistischer Test und Analysen. Für unsere beiden Modelle wird die Validierung anhand des von Gruppe 1 erstellten Kochrezeptes unter Anwendung des Statistikpaketes R durchgeführt. Hierfür sind einige manuelle Änderungen am Quellcode der beiden Modelle notwendig. Diese werden im folgenden näher gezeigt und grob erläutert. Für detaillierte Informationen vergleiche Kapitel 3.7.

Experiment mit n Replikationen für das ProC/B-Modell

Nach der Simulation des ProC/B-Modells wird der HI-SLANG Code *LiteraturCounter2.E1.hit* generiert und dann manuell für die n Durchläufe überarbeitet. Der endgültige Code wird in den Abbildungen 3.35 bis 3.37 dargestellt.

Die Ausführung des Befehls *„hit LiteraturCounter2.E1.hit“* erzeugt eine Tabelle *„procb.data“*, die die Ergebnisse der n Durchläufe der Simulation für die Laufzeit, Durchsatz und Auslastung repräsentiert (siehe Abb. 3.38).

```
%COMMON
%PARM = WARN
%BIND "PROC_AUS" TO /home/pg476/glasirin/procb.data
%END
%COPY "counter"
%COPY "semaphor"
%COPY "observer"
%COPY "prioserver"
TYPE voidRecordName RECORD; END TYPE voidRecordName;

TYPE F1Umgebung_Typ MODEL;

VARIABLE
Prozessanzahl : INTEGER DEFAULT
0;

COMPONENT modellobserver : observer(28800/1200);

TYPE F88Literaturrecherche_Typ COMPONENT;

PROVIDE
SERVICE P127Suche(data : POINTER FOR REC127_Suche);
END PROVIDE;

COMPONENT F122Rechensystem : server
(LET ACCEPT   := ALWAYS,
 LET SCHEDULE := IMMEDIATE,
 LET DISPATCH := SHARED (LET SPEED := 8*1),
 LET OFFER    := ALL);

TYPE REC127_Suche RECORD;
END {RECORD} TYPE REC127_Suche;

TYPE P127Suche SERVICE(data : POINTER FOR REC127_Suche);
USE
SERVICE L112Rechensystem_anfrage(B2H___amount : REAL DEFAULT
1.0);
END USE;

BEGIN
NEW REC127_Suche POINTER data;

L112Rechensystem_anfrage(
randint(1,19));

END {SERVICE} TYPE P127Suche;
```

```
REFER
P127Suche
TO
F122Rechensystem
EQUATING
P127Suche.L112Rechensystem_anfrage
WITH F122Rechensystem.request;
END REFER;

END {COMPONENT} TYPE F88Literaturrecherche_Typ;

COMPONENT F88Literaturrecherche : F88Literaturrecherche_Typ;

TYPE REC5_anfrage RECORD;
END {RECORD} TYPE REC5_anfrage;

TYPE P5anfrage SERVICE(data : POINTER FOR REC5_anfrage);
USE
SERVICE L134Suchen(data : POINTER FOR voidRecordName);
END USE;

BEGIN
NEW REC5_anfrage POINTER data;


IF
Prozessanzahl=10
THEN
{*** DUMMY-Kommentar da kein PKE vorhanden ist! ***};
ELSE
IF
Prozessanzahl<10
THEN

Prozessanzahl:=Prozessanzahl+1;L134Suchen(
NONE);

Prozessanzahl:=Prozessanzahl-1;ELSE
{Fehlerabfrage fuer Fehlenden ELSE-Zweig.}
WRITELN "FEHLER: Fehlender ELSE-Zweig im boolschen Oder-Konnektor ";
WRITELN "        wird benoetigt.";
WRITELN "        (siehe Umgebung.anfrage)";
WRITELN "Simulationsstop durch (absichtlichen) RUNTIME-ERROR.";
IF (1=1/0) THEN WRITELN "-> Vorzeitiger Stop <-"; END IF;
END IF;
END IF;
```

Abbildung 3.35: HI-SLANG Teil 1 ProC/B-Modell

```
END {SERVICE} TYPE P5anfrage;

TYPE UnbedQuellen SERVICE;
BEGIN
LOOP
BLOCK
VARIABLE
hitLoopSchleifenZaehler       : INTEGER;
hitLoopSchleifenProzessanzahl : INTEGER;
BEGIN
hitLoopSchleifenZaehler := 0;
hitLoopSchleifenProzessanzahl := 1;
WHILE hitLoopSchleifenZaehler < hitLoopSchleifenProzessanzahl
LOOP
CREATE 1 PROCESS P5anfrage(NONE);
hitLoopSchleifenZaehler := hitLoopSchleifenZaehler +1;
END LOOP;
END BLOCK;
hold(cox(1.0,1.5));
END LOOP;
END {SERVICE} TYPE UnbedQuellen;

TYPE StatusAusgabe SERVICE;
  VARIABLE prozent:INTEGER DEFAULT 0;
           schritt:REAL DEFAULT 0;
  BEGIN
      WRITELN "Die Simulation startet jetzt...";
      {*** Status-Ausgabe in Prozent alle 5% ***}
      schritt := 28800*0.05;
      LOOP
        prozent := prozent + 5;
        hold(schritt-0.00000001);
        WRITELN prozent::3, "%";
      END LOOP;
  END {SERVICE} TYPE StatusAusgabe;
```

```
REFER
P5anfrage
TO
F88Literaturrecherche
EQUATING
P5anfrage.L134Suchen
WITH F88Literaturrecherche.P127Suche;
END REFER;

BEGIN
WRITELN("Die Simulation startet jetzt...");
CREATE 1 PROCESS UnbedQuellen;
{*** Aktivierung der Statusausgabe, bei Observer deaktiviert ***}
{*** CREATE 1 PROCESS StatusAusgabe; ***}
END {MODEL} TYPE F1Umgebung_Typ;

EXPERIMENT versuch METHOD SIMULATIVE;

VARIABLE
replic   :INTEGER;
startseed :REAL;
myseed   :REAL;
uti      :REAL;
tput     :REAL;
turn     :REAL;
n        :REAL;
run      :INTEGER;

f1       :OUTFILE;

BEGIN
OPEN f1, "PROC_AUS" LENGTH 80;
 replic := 20;
   n      := 2**30//replic+1;
   startseed, myseed := 13;
       FOR run := 1 STEP 1 UNTIL replic
           LOOP
```

Abbildung 3.36: HI-SLANG Teil 2 ProC/B-Modell

```
EVALUATE MODEL F1Umgebung : F1Umgebung_Typ(LET SEED := myseed);
EVALUATIONOBJECT
messgeraet VIA F1Umgebung;
el22rechensy VIA
F1Umgebung.F88Literaturrecherche.F122Rechensystem
DEFAULT ESTIMATOR
  CONFIDENCE LEVEL 90
OUTPUT TABLE "Table", DUMPFILE "DUMP";
       myseed := get_seed (startseed, n);
              startseed := myseed;

              uti := get_result(,"UTILIZATION", "el22rechensy",,,run);
              tput := get_result(,"THROUGHPUT", "el22rechensy",,,run);
              turn := get_result(,"TURNAROUNDTIME", "el22rechensy",,,run);

              WRITELN FILE fl, uti, tput, turn;

       END LOOP;
       CLOSE fl;
END EXPERIMENT versuch;
BEGIN
MEASURE THROUGHPUT AT messgeraet;

MEASURE THROUGHPUT AT el22rechensy DUE TO ALL ABSCISSA run ESTIMATOR MEAN, STANDARDDEVIATION,
 CONFIDENCE LEVEL 95 GLOBALSTOP WIDTH 5;

MEASURE TURNAROUNDTIME AT el22rechensy DUE TO ALL ABSCISSA run ESTIMATOR MEAN, STANDARDDEVIATION,
 CONFIDENCE LEVEL 95 GLOBALSTOP WIDTH 5;

MEASURE UTILIZATION AT el22rechensy DUE TO ALL ABSCISSA run ESTIMATOR MEAN, STANDARDDEVIATION,
 CONFIDENCE LEVEL 95 GLOBALSTOP WIDTH 5;

CONTROL AT messgeraet STOP MODELTIME 28800;
END EVALUATE;
```

Abbildung 3.37: HI-SLANG Teil 3 ProC/B-Modell

```
9.323420E-001 7.458536E-001 8.675066E+000
9.209221E-001 7.390172E-001 8.615058E+000
9.024404E-001 7.168512E-001 8.398308E+000
9.346539E-001 7.467182E-001 8.855905E+000
9.244102E-001 7.427591E-001 8.442037E+000
9.380030E-001 7.512216E-001 8.643193E+000
9.070594E-001 7.307064E-001 8.389734E+000
9.240446E-001 7.346845E-001 8.649144E+000
9.281841E-001 7.430636E-001 8.636684E+000
9.397072E-001 7.460827E-001 8.921289E+000
9.089221E-001 7.345683E-001 8.328032E+000
9.340721E-001 7.386450E-001 9.038201E+000
9.324796E-001 7.309391E-001 8.900195E+000
9.135776E-001 7.214642E-001 8.873042E+000
9.277547E-001 7.387834E-001 8.690822E+000
9.343380E-001 7.412744E-001 9.019265E+000
9.220919E-001 7.166349E-001 8.781430E+000
9.275175E-001 7.474181E-001 8.617054E+000
9.344758E-001 7.419578E-001 8.756619E+000
9.123736E-001 7.465306E-001 8.391804E+000
```

Abbildung 3.38: procb.data

Experiment mit n Replikationen für das HIT-Modell Nach der Simulation des HIT-Modells wird der HI-SLANG Code *„bib1_vert_exp.hit“* generiert und dann manuell für die n Durchlaufe überarbeitet. Der endgültige Code wird in den Abbildungen und dargestellt.

Die Ausführung des Befehls *„hit bib1_vert_exp.hit“* erzeugt eine Tabelle *„hit.data“*, die die Ergebnisse der n Durchläufe der Simulation für die Laufzeit, Durchsatz und Auslastung repräsentiert (siehe Abb. 3.41).

```
%COMMON
%PARM = INDENT = 0
%BIND "HIT_AUS" TO /home/pg476/glasirin/hit.data
%END

%TITLE MODEL TYPE bibl_verteilung
TYPE bibl_verteilung MODEL;

   {DECLARATION OF SERVICE sitzung}
   {WITHIN COMPONENT/MODEL TYPE bibl_verteilung}

   TYPE sitzung SERVICE;

      USE {of service/procedure sitzung}
         SERVICE   anfrage (
            amount:REAL
            )
;
      END USE;  {of service/procedure sitzung}



      {START OF BODY OF SERVICE/PROCEDURE sitzung}
      {WITHIN COMPONENT/MODEL TYPE bibl_verteilung}

      BEGIN
         anfrage (randint (1,19)) ;
;

   END {SERVICE} TYPE sitzung;

   {END OF DECLARATION OF SERVICE  sitzung}
   {WITHIN COMPONENT/MODEL TYPE bibl_verteilung}


%TITLE MODEL/COMPONENT TYPE bibl_verteilung
   COMPONENT rechensystem : server (
      LET  DISPATCH  :=  SHARED  (
         LET speed := 8.0

         ) ) ;

   REFER {of component/model type bibl_verteilung}
      sitzung
   TO
      rechensystem
   EQUATING
      sitzung.anfrage WITH rechensystem.request;
   END REFER;  {of component/model type bibl_verteilung}


%TITLE ACTIVITIES OF MODEL TYPE bibl_verteilung
   {START OF ACTIVITIES (BODY) OF MODEL TYPE bibl_verteilung}

   BEGIN
      CREATE 1 PROCESS sitzung LIMIT 10 EVERY cox(1.0,1.5);
;

END {MODEL} TYPE bibl_verteilung;

{END OF DECLARATION OF MODEL TYPE  bibl_verteilung}


%TITLE EXPERIMENT bibl_vert_exp
{HI-SLANG code for experiment}
{                bibl_vert_exp}
{generated by    HITGRAPHIC V. 3.7.16}
{          at    06.12.05 12:41}
{    for user    glasirin}
{          on    bube sun4u SunOS 5.9}

EXPERIMENT bibl_vert_exp METHOD SIMULATIVE;
VARIABLE
replic : INTEGER;
startseed : REAL;
myseed : REAL;
uti : REAL;
tput : REAL;
turn: REAL;
n : REAL;
run: INTEGER;
uti_t :TEXT;
tput_t :TEXT;
turn_t :TEXT;

fl : OUTFILE;


%TITLE BODY OF EXPERIMENT bibl_vert_exp
{START OF BODY OF EXPERIMENT bibl_vert_exp}

BEGIN

   OPEN fl, "HIT_AUS" LENGTH 80;

   replic := 20;
```

Abbildung 3.39: HI-SLANG Teil 1 HIT-Modell

```
   n :=  2**30//replic+1;
   startseed, myseed := 17;
   FOR run := 1 STEP 1 UNTIL replic
     LOOP


%TITLE EVALUATION bibl_vert_eva
   {START OF EVALUATION OF MODEL TYPE bibl_verteilung}

   EVALUATE

      MODEL bibl_vert_eva : bibl_verteilung
(
 LET seed:= myseed
)
   ;


   EVALUATIONOBJECT
   {for model bibl_vert_eva}
      rs_reaktion VIA bibl_vert_eva.
         rechensystem
      DEFAULT
      OUTPUT       TABLE "TABLE",
      DUMPFILE "DUMP";

   {START OF MEASURE STATEMENTS OF EVALUATION bibl_vert_eva}

   BEGIN

   MEASURE THROUGHPUT DEGREE 10,
      TURNAROUNDTIME DEGREE 10,
      UTILIZATION DEGREE 10
      AT rs_reaktion
      DUE TO
         ALL
      ABSCISSA run
      ESTIMATOR
         MEAN,
         STANDARDDEVIATION,
         CONFIDENCE LEVEL 95
      GLOBALSTOP
         WIDTH 5
      ;
```

```
   CONTROL

      AT rs_reaktion
        STOP MODELTIME 28800

      AT rs_reaktion
        STOP CPUTIME 120

   ;
   END EVALUATE;

   {END OF EVALUATION  bibl_vert_eva FOR MODEL TYPE bibl_verteilung}



     myseed := get_seed(startseed, n);
     startseed := myseed;

     uti := get_result(,"UTILIZATION","rs_reaktion",,,run);
     tput := get_result(,"THROUGHPUT","rs_reaktion",,,run);
     turn := get_result(,"TURNAROUNDTIME","rs_reaktion",,,run);

     WRITELN FILE fl, uti, tput, turn;

     END LOOP;
     CLOSE fl;
;

END EXPERIMENT bibl_vert_exp;

{END OF EXPERIMENT  bibl_vert_exp}
```

Abbildung 3.40: HI-SLANG Teil 2 HIT-Modell

```
9.121683E-001 7.336217E-001 8.496927E+000
9.288425E-001 7.353850E-001 8.723402E+000
9.348759E-001 7.448931E-001 8.771201E+000
9.209644E-001 7.325673E-001 8.688626E+000
9.404658E-001 7.619351E-001 8.818952E+000
9.133285E-001 7.184890E-001 8.999235E+000
9.364227E-001 7.521210E-001 9.005321E+000
9.244086E-001 7.404096E-001 8.606005E+000
9.177080E-001 7.387848E-001 8.737727E+000
9.168059E-001 7.304751E-001 8.576509E+000
9.246551E-001 7.402789E-001 8.622616E+000
9.229038E-001 7.324466E-001 8.612265E+000
9.233059E-001 7.316737E-001 8.591564E+000
9.229239E-001 7.361581E-001 8.762580E+000
9.282776E-001 7.424304E-001 8.919614E+000
9.301320E-001 7.441634E-001 8.769227E+000
9.251907E-001 7.440750E-001 8.700751E+000
9.446980E-001 7.508003E-001 9.060860E+000
9.281165E-001 7.358214E-001 8.934560E+000
9.218613E-001 7.396242E-001 8.592857E+000
```

Abbildung 3.41: hit.data

Hypothesentest mittels R

Die Befehle der R-Umgebung, die bei der Validierung benutzt wurden, sind in der Datei *hit.R* gespeichert. Nach der Ausführung des Befehls *source("hit.R")* werden die Tabellen verglichen, und mit den Befehlen *t.test(turn_h,turn_p);*, *t.test(uti_h,uti_p);*, *t.test(tput_h,tput_p);*, kann man die Ergebnisse des Vergleiches zeigen lassen, wobei *turn* für *TURNAROUNDTIME*, *uti* für *UTILIZATION*, *tput* für *THROUGHPUT* stehen (Abb. 3.42).

```
hit <- scan("hit.data", list(0,0,0));
proc <- scan("procb.data", list(0,0,0));
uti_h <- hit[[1]];
tput_h <- hit[[2]];
turn_h <- hit[[3]];
uti_p <- proc[[1]];
tput_p <- proc[[2]];
turn_p <- proc[[3]];
t.test(uti_h,uti_p);
t.test(tput_h,tput_p);
t.test(turn_h,turn_p);
```

Abbildung 3.42: R-Befehle

Die Ergebnisse der Validierung mittels des R Paketes sind in Abbildung 3.43 dargestellt. Da die von den drei T-Test Durchläufen errechneten Konfidenzintervalle für die Übereinstimmung der Messwerte alle die *0* einschließen kommt man mittels dieser Ergebnisse der einzelnen Tests zu dem Schluss, dass beide Modelle die gleichen Systemleistungen modellieren.

```
> t.test(uti_h, uti_p)

        Welch Two Sample t-test

data:  uti_h and uti_p
t = 0.302, df = 35.727, p-value = 0.7644
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.005341886  0.007210446
sample estimates:
mean of x mean of y
0.9259028 0.9249685


> t.test(tput_h, tput_p)

        Welch Two Sample t-test

data:  tput_h and tput_p
t = 0.5069, df = 37.752, p-value = 0.6151
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.004638068  0.007736048
sample estimates:
mean of x mean of y
0.7393077 0.7377587


> t.test(turn_h, turn_p)

        Welch Two Sample t-test

data:  turn_h and turn_p
t = 1.1376, df = 35.368, p-value = 0.2629
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.05361218  0.19040388
sample estimates:
mean of x mean of y
 8.749540  8.681144
```

Abbildung 3.43: Vergleichsergebnisse

4 Projektmanagement

Da die Bottleneckanalyse des (Informatik-) Studiums an der Universität Dortmund eine recht komplexe Aufgabe darstellt, deren Bearbeitung den langen Zeitraum von einem Jahr umfasst, haben sich die Beteiligten unter der Anregung des Projektgruppenleiters Jürgen Mäter früh dafür entschieden ein Projektmanagement zu betreiben. Dies sollte gewährleisten, dass durchgängig der aktuelle Projektstatus im Auge behalten wurde, mit dem Ziel die geplanten Termine einzuhalten.

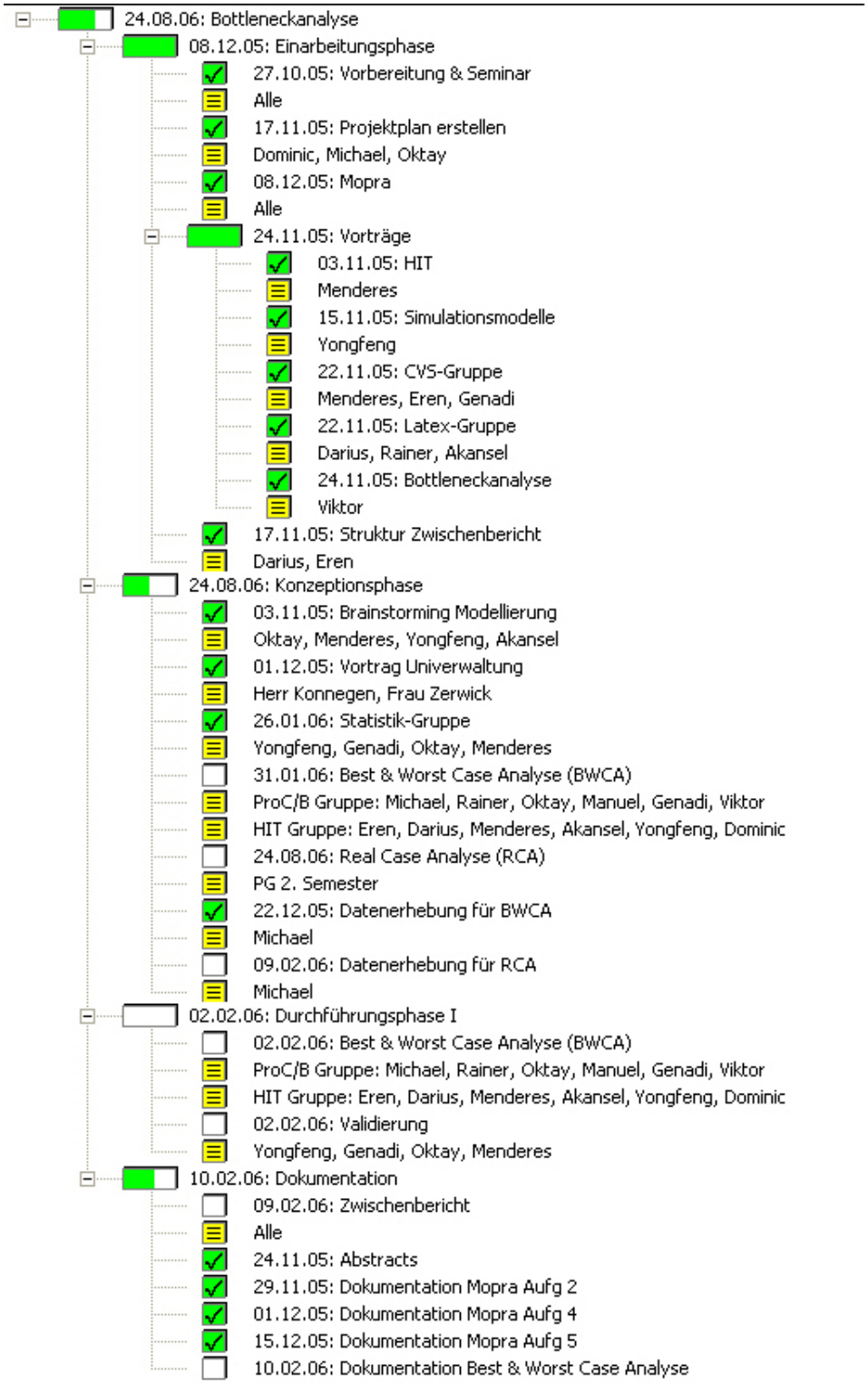
Aus diesem Grund wurde nach einem Monat Einarbeitungszeit eine für das Projektmanagement verantwortliche Gruppe festgelegt, bestehend aus Dominic Ketteltasche, Oktay Özdemir und Michael Thüner. Die Gruppe hat nach ihrer Ernennung einen ersten Projektplan aufgestellt und drei potentielle Tools aus der breiten Masse der Projektmanagement-Tools herausgesucht. Die Projektgruppe hat sich aus diesen auf das Tool ProMa FREEWARE 4.08 geeinigt.

Innerhalb der Arbeit in der Projektgruppe konnten viele Aufgaben parallel bearbeitet werden, so dass die zeitlichen Abhängigkeiten untereinander sehr eingeschränkt waren. Deshalb ist es in unserem Fall nicht nötig gewesen mit zum Beispiel Netzplänen zu arbeiten, weshalb die Wahl auf das genannte Tool gefallen ist. Es zeichnet sich durch seine einfache Baumstruktur aus, mit deren Hilfe das Projekt in Unterprojekte und Aufgaben unterteilt wird. Die Aufgaben können nach erfolgreicher Bearbeitung abgehakt werden, was direkt an dem Statusbalken des entsprechenden Unterprojekts sichtbar wird. Für jedes Unterprojekt und jede Aufgabe ist der geplante Endtermin einzutragen. Bei Bedarf können zu den einzelnen Aufgaben noch Notizen, wie zum Beispiel die verantwortlichen Gruppenmitglieder, hinterlegt werden.

Die erste Planung orientierte sich an dem Projektplan aus dem PG-Antrag und teilte sich auf in die vier Hauptphasen Einarbeitungsphase, Konzeptionsphase, Durchführungsphase und Dokumentation. Dabei hat die Planung bewusst die vorlesungsfreien Zeiten außer Acht gelassen. Die Einarbeitungsphase umfasste unter anderem die einleitende Seminarphase, einige weitere vorbereitende Vorträge sowie das Modellierungspraktikum. In der Konzeptionsphase fanden für den Best- und Worstcase und den Realcase die Datenerhebungen statt und es wurden die entsprechenden Modelle aufgestellt, wobei der Best- und der Worstcase in der Durchführungsphase auch modelliert wurden. Dabei ist zu beachten, dass sich die Phasen stellenweise überlappten. Die Dokumentation als solche ist nicht als eigentliche Phase zu verstehen, sondern steht dafür, dass durchgängig die absolvierten Arbeiten dokumentiert und in den Zwischenbericht eingefügt wurden.

Neben diesen Hauptaufgaben wurden regelmäßig kleinere aktuelle Aufgaben in den Sitzungen der Projektgruppe verteilt. Dabei wurden dann auch direkt die Verantwortlichen und der Endtermin festgelegt. Bei besonderer Wichtigkeit dieser Aufgaben wurden diese von der Gruppe Projektmanagement in den bestehenden Projektplan eingepflegt. Neben der Aktualisierung der Planung gehörte es zu den Aufgaben der Gruppe Projektmanagement regelmäßig auf wichtige einzuhaltende Termine hinzuweisen.

Trotzdem kam es aufgrund einiger Probleme im Modellierungspraktikum zu Terminverschiebungen, weshalb die gesamte Projektgruppe zu dem Entschluss kam, sich in den Weihnachtsferien zu treffen, um die verlorenen Zeit wieder aufzuholen.


24.08.06: Bottleneckanalyse
08.12.05: Einarbeitungsphase
27.10.05: Vorbereitung & Seminar
Alle
17.11.05: Projektplan erstellen
Dominic, Michael, Oktay
08.12.05: Mopra
Alle
24.11.05: Vorträge
03.11.05: HIT
Menderes
15.11.05: Simulationsmodelle
Yongfeng
22.11.05: CVS-Gruppe
Menderes, Eren, Genadi
22.11.05: Latex-Gruppe
Darius, Rainer, Akansel
24.11.05: Bottleneckanalyse
Viktor
17.11.05: Struktur Zwischenbericht
Darius, Eren
24.08.06: Konzeptionsphase
03.11.05: Brainstorming Modellierung
Oktay, Menderes, Yongfeng, Akansel
01.12.05: Vortrag Univerwaltung
Herr Konnegen, Frau Zerwick
26.01.06: Statistik-Gruppe
Yongfeng, Genadi, Oktay, Menderes
31.01.06: Best & Worst Case Analyse (BWCA)
ProC/B Gruppe: Michael, Rainer, Oktay, Manuel, Genadi, Viktor
HIT Gruppe: Eren, Darius, Menderes, Akansel, Yongfeng, Dominic
24.08.06: Real Case Analyse (RCA)
PG 2. Semester
22.12.05: Datenerhebung für BWCA
Michael
09.02.06: Datenerhebung für RCA
Michael
02.02.06: Durchführungsphase I
02.02.06: Best & Worst Case Analyse (BWCA)
ProC/B Gruppe: Michael, Rainer, Oktay, Manuel, Genadi, Viktor
HIT Gruppe: Eren, Darius, Menderes, Akansel, Yongfeng, Dominic
02.02.06: Validierung
Yongfeng, Genadi, Oktay, Menderes
10.02.06: Dokumentation
09.02.06: Zwischenbericht
Alle
24.11.05: Abstracts
29.11.05: Dokumentation Mopra Aufg 2
01.12.05: Dokumentation Mopra Aufg 4
15.12.05: Dokumentation Mopra Aufg 5
10.02.06: Dokumentation Best & Worst Case Analyse

5 Simulationsstudie

Das fünfte Kapitel widmet sich der eigentlichen Simulation. Dabei wird versucht sich an den 10 Schritten einer Simulationsstudie gemäß [Law 91] zu orientieren, von denen einige im Zuge der Simulationsstudie zusammengefasst worden sind. Diese wären: Problemstellung und Planung der Studie, Datenerhebung und Modellentwurf, Validierung, Modellimplementierung und Modellverifikation, Pilotläufe, ein weiterer Validierungsschritt, Experimentenentwurf, Produktionsläufe, Analyse der Produktionsläufe und Aufbereitung der Ergebnisse.

5.1 Problemstellung und Planung der Studie

Die Projektgruppe beschäftigt sich mit dem so genannten Best- bzw. Worstcase der Studiensituation. Anhand des im folgenden vorgestellten abstrakten Modells des Fachbereiches Informatik an der Universität Dortmund wurde die Übereinkunft getroffen, dass lediglich die Kapazität der Ressourcen einzuschränken (Worstcase) beziehungsweise als unbeschränkt (Bestcase) vorauszusetzen ist. Zudem werden die beiden genutzten Modellierungstools einer echten Bewährungsprobe unterzogen, so dass etwaige Probleme zu Tage treten und an die verantwortlichen Entwickler der Tools herangetragen werden können.

5.2 Modellentwurf und Datenerhebung

Dieser Abschnitt widmet sich dem Entwurf eines allgemeinen Strukturmodells einer Universität, des Ablaufes sowohl des Grundstudiums als auch des Hauptstudiums, gegliedert nach den Studiengängen Kerninformatik und Angewandte Informatik des Fachbereiches Informatik an der Universität Dortmund. Mittels dieser Entwürfe werden anschließend die benötigten Daten erhoben.

5.2.1 Struktur- und Ablaufdiagramme

Bevor das abstrakte Modell erstellt werden kann, müssen eine einheitliche Syntax und Semantik festgelegt werden. Diese werden in der Abbildung 5.1 dargestellt. Die gesamte Struktur des abstrakten Modells wurde in drei Teile aufgeteilt: *Strukturmodell der Universität Dortmund, Strukturmodell des Grundstudiums Kerninformatik bzw. Angewandte Informatik* und *Strukturmodell des Hauptstudiums Kerninformatik bzw. Angewandte Informatik*.

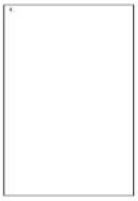
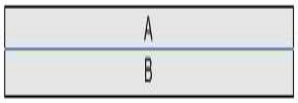
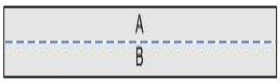
Symbol	Beschreibung
	Der Abhängigkeitspfeil wird zur Darstellung der Abhängigkeit zwischen zwei oder mehreren Vorlesungen benutzt. Vorlesungen, die durch diesen Pfeil verbunden sind, werden nur der Reihe nach „abgearbeitet“ und die Reihenfolge darf nicht geändert werden.
	Der Überganspfeil zeigt die Ablauffolge des Datenflusses bzw. der Studentenfluss während des Studiums.
	Die Vorlesung wird durch eine Elementeinheit in der Form eines horizontalen Rechteckes mit der Vorlesungsbezeichnung in der Mitte dargestellt.
	Das Semester wird durch eine Elementeinheit in der Form eines vertikalen Rechteckes mit der Semesternummer in der linken oberen Ecke dargestellt.
	Die Elementeinheit mit UND-Konnektor wird durch ein Rechteck mit durchgezogener Linie in der Mitte dargestellt und repräsentiert eine Menge von Elementen bzw. Vorlesungen (A, B), die unbedingt ausgewählt werden müssen, entweder parallel oder nacheinander.
	Die Elementeinheit mit ODER-Konnektor wird durch ein Rechteck mit gestrichelter Linie in der Mitte dargestellt und repräsentiert eine Menge von Elementen bzw. Vorlesungen (A,B), von deren nur eine ausgewählt werden muss, entweder A oder B.
	Die Ellipse mit der Bezeichnung in der Mitte repräsentiert die benötigten Ressourcen, zum Beispiel Personal oder Bücher.
	Das Rechteck mit der Überschrift wird zur Darstellung des allgemeinen Konzeptes oder globalen Einheiten wie *Universität* oder *Fachbereich* angewendet.

Abbildung 5.1: Legende

5.2.1.1 Strukturmodell der Universität Dortmund

Das *Strukturmodell* der Universität Dortmund bildet die Grundlage für die Simulation und die spätere *Bottleneckanalyse*. Das Modell ist dabei hierarchisch gegliedert und soll eine zunächst grobe Struktur der Universität Dortmund wiedergeben. Die einzelnen Hierarchiestufen umfassen

- Ebenen, aus denen die Uni (im groben) besteht
- Ressourcen, wie z.B. Räume, Personal und angebotene Vorlesungen/Veranstaltungen, die während des Studiums an der Uni benötigt werden

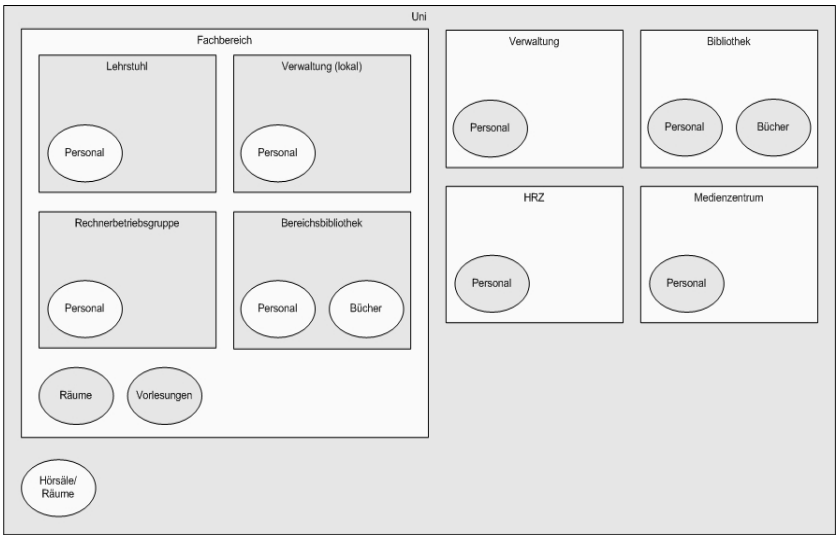


Abbildung 5.2: Strukturmodell der Universität

Auf der obersten Ebene steht die Universität selbst. Diese verfügt dabei über die Ressource *Hörsäle/ Räume*. Des weiteren liegen fünf weitere Ebenen vor, die direkt unter der Ebene *Uni* liegen. Diese sind

- der Typ Fachbereich (stellvertretend für alle Fachbereiche),
- die Verwaltung,
- die Bibliothek,
- das Medienzentrum und
- das Hochschulrechenzentrum (HRZ).

Bevor auf den Typ Fachbereich genauer eingegangen wird, sollen zunächst die Verwaltung, das Medienzentrum, das HRZ und die Bibliothek genauer betrachtet werden. Alle besitzen als Ressource das *Personal*. *Personal* umfasst die Menge an Arbeitskräften, welche dem entsprechenden Bereich zur Verfügung stehen. Die Bibliothek hat eine zusätzliche Ressource *Bücher* welche den Bestand an Büchern angibt. Der Typ Fachbereich besteht aus den folgenden Komponenten:

- dem Lehrstuhl,

- der Rechnerbetriebsgruppe,
- der lokalen Verwaltung und
- der Bereichsbibliothek.

Der Fachbereich selber besitzt zwei Ressourcen: *Räume* und *Vorlesungen*. *Räume* stellt dabei die dem Fachbereich zustehenden Räumlichkeiten, *Vorlesungen* die vom Fachbereich angebotenen Vorlesungen dar. Die Subebenen Lehrstuhl, Verwaltung und Rechnerbetriebsgruppe lassen sich wieder relativ gleich beschreiben. Sie alle besitzen eine Ressource *Personal*. Diese lässt sich in völlig analoger Weise beschreiben, wie bereits für die Ebenen Verwaltung, Bibliothek, HRZ und Medienzentrum. Einzig die Bereichsbibliothek des Fachbereichs besitzt eine weitere Ressource *Bücher*. Da sich die Bereichsbibliothek nicht von der allgemeinen Bibliothek im hier angegeben Aufbau unterscheidet, wird hier auf eine erneute Beschreibung verzichtet.

5.2.1.2 Strukturmodell des Grundstudiums Kerninformatik

Das Grundstudium des Kerninformatikstudiums an der Universität Dortmund hat einen bestimmten vorgegeben Ablaufplan. Dieser Ablaufplan ist in vier Semester unterteilt. Nach der Diplomprüfungsordnung 2001 [DPO 01] sollen folgende Prüfungen in den jeweiligen Semestern abgelegt werden:

- **1. Semester** :
  - Datenstrukturen, Algorithmen und Programmierung 1 (DAP1)
  - Rechnerstrukturen (RS)
  - Mathematik 1: Lineare Algebra und Analysis (M1)
- **2. Semester** :
  - Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)
  - Elektrotechnik und Nachrichtentechnik (ET/NT)
  - Betriebssysteme, Rechnernetze und verteilte Systeme 1 (BS1)
  - Mathematik 2: Diskrete Strukturen und Algebra (M2)
- **3. Semester** :
  - Softwaretechnik (SWT)
  - Hardwarepraktikum (HaPra)
  - Betriebssysteme Rechnernetze und verteilte Systeme 2 (BS2)
  - Wahrscheinlichkeitsrechnung und mathematische Statistik (WR)
  - Logik
- **4. Semester** :
  - Softwarepraktikum (SoPra)
  - Informationssysteme (IS)
  - Grundlagen der theoretischen Informatik (GTI)
  - Proseminar (ProSem)

Die Übergangspfeile in der Abbildung 5.3 zeigen an, dass die Semester nacheinander durchlaufen werden.
Die Teilnahmevoraussetzungen für *HaPra* bzw. *SoPra* sind von der erfolgreichen Teilnahme an *RS* bzw. *DAP1*, *DAP2* und *SWT* abhängig. Diese werden durch die Abhängigkeitspfeile in der Abbildung 5.3 dargestellt.
Für die gesamte Studienzeit wählt der Studierende eines von insgesamt 22 angebotenen Nebenfächern aus. Diese haben im Grundstudium jeweils einen Umfang von 12-19 Semesterwochenstunden (SWS). Die am häufigsten gewählten Nebenfächer sind *BWL*, *Theoretische Medizin*, *Elektrotechnik* und *Mathematik*. Wobei für Studenten mit den Nebenfächer *Mathematik* und *Elektrotechnik* der Ablaufplan des Grundstudiums teilweise anders aussieht. Für Studenten mit dem Nebenfach *Elektrotechnik* wird das *HaPra* durch das *EPra* (Digitalelektronisches Praktikum) ersetzt, was die gleiche Anzahl an SWS in Anspruch nimmt. Für Studenten mit dem Nebenfach *Mathematik* werden die Vorlesungen *M1 (Lineare Algebra und Analysis)* und *M2 (Diskrete Strukturen und Algebra)* durch gleichwertige Vorlesungen ersetzt, die aber den Lernstoff ausführlicher behandeln.

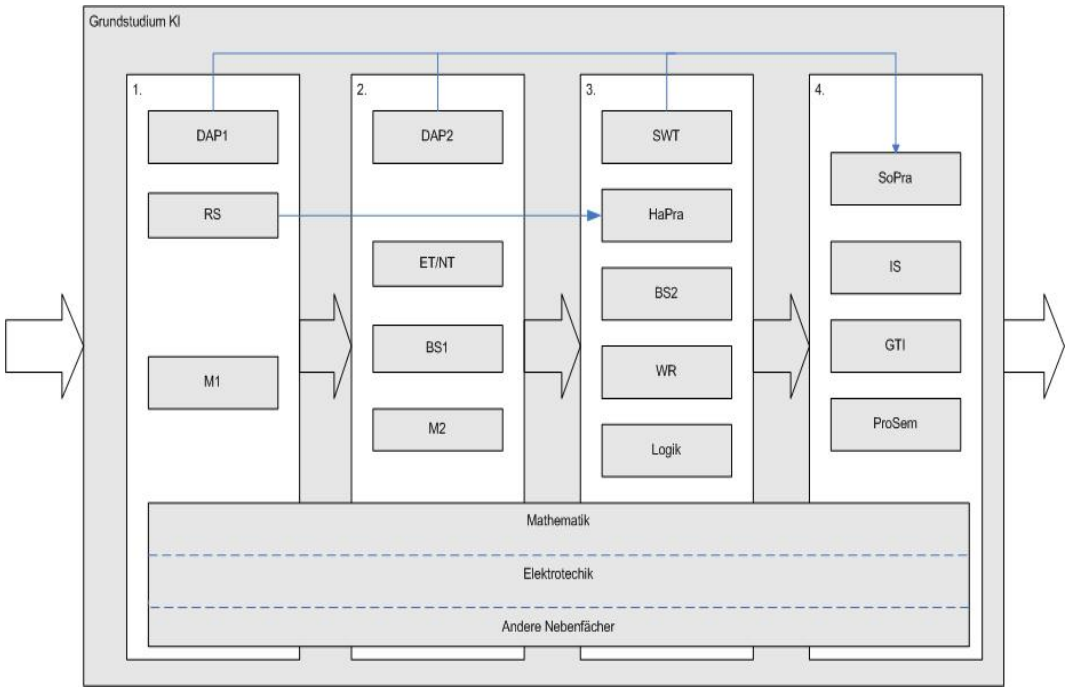


Abbildung 5.3: Ablaufplan des Grundstudiums für Kerninformatik

5.2.1.3 Strukturmodell des Grundstudiums Angewandte Informatik

Als Alternative zu der Kerninformatik wird an der Universität Dortmund Angewandte Informatik angeboten. Der Ablaufplan des Grundstudiums wird auch in vier Semester unterteilt, allerdings unterscheidet er sich von dem Ablaufplan des Grundstudiums der Kerninformatik in mehreren Punkten. Das Grundstudium besteht (außer im Anwendungsfach) aus folgenden Pflicht-Lehrveranstaltungen:

- **1. Semester** :
  - Datenstrukturen, Algorithmen und Programmierung 1 (DAP1)

  - Rechnerstrukturen (RS)
  - Höhere Mathematik 1 (HM1)

- **2. Semester** :

  - Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)
  - Betriebssysteme, Rechnernetze und verteilte Systeme 1 (BS1)
  - Höhere Mathematik 2 (HM2)
  - Betriebswirtschaftslehre (BWL)

- **3. Semester** :

  - Softwaretechnik (SWT)
  - Betriebssysteme Rechnernetze und verteilte Systeme 2 (BS2)
  - Wahrscheinlichkeitsrechnung und mathematische Statistik (WR)
  - Höhere Mathematik 3 (HM3)
  - Betriebswirtschaftslehre (BWL)

- **4. Semester** :

  - Softwarepraktikum (SoPra)
  - Informationssysteme (IS)
  - Theoretische Informatik für Studierende der Angewandten Informatik (TIfAI)
  - Proseminar (ProSem)
  - Betriebswirtschaftslehre (BWL)

Die Übergangspfeile in der Abbildung 5.4 zeigen an, dass die Semester nacheinander durchlaufen werden. Die Teilnahmevoraussetzungen für SoPra sind von der erfolgreichen Teilnahme an *DAP1*, *DAP2* und *SWT* abhängig. Diese werden durch die Abhängigkeitspfeile in der Abbildung 5.4 dargestellt.
Für die gesamte Studienzeit wählt der Studierende eines von insgesamt acht (Stand Februar 2004) angebotenen Anwendungsfächern aus. Diese haben im Grundsstudium jeweils einen Umfang von 12 bis 19 Semesterwochenstunden (SWS).
Für Studenten mit den Anwendungsfächern *Architektur* und *Logistik* gilt, anstatt des Leistungsnachweises über *Höhere Mathematik I* ein Leistungsnachweis über *Lineare Algebra und Analysis* (*M1*) zu erwerben und anstatt der Fachprüfung über *Höhere Mathematik II und III* eine schriftliche Fachprüfung über *Diskrete Strukturen und Algebra* (*M2*) zu bestehen und ein Leistungsnachweis über *Logik* zu erwerben.

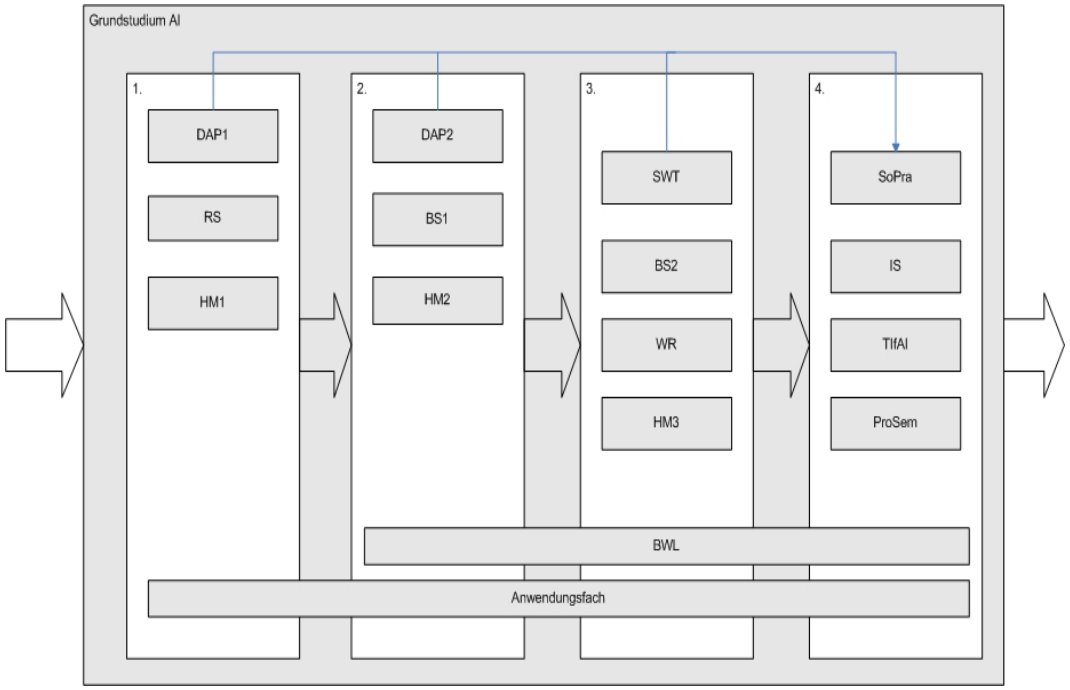


Abbildung 5.4: Ablaufplan des Grundstudiums für Angewandte Informatik

5.2.1.4 Strukturmodell des Hauptstudiums Kerninformatik

Das Hauptstudium gliedert sich in zwei Abschnitte und hat einen Umfang von fünf Semestern. Das fünfte und das sechste Semester bilden zusammen den ersten Abschnitt, welcher aus Pflicht-, Wahlpflicht - Lehrveranstaltungen und Veranstaltungen für das Nebenfach besteht.
Zu den Pflicht-Lehrveranstaltungen gehören die drei Scheinprüfungen *Informatik und Gesellschaft* (*IuG*), *Softwarekonstruktion* (*SWK*) und *Übersetzerbau* (*Üb*). Diese Lehrveranstaltungen haben einen Umfang von jeweils 3 SWS (4,5 Leistungspunkte (LP)).
Aus dem Bereich der Wahlpflicht-Lehrveranstaltungen sind ebenfalls drei Vorlesungen von jeweils neun Leistungspunkten zu belegen. Dabei hat man eine Auswahl aus zwei gegebenen *Katalogen A* und *B*. Jeder Katalog muss mit mindestens einer Vorlesung abgedeckt werden. Die Kataloge teilen sich wie folgt in Vorlesungen auf:

Katalog A (A):

- Mensch-Maschine-Interaktion
- Rechensysteme
- Eingebettete Systeme
- Modellgestützte Analyse und Optimierung

Katalog B (B):

- Effiziente Algorithmen und Komplexitätstheorie
- Darstellung, Verarbeitung und Erwerb von Wissen

- Formale Methoden des Systementwurfs

Das im Grundstudium gewählte Nebenfach wird im Hauptstudium fortgesetzt und umfasst Lehrveranstaltungen im Umfang von 12 bis 16 SWS (18 bis 24 Leistungspunkten).
Die Pflicht-Lehrveranstaltungen werden in der Abbildung 5.5 gemäß Diplomprüfungsordnung auf das fünfte und sechste Semester aufgeteilt. Außerdem wird die Wahl der drei Wahlpflicht-Lehrveranstaltungen in der Abbildung 5.5 mit Hilfe des UND- und ODER-Konnektors über das fünfte und sechste Semester dargestellt. Das Nebenfach wird in der Abbildung 5.5 zusammenfassend als ein dreischichtiger ODER-Konnektor über das fünfte und sechste Semester dargestellt.
Nachdem der erste Abschnitt eine verbreiterte Orientierung über fortgeschrittene Erkenntnisse der Informatik vermittelt, dient der zweite Abschnitt der Vertiefung. Dies geschieht mit den Wahl-Lehrveranstaltungen und der Diplomarbeit. Es ist zu beachten, dass sich die Wahl-Lehrveranstaltungen aufteilen in Leistungsnachweise und Fachprüfungen. Die Leistungsnachweise sind in Lehrveranstaltungen, in mindestens einem Seminar (vier Leistungspunkte) und einer Projektgruppe (24 Leistungspunkte) zu erwerben. Die Fachprüfungen müssen alle aus einem Schwerpunktgebiet sein und mindestens 18 Leistungspunkte erbringen. Die Leistungspunkte der Fachprüfungen müssen zusammen mit denen der Leistungsnachweise aus den Lehrveranstaltungen mindestens auf eine Summe von 30 Leistungspunkten kommen. Bei der Wahl des Schwerpunktgebietes hat man folgende Auswahlmöglichkeiten:

- Software-Konstruktion
- Rechnerarchitektur, eingebettete Systeme und Simulation
- Verteilte Systeme
- Algorithmen, Komplexität und formale Modelle
- Sicherheit und Verifikation
- Computational Intelligence und Natural Computing
- Intelligente Systeme

Die Aufteilung in diesem zweiten Abschnitt wird in der Abbildung 5.5 analog zum ersten Abschnitt repräsentiert.

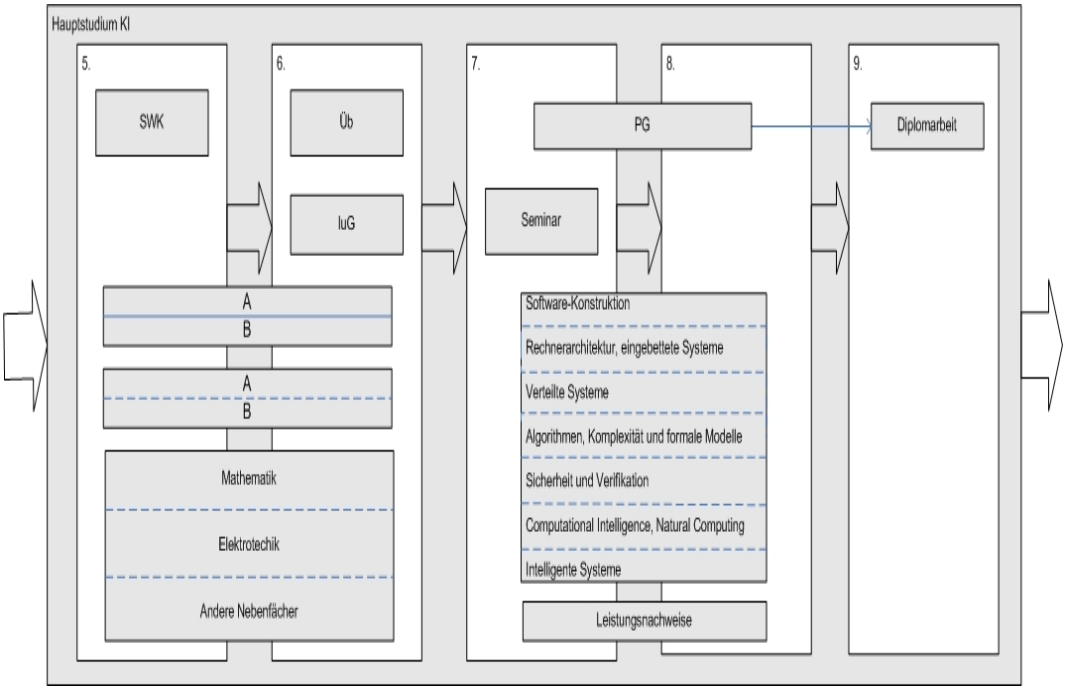


Abbildung 5.5: Ablaufplan des Hauptstudiums der Kerninformatik

5.2.1.5 Strukturmodell des Hauptstudiums Angewandte Informatik

Das Hauptstudium soll nach der DPO in fünf Semestern (einschließlich Diplomarbeit) abgeschlossen werden. Dabei wird die genaue Unterteilung der Lehrveranstaltungen in die einzelnen Semester den Studenten selbst überlassen. Das hier vorgeschlagene Strukturmodell des Hauptstudiums wurde nach dem Prinzip der gleichmäßigen Aufteilung der Semesterwochenstunden pro Semester erzeugt und in der Abbildung 5.6 dargestellt.
Im fünften und sechsten Semester sind Leistungspunkte der Lehrveranstaltungen zu erwerben, die in den Katalogen des Wahlpflicht- und nichttechnischen Wahlpflichtbereich angeboten werden. Wobei aus dem Katalog *Nichttechnische Wahlpflicht-Lehrveranstaltungen* die Lehrveranstaltungen im Umfang von 6 bis 8 SWS (9-12 LP) zu absolvieren sind, und aus dem Katalog *Wahlpflicht-Lehrveranstaltungen* eine Fachprüfung im Umfang von 9 LP zu bestehen ist. Die beiden Kataloge sind unten aufgelistet:

Katalog *Nichttechnische Wahlpflicht-Lehrveranstaltungen*:

1. Bereich: Betriebswirtschaftslehre

    - Kostenrechnung und Controlling (2 SWS)
    - Wirtschaftsinformatik (3 SWS)
    - Investition und Finanzierung (3 SWS)
    - Produktionswirtschaft (3 SWS)
    - Markt und Absatz (6 SWS)
    - Einführung in die Arbeits- und Industriesoziologie (4 SWS)
    - Unternehmensführung (6 SWS)

- Die ersten 3 der 4 LVs des Faches *Betriebsführung*, (angeboten von der Fakultät *Maschinenbau*) (6 SWS)
- Seminare aus dem Hauptstudium des Studiengangs *Wirtschaftswissenschaften*
- Lehrveranstaltungen zum Thema *Existenzgründung* oder *Entrepreneurship*

2. Bereich: Organisationspsychologie (Die Lehrveranstaltungen sind aus dem Studiengang Organisationspsychologie zu wählen).

3. Bereich: Rechtswissenschaften

   - Wirtschaftsprivatrecht I und II (jeweils 2V+1Ü)
   - Datenschutz (rechtliche Aspekte)
   - Computer-Strafrecht, Telekommunikations-Recht, IT-Sicherheits-Recht
   - Produkthaftungs-, Urheber- und Patent-Recht
   - Einführung in das Recht

Katalog *Wahlpflicht-Lehrveranstaltungen*:

- Mensch-Maschine-Interaktion
- Rechensysteme
- Eingebettete Systeme
- Modellgestützte Analyse und Optimierung
- Effiziente Algorithmen und Komplexitätstheorie
- Darstellung, Verarbeitung und Erwerb von Wissen
- Formale Methoden des Systementwurfs

Außerdem sind noch weitere Leistungsnachweise über verschiedene Informatik- und/oder Anwendungsfach-Lehrveranstaltungen abzulegen, die mit den Fachprüfungen aus dem siebten und achten Semester einen Umfang von mindestens 30 LP haben, sowie ein Leistungsnachweis über die Pflicht-Lehrveranstaltung *Informatik und Gesellschaft* im sechsten Semester zu erwerben. Der Seminarschein soll im siebten Semester erworben werden. Die Projektgruppe im Umfang von 24 LP wird während dem siebten und achten Semester angeboten. Die Fachprüfungen über die Wahlbereich-Lehrveranstaltungen müssen den Umfang von mindestens 18 LP haben und sind ebenfalls innerhalb des siebten und achten Semesters zu belegen. Zu den Wahlbereich-Lehrveranstaltungen gehören alle Informatik- und Anwendungsfach-Lehrveranstaltungen, die nicht in der oben genannten Katalogen aufgeführt sind. Das im Grundstudium ausgewählte Anwendungsfach wird im Hauptstudium fortgesetzt und umfasst Lehrveranstaltungen, die sich über vier Semester verteilen und einen Umfang von 31,5 bis 39 Leistungspunkten haben, wobei 6 LP davon über eine Studienarbeit im Anwendungsfach zu erwerben sind. Das neunte Semester wird ausschließlich für das Schreiben der Diplomarbeit genutzt.

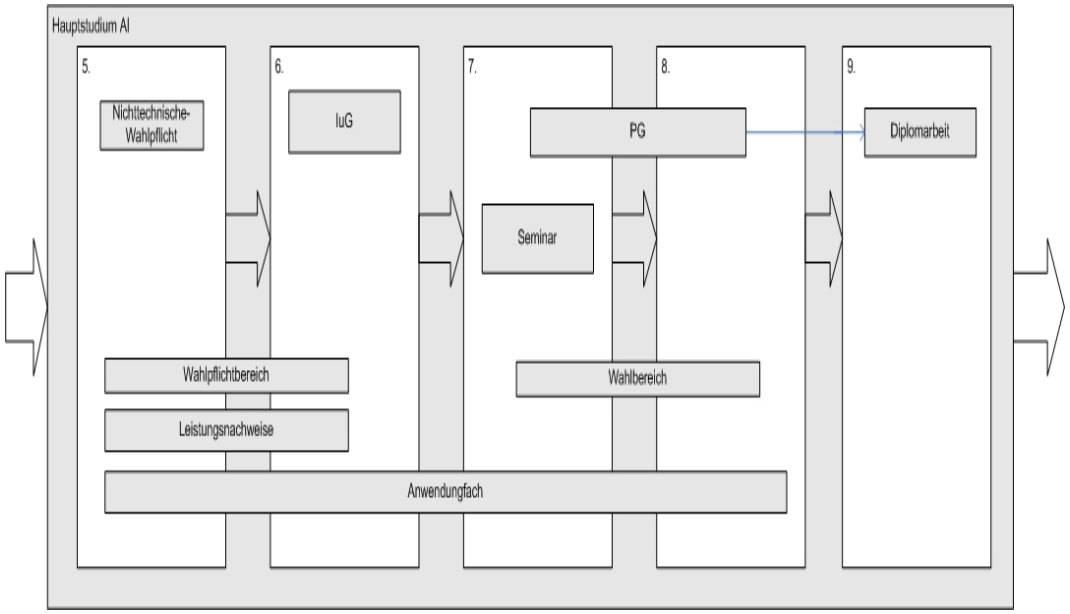


Abbildung 5.6: Ablaufplan des Hauptstudiums der Angewandten Informatik

5.2.2 Datenerhebung/-analyse

Zu Beginn der Projektgruppe wurde das Ziel gesetzt, die Simulationsmodelle entsprechend der Diplomprüfungsordnung 2001 so realistisch wie möglich zu implementieren. Um das realisieren zu können waren die Statistikdaten von der Universitätsverwaltung, vom Dekanat des Fachbereichs Informatik und vom Prüfungsamt notwendig. Mit freundlicher Unterstützung der Mitarbeiter oben genannter Instanzen, wurden während der ersten drei Monate sehr große Menge an Daten zusammengetragen. Dazu zählen unter anderem:

- Neuimmatrikulationen bzw. Einschreibungszahlen von Informatikstudenten
- Erfolgsquoten der Vordiplome und Diplome für Kerninformatik, Angewandte Informatik und Lehramt
- Absolventenzahlen nach den verschiedenen DPOs
- Projektgruppenstatistiken
- Schwundquoten von Studenten, aufgeschrieben nach der Semesteranzahl
- Abbruchzahlen nach den beiden DPOs ’96/97 und 2001 (bestandenes bzw. nichtbestandenes Diplom)
- Tabellen bestandener Prüfungen für die beiden DPOs (aufgelistet nach den einzelnen Fächern)
- Verteilung auf die Schwerpunktgebiete für die DPO 2001

Ein weiterer Punkt, um Modelle der Realität näher zu bringen, ist nicht nur den Ablauf der einzelnen Semester zu verstehen und alle Vorlesungen, die nach der DPO 2001 angeboten werden, zu berücksichtigen, sondern sowohl das angeforderte Personal, wie Professoren und wissenschaftliche

Hilfskräfte (HiWis)[1], als auch deren erforderliche SWS-Anzahl zu betrachten. Die Semesterabläufe und Vorlesungen, die durch die DPO 2001 (Stand Februar 2004) festgelegt sind, wurden schon im Unterkapitel 5.2.1 beschrieben. An dieser Stelle sollte noch erwähnt werden, dass der Studiumumfang (einschließlich Diplomarbeit), etwa 160-180 SWS beträgt, was durchschnittlich 18-20 SWS pro Semester ausmacht. Die meisten Lehrveranstaltungen werden einzeln abgeprüft, so dass bei der mündlichen Prüfung 1 SWS vom Professor und 1 SWS vom HiWi benötigt wird, um einen Studenten zu prüfen. Bei der schriftlichen Prüfung werden unterschiedliche Professoren- und HiWi-SWS benötigt, was an dem Umfang und der Schwierigkeit der Lehrveranstaltung liegt, wobei die Anzahl der zu prüfenden Studenten nur durch die Größe des Hörsaals begrenzt wird. Laut der DPO 2001 gilt für die Dauer schriftlicher Fachprüfungen folgendes:

- Lehrveranstaltungen bis 3 SWS: 30 bis 60 Minuten,
- Lehrveranstaltungen von 3 bis 6 SWS: 60 bis 120 Minuten,
- Lehrveranstaltungen über 6 SWS: 120 bis 180 Minuten.

Als Faustregel kann angenommen werden, dass 1 SWS ungefähr 45 Minuten Vorlesung bzw. Übung entspricht. Mit der DPO 2001 wurde ein Leistungspunkte-System (Credit Point System) eingeführt, das dem ECTS (European Credit Transfer System) entspricht. Mit diesem System wird jeder Lehrveranstaltung eine bestimmte Anzahl an Leistungspunkten (LP) zugeordnet. Für jede Studentin bzw. für jeden Studenten wird ein Punktkonto geführt, so dass man die Leistung einzelner Studenten leichter beurteilen kann. Die Leistungspunkte sind proportional zur Anzahl der Semesterwochenstunden. Mit einer einfachen Regel kann man Leistungspunkte, die jeder Lehrveranstaltung zugeordnet sind, in Semesterwochenstunden überführen und umgekehrt. Diese Regel lautet: 1 SWS ist gleich 1,5 Leistungspunkte. Bei den Seminaren und Praktika werden mehr Leistungspunkte pro SWS vergeben, um die Schwierigkeitsgrade zu verdeutlichen, so werden z.B. bei dem Proseminar für 2 SWS 4 LP auf das Konto geschrieben. Den meisten Prüfungen, die nach der DPO 2001 angeboten werden, werden entweder 9 oder 4,5 Leistungspunkte zugeordnet, was 6 bzw. 3 SWS entspricht. Die 6 SWS-Lehrveranstaltungen bestehen in der Regel aus 2 Vorlesungen, je 2 SWS, und einer Übung, die auch 2 SWS beansprucht. 3 SWS-Lehrveranstaltungen werden in eine 2 SWS-Vorlesung und 1 SWS-Übung aufgeteilt. Jede Vorlesung wird von einem Professor gehalten, jede Übungsdurchführung beansprucht einen HiWi. Diese SWS-Aufteilung und das Leistungspunkte-System können zur Ermittlung der Auslastung und der Anzahl der Professoren und HiWis an der Universität Dortmund behilflich sein.
Für die Erstellung des Simulationsmodells müssen noch

- die Verteilung der Einschreibungszahlen,
- die Aufteilung der Studenten in Kerninformatik und Angewandte Informatik,
- die Auswahl der Kataloge im Wahlpflichtbereich,
- Diplomarbeiten und Seminare,
- Angebote und Nachfrage für Projektgruppen,
- Schwerpunktgebiete,
- Nebenfächer und

[1] Im weiteren werden wissenschaftliche Hilfskräfte (HiWi) und wissenschaftliche Mitarbeiter (WiMi) als Synonyme verwendet.

- Schwundquoten

ermittelt werden. Diese Ermittlungen werden in den Unterkapiteln ausführlicher beschrieben.

5.2.2.1 Verteilung der Einschreibungszahlen

Nachdem das Strukturmodell für das Simulationsmodell erstellt wurde, stellte sich heraus, dass man aus den erworbenen Daten für den Best- bzw. Worstcase, die Verteilungen von jährlich eingeschriebenen Studenten ermitteln muss, um die Prozesse in den Modellen entsprechend dieser Verteilung zu kreieren. Als Werkzeuge standen dabei zwei Tools zur Verfügung, das **ARENA**-Tool [Aren 04] und das **R**-Tool [Vena 05]. Beide Tools berechnen eine passende Verteilung für bestimmte Eingabemengen. Da aber nur eine geringe Anzahl an Daten vorliegt, sind die Ergebnisse des Chi-Quadrat Tests nicht akzeptabel. Der Chi-Quadrat Test muss mindestens eine Eingabemenge von 50 Daten haben, um ein glaubwürdiges Ergebnis auszugeben. Trotz der geringen Anzahl der Eingabedaten, berechnet das **ARENA**-Tool eine passende Verteilung und gibt das Ergebnis des Chi-Quadrat Tests aus. Das Ergebnis vom Chi-Quadrat Test ist in diesem Fall nicht glaubwürdig. Die vom **ARENA**-Tool ausgerechneten Verteilungen werden nochmals im **R**-Tool mit dem Kolmogorov-Smirnov-Test (KS-Test) untersucht. Die Ergebnisse vom KS-Test sind in diesem Fall glaubwürdig, weil dieser Test keine Anforderungen an die Größe der Datenmenge hat. Die Ausgabe des KS-Tests besagt aber, dass keine der von **ARENA** geschätzten Verteilungen zu den gegebenen Eingabedaten passt. Nachdem jeder Versuch, die Verteilung der Daten zu finden, misslungen war, wurde entschieden, die diskrete Gleichverteilung für die Anzahl der eingeschriebenen Studenten zu wählen. Dabei wurden nur die Daten der letzten 16 Jahre, also von 1989 bis 2004, betrachtet. Diese Entscheidung wurde aus folgendem Grund getroffen: die Informatik als Studienfach wird erst seit 1972 an der Universität Dortmund angeboten und hatte damals noch nicht die große Bedeutung wie heute, was zu einer deutlichen Erhöhung der Studentenanzahl erst ab Mitte der 80er Jahre führte. Der Mittelwert bzw. die durchschnittliche Anzahl von Studenten, die sich jedes Jahr einschreiben, entspricht in diesem Fall 474 Studenten pro Jahr. Die berechnete Standardabweichung beträgt 177 Studenten pro Jahr. Als Parameter für die obere bzw. untere Grenze der diskreten Gleichverteilung wurde das Minimum bzw. das Maximum genommen, die aus dem Mittelwert und der Standardabweichung bestimmt wurden. Daraus ergibt sich folgendes Intervall für die diskrete Gleichverteilung $[168, 780]$. Im weiteren erwies sich die Annahme einer diskreten Gleichverteilung als falsch (siehe 5.3.1).

5.2.2.2 Aufteilung der Studenten in Kerninformatik und Angewandte Informatik

Der prozentuale Anteil an Studenten der Kerninformatik (KI) und Studenten der Angewandten Informatik (AI) von dem kummulierten Wert der Einschreibungen aller eingeschriebenen Informatikstudenten kann aus den vorhandenen Statistiken ermittelt werden. Der Anteil beträgt 85% für KI und 15% für AI.

5.2.2.3 Auswahl der Kataloge im Wahlpflichtbereich

Hier stehen keine Daten zur Verfügung, weswegen die Verteilung der Studenten auf die Kataloge A und B - also ob zwei Vorlesungen aus Katalog A oder zwei aus B gehört/geprüft werden - geschätzt wurde. Eine Schätzung auf 70% Katalog A und 30% Katalog B wurde über die Wahl der PG-Mitglieder und anderer Studenten als angemessen erachtet. Eine komplette Erhebung dieser Daten wäre zu zeitaufwendig.

5.2.2.4 Diplomarbeiten und Seminare

Zu den Diplomarbeiten und Seminaren liegen keine Statistiken vor, so dass man auf die Erfahrungen der PG-Teinehmer und der Studenten zurückgreifen muss, die gerade ihre Diplomarbeit schreiben oder ihr Studium schon abgeschlossen haben. Daraus folgt, dass praktisch nie zu wenig Seminare oder Diplomarbeiten angeboten werden und für die Studenten auch die Möglichkeit besteht, bei den Diplomarbeiten eigene Themen vorzuschlagen. An dieser Stelle wird auf die Annahmen und die Simulationsmodellbeschreibungen bei dem Bestcase bzw. Worstcase hingewiesen (Kapitel 5.3.2 und 5.3.3).

5.2.2.5 Angebot und Nachfrage für Projektgruppen

Laut der zur Verfügung stehenden Statistik, ist in fast jedem Semester die Anzahl angebotener Projektgruppenplätze geringer als die Anzahl der Bewerber, was zu einem Engpass im Realcase oder im Worstcase führen kann, aber für den Bestcase eher irrelevant ist. Die entsprechenden Annahmen für den Worstcase werden in dem Kapitel 5.3.3 näher erläutert.

5.2.2.6 Schwerpunktgebiete

Leider war es unmöglich eine genaue Aufteilung in die Schwerpunktgebiete zu ermitteln, da die zur Verfügung gestellten Daten nicht ausreichend sind und es sehr viele Möglichkeiten gibt aus dem vorhandenen Prüfungsangebot zu wählen. An dieser Stelle wird auf das Kapitel (5.3.2) hingewiesen, um genaue Annahmen in diesem Fall zu erleuchten.

5.2.2.7 Nebenfächer

Aus den vorhandenen Statistiken zu den bestandenen Vordiplom-Prüfungen in Nebenfächern lässt sich die Schlussfolgerung ziehen, dass sich 80% aller Studenten der Informatik auf vier Nebenfächer BWL, Theoretische Medizin, Mathematik und Elektrotechnik verteilen. Für den Best- bzw. Worstcase wurde entschieden, diese nicht genauer, d.h. vor allem mit ihren Auswirkungen auf das Hauptfach, zu modellieren. Alle Nebenfächer werden in einem zusammengefasst, in dem nur der Aufwand an Semesterwochenstunden bzw. zu erbringenden Leistungspunkten modelliert wird.

5.2.2.8 Schwundquoten

Die Schwundquoten an der Universität Dortmund sind zwar erheblich, doch es steht keine Aufteilung der nach ihren Ursachen differenzierten Schwundquoten zur Verfügung. Deswegen kann man nicht eindeutig sagen, ob diese Quote zu vernachlässigen ist. Aufgrund der ausschließlichen Beschäftigung mit der Best- bzw. Worstcase-Analyse, wurden die Schwundquoten nicht betrachtet und die Datenauswertung an dieser Stelle eingestellt.

5.3 Validierung und Annahmen

Gemäß der oben aufgeführten Schritte einer Simulationsstudie muss nun die Verteilungsfunktion der ankommenden Studenten validiert werden. Weiterhin werden globale Annahmen für die speziellen Fälle des Best- und des Worstcase des abstrakten Modells getroffen.

5.3.1 Bestimmung des Intervalls für die diskrete Gleichverteilung

Aus den wenigen Daten, die der Gruppe zur Verfügung standen, konnte keine zufrieden stellende Verteilung für die Anzahl der eingeschriebenen Studenten gefunden werden. Aus diesem Grund wurde entschieden, eine diskrete Gleichverteilung in den Modellen zu wählen (siehe Kapitel 5.2.2.1). Die Parameter für die Verteilung sind Minimum und Maximum, die nicht direkt bestimmbar sind, so dass man diese erst über einen kleinen Umweg berechnen muss. Als erstes wird der Mittelwert und die Standardabweichung der vorhandenen Daten berechnet. Der Mittelwert ist in diesem Fall gleich 474 Studenten pro Jahr und die Standardabweichung ist 177. Diese Werte liegen zwar in vernünftigen Grenzen, können aber zur Bestimmung der Parameter der diskreten Gleichverteilung nicht direkt verwendet werden, da sie über die Formeln für kontinuierliche Stichproben ermittelt wurden. Um den Mittelwert und die Standardabweichung in unserem Fall benutzen zu können, müssen zwei Annahmen getroffen werden, nämlich, dass der berechnete Mittelwert und die Standardabweichung äquivalent dem Mittelwert und der Standardabweichung unserer diskreten Daten sind. Mit diesen Annahmen können Mittelwert und Varianz, die aus der Standardabweichung abgeleitet wurden, in die Formel der diskreten Gleichverteilung eingesetzt werden. Durch das Lösen zweier Gleichungen mit zwei Unbekannten kann man die beiden Parameter für die Verteilung bestimmen. Der Rechenweg ist im folgenden dargestellt:

diskrete Gleichverteilung:

$Mittelwert \qquad \overline{X} = \frac{i+j}{2}$

$Varianz \qquad Var(X) = \frac{(j-i+1)^2-1}{12}$

wobei i und j Minimum bzw. Maximum sind

ermittelte Größen:

$Mittelwert \qquad \overline{X} = 474$

$Standardabweichung \quad \sigma = 177$

$Var(x) = \sigma^2$

Nach dem Einsetzen:

$$\frac{i+j}{2} = 474 \qquad \frac{(j-i+1)^2-1}{12} = 177^2$$

$$i + j = 474 * 2$$

$$j = 948 - i$$

$$\frac{(949-2i)^2-1}{12} = 177^2$$

$$(949-2i)^2 - 1 = 31329 * 12$$

$$(949-2i)^2 = 375949$$

$$949 - 2i = \sqrt{375949}$$

$$2i = 335,853$$

$$\underline{i = 167,93 \approx 168}$$

$$j = 948 - 168$$

$$\underline{j = 780}$$

Die so errechneten Werte des Minimums und des Maximums stellen die Grenzen des Intervalls für die diskrete Gleichverteilung dar. Um zu überprüfen, ob die diskrete Gleichverteilung auf dem Intervall $[168, 780]$ auch unsere Eingangsdaten beschreibt, wird ein KS-Test durchgeführt. Das ausgewählte Statistiktool R liefert zwar die Möglichkeit, diesen Test durchzuführen, schließt aber den Vergleich mit der diskreten Gleichverteilung aus. An dieser Stelle muss ein wenig in die Trickkiste gegriffen und eine Reihe von Daten erzeugt werden, die diskret und gleichverteilt sind. Erst dann wird der KS-Test mit den Originaldaten und künstlich erzeugten Daten durchgeführt. Die Erzeugung der Daten wird in HIT ausgeführt, da hier die Möglichkeit besteht, mit der Funktion *randint*, die diskret gleichverteilten Daten zu erzeugen. Der dafür benutzte HI-SLANG Code des Programms *randint.hit* wird im weiteren kurz beschrieben.

```
%COMMON
%PARM = INDENT = 0
%BIND "hit_aus" TO /home/pg476/glasirin/ankunft.data
%END

EXPERIMENT r_exp METHOD SIMULATIVE;
VARIABLE n,z:INTEGER;
          f:OUTFILE;
         run:INTEGER;
      replic:INTEGER;
BEGIN
   OPEN f, "hit_aus" LENGTH 80;
   replic := 5 ;
   FOR run := 1 STEP 1 UNTIL replic
      LOOP
         n:=0;
         WHILE n<16
          LOOP
             z := randint(168,780);
             WRITE FILE f,z;
             n:=n+1;
          END LOOP;
      END LOOP;
   CLOSE f;
END EXPERIMENT r_exp ;
```

Als erstes wird ein Pfad zu einer Datei angelegt, in welche die erzeugten Daten gespeichert werden. Dann kommt der eigentliche Experimentblock, der die Variablendeklarationen und zwei Schleifen enthält. Die Variablen *replic* und *run* werden mit den Werten 5 bzw. 1 initiiert und in der äußeren *FOR*-Schleife als Zähler eingesetzt. Die *FOR*-Schleife wird fünf Mal ausgeführt und die verschachtelte *WHILE*-Schleife wird 16 Mal ausgeführt. Dadurch werden fünf verschiedene 16-elementige Datenreihen erzeugt, deren Daten durch den Aufruf *randint(168,780)* generiert werden. (Die Anzahl der Elemente sollte 16 sein, da die erzeugten Datenreihen später mit dem Array der Originaldaten beim KS-Test verglichen werden). Diese Datenreihen werden dann in *ankunft.data* gespeichert. Die An-

kunftszahlen der Studenten werden in eine separate Datei *ankunftszahl.data* gespeichert. Daraufhin wird mit dem Befehl

```
source("r_script",echo=T)
```

folgender R-Skript ausgeführt.

```
x<-scan("ankunftszahl.data")
y<-scan("ankunft.data")
dim(y)<-c(16,5)
ks.test(x,y[,1])
ks.test(x,y[,2])
ks.test(x,y[,3])
ks.test(x,y[,4])
ks.test(x,y[,5])
```

Zuerst werden die beiden Dateien *ankunftszahl.data* und *ankunft.data* eingelesen und den Arrays *x* und *y* zugewiesen. Das Array *y* enthält 80 Elemente. Mit dem Befehl *dim(y)<-c(16,5)* wird das Array in fünf Teilarrays aufgeteilt, die jeweils 16 Elemente enthalten. Im nächsten Schritt wird bei dem KS-Test das Array *x* den einzelnen Teilarrays gegenübergestellt. Die Anzahl der Tests wurde auf fünf festgelegt, um die KS-Test-Ergebnisse besser auswerten zu können. Im weiteren werden die Testdurchläufe präsentiert und die Ergebnisse dargestellt.

Die anschließenden KS-Tests liefern folgende Ergebnisse:

```
> ks.test(x, y[,1])

        Two-sample Kolmogorov-Smirnov test

data:  x and y[,1]
D = 0.375, p-value = 0.2145
alternative hypothesis: two.sided


> ks.test(x, y[,2])

        Two-sample Kolmogorov-Smirnov test

data:  x and y[,2]
D = 0.375, p-value = 0.2145
alternative hypothesis: two.sided


> ks.test(x, y[,3])

        Two-sample Kolmogorov-Smirnov test

data:  x and y[,3]
D = 0.25, p-value = 0.6994
```

```
alternative hypothesis: two.sided


> ks.test(x, y[,4])

        Two-sample Kolmogorov-Smirnov test

data:  x and y[,4]
D = 0.3125, p-value = 0.4154
alternative hypothesis: two.sided


> ks.test(x, y[,5])

        Two-sample Kolmogorov-Smirnov test

data:  x and y[,5]
D = 0.1875, p-value = 0.9412
alternative hypothesis: two.sided

Warning messages:
1: cannot compute correct p-values with ties in: ks.test(x, y[,3])
2: cannot compute correct p-values with ties in: ks.test(x, y[,4])
3: cannot compute correct p-values with ties in: ks.test(x, y[,5])
```

Bei zwei der fünf Tests sind die Werte für *p-value* gleich 21.45%, was kleiner als 50% ist und an den zufällig erzeugten Daten in den Arrays *y[,1]* und *y[,2]* liegt. In diesen beiden Fällen kann man deutlich sagen, dass unsere diskreten Daten nicht gleichverteilt sind. Die drei anderen Tests liefern deutlich bessere Ergebnisse als die ersten zwei. Sie sind aber mit der *Warning message: „cannot compute correct p-value with ties in: ks.test(x,...)“* versehen. Diese Warnung wird dann ausgegeben, wenn mindestens eine Zahl in dem Array *x* exakt mit einer der Zahlen, die in den Teilarrays vorkommen, übereinstimmen oder wenn in einem Teilarray zwei oder mehr gleiche Zahlen vorkommen. Die eindeutige Auswertung der Ergebnisse der letzten drei Tests lässt sich nicht durchführen, so dass an dieser Stelle einzuräumen ist, dass die diskrete Gleichverteilung, die für die gegebenen Eingangsdaten angenommen und für richtig gehalten wurde, nicht zu akzeptieren ist und vor allem bei der Worstcase-Simulation zu falschen Ergebnissen führen kann. Die Benutzung der kontinuierlichen Gleichverteilung mit gleichen Parametern liefert ähnliche Ergebnisse. Auf weitere Versuche, eine weitere Verteilung zu bestimmen, wurde verzichtet und entschieden, für das spätere Realcase-Modell eine Tendenz der letzte Jahren zu verfolgen und daraus eine durchschnittliche Zahl zu berechnen, die dann anstatt der Verteilung eingesetzt wird.

5.3.2 Annahmen für den Bestcase

Für den Bestcase wird die Annahme getroffen, dass alle benötigten Ressourcen in ausreichendem Maße vorhanden sind und somit kein Bottleneck entstehen kann. Das bedeutet, dass erstens die Räume und Hörsäle alle Studenten aufnehmen können, und zweitens, dass es möglich ist, die Professoren für alle Vorlesungen einzusetzen. Ebenso können die WiMis (Wissenschaftliche Mitarbeiter) alle anfallenden Arbeiten erledigen, wie zum Beispiel Übungsgruppen leiten. Außerdem wird davon aus-

gegangen, dass die Studenten alle Prüfungen beim ersten Versuch erfolgreich absolvieren, und dass sich Jahr durchschnittlich 474 Informatik-Studenten neu einschreiben. Die Einschreibezahlen befinden sich gleichverteilt im Intervall von [297,651].
Diese Verteilung und das angegebene Intervall haben sich im Nachhinein als falsch erwiesen, und eine korrekte Verteilung konnte nicht gefunden werden (siehe 5.3.1). Dabei besteht die Annahme, dass die richtige Verteilung keinen großen Einfluss auf das bestehende Modell hat.
Da alle Nebenfächer den gleichen Umfang haben, werden die einzelnen Nebenfächer in den Modellen nicht unterschieden und zur Vereinfachung und besseren Übersicht im weiteren pauschal nur als *Nebenfach* behandelt. Außerdem wurde die Annahme getroffen, dass in jedem Semester des Grundstudiums das Nebenfach als vier SWS (Semesterwochenstunden) dargestellt wird. Im Hauptstudium belegt das Nebenfach im fünften und sechsten Semester auch jeweils vier SWS.
Desweiteren werden die Wahlpflichtkataloge A und B nicht im Detail betrachtet, sondern als drei Vorlesungen modelliert, die jeweils einen Umfang von 6 SWS haben und mit einer mündlichen Prüfung abgeschlossen werden. Hierbei wird davon ausgegangen, dass sich 70% der Studenten für zwei Vorlesungen aus dem Katalog A und 30% der Studenten für zwei Vorlesungen aus dem Katalog B entscheiden. Weiterhin werden die Projektgruppe und die Leistungsnachweise als Vorlesungen im Umfang von 4 SWS behandelt, die ihre Ressourcen von den ihnen zugeordneten Lehrstühlen beziehen, welche allerdings keine real existierenden Lehrstühle nachbilden, sondern aus Abstraktionsgründen als Dummyklassen implementiert werden (siehe 5.4.5). Außerdem wird das Schwerpunktgebiet als eine Vorlesung im Umfang von 6 SWS, welche im siebten und achten Semester gehört wird, modelliert, wobei in jedem Semester zwei Prüfungen abzulegen sind. Bei der Wahl des Schwerpunktgebietes wird eine Gleichverteilung unter allen möglichen Schwerpunktgebieten angenommen. Zu guter letzt wird davon ausgegangen, dass im neunten Semester nur die Diplomarbeit geschrieben wird und keine Veranstaltungen besucht werden. Hierbei wird vorausgesetzt, dass immer genügend Themen für Diplomarbeiten angeboten werden.
Ressourcen wie zum Beispiel Bibliothek, Hochschulrechenzentrum, etc. werden nicht benötigt und außer Acht gelassen, da diese einen unnötigen Overhead im Modell darstellen würden.
In den Modellen werden *Wochen* als Zeiteinheiten benutzt und die Ressourcenanforderungen finden in *Semesterwochenstunden* statt.

5.3.3 Annahmen für den Worstcase

Die Annahmen für den Worstcase stimmen in großen Teilen mit den Annahmen für den Bestcase überein und werden somit an dieser Stelle nicht erneut aufgeführt.
In der Realität werden zuerst die Vorlesungen ausreichend mit Ressourcen bedient, die in jedem Fall stattfinden müssen. Eine Einschränkung in diesem Bereich hätte also nur zufolge, dass Vorlesungen aus dem Schwerpunktgebiet nicht stattfinden würden. Aus diesem Grund werden in den Worstcase-Modellen nur die zur Verfügung stehenden Plätze für das SoPra (192 Plätze) und die PG (144 Plätze) eingeschränkt. Weiterhin werden den PGs keine Professoren zugeteilt. Da sich im Durchschnitt jedes Jahr 474 Studenten neu einschreiben, sind diese Werte ausreichend geeignet einen Bottleneck zu bilden. Weiterhin wird angenommen, dass für alle Vorlesungen nur noch Globalübungen stattfinden.

5.3.4 Zusammenfassung der Annahmen

- Aufteilung der Informatikstudenten: 85% Kerninformatik, 15% Angewandte Informatik
- Lehramt Informatik wird nicht betrachtet

- ausreichende Ressourcen im Best- und Worstcase hinsichtlich der Kompetenz der Professoren und WiMis, Größe und Anzahl der Hörsäle und Räume
- durchschnittlich 474 Neu-Einschreibungen pro Jahr, Neu-Einschreibungen gleichverteilt im Intervall [297, 651]
- alle Prüfungen werden bestanden
- Modellierung von nur einem stellvertretenden Nebenfach (4 SWS vom ersten bis zum sechsten Semester)
- Katalog A und B pauschal als drei Vorlesungen mit 6 SWS und mündlicher Prüfung
- 70% Wahl von zwei Vorlesungen aus Katalog A, 30% analog Katalog B
- PG und Leistungsnachweise als je 4 SWS
- Schwerpunktgebiet als 6 SWS (je zwei Prüfungen im siebten und achten Semester)
- Gleichverteilung der sieben Schwerpunktgebiete
- Seminar und Proseminar nehmen keine Ressourcen in Anspruch
- neuntes Semester nur Diplomarbeit
- unbenötigte Ressourcen werden nicht modelliert
- Wochen als Zeiteinheiten
- Ressourcenanforderungen in SWS
- Einschränkungen der SoPra (192) und PG (144)-Plätze im Worstcase
- keine Professoren für die PGs im Worstcase
- Globalübungen im Worstcase

Nach Entwurf und Validierung des Modells schließt sich die Implementierungen für den Best- und Worstcase in den jeweiligen Modellierungstools an.

5.4 Implementierung und Verifikation des HIT-Modells für den Bestcase

Im folgenden wird zuerst die grobe Struktur des HIT-Modells beschrieben und anschließend detailliert auf die einzelnen Ebenen des Modells eingegangen.

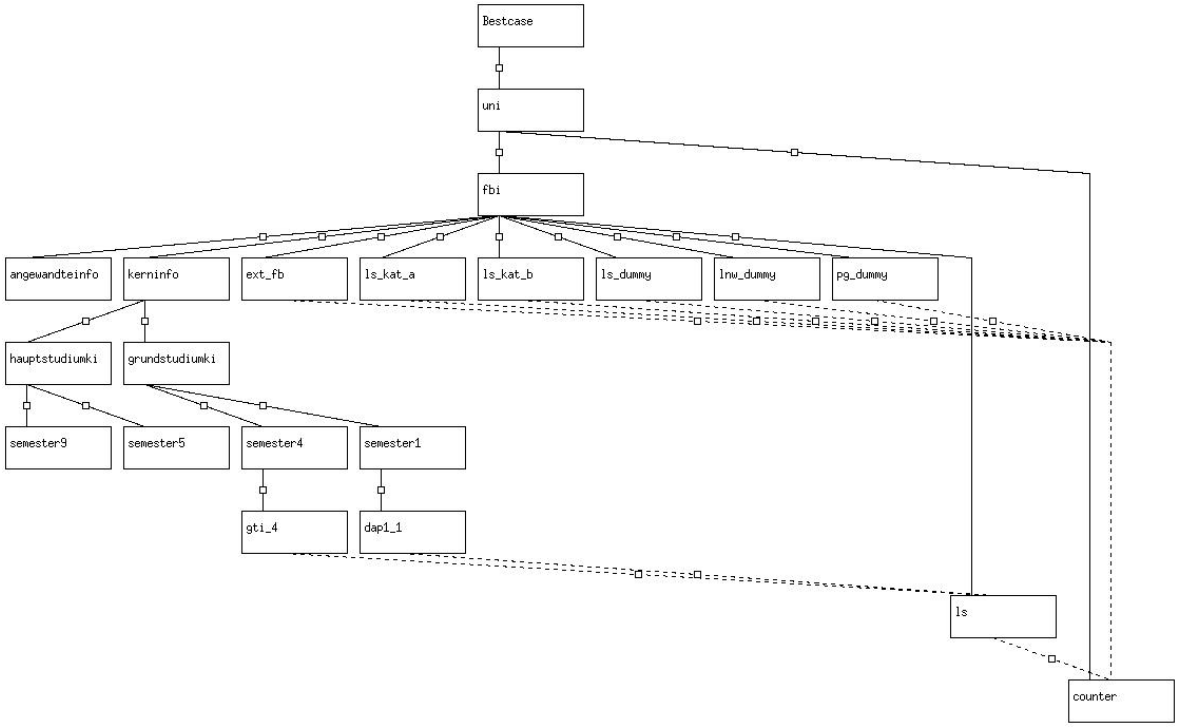


Abbildung 5.7: Struktur des HIT-Modells

Die Struktur des HIT-Modells ist in der Abbildung 5.7 zu sehen (Typstruktur des HIT-Modells). Die Abbildung stellt aus Gründen der Übersicht nur Ausschnitte des Modells dar, enthält aber alle relevanten Komponententypen und veranschaulicht den hierarchischen Aufbau.

Auf oberster Hierarchieebene befindet sich der Modelltyp *Bestcase*. Als hierarchisch oberster Komponententyp fungiert die *uni*, in welche sich der angehende Student nach dem Entschluss zu studieren einschreibt und die ihm das gewünschte Studium anbietet. Abhängig von dem gewählten Studiengang wird er dem entsprechenden Fachbereich zugeordnet. Im Rahmen der Bottleneck-Analyse wird nur der Fachbereich Informatik (*fbi*) betrachtet. Hierbei werden die zwei Studiengänge Kerninformatik (*kerninfo*) und Angewandte Informatik (*angewandteinfo*) angeboten. Das Lehramt-Informatik wird nicht weiter berücksichtigt, da sich jedes Jahr die Anzahl der Einschreibungen auf weniger als ein Prozent bei den Informatikstudenten beläuft. Bei Kerninformatik und Angewandter Informatik unterteilt sich das Studium in Grund- und Hauptstudium (*grundstudiumki*, *hauptstudiumki*), wobei das

Grundstudium vier Semester einnimmt, und das Hauptstudium fünf Semester (*semesterX*). Die Semester teilen sich bei der Kerninfomatik in die nach der DPO 2001 vorgesehenen Veranstaltungen auf (exemplarisch für das vierte Semester *gti_4*). Die Angewandte Informatik wird nur als Verzögerung im Umfang von neun Semestern dargestellt, um die Komplexität des Bestcase-Modells gering zu halten.
Zu erwähnen ist, dass der Fachbereich sich in elf Lehrstühle unterteilt (*ls*), die für die jeweiligen Vorlesungen verantwortlich sind, und die zugehörigen Ressourcen bereitstellen. Zusätzlich zu den normalen Lehrstühlen befinden sich in dem Modell die Dummies ls_kat_b, ls_kat_a, pg_dummy, ls_dummy, lnw_dummy und ext_fb. Auf diese Dummies, welche ebenfalls Ressourcen bereitstellen, wird an geeigneter Stelle eingegangen (siehe Abschnitt 5.4.5).
Die Counter verwalten die Uni-eigenen Ressourcen, die die gerade erwähnten Lehrstühle und Dummies anfordern.

Nun wird der Modelltyp *Bestcase* und die einzelnen Komponententypen beschrieben. Diese wären *uni*, *fbi*, *ls*, *kerninfo*, *grundstudiumki*, *hauptstudiumki*, die einzelnen *semester*, *gti* als Beispiel für Vorlesungen mit mündlichen Prüfungen, *dap1* als Beispiel für Vorlesungen ohne mündlichen Prüfungen, *pg*, *Seminare*, *Nebenfächer* und *Praktika*.

5.4.1 Bestcase

Auf der obersten Ebene befindet sich der Student, der an der Universität studiert und von dieser den Abschluss erhält.
Dazu ruft der Service *student* den used-service *studieren* auf, welcher mit dem provided-service *abschluss* der Komponente *universitaet* verknüpft ist. Dies wird in Abbildung 5.8 dargestellt. Außerdem werden hier die globalen Variablen für *prof*, *wimi*, *hs* und *raum* aufgeführt, die auf den Wert 1.000.000 gesetzt sind. Diese Variablen bilden ein Hilfskonstrukt, da es nicht möglich ist die Anzahl der verfügbaren Ressourcen aus der Komponente *Counter* direkt abzulesen. Jedes Jahr schreiben sich zwischen 297 und 651 Studenten ein, was mit Hilfe der diskreten Gleichverteilung (randint) realisiert wird. Somit wird alle 52 Wochen die entsprechende Anzahl an Prozessen *student* erzeugt. Um dieses Ergebnis zu erzielen, wird der Service *Ankunft* benötigt, welcher zu Beginn innerhalb der Activities einmal angestoßen wird (CREATE 1 PROCESS Ankunft ;) und ohne Unterbrechung weiterläuft. Der Service hat den folgenden Code:

```
LOOP
  CREATE randint(297,651) PROCESS student;
  spend(52);
END LOOP;
```

Diese Schleife erzeugt alle 52 Wochen eine Anzahl an Prozessen *student*, welche gleichverteilt in dem Intervall [297,651] liegt.

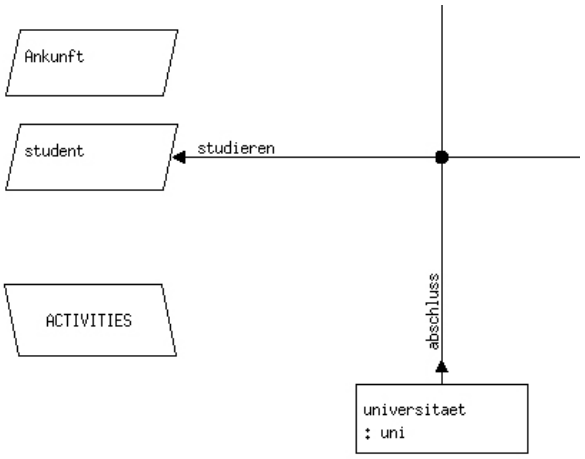


Abbildung 5.8: Bestcase

5.4.2 uni

In der Universität bietet der Fachbereich Informatik dem Studenten zwei Studiengänge an. Implementiert wird dies dadurch, dass der Service *abschluss* den used-service *studieren* aufruft, welcher mit dem provided-service *studiengang* der Komponente *fachbereich* vom Typ *fbi* verknüpft ist. Die Universität stellt für das Studium die Ressourcen wissenschaftliche Mitarbeiter (WiMis), Hörsäle (hs), Professoren (profs) und Räume zur Verfügung, die als *Counter* modelliert wurden (zu sehen in Abbildung 5.9).

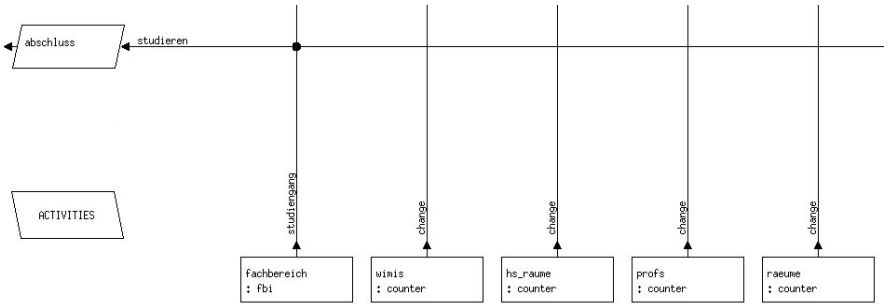


Abbildung 5.9: uni

5.4.3 fbi

Der Fachbereich Informatik (fbi) bietet den Service studiengang an, welcher die Verteilung der Studenten auf die spezifischen Studiengänge Kern- und Angewandte Informatik realisiert. Dieser Sachverhalt wird von dem Modell durch die Komponenten *kerninformatik* (vom Typ *kerninfo*) und *angewandte* (vom Typ *angewandteinfo*) umgesetzt. Aus den vergangenen Jahren hat die Datenanalyse ergeben, dass 85 Prozent aller Studierenden Kerninformatik wählen und 15 Prozent Angewandte Informatik.

Diese Verteilung wird im HIT-Modell mit Hilfe einer Branch-Anweisung realisiert:

```
BRANCH
PROB 0.85 : waehleki;
PROB 0.15 : waehleai;
END BRANCH;
```

Den jeweiligen Lehrstühlen des Fachbereichs sind die einzelnen Vorlesungen und die entsprechenden Prüfungen zugeordnet. Im Modell werden die Komponenten *ls1* bis *ls12* vom Typ *ls* in dem Komponententyp *fbi* eingebettet. Diese Komponenten bieten die Services *pruefung, vorlesung* und *vorl_beenden* an. Das gleiche gilt auch für die Dummy-Lehrstühle (siehe Abschnitt 5.4.5). Der Dummy *externe_fb* vom Typ *ext_fb* (externer Fachbereich) bietet nur die Vorlesungen an, die nicht von den Informatik-Lehrstühlen gehalten werden, wie z.B. Nebenfachvorlesungen, Mathematik- und E-Technikveranstaltungen. Der *pg*-Dummy vom Typ *pg_dummy* bietet weder Prüfungen noch Vorlesungen an, sondern nur die Projektgruppe. Der Aufbau des Komponententyps *fbi* findet sich in Abbildung 5.10 wieder, wobei aus Platzgründen nicht alle Lehrstühle und keine Dummies abgebildet werden.

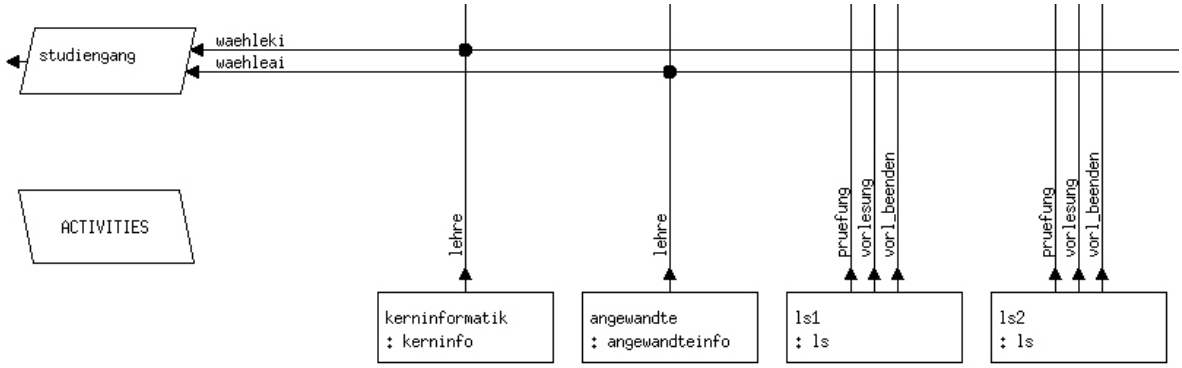


Abbildung 5.10: fbi

5.4.4 ls

Die Lehrstühle stellen für ihre Vorlesungen und Prüfungen Professoren (*profs*), WiMis (*wimis*), Hörsäle (*hs_raume*) und Räume (*raeume*) zur Verfügung. Hierbei ist zu berücksichtigen, dass diese mit den Ressourcen des Komponententyps *uni* geshared sind, was in Abbildung 5.11 zu sehen ist.

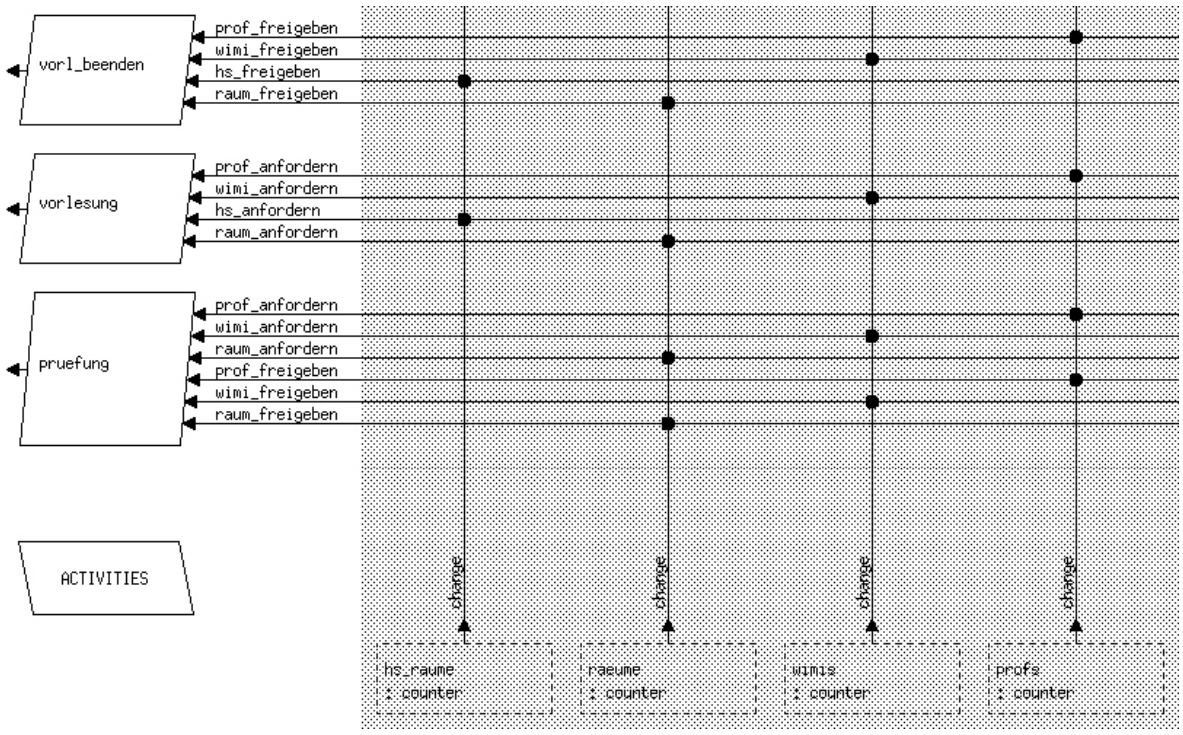


Abbildung 5.11: ls

Die Ressourcenanforderung unterscheidet sich nach der Anzahl der Semesterwochenstunden. Die Aufteilung der Semesterwochenstunden in Vorlesungen und Übungen sind durch die DPO 2001 [DPO 01] festgelegt. Die Ressourcenanforderung im provided-service *vorlesung* ist im Code wie folgt geregelt:

```
CASE sws

WHEN 6 :
   IF prof >= 4 AND wimi >= ue_gr AND raum >= ue_gr AND hs >= 4
     THEN
        prof_anfordern([-4]);
        prof := prof - 4;
        wimi_anfordern([-ue_gr]);
        wimi := wimi - ue_gr;
        hs_anfordern([-4]);
        hs := hs -4 ;
        raum_anfordern([-ue_gr]);
        raum := raum - ue_gr ;

        RESULT TRUE;

     ELSE
        RESULT FALSE;
```

```
     END IF;

WHEN 5 :
   IF prof >= 3 AND wimi >= ue_gr AND raum >= ue_gr AND hs >= 3
     THEN
        prof_anfordern([-3]);
        prof := prof - 3;
        wimi_anfordern([-ue_gr]);
        wimi := wimi - ue_gr;
        hs_anfordern([-3]);
        hs := hs -3 ;
        raum_anfordern([-ue_gr]);
        raum := raum - ue_gr ;

        RESULT TRUE;

     ELSE
        RESULT FALSE;

     END IF;

WHEN 3 :
   IF prof >= 2 AND wimi >= ue_gr AND raum >= ue_gr AND hs >= 2
     THEN
        prof_anfordern([-2]);
        prof := prof - 2;
        wimi_anfordern([-ue_gr]);
        wimi := wimi - ue_gr;
        hs_anfordern([-2]);
        hs := hs -2 ;
        raum_anfordern([-ue_gr]);
        raum := raum - ue_gr ;

        RESULT TRUE;


     ELSE
        RESULT FALSE;

     END IF;

{SOPRA}
WHEN 2 :
   IF  wimi >= ue_gr AND raum >= ue_gr
     THEN
        wimi_anfordern([-ue_gr]);
        wimi := wimi - ue_gr;
```

```
         raum_anfordern([-ue_gr]);
         raum := raum - ue_gr ;

         RESULT TRUE;

      ELSE
         RESULT FALSE;

      END IF;

{HAPRA}
WHEN 1 :
   IF  wimi >= ue_gr AND raum >= ue_gr
     THEN
         wimi_anfordern([-ue_gr]);
         wimi := wimi - ue_gr;
         raum_anfordern([-ue_gr]);
         raum := raum - ue_gr ;

         RESULT TRUE;


      ELSE
         RESULT FALSE;

      END IF;

ELSE    :
         RESULT FALSE;

END CASE;
```

Bei 6 SWS wird ein Professor und ein Hörsaal für vier SWS angefordert. Die benötigte Anzahl SWS für WiMis und Räume (ue_gr) wird in den einzelnen Vorlesungen berechnet. Der berechnete Wert wird bei der Ressourcenanforderung übergeben. Zusätzlich zu dem jeweiligen *_anfordern*-Befehl wird ebenfalls die entsprechende Kapazität, die noch zur Verfügung steht, reduziert. Die Vorlesung kann nur stattfinden (RESULT TRUE;) wenn alle benötigten Ressourcen angefordert werden können, was nur nach einer vorangehenden Überprüfung der vorhandenen Kapazitäten möglich ist. Für den Fall, dass die Kapazitäten nicht ausreichen, wird die Vorlesung nicht angeboten (RESULT FALSE;).
Bei 5 und 3 SWS läuft die Ressourcenanforderung analog mit entsprechenden Werten ab.
Um die Veranstaltungen SoPra und HaPra auf ähnliche Weise modellieren zu können, wurden sie den Fällen 2 beziehungsweise 1 SWS zugeordnet, obwohl dies nicht ihre eigentliche SWS-Anzahl repräsentiert.
Der Code des Service *vorl_beenden* ist ähnlich dem Code des Service *vorlesung*. Die Ressourcen werden hier freigegeben und nicht angefordert. Der ganze Vorgang der Freigabe findet erst nach der Vorlesungszeit statt (spend(15)).

Damit ein Student eine Prüfung ablegen kann, muss mindestens ein Professor und ein WiMi für eine SWS zur Verfügung stehen. Dies ist im Code wie folgt umgesetzt:

```
IF prof >= 1 AND wimi >= 1 THEN

     prof_anfordern([-1]);
     wimi_anfordern([-1]);

     RESULT TRUE;

     prof_freigeben([+1]);
     wimi_freigeben([+1]);

  ELSE

     RESULT FALSE ;

END IF;
```

5.4.5 Dummies

Die Dummies *ls_kat_a*, *ls_kat_b* und *ls_dummy* sind identisch aufgebaut. Der Dummy für den *externen_fb* stellt keine mündlichen Prüfungen zur Verfügung, so dass diese Aktivität entfallen kann. Bei dem *pg_dummy* wird auf die Ressource Hörsäle verzichtet, da diese nicht gebraucht wird. Die Dummies werden benötigt, weil die Vorlesungen für die Schein-Prüfungen, die Vorlesungen aus den Schwerpunktgebieten, sowie die Vorlesungen aus den Katalogen nicht eindeutig den Lehrstühlen zugeordnet werden können. Aus diesem Grund werden in diesen Bereichen die Ressourcen gesammelt und ihr Bedarf an alle Lehrstühle in gleichem Umfang verteilt. Die Ressourcen des *externe_fb*'s werden nicht auf die Lehrstühle verteilt, sondern greifen direkt auf die Ressourcen der Universität zurück.

5.4.6 kerninfo

Der Studiengang Kerninformatik teilt sich in Grundstudium (Komponente *grundstudium*) und Hauptstudium (Komponente *hauptstudium*) auf, nach welchen der Student sein Vordiplom beziehungsweise Diplom erhält. Dies ist in Abbildung 5.12 dargestellt, in welcher die Komponententypen *grundstudiumki* und *hauptstudiumki* die Dienste *vordiplom* und *diplom* bereitstellen.
Im HIT-Code werden die used-services *vordiplom* und *diplom* nacheinander aufgerufen, da dies auch dem Ablauf in der Realität entspricht, wo der Student zunächst sein Grundstudium erfolgreich zu beenden hat, bevor er das Hauptstudium mit abschließendem Diplom absolviert.

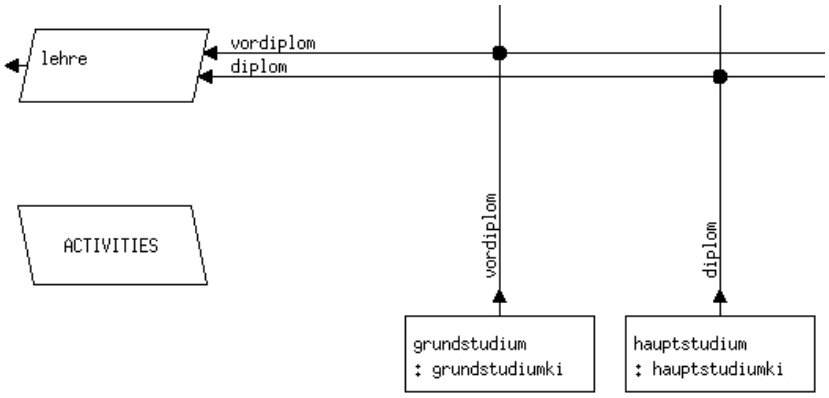


Abbildung 5.12: kerninfo

5.4.7 grundstudiumki und hauptstudiumki

Das Grundstudium ist in vier Semester (sem1 bis sem4), das Hauptstudium in fünf Semester (sem5 bis sem9) unterteilt.
Die jeweiligen used-services sem1 bis sem4 des Dienstes *vordiplom* werden sequentiell aufgerufen, so dass die Modellannahme, die besagt, dass ein Student alle Veranstaltungen eines Semesters erfolgreich bestritten haben muss, bevor er das nächste Semester in Angriff nehmen kann, erfüllt ist (analog für das Hauptstudium). Das ist für das Grundstudium in Abbildung 5.13 und für das Hauptstudium in Abbildung 5.14 und in dem folgenden Code (repräsentativ für das Grundstudium) zu sehen.

```
sem1;
sem2;
sem3;
sem4;
```

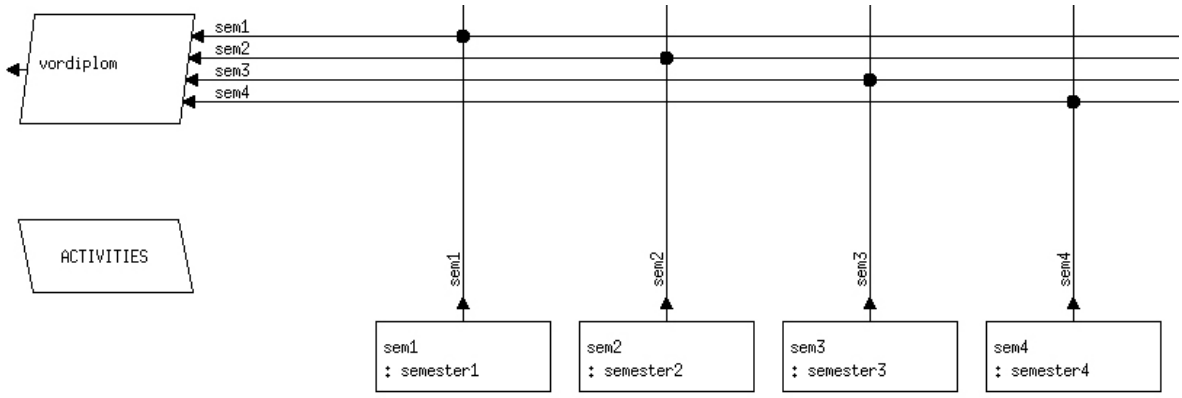


Abbildung 5.13: grundstudiumki

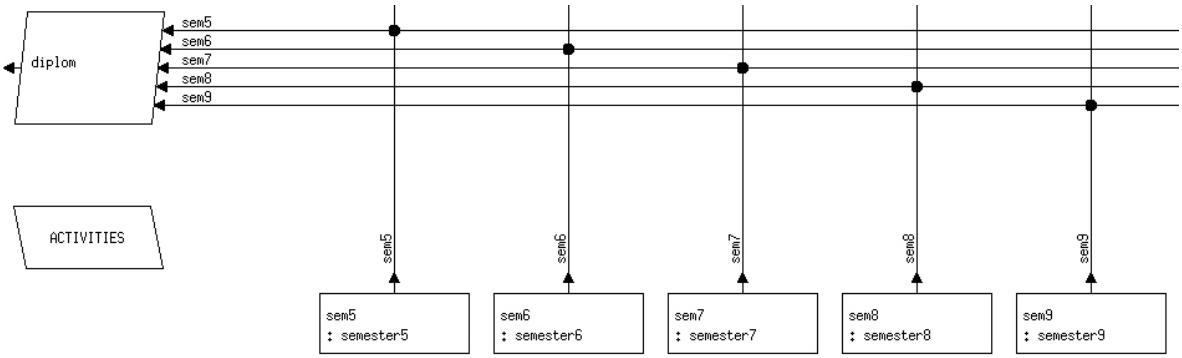


Abbildung 5.14: hauptstudiumki

5.4.8 semester1 bis semester4

An dieser Stelle wird stellvertretend nur der Komponententyp *semester4* betrachtet, da die Grundstruktur in allen vier Semestern gleich ist. Im vierten Semester wird zusätzlich die Vorlesung GTI angeboten, die mit einer mündlichen Prüfung abgeschlossen wird. In den anderen Veranstaltungen werden Prüfungen in der jeweiligen Vorlesungszeit durchgeführt und somit keine zusätzlichen Ressourcen angefordert.
semester1 besteht aus den Komponenten *dap1*, *m1*, *rs*, *nebenfach*.
semester2 besteht aus den Komponenten *dap2*, *m2*, *bs1*, *et_nt*, *nebenfach*.
semester3 besteht aus den Komponenten *hapra*, *bs2*, *wr*, *logik*, *nebenfach*.
semester4 besteht aus den Komponenten *proseminar*, *gti*, *sopra*, *is*, *nebenfach*, was in der folgenden Abbildung 5.15 aufgeführt wird.

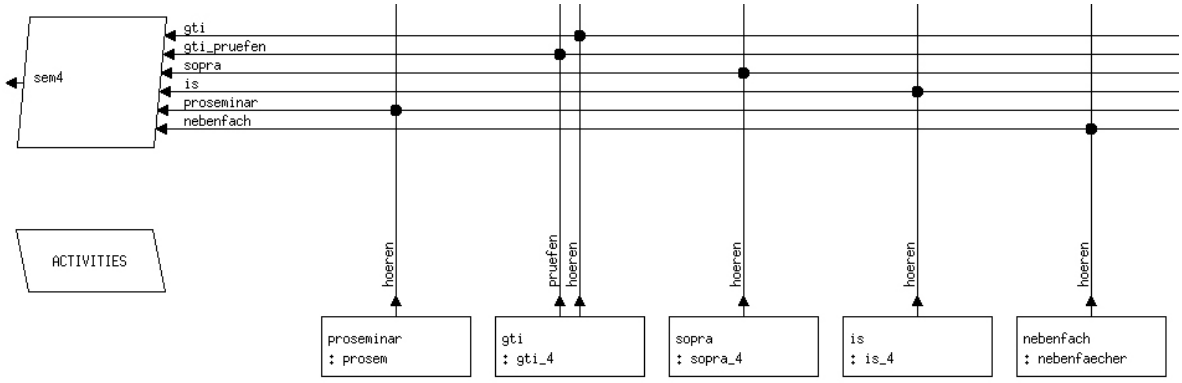


Abbildung 5.15: semester4

Der HIT-Code beschränkt sich ebenfalls auf das semester4:

```
CONCURRENT
 gti;
 gti_pruefen;
TO
 sopra;
TO
 is;
TO
 proseminar;
TO
 nebenfach;
TO
 spend(26);
END CONCURRENT;
```

Die aufgeführten Veranstaltungen finden alle parallel statt. Dies ist im Code durch den CONCURRENT Block realisiert. Der Student verbringt in diesem, wie in jedem anderen Semester, 26 Wochen, was durch die Anweisung *spend(26)* sichergestellt wird, bevor er in das nächste Semester übergeht. Die gti-Prüfung, ebenso wie jede andere Prüfung, kann der Student erst ablegen, nachdem er die entsprechende Vorlesung besucht hat, denn die Anweisung gti_pruefen folgt auf den Aufruf *gti*.

5.4.9 semester5 bis semester9

Die Semester im Hauptstudium sind analog zu denen im Grundstudium aufgebaut. Sie enthalten sowohl Vorlesungen mit mündlichen Prüfungen, als auch Veranstaltungen, deren Prüfungen in der entsprechenden Vorlesungszeit durchgeführt werden ohne weitere Ressourcen anzufordern.
semester5 besteht aus den Komponenten *swk*, *nebenfach*, *katalog_a* und *katalog_b*.
semester6 besteht aus den Komponenten *iundg*, *ueb*, *nebenfach*. Und einer weiteren Vorlesung aus *katalog_a* oder *katalog_b*.
semester7 besteht aus den Komponenten *seminar*, *pg*, *schwerpunkt*, *leist_nachweis7*.
semester8 besteht aus den Komponenten *schwerpunkt*, *leist_nachweis8*, *pg*.
semester9 besteht aus der Komponente *diplom*.
Die semester5 bis 8 sind vom Code her gleich aufgebaut wie die Semester im Grundstudium. Jedoch gibt es in semester6 eine Erweiterung, da der Student sich entscheiden muss, ob seine dritte Vorlesung im Wahlbereich aus dem Katalog A (Komponententyp *kat_a*) oder B (Komponententyp *kat_b*) stammt. Aufgrund der Annahme (siehe Abschnitt 5.3.4) wird der Katalog A mit einer Wahrscheinlichkeit von 70 Prozent ein weiteres Mal gewählt. Die verbleibenden 30 Prozent entfallen zwangsläufig auf die Wahl einer Fachprüfung im Katalog B.

```
CONCURRENT
 iug;
TO
 ueb;
TO
 nebenfach;
TO
 BRANCH
 PROB 0.7 : katalog_a;
            pruefen_kat_a;
 PROB 0.3 : katalog_b;
            pruefen_kat_b;
 END BRANCH;
TO
 spend(26);
END CONCURRENT;
```

Da für das Semester9 nur die Diplomarbeit vorgesehen ist, wird es im Modell nur als Verzögerung (spend(26)) modelliert. In dem HIT-Modell sieht dies wie in Abbildung 5.16 aus.

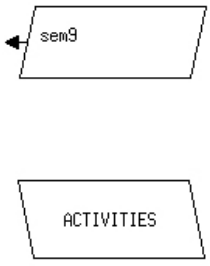


Abbildung 5.16: semester9

5.4.10 gti

Alle Vorlesungen mit mündlichen Prüfungen sind identisch aufgebaut. Aus diesem Grunde wird an dieser Stelle nur der Komponententyp *gti* beschrieben. Dabei werden die passenden Ressourcen wie Professor, WiMi, Hörsäle und Räume für die Vorlesungen und Prüfungen angefordert. Diese Ressourcen kommen von den jeweiligen Lehrstühlen beziehungsweise von den entsprechenden Dummies, welche in Abbildung 5.17 abgebildet ist.

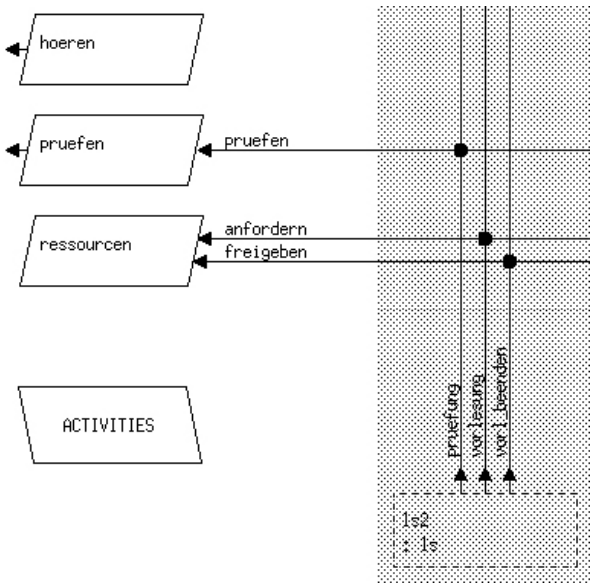


Abbildung 5.17: gti

Die Ressourcen werden alle 52 Wochen, also einmal im Jahr, angefordert, in dem der Prozess *ressourcen* in den *Activities* angestoßen wird (CREATE 1 PROCESS ressourcen EVERY 52). Eine Ausnahme bilden die Kataloge A und B, das Nebenfach, das Schwerpunktsgebiet und die Leistungsnachweise, deren Ressourcen alle 26 Wochen angefordert werden. Somit kann man die Verteilung der Ressourcen auf Sommer und Wintersemester aufschlüsseln. Die gti-Vorlesung besteht aus den drei Services *hoeren*, *pruefen* und *ressourcen*, welche im folgenden näher erläutert werden. Der Code von *hoeren* ist wie folgt aufgebaut:

```
anz := anz+1;
spend(2);

WHILE NOT (gti)

  LOOP
       spend(52);
  END LOOP;

spend(15);
anz := anz-1;
```

Für jeden Studenten, der die Vorlesung besuchen muss, wird die Anzahl der Hörer (anz) um eins erhöht. *spend(2)* stellt die Zeit zwischen Semesterbeginn und Vorlesungsbeginn dar. In der folgenden while-Schleife wird überprüft, ob die Vorlesung angeboten wird. Dies geschieht mit der Variablen *gti*, welche standardmäßig auf FALSE gesetzt ist. Wenn die Vorlesung angeboten wird, so wird ihr Wert im Service *vorlesung* der Komponente Lehrstuhl (ls) auf TRUE gesetzt. Wenn die Vorlesung nicht angeboten wird, müssen die Studenten 52 Wochen auf die nächste gti-Vorlesung warten. Wenn die Vorlesung jedoch stattfindet, so dauert sie 15 Wochen und danach wird die Anzahl der Studenten wieder heruntergesetzt.

Nachdem der Student die Vorlesung gehört hat, kann er nun den Service *pruefen* aufrufen, dessen Code wie folgt aussieht:

```
LOOP
 LOOP
 geprueft := pruefen;
 x := x+1;
 spend(1);
 END LOOP UNTIL geprueft  OR x=7;
 x := 0 ;
 IF NOT (geprueft) THEN
    spend(44);
 END IF;
 END LOOP UNTIL geprueft;
```

Zuerst wird das Ergebnis der Prüfungsanfrage auf die Variable *geprueft* gesetzt. Dies geschieht mit dem Aufruf *pruefen*, dessen Rückgabewert per Default auf FALSE gesetzt ist (*geprueft := pruefen;*). Wenn die Prüfung stattgefunden hat, so wird ihr Wert im Service *pruefung* der Komponente Lehrstuhl (ls) auf TRUE gesetzt. Für den Fall, dass der Student noch keine Prüfung abgelegt hat, fragt er jede Woche erneut an, ob die notwendigen Ressourcen für eine Prüfung vorhanden sind. Dies geschieht bis zum *Wert x=7*, welcher dafür steht, dass das Semesterende erreicht wurde. Wenn der Student bis zu diesem Zeitpunkt noch nicht geprüft wurde, muss er 44 Wochen warten, bis er erneut sieben mal die Gelegenheit hat, eine Prüfung anzufordern.

Der Code des Service *ressourcen* sieht wie folgt aus:

```
spend(26);
spend(1);
gr   := anz//20;
rest := anz MOD 20;
IF rest >10 THEN
   gr := gr +1 ;
END IF;
gr := gr * 2 ;
gti := anfordern(6,gr);
IF gti THEN
 spend(16);
 freigeben(6,gr);
 gti := FALSE;
 END IF;
```

Der Befehl *spend(26)* ist in allen Vorlesungen des Sommersemesters zu finden, da die Ressourcenanforderungen, wie gerade beschrieben, einmal pro Jahr erzeugt werden. *spend(1)* soll sicher stellen, dass alle Studenten, die die Vorlesung hören wollen im System bekannt sind und dass die Vorlesung eine Woche vor Vorlesungsbeginn angeboten wird. Danach wird die Anzahl der Übungsgruppen festgelegt. Übungsgruppen haben in der Regel eine Größe von 20 Studenten, ausgenommen sind HaPra, SoPra und PG. Da die Anzahl der Studenten in der Regel kein Vielfaches von 20 sein wird, wird überprüft, ob der Rest der noch nicht eingeteilten Studenten auf die bestehenden Gruppen verteilt wird

(*rest* kleiner gleich 10) oder ob eine zusätzliche Gruppe angeboten werden muss. Anschließend wird der gerade berechnete Wert noch mit zwei multipliziert, da die Übungen in der Vorlesung *gti* zwei SWS ausmachen. Im folgenden werden die Ressourcen für die SWS der kompletten Vorlesung und Übung angefordert und die oben erwähnte Statusvariable *gti* gemäß dem Rückgabewert gesetzt (gti := anfordern(6,gr)). Wenn die Vorlesung stattgefunden hat, müssen danach Ressourcen wieder frei gegeben (freigeben(6,gr)) und die Variable *gti* auf FALSE gesetzt werden. Die Freigabe der Ressourcen findet erst nachdem die Vorlesungszeit verstrichen ist (spend(16)) statt.

5.4.11 dap1

Alle Vorlesungen ohne mündliche Prüfungen sind identisch aufgebaut. Aus diesem Grunde wird an dieser Stelle nur der Komponententyp *dap1* beschrieben. Analog zu den Vorlesungen mit mündlichen Prüfungen werden auch hier die Ressourcen für die Vorlesungen angefordert. Allerdings entfallen hier die Ressourcenanforderungen für die Prüfungen. Der Aufbau ist der Abbildung 5.18 zu entnehmen.

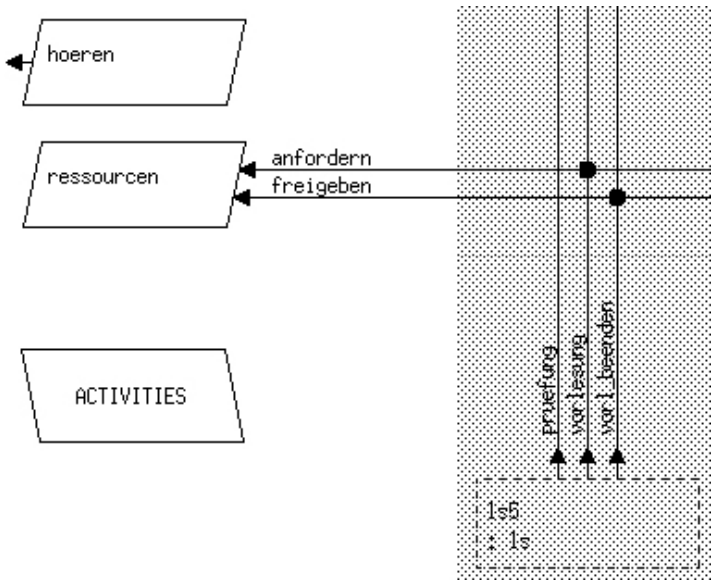


Abbildung 5.18: dap1

Der Code der beiden vorhandenen Services *hoeren* und *ressourcen* ist bis auf die übergebenen Parameter identisch mit dem gerade beschriebenen Code der Vorlesungen mit mündlichen Prüfungen.

5.4.12 PG

Für den Erhalt des Diploms ist ebenfalls der Scheinerwerb für die Projektgruppe Pflicht. Die Komponente ist analog zu den Vorlesungen aufgebaut und bezieht seine Ressourcen von dem bereits erwähnten pg_dummy. Es ist aufgeteilt in Semester sieben und acht (pg_7, pg_8). Aufbau und Code ist derselbe wie bei den Vorlesungen ohne mündliche Prüfungen. Da die Ressourcen in pg_7 für das ganze Jahr erzeugt werden, wird die Komponente pg_8 nur als Verzögerung (spend((26)) modelliert.

5.4.13 sonstige Komponententypen

Seminare
Das Proseminar aus dem vierten Semester und das Seminar aus dem siebten Semester werden gemäß Modellannahme nur als Verzögerung (spend(17)) modelliert. Aus diesem Grund entfällt die Ressourcenanforderung.

Nebenfächer
Da im Laufe des Studiums Prüfungen in einem selbst gewählten Nebenfach absolviert werden müssen, braucht man den so genannten *ext_fb*, der die benötigten Ressourcen zur Verfügung stellt. Es wird davon ausgegangen, dass die Ressourcenanforderung unerheblich für den Vorlesungsbetrieb des FBI sind. Aus diesem Grund ist sein Aufbau und Code identisch mit den vorher beschriebenen Vorlesungen ohne mündliche Prüfungen.

Praktika
Eine andere Vorraussetzung für das Vordiplom ist die erfolgreiche Teilnahme an Praktika. Diese sind im dritten Semester das HaPra und im vierten Semester SoPra, die in dem Modell fast identisch aufgebaut sind. Der Unterschied ist, dass die Ressourcen von anderen Lehrstühlen angeboten und genutzt werden. Bis auf die Größe der Gruppen ist der Aufbau und Code derselbe wie bei den Vorlesungen ohne mündlichen Prüfungen. Diese beträgt für HaPra 30 Studenten und für SoPra acht Studenten.

5.4.14 Verifikation

Aufgrund der kontinuierlich durchgeführten Kontrollen während der Implementierung, war eine explizite Verifikation nicht notwendig. Vielmehr wurde die jeweils aktuelle Implementierung erst fortgesetzt nachdem die bereits existierenden Hierarchieebenen auf ihre korrekte Funktion hin geprüft worden sind.

5.5 Implementierung des HIT-Modells für den Worstcase

Im folgenden werden nur die Änderungen gegenüber dem Bestcase aufgeführt.

5.5.1 Vorlesungen

Die Änderungen für den Worstcase in den Vorlesungen werden wieder stellvertretend mit Hilfe der Veranstaltung *gti* beschrieben. Der Aufbau der Komponenten ist identisch mit dem Aufbau beim Bestcase, lediglich der Code hat sich minimal geändert.

```
spend(26);
gti := anfordern(6,2);
IF gti THEN
 spend(16);
 freigeben(6,2);
 gti := FALSE;
 END IF;
```

Der Codeteil, welcher für die Berechnung der Anzahl der Übungsgruppen zuständig war, ist überflüssig geworden und somit entfallen. Bei dem *anfordern*- und *freigeben*-Befehl werden nun keine Variablen mehr für die SWS der Übungsgruppen übergeben, sondern die konkrete SWS-Anzahl.

5.5.2 PG und SoPra

Die Ressourcenanforderung über den *pg_dummy* bei der PG beziehungsweise über die Lehrstühle beim SoPra wurde dahingehend verändert, dass die Komponenten PG und SoPra nun je einen Counter erhalten. Der Aufbau ist in Abbildung 5.19 stellvertretend für die PG zu sehen.

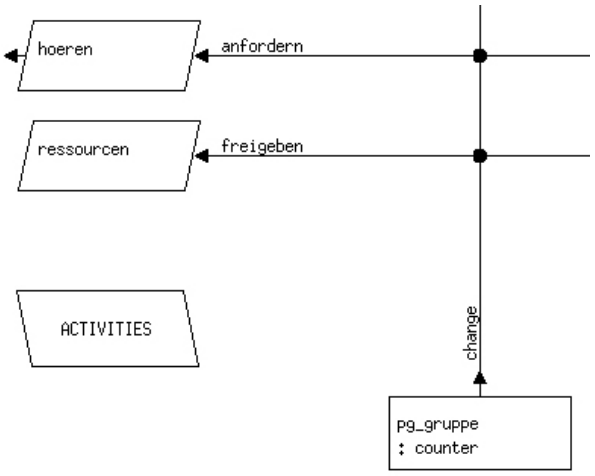


Abbildung 5.19: pg im Worstcase

Der Code des Service *hoeren* sieht wie folgt aus

```
anz := anz+1;
spend(2);
anfordern([-1]);
anz := anz-1;
spend(24);
```

und arbeitet mit dem Code des Service *ressourcen*

```
spend(1);
IF anz>0 THEN
  freigeben([+144]);
END IF;
```

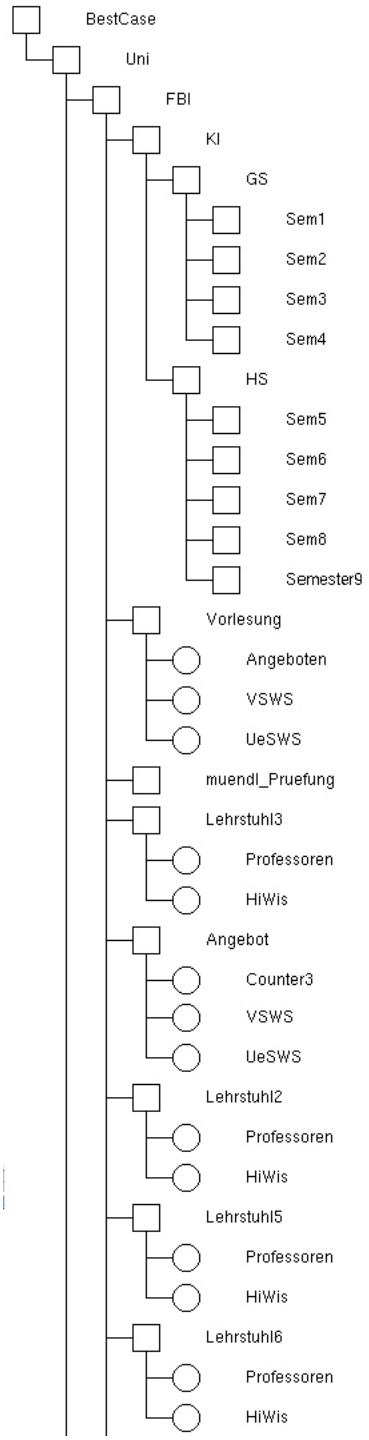
zusammen.
Die Ressourcen werden alle 52 Wochen angefordert (CREATE 1 PROCESS ressourcen EVERY 52;). Zu Beginn des Semesters wird die Anzahl der Studenten festgehalten, die an einer PG teilnehmen möchten. Nach einer Woche wird überprüft, ob Studenten vorhanden sind. Im positiven Fall werden 144 PG-Plätze freigegeben. Aus diesem Grund sehen die Parameter (Actual Parameters) wie folgt aus: ([0], [144], [0]).
Eine weitere Woche später werden die Ressourcen angefordert und die Anzahl der Studenten, die auf einen PG-Platz warten, um die Anzahl reduziert, die einen Platz bekommen haben. Mit *spend(24)* wird die Zeit bis zum Semesterende überbrückt. Die PG wird im achten Semester wieder durch einen simplen spend(26)-Befehl realisiert. Die Komponente SoPra ist identisch aufgebaut, mit dem einzigen Unterschied, dass 192 Plätze zur Verfügung stehen.

5.6 Implementierung und Verifikation des ProC/B-Modells für den Bestcase

Im folgenden Abschnitt wird die Modellierung und Umsetzung des Bestcases in ProC/B beschrieben und näher erläutert. Dabei wird speziell auf die Modellierungskonstrukte und deren Parameter eingegangen sowie deren Bedeutung erklärt. Die Annahmen, die für das Modell getroffen wurden, sind bereits in Kapitel 5.3.2 erläutert worden. Die Grundlagen für die gesamte Modellierung bilden die im vorherigen Kapitel 5.2.1 gezeigten Stuktur- und Ablaufdiagramme.

5.6.1 Aufbau und Struktur der ProC/B Modellierung

Der Aufbau des gesamten Simulationsmodells erfolgt hierarchisch, siehe Abbildungen 5.20 und 5.21. Dabei werden die einzelnen Ebenen ineinander geschachtelt, sofern dies dem realen Aufbau entspricht. Dadurch werden die Abhängigkeiten und Zugehörigkeiten entsprechend nahe an den realen Strukturen gehalten. Die Wurzel *Bestcase* bildet nur die Umgebung für die Modellierung, in die dann alle weiteren Konstrukte eingebunden werden. Der Knoten *Uni* ist der nächst tiefere Knoten im Strukturbaum, welcher dann in die feineren Strukturen aufgeteilt wird. Die nächste Hierarchieebene besteht aus den Funktionseinheiten *FBI*, dem Fachbereich Informatik, und allen weiteren Fachbereichen der Uni, hier zusammengefasst unter *ExterneFBs*. Eine Modellierung der externen Fachbereiche ist notwendig, da ein Studiengang am Fachbereich Informatik notwendigerweise auch Ressourcen von fremden Fachbereichen benötigt. Als weitere Ressource besitzt die Uni *Hörsäle*; diese werden zentral auf Uni-Ebene verwaltet und vom nur Fachbereich genutzt, sind diesem jedoch nicht zugehörig. Dem Knoten *FBI* wird dann der Knoten *KI* (Kerninformatik) zugeordnet in dem die Abläufe des Kerninformatikstudiums modelliert werden. Weiterhin werden dem *FBI* die Vorlesungen zugeordnet, da dieser die Vorlesungen für die Studiengänge zentral anbietet. Als weitere zugehörige Erweiterung des Fachbereiches wurden die benötigten Lehrstühle des Fachbereiches mit ihren *Professoren* und *HiWis* modelliert. Als letzte zugehörige Ressource besitzt der Fachbereich eigene *Räume*, die für bestimmte Veranstaltungen, wie Seminare oder Übungsgruppen, genutzt werden können. Der Bereich *KI* wird weiter für die einzelnen, expliziten Abläufe in Grundstudium *GS* und Hauptstudium *HS* verfeinert. Weiter werden dann die einzelnen Semester und deren internen Abläufe in den Funktionseinheiten *Sem1* bis *Sem8* und *Semester9* modelliert. Ähnlich dem Fachbereich Informatik werden auch bei den externen Fachbereichen *ExterneFBs* Vorlesungen von bestimmten Lehrstühlen angeboten, diese finden zum Teil in den eigenen Räumen statt.



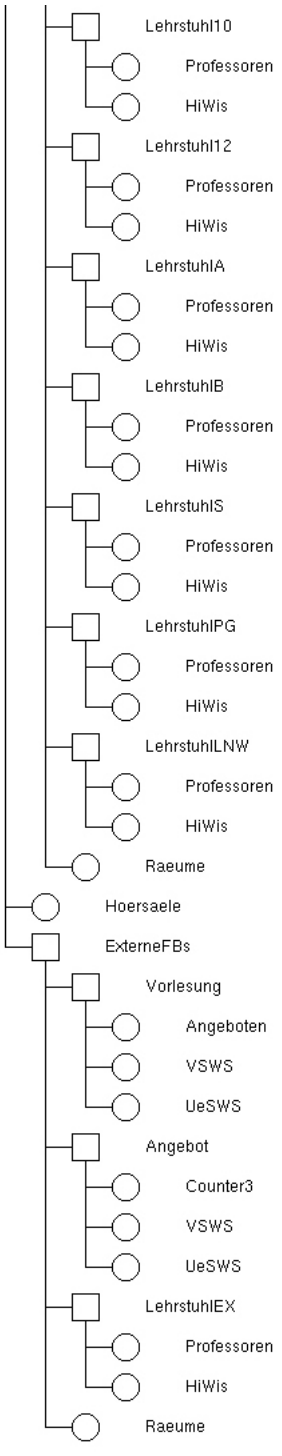


Abbildung 5.20: Struktur des ProC/B Modells - Teil1

Abbildung 5.21: Struktur des ProC/B Modells - Teil2

5.6.2 Hinweise zur Dokumentation der einzelnen Funktionseinheiten

In den folgenden Abschnitten werden alle Funktionseinheiten des ProC/B-Modells erläutert und dargestellt. Hierbei werden im Besonderen die Aufgabe der Funktionseinheiten, ihre Dienste und Funktionen und die jeweiligen Ressourcen berücksichtigt.
Bei der in vielen der Funktionseinheiten eingebundenen Ressource *EXTERNAL*, mit den Diensten *Nhoeren, hoeren, pruefen*, handelt es sich um die Einbindung der drei Dienste aus den Funktionseinheiten *Vorlesung* (im FBI) *Vorlesung* (in ExterneFBs) und *muendl_Pruefung* als so genannte *externe Funktionseinheit*. Diese Einbindung dient vielerorts nur der Möglichkeit auf die drei genannten Dienste auch in tiefer gelegen Ebenen zugreifen zu können. Auf Verwendung eines der Dienstes wird dann in der entsprechenden Funktionseinheit hingewiesen. In den Fällen in denen keiner der Dienste benötigt wird, ist auf eine explizite Erwähnung der externen Funktionseinheit verzichtet worden.
Die Beschreibung der Funktionseinheiten hat einen hierarchischen Aufbau. Ausgehend von der Funktionseinheit *Bestcase* auf der obersten Hierarchieebene werden zunächst alle Funktionseinheiten der zweiten Ebene beschrieben, dann der dritten, usw. Wobei die Reihenfolge der Beschreibung, von auf derselben Ebene liegenden Funktionseinheiten, nicht unbedingt der in den Abbildungen 5.20 und 5.21 visualisierten Darstellung entspricht, da einige Funktionseinheiten der Übersicht halber zusammengefasst, bzw. nacheinander beschrieben werden. Weiterhin kann es vorkommen, dass einige Abbildungen nicht in der Reihenfolge eingebunden werden in der sie im Text erwähnt werden. Dies liegt daran das auf eine möglichst gute Ausnutzung des vorhandenen Platzes Wert gelegt wird.

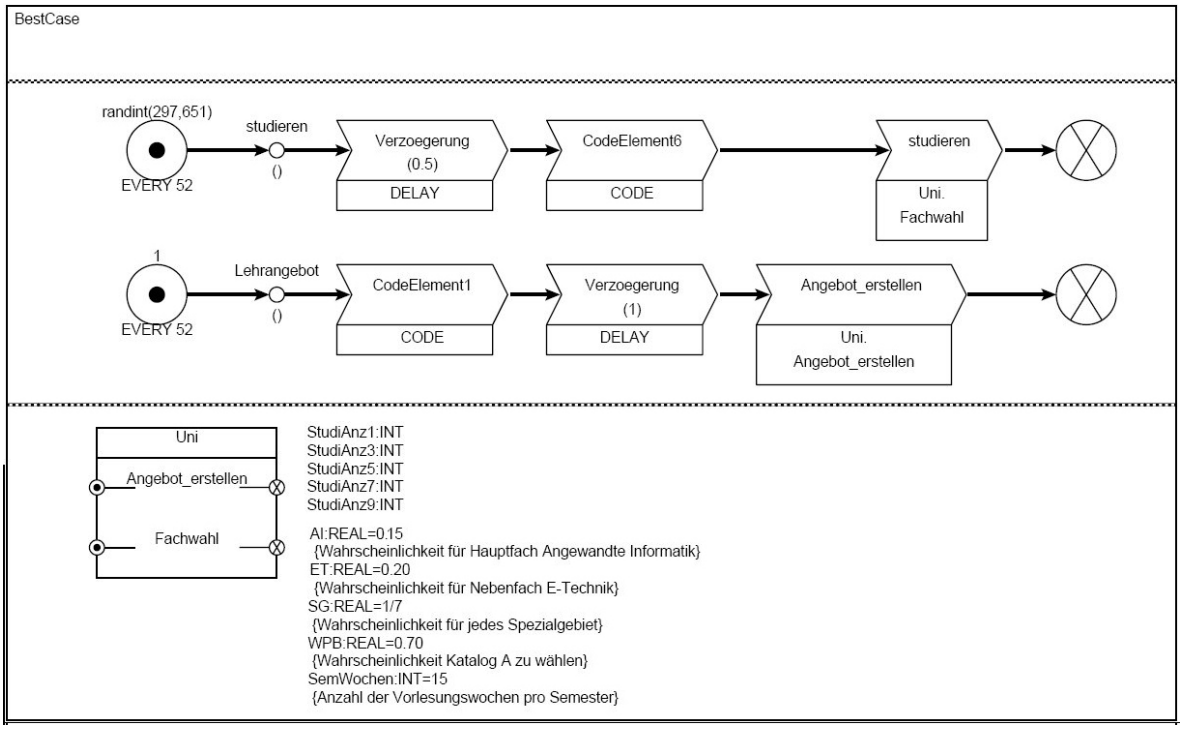


Abbildung 5.22: Bestcase Umgebung

5.6.3 Bestcase

Die Funktionseinheit *Bestcase* bildet nur die Umgebung für die Modellierung, in die alle weiteren Konstrukte eingebunden werden (siehe Abb. 5.22). Die Umgebung für den Bestcase besteht aus zwei Prozessketten.
Die Prozesskette *studieren* simuliert den Verlauf des Studiums. Hier besteht die Möglichkeit die einzelnen Parameter des Modells zentral zu variieren und an ggf. neue Annahmen anzupassen. Sie besteht aus drei Prozesskettenelementen. *Verzoegerung* ist eine Verzögerung um 0,5 Zeiteinheiten damit es keine Konflikte in den Zuweisungen zwischen den Codeelementen *CodeElement1* und *CodeElement6* gibt. Es sorgt also dafür, dass das *CodeElement1* vor dem *CodeElement6* abgearbeitet wird. Die Prozesskette *studieren* benötigt für alle weiteren Vorgänge und Abläufe die Funktionseinheit *Uni* und ruft dort den Dienst *Fachwahl* auf, da im Fachbereich Informatik in dem Modell zwei verschiedene Studiengänge zur Auswahl stehen. Die Verteilung der ankommenden Studenten für die Gesamtsimulation wird an der Quelle eingegeben. Hier werden die Prozesse mit der jeweiligen Verteilung generiert und gestartet.
Die Prozesskette *Lehrangebot* dient dazu das Studienangebot zu erstellen. Das Prozesskettenelement *Verzoegerung* stellt sicher, dass zuerst alle Studenten das *CodeElement6* des Prozesses *studieren* passiert haben und dort die aktuelle Anzahl der Studenten (Prozesse) ermittelt wurde und erst dann in dem Prozesskettenelement *Angebot_erstellen*, welches den Dienst *Angebot_erstellen* der Funktionseinheit *Uni* aufruft, das Angebot erstellt wird.
Die globalen Variablen *AI, ET, Mathe, SG* und *WPB* geben die jeweiligen Wahrscheinlichkeiten an, mit denen die Studenten den Studiengang *Angewandte Informatik*, eines der Nebenfächer, das Spezialgebiet und im Wahlpflichtbereich eine Vorlesung aus dem Vorlesungskatalog A (praktische Informatik) wählen. Diese Werte können an dieser Stelle global geändert werden und somit neue Eingabewerte für die Simulation geschaffen werden.
Die globalen Variablen *StudiAnz1, StudiAnz3, StudiAnz5, StudiAnz7, StudiAnz9* werden vom System benutzt und geben an wie viele Studenten sich momentan in dem jeweiligen Semester befinden. Da nur alle zwei Semester neue Studenten aufgenommen werden, benötigt man entsprechend nur Variablen für die ungeraden Semester. Es wird mittels Verzögerungselementen innerhalb der Prozesskette *studieren* und in ihren Bestandteilen sichergestellt, dass zuerst die Angebotserstellung durchlaufen wird (vgl. Abb. 5.40 auf Seite 129).
Codeelemente der Funktionseinheit *Bestcase*:

```
CodeElement CodeElement6
 StudiAnz1 := StudiAnz1+1;


CodeElement CodeElement1
 StudiAnz9 := StudiAnz7;
 StudiAnz7 := StudiAnz5;
 StudiAnz5 := StudiAnz3;
 StudiAnz3 := StudiAnz1;
 StudiAnz1 := 0;
```

5.6.4 Uni

Die Funktionseinheit *Uni* besitzt die Prozessketten *Fachwahl* und *Angebot_erstellen* (siehe Abb. 5.23). Die Prozesskette *Fachwahl* enthält ein Prozesskettenelement *Informatik*. Dieses ruft den Dienst *DPO-Wahl* der Funktionseinheit *FBI* auf.
Die Prozesskette *Angebot_erstellen* enthält zwei Prozesskettenelemente *anbieten* die mit einem *UND-Konnektor* verbunden sind, also parallel ablaufen. Beide rufen den Dienst *Angebot_erstellen* auf, allerdings von unterschiedlichen Funktionseinheiten (*FBI* und *ExterneFBs*).
Als Ressourcen stehen die Funktionseinheiten *FBI* (Fachbereich Informatik), *ExterneFBs* (externe Fachbereiche) und *Hoersaele*. Letztere ist als *Storage* modelliert und verwaltet entsprechend die verfügbaren Kapazitäten an Hörsälen in *SWS* (Semesterwochenstunden).

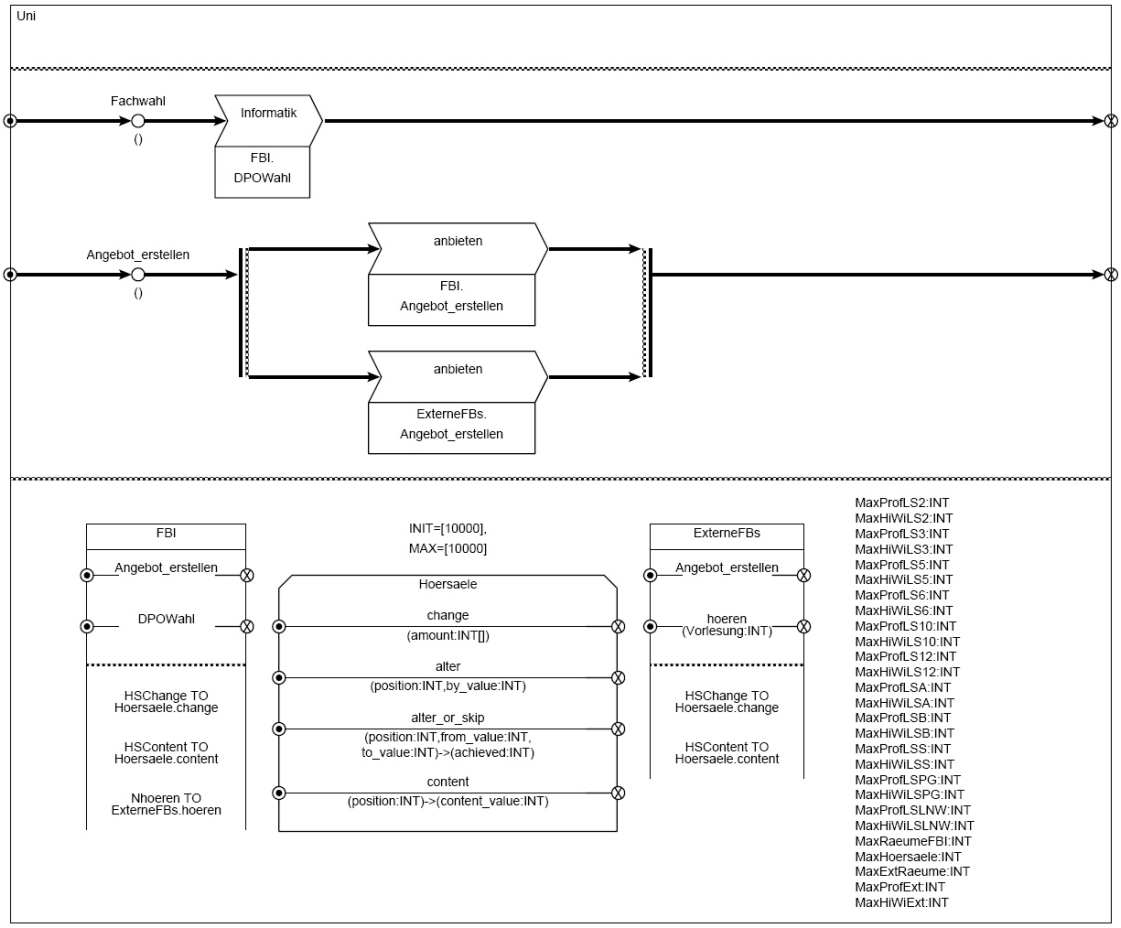


Abbildung 5.23: Funktionseinheit Uni

5.6.5 FBI

Die Funktionseinheit *FBI* enthält zwei Prozessketten (siehe Abb. 5.24).
In der Prozesskette *DPOWahl* werden am *ODER-Konnektor* die global gesetzten Wahrscheinlichkeiten für die Auswahl der Studiengänge ausgewertet. Das Prozesskettenelement *Kerninformatik* ruft den Dienst *studieren* der Funktionseinheit *KI* auf. Das Prozesskettenelement *AngewandteInformatik* wird

als Verzögerung über 234 Wochen, was genau 9 Semestern Regelstudienzeit entspricht, modelliert. Dies dient nur dazu, die Studenten die sich für Angewandte Informatik entscheiden aus dem Gesamtstrom der ankommenden Studenten heraus zu rechnen.
Die Prozesskette *Angebot_erstellen* besitzt ein Prozesskettenelement *anbieten* in dem der Dienst *angebote* der Funktionseinheit *Angebot* aufgerufen wird.
Als Ressourcen stehen folgende Funktionseinheiten zur Verfügung: die verschiedenen Lehrstühle, *muendl_Pruefung*, *Angebot*, *Vorlesung* und *KI*. Zusätzlich besitzt die Funktionseinheit als Ressource ein Storage *Raeume*, in welchem die räumlichen Kapazitäten in der Einheit SWS verwaltet werden. (siehe Abbildungen 5.25 und 5.26)

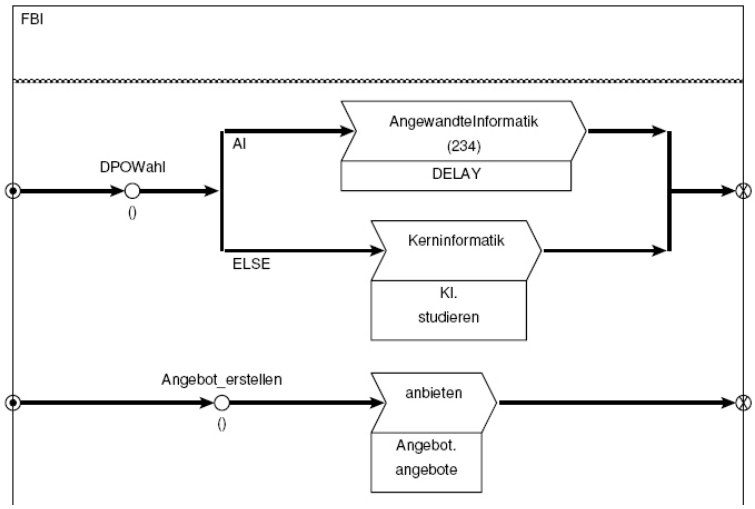


Abbildung 5.24: Funktionseinheit FBI Prozessketten

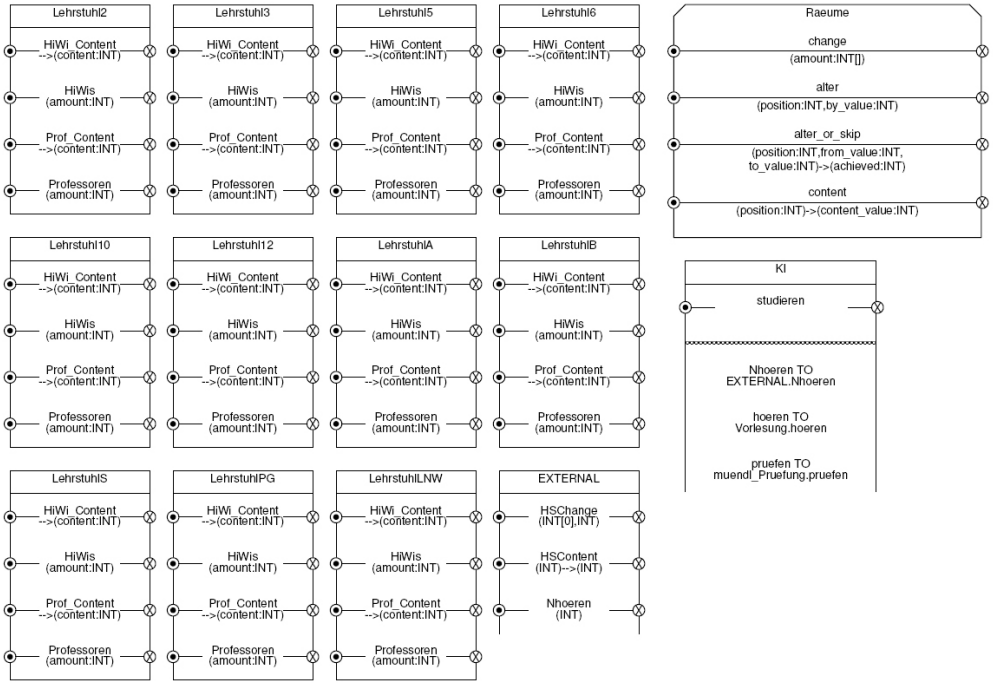


Abbildung 5.25: Funktionseinheit FBI Ressourcen - Teil1

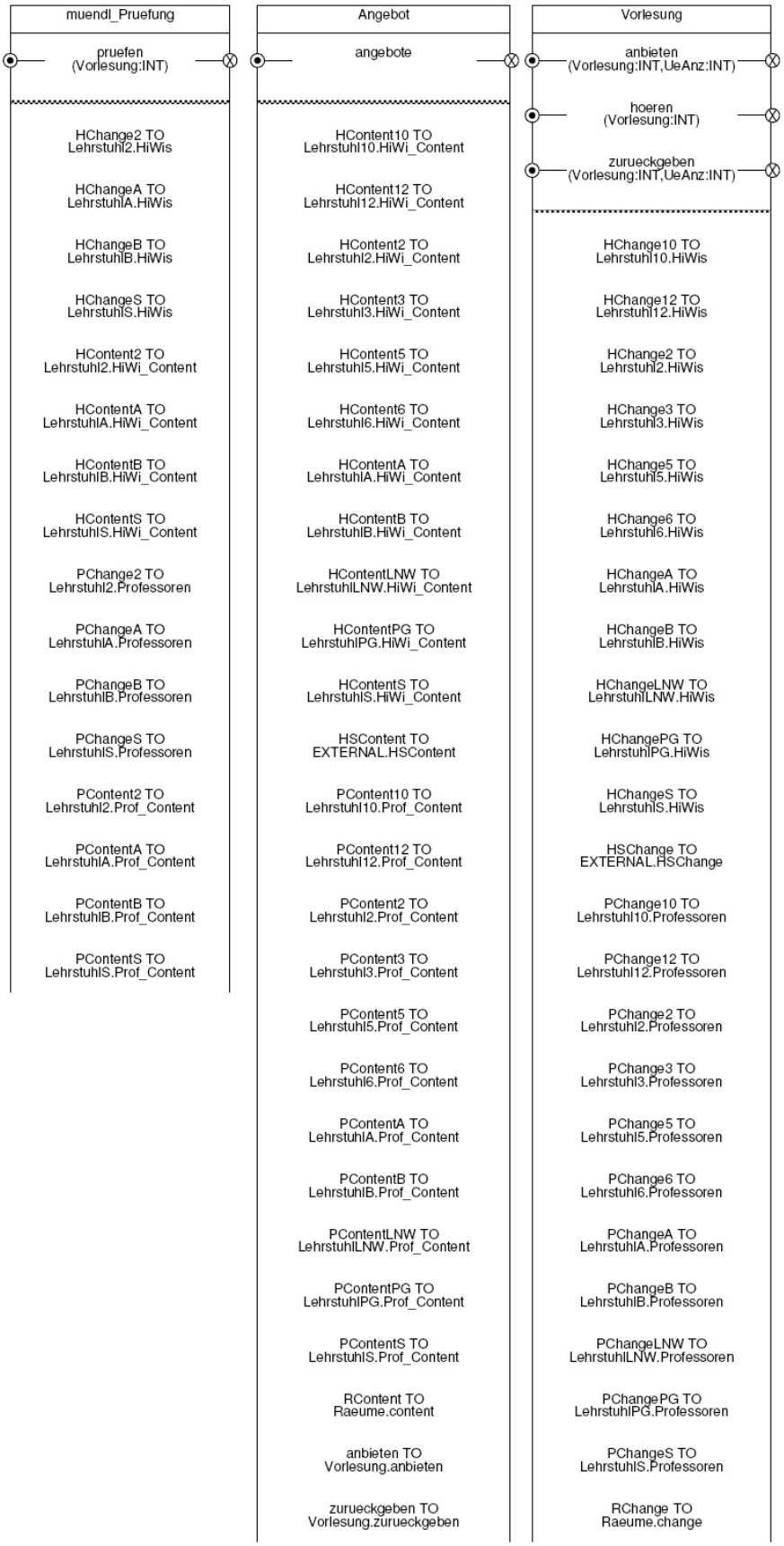


Abbildung 5.26: Funktionseinheit FBI Ressourcen - Teil2

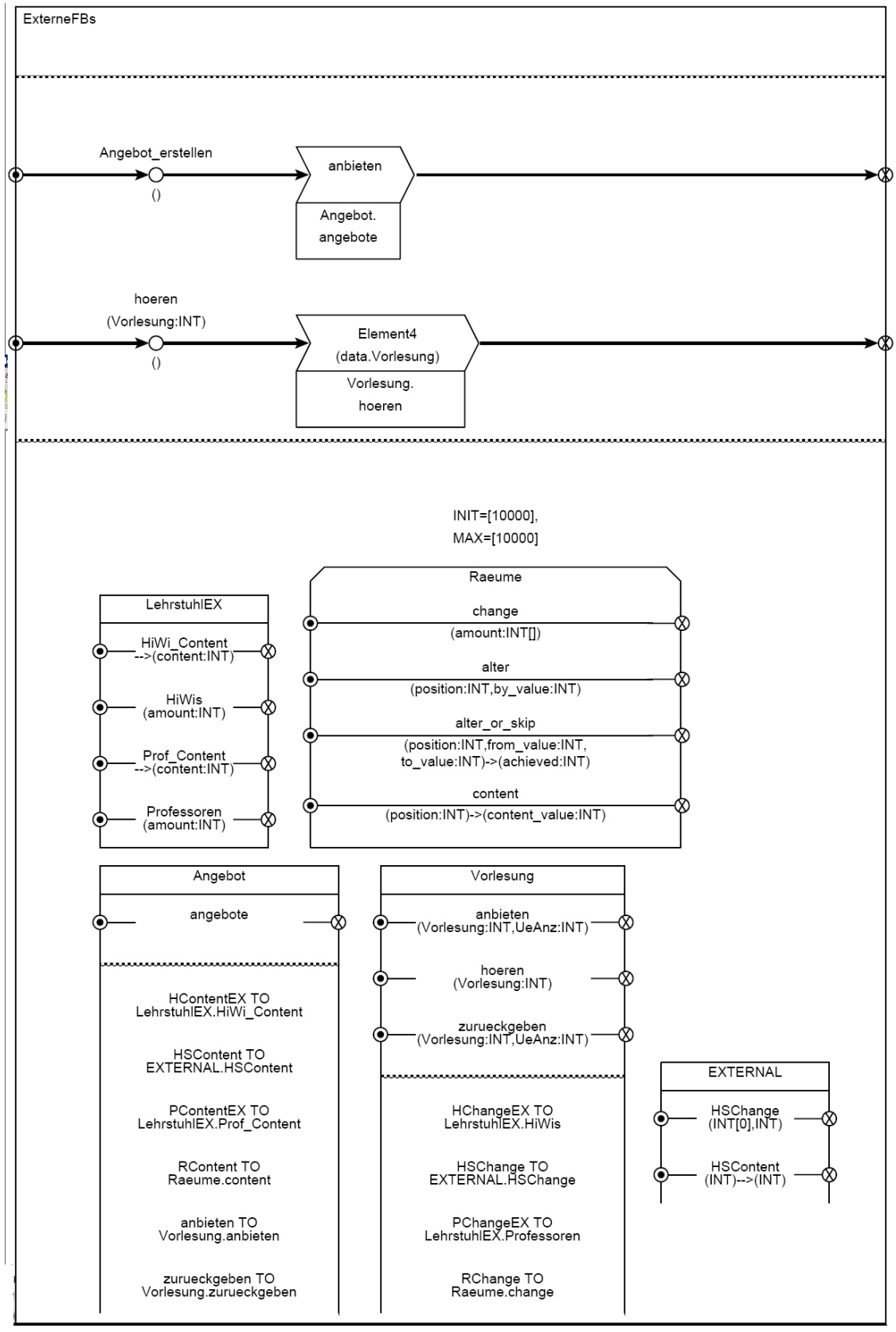


Abbildung 5.27: Funktionseinheit externeFBs

5.6.6 ExterneFBs

Die Funktionseinheit *ExterneFBs* repräsentiert alle externen Fachbereiche. Diese stellt Ressourcen bereit, die nicht zum Fachbereich Informatik gehören (siehe Abb. 5.27).
Sie besteht aus zwei Prozessketten, die jeweils ein Prozesskettenelement enthalten.
Das Prozesskettenelement *anbieten* der Prozesskette *Angebot_erstellen* ruft den Dienst *angebote* der Funktionseinheit *Angebot* auf.
Die Prozesskette *hoeren* enthält einen Parameter *Vorlesung* und ein Prozesskettenelement *Element4*, welches den Dienst *hoeren* der Funktionseinheit *Vorlesung* aufruft.
Als Ressourcen stehen die Funktionseinheiten *LehrstuhlEX*, *Angebot*, *Vorlesung* und das Storage *Raeume* zur Verfügung.

5.6.7 Angebot (im FBI)

In der Funktionseinheit *Angebot* werden alle vom Fachbereich Informatik angebotenen Vorlesungen (*DAP1, RS, DAP2, BS1, SWT, HAPRA, BS2, SoPra, IS, GTI, SWK, Übau, IuG, KatalogA, KatalogB, Schwerpunkt, PG und LNW*) erstellt und die dazugehörigen Ressourcen reserviert. Die aufgezählte Reihenfolge entspricht auch der jeweiligen Position in den Storages *VSWS* und *UeSWS*, die als Ressource zur Verfügung stehen.
Die Funktionseinheit enthält eine Prozesskette *angebote*. Diese besteht aus acht Blöcken (entspricht den ersten acht Semestern, da das neunte Semester nur für die Diplomarbeit vorgesehen ist) die nacheinander durchlaufen werden. Da alle Blöcke identisch aufgebaut sind, wird hier nur der Aufbau eines Blocks beschrieben (dementsprechend wurde die Darstellung in Abb. 5.28 und 5.29 um den nicht beschriebenen Teil gekürzt). Alle anderen sind analog.
Am Anfang jedes Blocks befindet sich ein UND-Konnektor, der dafür sorgt, dass alle an ihm angeschlossenen Prozesse scheinbar parallel abgearbeitet werden. Da aber eine echte parallele Abarbeitung zu Konflikten mit den Ressourcen führen kann, werden diese nacheinander durchlaufen. Dafür sorgt das Prozesskettenelement *Element2*, welcher den Dienst *change* der Funktionseinheit *Counter3* mit dem Parameterwert -1 aufruft. Solange der Counter den Wert -1 hat, werden keine anderen Prozesse des UND-Konnektors durchlaufen um Ressourcenkonflikte zu vermeiden. Im nächsten Prozesskettenelement *UeAnz_Berechnen* wird die Anzahl der notwendigen Übungsgruppen berechnet (20 Studenten pro Übungsgruppe). In den folgenden Prozesskettenelementen *P_auslesen, H_auslesen, R_auslesen, HS_auslesen* werden nacheinander die Professoren, HiWis, Räume und Hörsäle über die externe Funktionseinheit ausgelesen. Anschließend werden die vorhanden Ressourcen an Vorlesungs-SWS (*Element1*) und Übungs-SWS (*Element2*) ausgelesen. Im Prozesskettenelement *Ressourcen_Pruefung* wird überprüft, ob genug Ressourcen (Übungs-SWS, Vorlesungs-SWS, Professor-SWS und Hörsaal-SWS) vorhanden sind. Falls das nicht der Fall ist, wird die Variable *data.anbieten* auf *FALSE* gesetzt. Der ODER-Konnektor sorgt dafür, dass die entsprechende Vorlesung im Falle einer erfolgreichen Ressourcenüberprüfung im Prozesskettenelement *anbieten* über den Aufruf des Dienstes *anbieten* der Funktionseinheit *EXTERNAL* als „angeboten“ markiert wird. Das Prozesskettenelement *Element6* sorgt mit dem Parameterwert -1 dafür, dass jetzt die nächste Prozesskette des UND-Konnektors abgearbeitet werden kann. Das Prozesskettenelement *Element11* ist eine Verzögerung, die dafür sorgt, dass die angebotene Vorlesung eine Woche vor und eine Woche nach der Vorlesungszeit zur Verfügung steht. Mit dem folgendem Prozesskettenelement *zurueckgeben* wird die Vorlesung wieder zurückgegeben, d.h. alle reservierten Ressourcen werden wieder an die Lehrstühle zurückgegeben. *CodeElement6* sorgt dafür, dass die maximal in einem Semester benötigten Ressourcen auch nur über ein Semester gemessen werden. Hier wird das Maximum aller Ressourcen wieder

auf 0 zurückgesetzt. Das *CodeElement6* ist nur im Block des ersten und des zweiten Semesters enthalten um zu verhindern, dass die Werte mehrfach zurückgesetzt werden, da am Ende des dritten Semesters der nachfolgende Jahrgang wieder am Ende des ersten Semesters ist. Bevor der nächste Block abgearbeitet werden kann, wird die Prozesskette mit dem Prozesskettenelement *SemVerz* um 9 Zeiteinheiten verzögert.
Als Ressourcen enthält die Funktionseinheitt zwei Storages *VSWS* und *UeSWS* in denen die Vorlesungssemesterwochenstunden bzw. die Übungssemesterwochenstunden der einzelnen Vorlesungen verwaltet werden und einen Counter *Counter3*, der nur als Hilfsmittel zur Vermeidung von Ressourcenkonflikten benutzt wird. Die Ressourcen werden in Abbildung 5.30 dargestellt.

Codeelemente der Funktionseinheit *Angebot*:

```
CodeElement UeAnz_Berechnen
data.UeAnz:=StudiAnz1//20;
IF (data.UeAnz*20 <> StudiAnz1) THEN data.UeAnz:= data.UeAnz+1;
END IF;

CodeElement Ressourcen_Pruefung
IF data.UeSWS*data.UeAnz>data.HContent THEN data.anbieten:=FALSE;
END IF;
IF data.UeSWS*data.UeAnz>data.RContent THEN data.anbieten:=FALSE;
END IF;
IF data.VSWS>data.PContent THEN data.anbieten:=FALSE; END IF;
IF data.VSWS>data.HSContent THEN data.anbieten:=FALSE; END IF;

CodeElement CodeElement6
MaxProfLS2:=0;   MaxHiWiLS2:=0;
MaxProfLS3:=0;   MaxHiWiLS3:=0;
MaxProfLS5:=0;   MaxHiWiLS5:=0;
MaxProfLS6:=0;   MaxHiWiLS6:=0;
MaxProfLS10:=0;  MaxHiWiLS10:=0;
MaxProfLS12:=0;  MaxHiWiLS12:=0;
MaxProfLSA:=0;   MaxHiWiLSA:=0;
MaxProfLSB:=0;   MaxHiWiLSB:=0;
MaxProfLSS:=0;   MaxHiWiLSS:=0;
MaxProfLSPG:=0;  MaxHiWiLSPG:=0;
MaxProfLSLNW:=0; MaxHiWiLSLNW:=0;
```

5.6.8 Angebot (in ExterneFBs)

Die Funktionseinheit *Angebot* (in ExterneFBs) hat den gleichen Aufbau wie die Funktionseinheit *Angebot*. Hier werden alle Vorlesungen erstellt, die nicht zum Fachbereich Informatik gehören und auf externe Ressourcen zugreifen (vergleiche Abb. 5.28, 5.29 und 5.30).

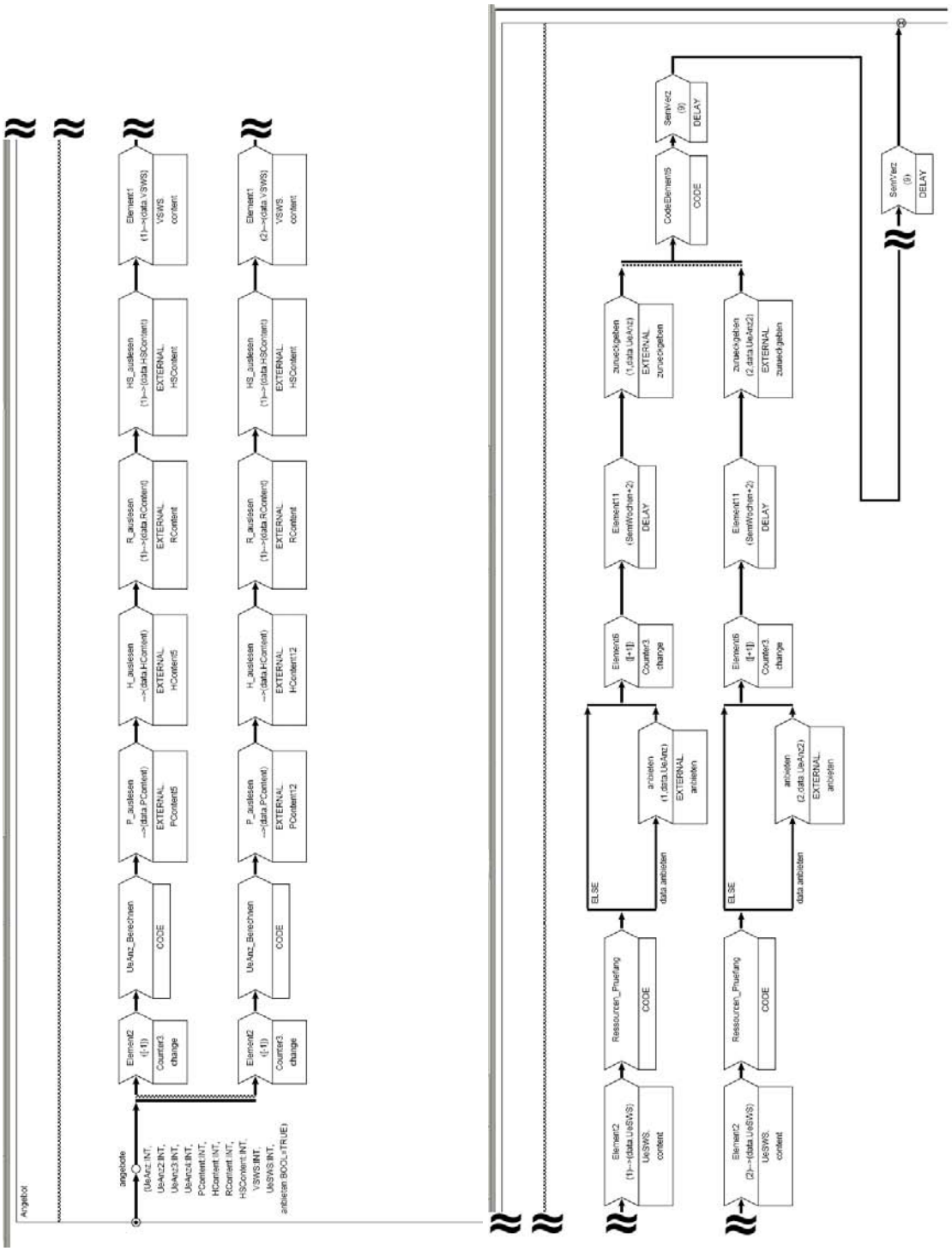


Abbildung 5.28: Funktionseinheit Angebot - Prozesskette - Teil1(gekürzt)

Abbildung 5.29: Funktionseinheit Angebot - Prozesskette - Teil2(gekürzt)

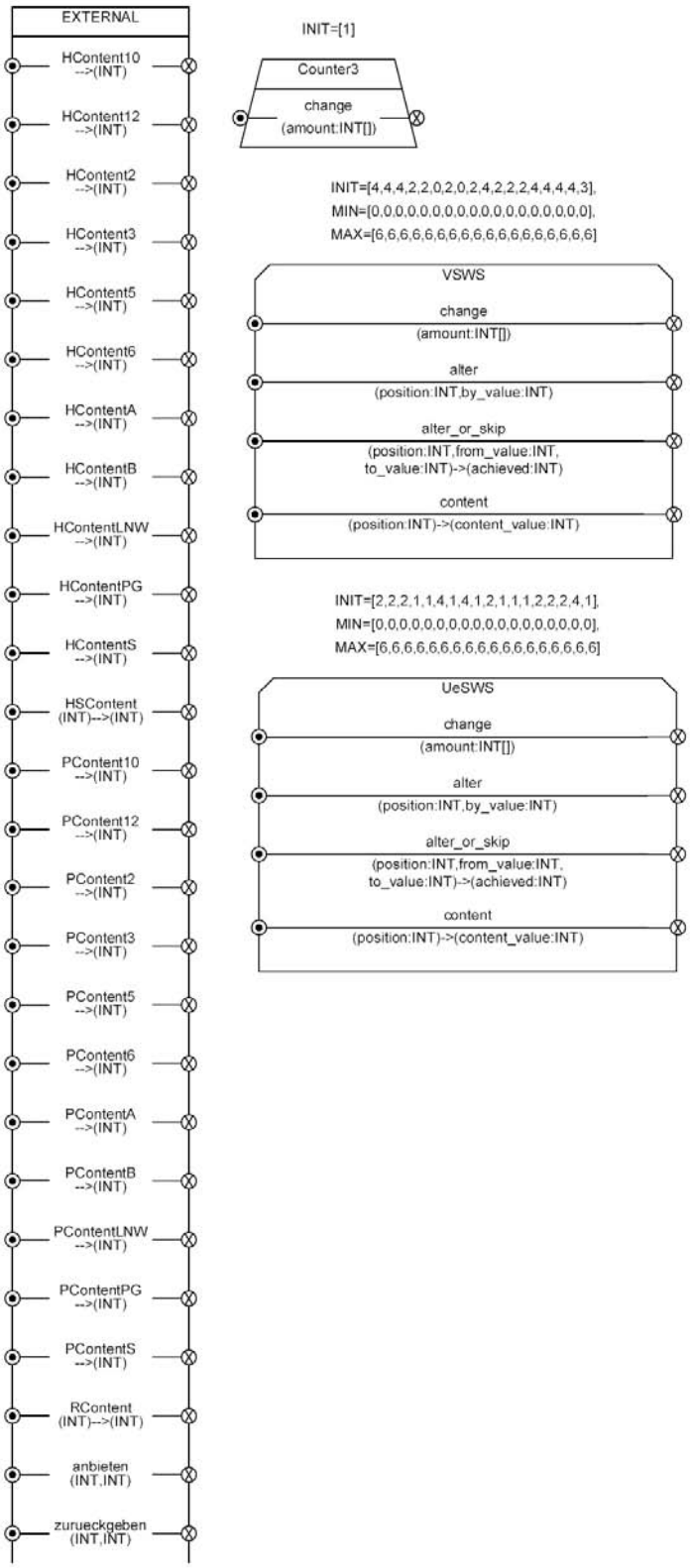


Abbildung 5.30: Funktionseinheit Angebot - Ressourcen

5.6.9 KI

Die Funktionseinheit *KI* modelliert den Aufbau und Ablauf für den Studiengang Kerninformatik (siehe Abb. 5.31). Die weitere Verfeinerung der Abläufe findet in den eingebunden Funktionseinheiten *GS* und *HS* statt. Zu diesem Zweck rufen die beiden Prozesskettenelemente *Grundstudium* und *Hauptstudium* der Prozesskette *studieren* jeweils den Dienst *Ablauf* der entsprechenden Funktionseinheit auf.

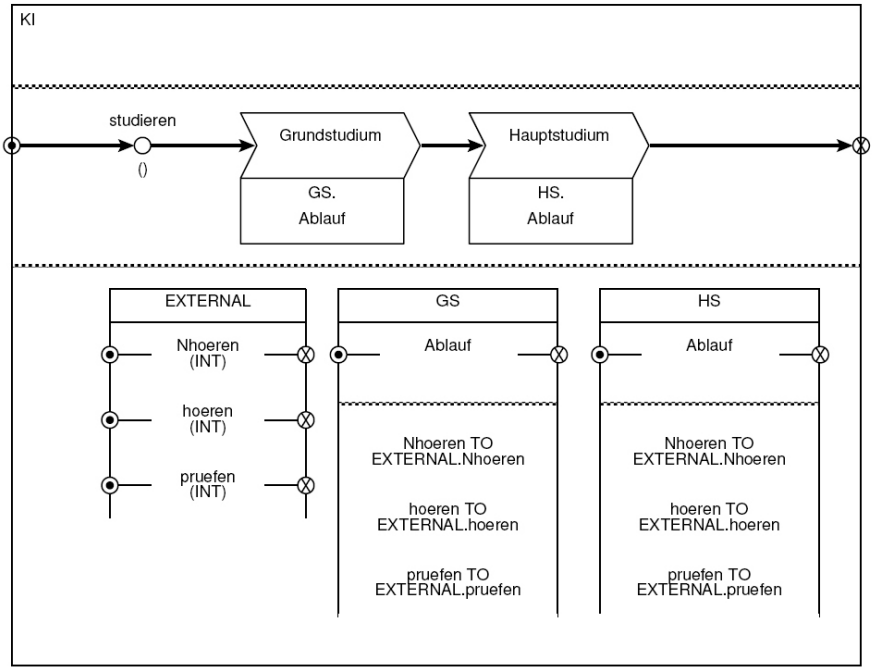


Abbildung 5.31: Funktionseinheit KI

5.6.10 muendl_Pruefung

Im Grundstudium muss nur eine mündliche Prüfung *GTI* bestanden werden, im Hauptstudium sind mehrere mündliche Prüfungen notwendig, um das Studium erfolgreich zu absolvieren. Dazu zählen die mündliche Prüfungen aus den *Katalogen A* und *B* und die Prüfungen aus den *Schwerpunktgebieten*. Die mündlichen Prüfungen werden im Modell durch die Funktionseinheit *muendl_Pruefung* realisiert (siehe Abb. 5.33).
Die Funktionseinheit *muendl_Pruefung* enthält eine Prozesskette *pruefen*.
Zunächst werden die lokalen Variablen *Versuche* und *Bestanden* definiert und mit dem Startwert „0“ bzw. „FALSE“ initialisiert. Dann folgt die Schleife *Bestanden_Loop* mit der verschachtelten Schleife *Versuche_Loop*. In der äußeren Schleife wird die Variable *Bestanden* überprüft und wenn diese den Wert „TRUE“ hat, wird die Schleife und somit auch die Funktionseinheit *muendl_Pruefung* verlassen, falls das nicht der Fall ist, wird die innere Schleife *Versuche_Loop* ausgeführt. Hier wird der Parameter *data.Vorlesung*, der von oberer Schicht übergegeben wurde, ausgewertet und mit dem ODER-Konnektor ein entsprechender Weg ausgewählt. Übersichtshalber ist die Aufteilung der mündlichen Prüfungen mit den Zahlen 10, 14, 15, 16 gekennzeichnet, die den Vorlesungen entsprechen. Die Zahl 10 steht für die *GTI*, die 14 und 15 entsprechend für die Prüfungen aus dem *Katalog A* bzw. *Katalog B* und die 16 für das *Schwerpunktgebiet*. Im weiteren wird exemplarisch nur eine Prozesskette für

die *GTI* Prüfung beschrieben. Die Abläufe für andere mündliche Prüfungen sind analog zu dieser. Nach der Auswertung des Parameters *data.Vorlesung*, in diesem Fall *data.Vorlesung* =10, wird in den Prozesskettenelementen *Element37* und *Element38* überprüft, ob zur Zeit noch Professoren und HiWis für die mündliche Prüfung zur Verfügung stehen. Dafür werden die Dienste *PContent2* und *HContent2* aufgerufen, die über die *Storages* im *Lehrstuhl2* implementiert sind. Es gibt vier Antwortmöglichkeiten:

1. die Antworten sind in beiden Fällen negativ
2. die Antwort ist im ersten Fall positiv, im zweiten Fall negativ
3. die Antwort ist im ersten Fall negativ, im zweiten Fall positiv
4. die Antworten sind in beiden Fällen positiv

Für die Fälle 1 bis 3 wird am nächsten ODER-Konnektor der Weg ausgewählt, der aus einer Verzögerungseinheit (*Wartezeit*) und einem Codeelement (*CodeElement19*) besteht. *Wartezeit* stellt eine einwöchige Verzögerung dar und das *CodeElement19* macht nichts anderes als die Anzahl der Versuchen um 1 zu inkrementieren. Für den Fall 4 wird der Weg mit den vier Prozesskettenelementen und einem *CodeElement5* ausgewählt. Die Elemente *Prof_anfordern* und *HiWi_anfordern* fordern über die externe Dienste *PChange2* und *HChange2* der Funktionseinheit *Lehrstuhl2* den Professor und die HiWis an. Das *CodeElement5* ändert den Wert der Variable *Bestanden* auf „TRUE“ und setzt den Wert der Variable *Versuche* auf „8“. Der Wert 8 gibt die Anzahl der Wochen an in denen der Student versuchen kann einen Prüfungstermin zu erhalten, er ergibt sich aus folgender Berechnung: Ein Semester dauert insgesamt 26 Wochen, von diesen 26 Wochen sind 2 Wochen Vorlaufzeit zu Beginn des Semesters bis die Vorlesungen beginnen. Es bleiben also 24 Wochen übrig, von diesen sind insgesamt 15 Wochen Vorlesungswochen. Erst im Anschluss an diese 15 Wochen können Prüfungen abgelegt werden. Es bleiben also für Prüfungsanforderungen 24-15=9 Wochen übrig, dieser Wert wurde nochmals für das Simulationsmodell um 1 Woche reduziert, um möglichen Synchronisationsproblemen vorzubeugen und um sicherzugehen das keine Prüfungsanforderung nach Ablauf des Semesters stattfindet. Danach werden der Professor und der HiWi über die gleichen externen Dienste *PChange2* und *HChange2* freigegeben. Unabhängig davon, welcher Weg ausgewählt wurde, kommt man auf jeden Fall zu dem letzten Prozesskettenabschnitt, der aus zwei Schleifenelementen und einer ODER-Verzweigung besteht. Im ersten Schleifenelement *Loop4* wird die Bedingung zum Aussprung aus der inneren Schleife implementiert. Die Bedingung lautet *until data.Versuche=8* und prüft, ob die Variable *Versuche* den Wert 8 hat. Diese Bedingung begrenzt die Anzahl der Prüfungsversuche auf acht. Wenn die Anzahl der Versuchen „8“ übersteigt, wird die in der Verzweigung implementierte Verzögerung ausgewählt und 44 Wochen gewartet, bis ein neuer Prüfungsversuch unternommen werden kann. Die 44 Wochen ergeben sich aus der Tatsache, dass eine neue Prüfungsanforderung nach dem Ende des Semesters erst weiter dann erfolgen kann, wenn im übernächsten Semester die Vorlesung erneut angeboten wurde. Es muss also ein ganzes Jahr (52 Wochen), abzüglich der vorlesungsfreien Zeit am Ende des Semesters, die für Prüfungsanforderungen zur Verfügung steht (8 Wochen), gewartet werden. Ansonsten wird die Prüfung als bestanden angesehen und die äußere Schleife verlassen. Dies geschieht im Prozesskettenelement *Loop7*, welches als Bedingung *until data.Bestanden* enthält, und überprüft, ob die Prüfung bestanden wurde. Bei der Erfüllung dieser Bedingung wird die äußere Schleife und damit auch die Funktionseinheit *muendl_Pruefung* verlassen.
Die Ressourcen der Funktionseinheit *muendl_Pruefung* werden in Abb. 5.32 dargestellt.

Codeelemente der Funktionseinheit *muendl_Pruefung*:

```
CodeElement CodeElement5
  data.Bestanden := TRUE;
  data.Versuche := 8;

CodeElement CodeElement19
  data.Versuche := data.Versuche+1;
```

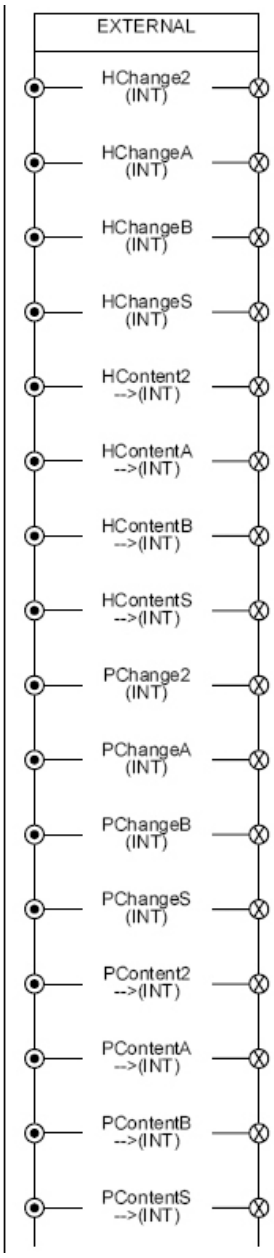


Abbildung 5.32: Funktionseinheit muendl_Pruefung - Ressourcen

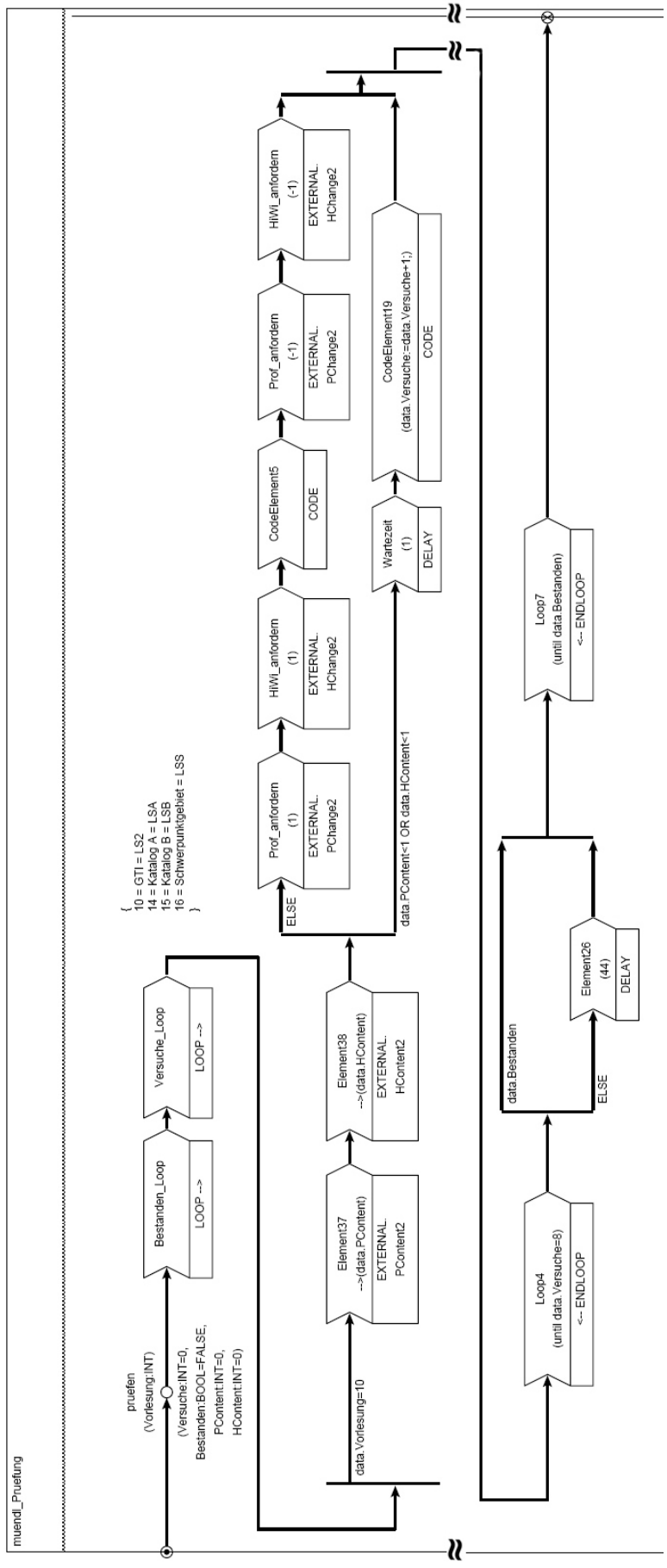


Abbildung 5.33: Funktionseinheit muendl_Pruefung - Prozesskette

5.6.11 Vorlesung (im FBI)

In der Funktionseinheit *Vorlesung* werden die Abläufe der einzelnen Vorlesungen und Veranstaltungen modelliert (siehe Abb. 5.34 und 5.35).
Die Funktionseinheit enthält drei Prozessketten *hoeren*, *anbieten* und *zurueckgeben*.
Die Prozesskette *hoeren* enthält die Prozesskettenelemente *Angebot* und *hoeren*. *Angebot* verringert, mit dem Aufruf der Funktion *alter* des Storages *Angeboten*, den Wert an der entsprechenden Position um eine Einheit. Das entspricht einem Studenten der diese Vorlesung hört. Das Prozesskettenelement *hoeren* ist eine Verzögerung um *SemWochen*, was der Dauer der Vorlesung entspricht. Die Prozesskette *hoeren* erhält als Parameter die Variable *Vorlesung* um bestimmen zu können um welche Vorlesung es sich handelt.
Die Prozesskette *anbieten* enthält die Prozesskettenelemente *VSWS_Auslesen* (Vorlesungs-Semester-Wochenstunden), *UeSWS_Auslesen* (Uebungs-Semester-Wochenstunden), *Prof_anfordern*, *Uebungs-Anzahl_Vorzeichen*, *HiWi_anfordern*, *UeRaum_anfordern*, *VRaum_anfordern* und *Angebot_setzen*. In den Prozesskettenelemeten *VSWS_Auslesen* und *UeSWS_Auslesen* werden zunächst, wie der Name schon sagt, die Anzahl an benötigten Semesterwochenstunden für die Vorlesung und die Übung von den jeweiligen Storages ausgelesen. Diese Werte werden zur weiteren Verarbeitung in den lokalen Variablen *VContent* und *UeContent* gespeichert. Im folgenden Codeelement *UebungsAnzahl_Vorzeichen* werden die Vorzeichen der beiden Variablen negiert. Dadurch wird die Reduzierung der freien Ressourcen im weiteren Verlauf der Prozesskette möglich. Der folgende ODER-Konnektor wertet den übergebenen Parameters *Vorlesung* aus, und leitet entsprechend der Zugehörigkeit der Vorlesung, die Anforderungen der Ressourcen an den entsprechenden Lehrstuhl weiter. Für den Fall, dass der Parameter den Wert „1“ hat, müssen die Ressourcen vom *Lehrstuhl5* angefordert werden. Es wird der erste Pfad des ODER-Konnektors durchlaufen. Im Prozesskettenelement *Prof_anfordern* wird der externe Dienst *PChange5*, der Funktionseinheit *Lehrstuhl5* aufgerufen. Über diesen Dienst wird von den Professoren-Ressourcen des *Lehrstuhl5* die benötigte Menge für die Vorlesung angefordert. Dieses Prinzip wird auch im folgenden Element *HiWi_anfordern* benutzt, mittels dessen die benötigten SWS an HiWi-Ressourcen bereitgestellt werden. Im Anschluss an den ODER-Konnektor wird im Element *UeRaum_anfordern* über den externen Dienst *RChange*, welcher den Dienst *change* des Storages *Raeume* der Funktionseinheit *FBI* einbindet, die entsprechende Menge SWS an Übungsräumen angefordert. Vergleichbar werden im folgenden Prozesskettenelement *VRaum_anfordern* über den externen Dienst *HSChange*, welcher auf den Dienst *change* des Storages *Hoersaele* der Funktionseinheit *Uni* referenziert, die benötigten Ressourcen an Hoersal-SWS angefordert. Beim letzten Prozesskettenelement *Angebot_setzen* wird der Dienst *alter* vom Storage *Angeboten* aufgerufen und damit die Vorlesung als „angeboten“ markiert.
Die Prozesskette *zurueckgeben* ist ähnlich aufgebaut wie die Prozesskette *anbieten*, nur dass hierbei, falls die Vorlesung angeboten wird, alle Ressourcen (*Professoren, HiWis und Räume*) an die entsprechenden Lehrstühle zurückgegeben werden. Die Ressourcen der Funktionseinheit *Vorlesung* sind in Abb. 5.36 dargestellt.

Codeelemente der Funktionseinheit *Vorlesung*:

```
CodeElement UebungsAnzahl_Vorzeichen
  data.UeContent := data.UeContent*dataUeAnz;
  data.UeContent := -data.UeContent;
  data.VContent := -data.VContent
```

```
CodeElement UebungsAnzahl
  data.UeContent := data.UeContent*data.UeAnz;
```

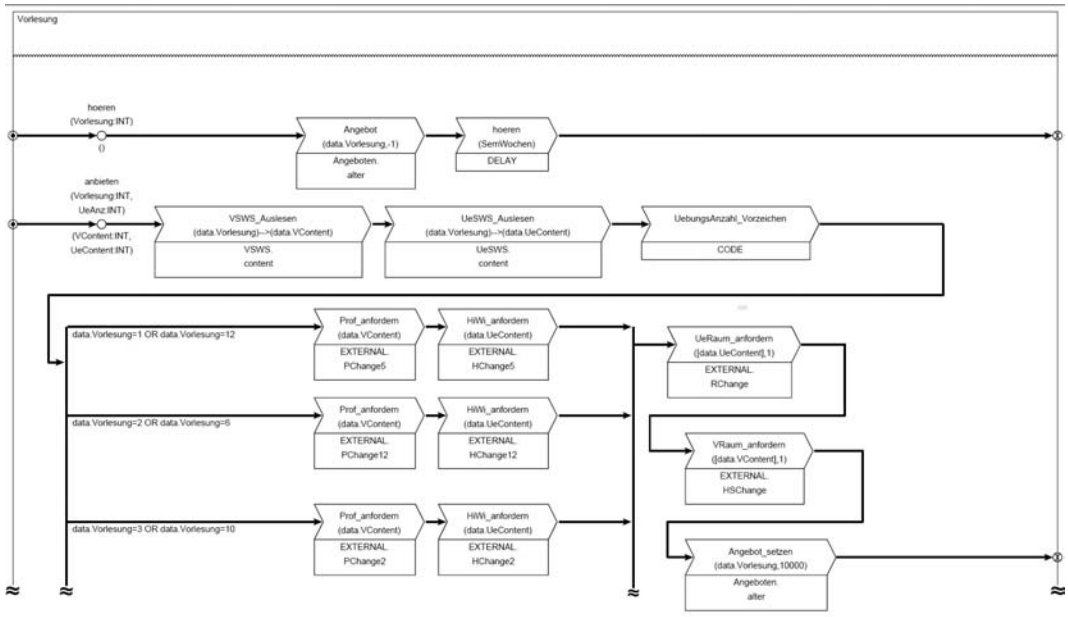


Abbildung 5.34: Funktionseinheit Vorlesung - Prozesskette - Teil1

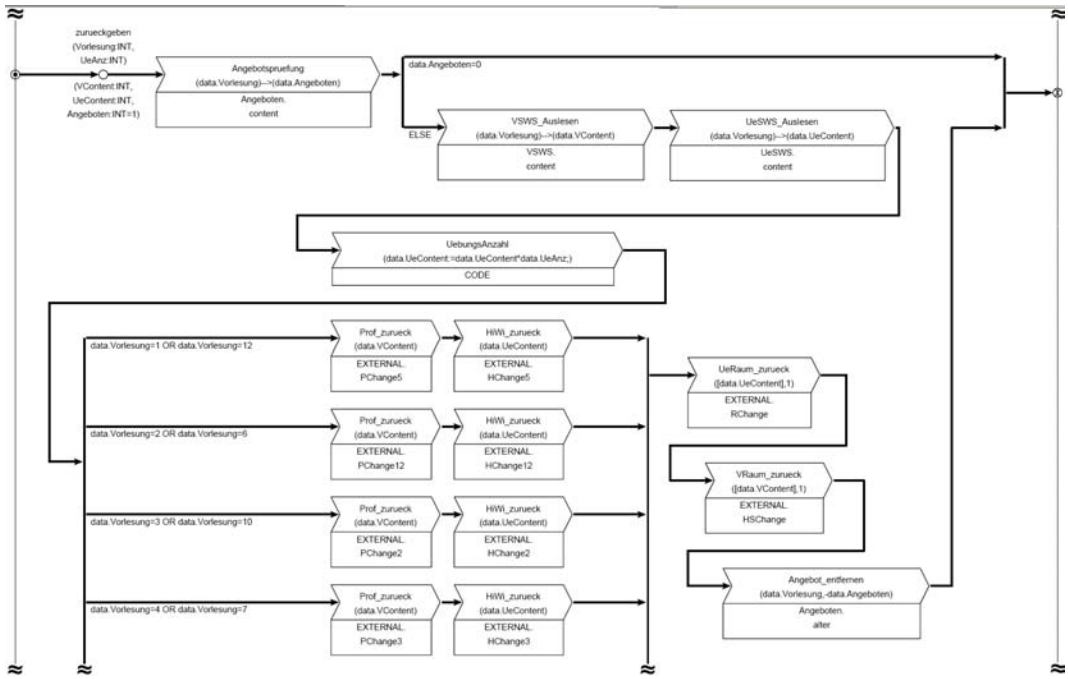


Abbildung 5.35: Funktionseinheit Vorlesung - Prozesskette - Teil2

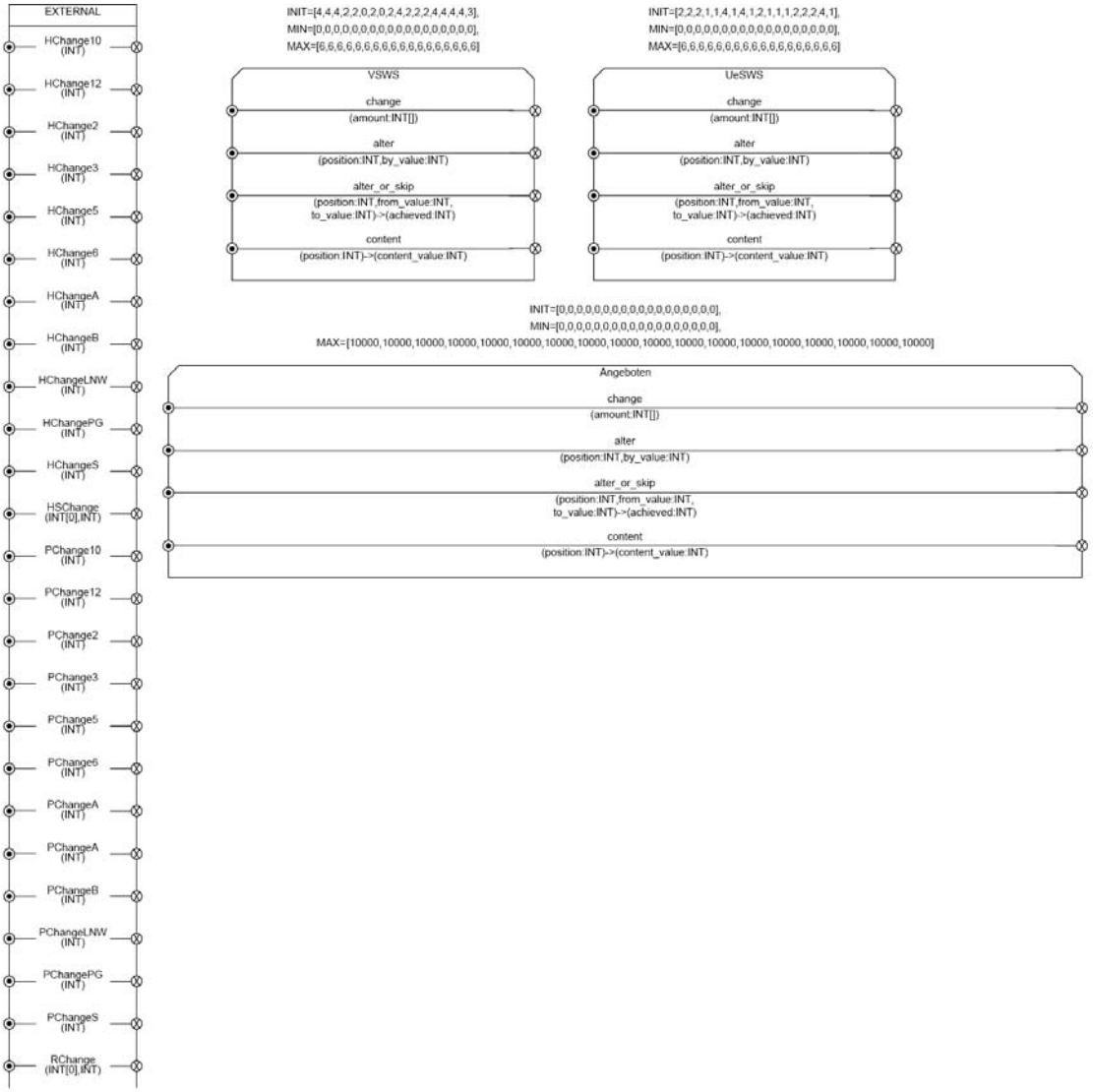


Abbildung 5.36: Funktionseinheit Vorlesung - Ressourcen

5.6.12 Vorlesung (in ExterneFBs)

Die Abläufe aller externen Vorlesungen werden in der Funktionseinheit *Vorlesung* (in ExterneFBs) modelliert. Diese besteht aus drei Prozessketten (siehe Abb. 5.38).
Die Prozesskette *hoeren* enthält die Prozesskettenelemente *Angebot* und *hoeren*. Das Prozesskettenelement *Angebot* verringert, mit dem Aufruf der Funktion *alter* des Storages *Angeboten*, den Wert an der entsprechenden Position um eine Einheit. Das entspricht einem Studenten der diese Vorlesung hört. Das Prozesskettenelement *hoeren* ist eine Verzögerung um *SemWochen*, was der Dauer der Vorlesung entspricht. Als Parameter erhält die Prozesskette *hoeren* die Variable *Vorlesung*. Damit wird angegeben um welche Vorlesung es sich handelt.

Die Prozesskette *anbieten* besteht aus acht Prozesskettenelementen, die nacheinander durchlaufen werden. Über die Parameter wird angegeben um welche Vorlesung es sich handelt und wie viele

SWS an Übungen notwendig sind. Die Prozesskettenelemente *VSWS_Auslesen* und *UeSWS_Auslesen* lesen aus der Funktionseinheit *VSWS* bzw. *UeSWS* die noch vorhandenen SWS aus. Über die weiteren Prozesskettenelemente werden über externe Funktionseinheiten nacheinander die Professoren, HiWis, Übungsräume und Hörsäle angefordert. Anschließend werden im Prozesskettenelement *Angebot_setzen* die Werte am Storage *Angeboten* an der Position *Vorlesung* anhand des aktuellen Parameterwertes angepasst.
Die Prozesskette *zurueckgeben* besteht aus neun Prozesskettenelementen und einem ODER-Konnektor. Über die Parameter wird der Prozesskette die Vorlesung und die Übungsanzahl übergeben. Zunächst wird im Prozesskettenelement *Angebotspruefung* über den Dienst *content* der Wert aus dem Storage *Angeboten* an der entsprechenden Position ausgelesen. Ist dieser Wert gleich Null, wird der Prozess beendet, ansonsten wird zunächst in den Prozesskettenelementen *VSWS_Auslesen* und *UeSWS_Auslesen* die vorhandenen Vorlesungs-Semesterwochenstunden bzw. die Übungs-Semesterwochenstunden ausgelesen. Anschließend werden über die Prozesskettenelemente *Prof_zurueck*, *HiWi_zurueck*, *UeRaum_zurueck* und *VRaum_zurueck* die reservierten Ressourcen wieder freigegeben. Im letzten Prozesskettenelement wird die Vorlesung als „nicht angeboten“ markiert, was über den Aufruf des Dienstes *alter* der Funktionseinheit *Angeboten* realisiert wird.
Als Ressourcen besitzt diese Funktionseinheit *VSWS*, *UeSWS* und *Angeboten*, die alle als Storage modelliert sind. Diese sind in Abbildung 5.37 dargestellt.

Codeelemente der Funktionseinheit *Vorlesung (in ExterneFBs*:

```
CodeElement UebungsAnzahl_Vorzeichen
  data.UeContent := data.UeContent*dataUeAnz;
  data.UeContent := -data.UeContent;
  data.VContent := -data.VContent

CodeElement UebungsAnzahl
  data.UeContent := data.UeContent*data.UeAnz;
```

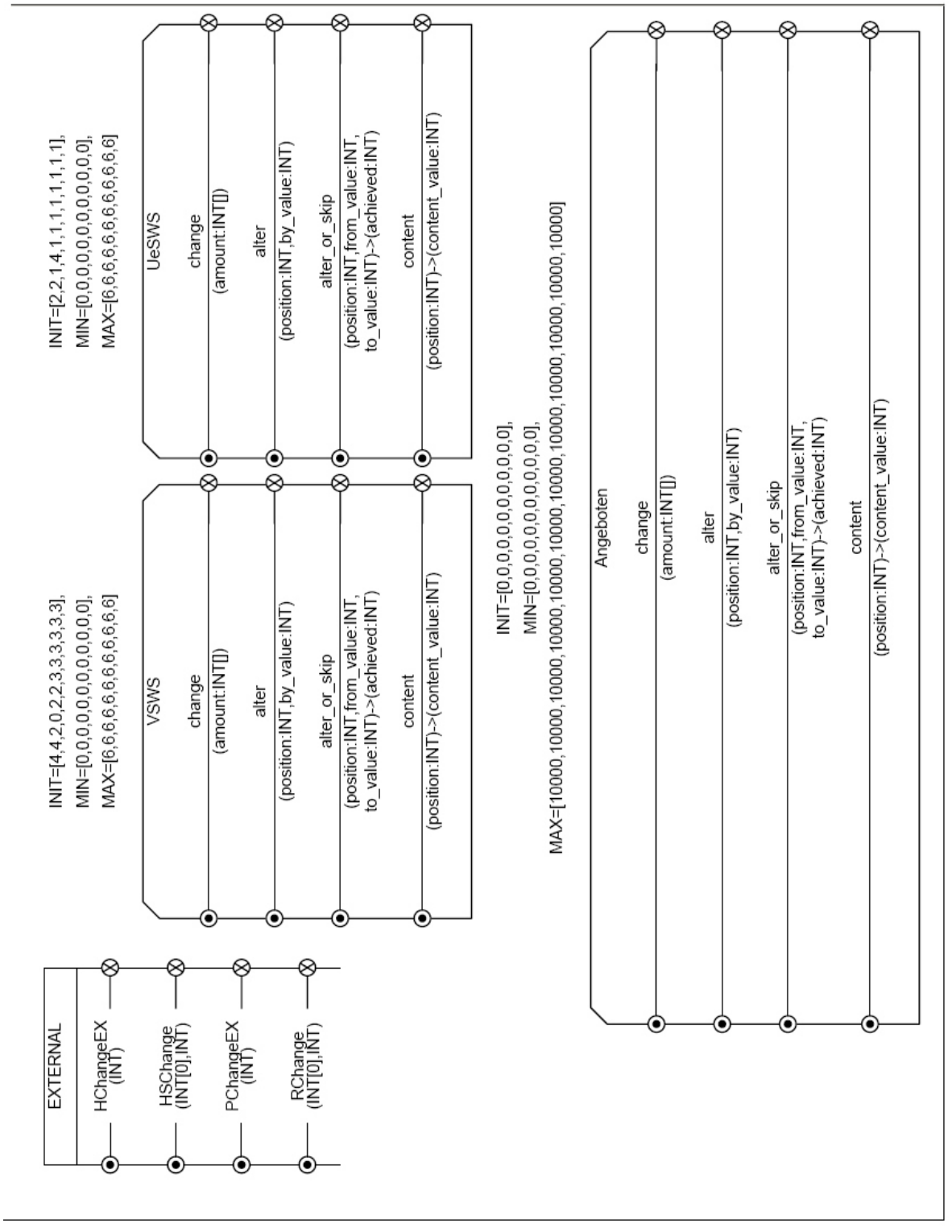


Abbildung 5.37: Funktionseinheit Vorlesung (in ExterneFBs) - Ressourcen

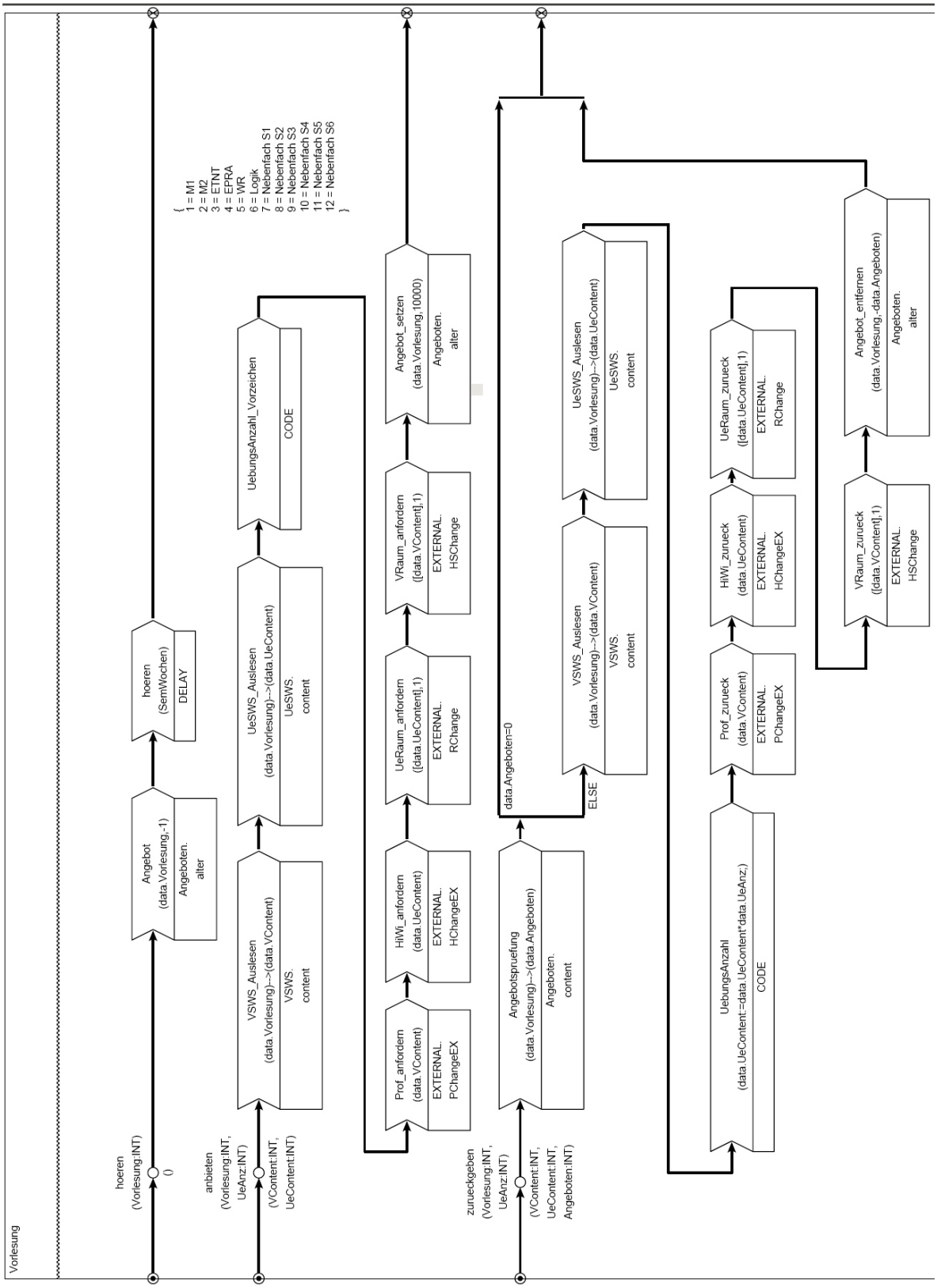


Abbildung 5.38: Funktionseinheit Vorlesung (in ExterneFBs) - Prozesskette

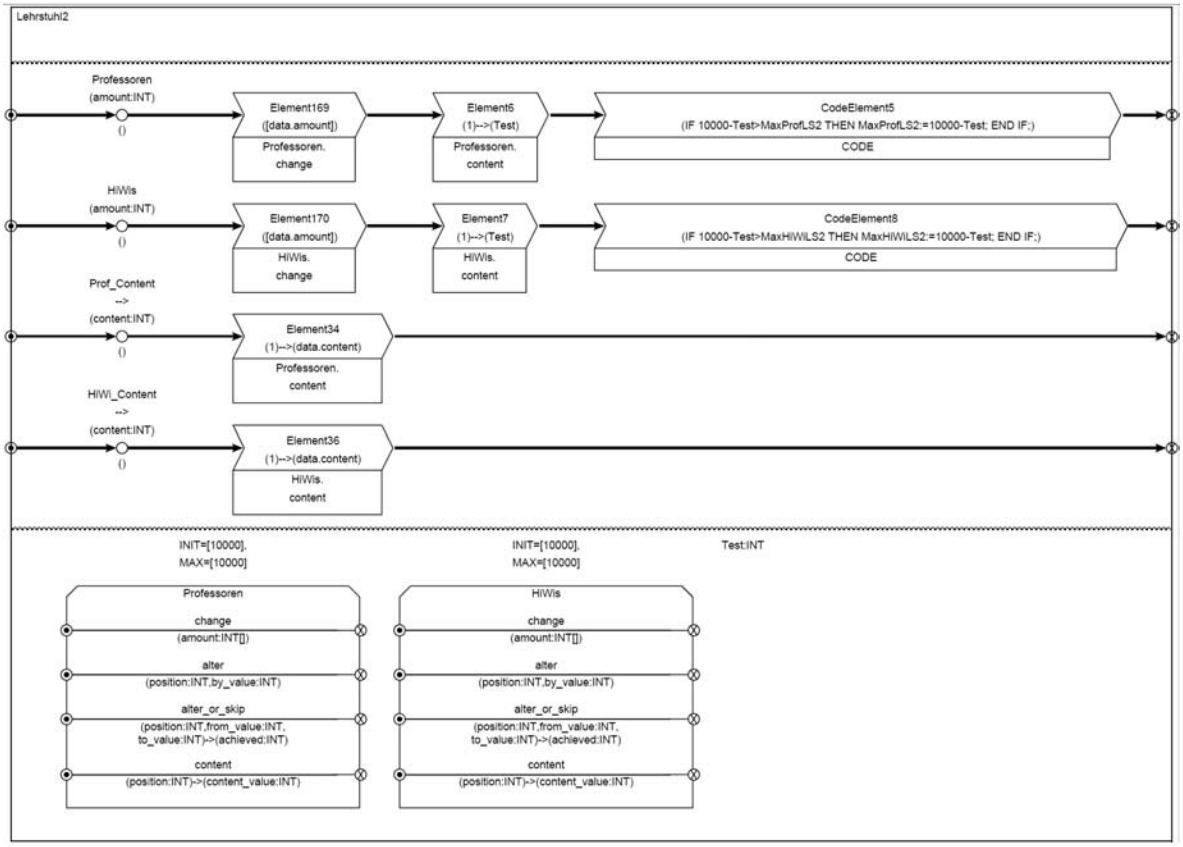


Abbildung 5.39: Funktionseinheit Lehrstuhl2

5.6.13 Lehrstuhl2

Insgesamt werden zwölf Lehrstühle modelliert. Da in dem Modell nicht alle am Fachbereich Informatik real existierende Lehrstühle benötigt werden, werden nur folgende realen Lehrstühle modelliert: *Lehrstuhl2*, *Lehrstuhl3*, *Lehrstuhl5*, *Lehrstuhl6*, *Lehrstuhl10*, *Lehrstuhl12*.
Die weiteren Lehrstühle sind speziell erzeugte Dummy-Lehrstühle. *LehrstuhlA* und *LehrstuhlB* stellen Ressourcen für die Vorlesungen aus dem Katalog A bzw. Katalog B zur Verfügung, die in den Semestern 5 und 6 benötigt werden. Analog für Leistungsnachweise (*LehrstuhlLNW*) und Schwerpunktgebiet (*LehrstuhlS*). Die Projektgruppe wird als eigener Lehrstuhl (*LehrstuhlPG*) modelliert. Die Modellierung dieser Dummy-Lehrstühle ist notwendig, da bei einigen der Veranstaltungen sich die Ressourcenanforderungen auf verschiedene reale Lehrstühlen verteilen können. Daher wurden die Anforderungen dieser Veranstaltungen an den Dummy-Lehrstühlen gebündelt. *LehrstuhlEX* stellt alle externen Ressourcen zur Verfügung, also Professoren und HiWis, die nicht am Fachbereich Informatik tätig sind.
Da alle Lehrstühle den gleichen internen Aufbau haben, wird hier exemplarisch nur ein Lehrstuhl beschrieben (siehe Abbildung 5.39).
Die Funktionseinheit *Lehrstuhl2* enthält vier Prozessketten. Mit Hilfe der Prozessketten *Professoren* und *HiWis* wird die Anzahl aktuell verfügbaren der Professoren bzw. der HiWis des Lehrstuhls in Abhängigkeit des Parameters *amount* erhöht bzw. vermindert. Diese werden in den Storages *Professoren* bzw. *HiWis* als SWS angegeben und verwaltet. Zusätzlich werden in beiden Prozessketten noch die

Werte für die maximal gleichzeitig angeforderten Kapazitäten der jeweiligen Ressource berechnet. Im Prozesskettenelement *Element6* wird der Wert an Position 1 des Storages *Professoren* ausgelesen und in der lokalen Variable *Test* gespeichert. Im folgenden Code-Element *CodeElement5* wird dann von den maximal zur Verfügung stehenden Ressourcen (hier 10000 SWS) der Betrag der Variable *Test* subtrahiert. Daraus ergibt sich der aktuelle Verbrauch an Ressourcen. Wenn dieser Wert nun größer ist als das gespeichert Maximum in der Variablen *MaxProfLS2*, dann wird dieser Wert als neues Maximum gespeichert.
Die Prozessketten *Prof_Content* und *HiWi_Content* dienen nur der Ausgabe. Hier werden in den Prozesskettenelementen *Element34* bzw. *Element36* die jeweiligen Werte aus den Storages ausgelesen und als Parameter übergeben.
Als Ressourcen stehen der Funktionseinheit die beiden Storages *Professoren* und *HiWis* zur Verfügung, in denen die verfügbaren Kapazitäten an Professoren und HiWis in der Größe SWS gespeichert werden.

Codeelemente der Funktionseinheit *Lehrstuhl2*:

```
CodeElement CodeElement5
  IF 10000-Test>MaxProfLS2 THEN MaxProfLS2:=10000-Test; END IF;

CodeElement CodeElement8
  IF 10000-Test>MaxHiWiLS2 THEN MaxHiWiLS2:=10000-Test; END IF;
```

5.6.14 GS

Hier wird der Ablauf für das Grundstudium spezifiziert (siehe Abb. 5.40). Nacheinander durchläuft der Prozess die Elemente *S1, S2, S3, S4* die jeweils mit Hilfe der entsprechenden Funktionseinheiten *Sem1, Sem2, Sem3, Sem4* die Abläufe für die einzelnen Semester des Grundstudiums modellieren. Die vier mit *Vorlesungsfrei* bezeichneten DELAY-Elemente dienen dazu die zwei vorlesungsfreien Wochen zu Semesterbeginn zu modellieren, damit letztendlich ein Semester auch die vollen 26 Wochen dauert. Die Abweichung des Delay-Wertes vor Beginn des ersten Semesters erklärt sich dadurch, da in der Funktionseinheit *Bestcase* bereits ein Verzögerungselement mit dem Wert 0,5 vor dem Beginn des ersten Semesters benötigt wird. Dadurch ergibt sich folglich an dieser Stelle ein Wert Verzögerungswert von 1,5. Durch die Verzögerung speziell vor dem ersten Semester wird sichergestellt, dass die Angebotserstellung vor dem Aufruf der eigentlichen Studienabläufe erfolgt.

5.6.15 HS

Die Prozesskette *Ablauf* der Funktionseinheit *HS* (siehe Abb. 5.41) besteht insgesamt aus zehn Prozesskettenelementen. Diese gliedern sich in DELAY-Elemente, die die vorlesungsfreie Zeit zu Beginn des Semesters repräsentieren, und den übrigen Elementen, die mit Hilfe der entsprechenden eingebundenen Funktionseinheiten die Abläufe der einzelnen Semester modellieren.
Die eingebundenen Funktionseinheiten sind *Sem5*, *Sem6*, *Sem7*, *Sem8*, *Semester9*. Jeder Prozess bekommt zu Beginn in dieser Funktionseinheit zwei Variablen zugewiesen (*Wahlpflicht1*, *Wahlpflicht2*), in denen die Zugehörigkeit der beiden im fünften Semester gewählten Wahlpflichtvorlesungen gespeichert wird. Diese Datenhaltung ist notwendig, um bestimmen zu können aus welchem Katalog die dritte Pflichtveranstaltung in Semester 6 stammen muss. Dies stellt sicher, dass die Forderung nach mindestens einer Vorlesung aus jedem Katalog erfüllt wird.

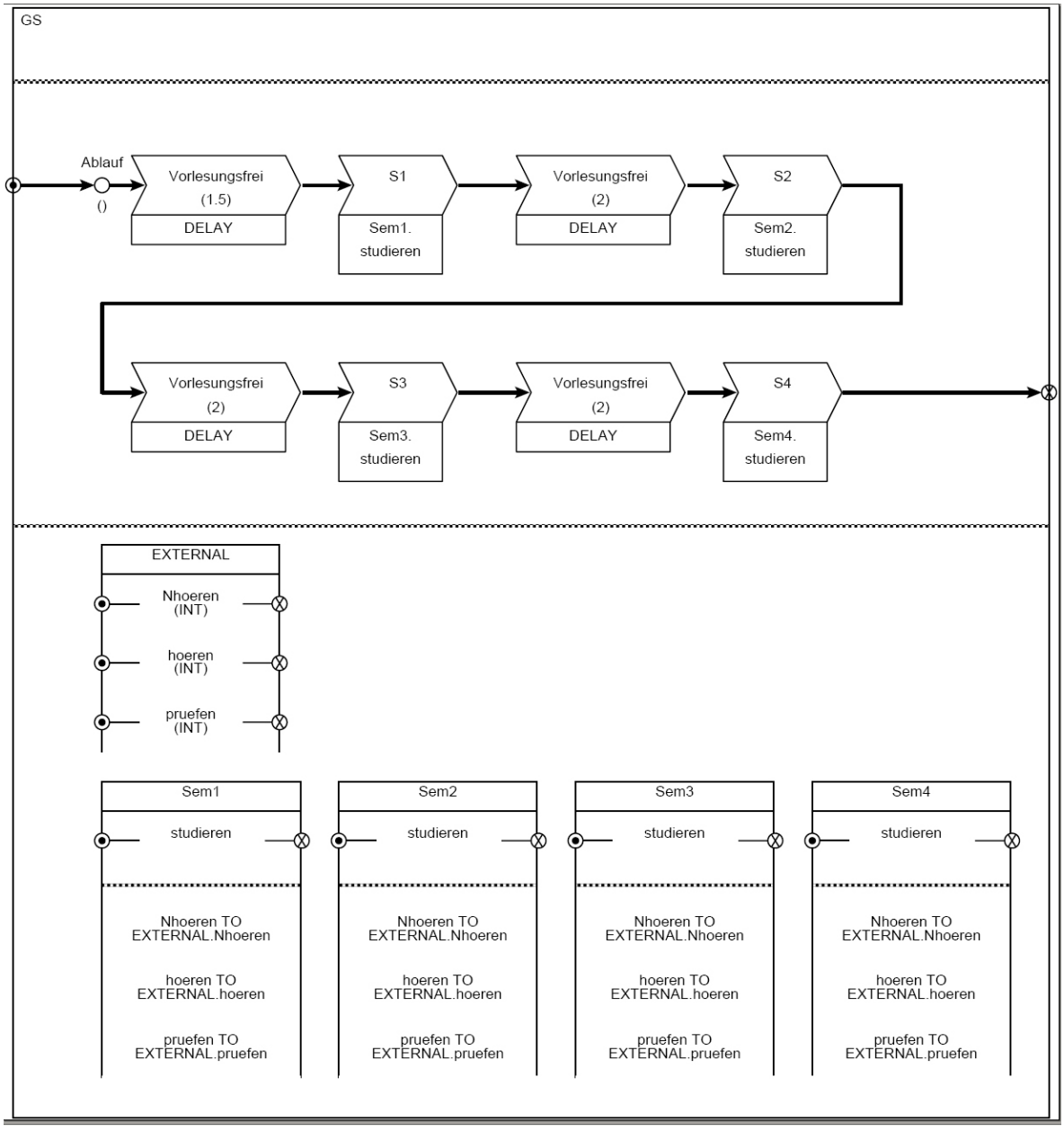


Abbildung 5.40: Funktionseinheit GS

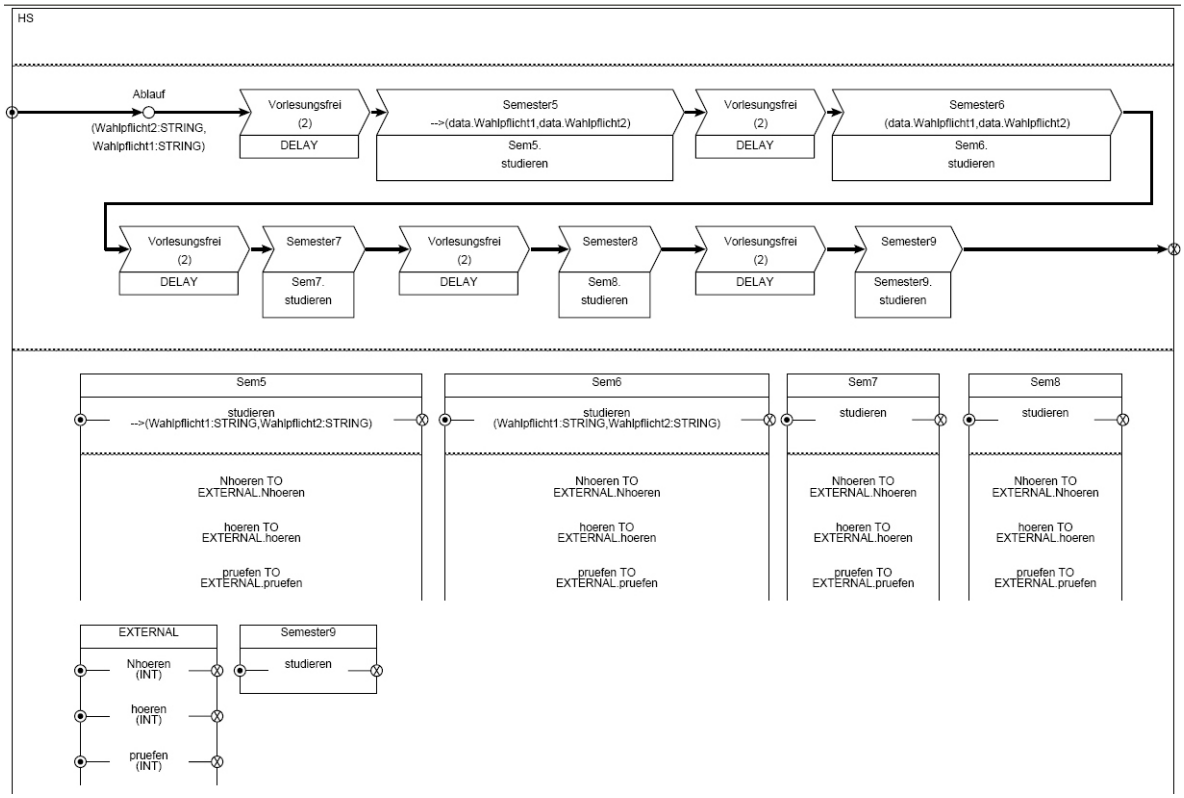


Abbildung 5.41: Funktionseinheit HS

5.6.16 Sem1

Die Funktionseinheiten *Sem1* bis *Sem8* und *Semester9* bestehen jeweils aus einer Prozesskette *studieren*.
In der Funktionseinheit *Sem1* (siehe Abb. 5.42) befinden sich die festgelegten Pflichtveranstaltungen des ersten Semesters. Im ersten Semester sind dies laut DPO (Diplomprüfungsordnung) die Vorlesungen *DAP1* (Datenstrukturen, Algorithmen und Programmierung 1), *RS* (Rechenstrukturen), *M1* (Lineare Algebra und Analysis) und *Nebenfach* (eine Vorlesung des jeweils gewählten Nebenfaches). Die ersten beiden Veranstaltungen rufen den externen Dienst *hoeren* der Funktionseinheit *Vorlesung* auf, da es sich bei den beiden Vorlesungen um Vorlesungen des Fachbereiches Informatik handelt. Die Vorlesung *M1* und *Nebenfach* benötigen Ressourcen von fremden, externen Fachbereichen und rufen daher den Dienst *Nhoeren* auf, über den schließlich der eigentliche Dienst *hoeren* der Funktionseinheit *ExterneFBs* aufgerufen wird. Die Parameter der Prozesskettenelemente dienen dazu die Veranstaltungen im weiteren Simulationsablauf eindeutig identifizieren zu können. Die Zuordnung erfolgt separat für interne und externe Vorlesungen. Diese Nummerierung der Vorlesungen ist notwendig, damit in den Vorlesungs- und Prüfungsprozessketten die jeweiligen Ressourcen vom richtigen Lehrstuhl angefordert werden können. Diese Nummerierung findet innerhalb jedes Semesters statt, es wird in der Dokumentation der weiteren Semester daher auf eine erneute Erklärung des Parameters verzichtet. Das DELAY-Element sichert den Verbleib der Prozesse für die Dauer des Semesters in selbigem; es bildet eine Sicherheitsfunktion.

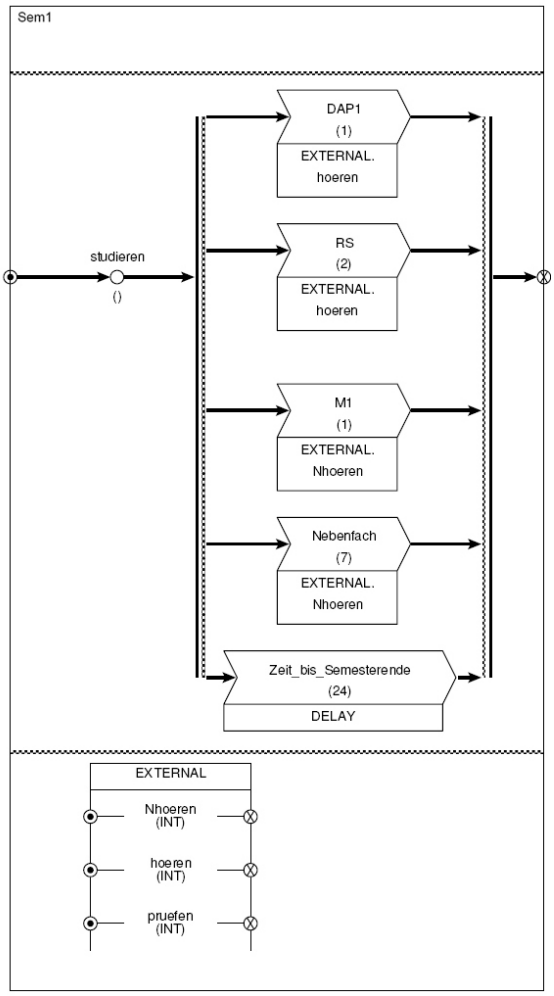


Abbildung 5.42: Funktionseinheit Sem1

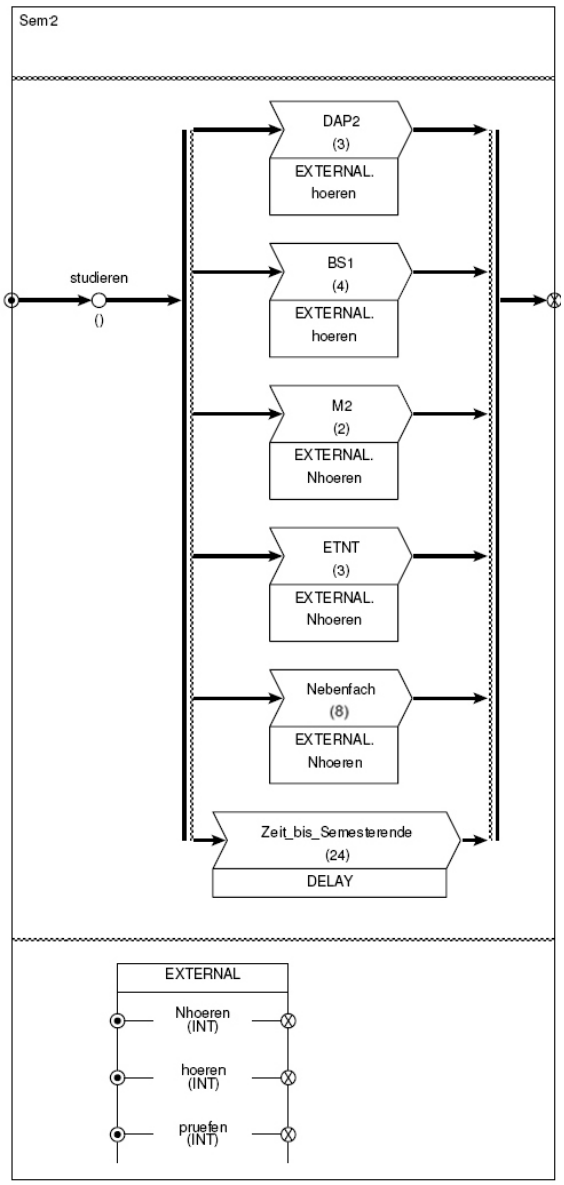


Abbildung 5.43: Funktionseinheit Sem2

5.6.17 Sem2

Im zweiten Semester (siehe Abb. 5.43) sind folgende Vorlesungen für den Studenten vorgesehen: *DAP2* (Datenstrukturen, Algorithmen und Programmierung 2), *BS1* (Betriebssysteme, Rechnernetze und verteilte Systeme 1), *M2* (Diskrete Strukturen und Algebra), *ETNT* (Elektrotechnik und Nachrichtentechnik) und *Nebenfach* (eine Vorlesung des jeweils gewählten Nebenfaches). Vom Aufbau und den Dienstaufrufen verhält sich die Funktionseinheit genauso wie die des ersten Semesters, von daher wird aus Gründen der Analogie hier auf weitere Beschreibungen verzichtet.

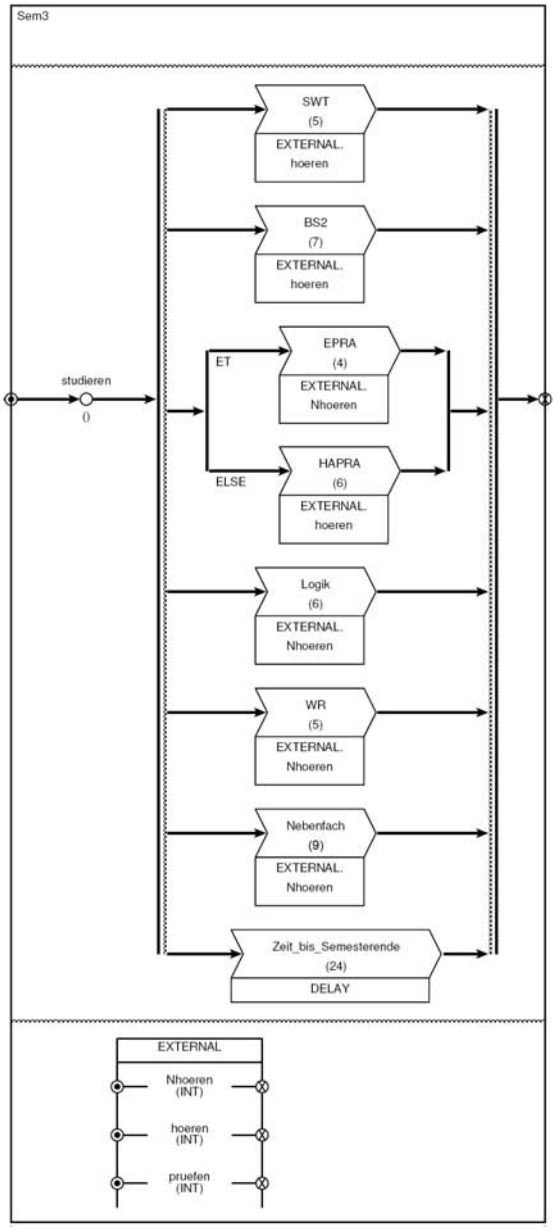


Abbildung 5.44: Funktionseinheit Sem3

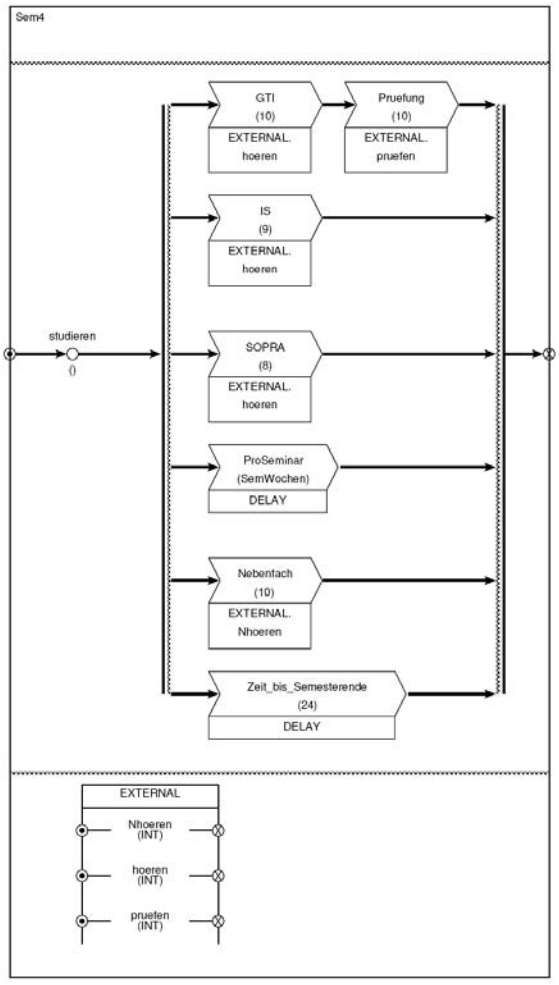


Abbildung 5.45: Funktionseinheit Sem4

5.6.18 Sem3

Im dritten Semester (siehe Abb. 5.44) sind folgende Vorlesungen für den Studenten vorgesehen: *SWT* (Softwaretechnik), *BS2* (Betriebssysteme 2), *Logik*, *WR* (Wahrscheinlichkeitsrechnung), *Nebenfach* (eine Vorlesung des jeweils gewählten Nebenfaches). Außerdem ist in Abhängigkeit des gewählten Nebenfaches ein Praktikum als Pflichtveranstaltung vorgesehen. Dazu wird hier mittels eines ODER-Konnektors das Nebenfach *ET* (Elektrotechnik) gesondert betrachtet, weil in diesem Fall der Student das *EPRA* (Digitalelektronisches Praktikum) besuchen muss. Alle anderen Informatikstudenten müs-

sen am *HAPRA* (Hardware-Praktikum) teilnehmen. Die restlichen Strukturen entsprechen den übrigen Semester-Funktionseinheiten.

5.6.19 Sem4

Im vierten Semester (siehe Abb. 5.45) sind folgende Vorlesungen für die Studenten vorgesehen: *GTI* (Grundbegriffe der theoretischen Informatik), *IS* (Informationssysteme), *SOPRA* (Software - Praktikum), ein *Proseminar* und das *Nebenfach* (eine Vorlesung des jeweils gewählten Nebenfaches). Da GTI im Gegensatz zu allen bisherigen Vorlesungen mit einer mündlichen Prüfung abgeschlossen wird, benötigt man an dieser Stelle nun erstmals den extern eingebundenen Dienst *pruefen* der Funktionseinheit *muendl_Pruefung*. Der Rest entspricht wieder dem bereits bekannten Ablauf.

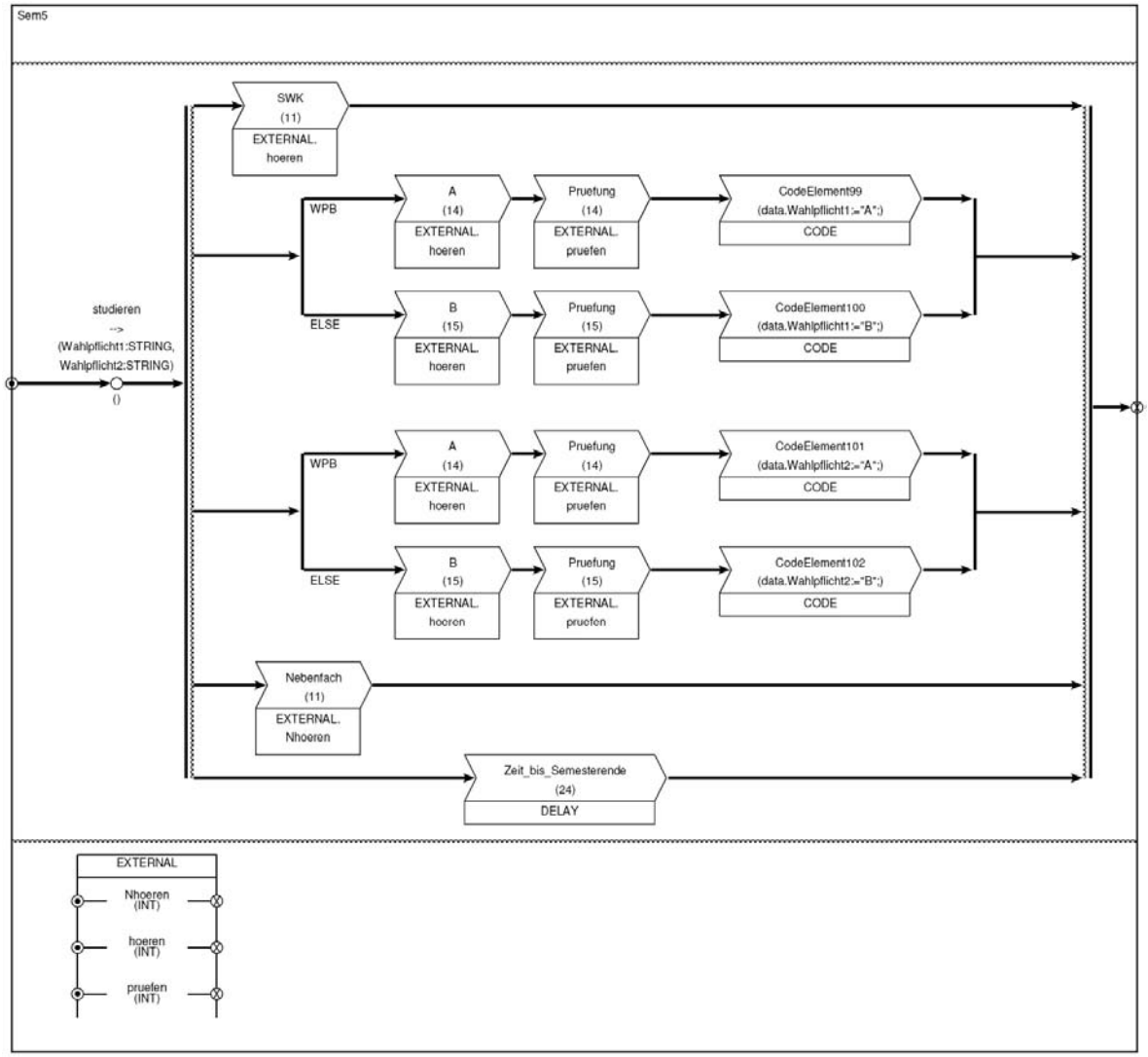


Abbildung 5.46: Funktionseinheit Sem5

5.6.20 Sem5

Im fünften Semester (siehe Abb. 5.46) sind folgende Vorlesungen für die Studenten vorgesehen: eine Pflicht-Lehrveranstaltung *SWK* (Softwarekonstruktion), zwei Wahlpflicht-Lehrveranstaltungen aus den Katalogen A und B (diese können in diesem Semester beliebig kombiniert werden) und *Nebenfach* (eine Vorlesung des jeweils gewählten Nebenfaches). Da zwei Vorlesungen aus dem Wahlpflichtbereich frei gewählt werden können, wird hier jeweils mit einem ODER-Konnektor modelliert. An diesem wird die, in der globalen Variable *WPB* abgelegte, Wahrscheinlichkeit mit der der Student eine Wahlpflichtvorlesung aus dem *Wahlpflichtkatalog A* wählt ausgewertet und die Prozesse entsprechend verteilt. Jede dieser gewählten Vorlesungen wird mit einer Prüfung beendet, daher wird auf den externen Dienst *pruefen* der Funktionseinheit *muendl_Pruefung* zugegriffen. Anschließend wird in den Code-Elementen *CodeElement199*, *CodeElement100*, *CodeElement101*, *CodeElement102* die entsprechende Wahl gespeichert um im sechsten Semester die ausstehende Veranstaltung bestimmen zu können.

5.6.21 Sem6

Im sechsten Semester (siehe Abb. 5.47) sind folgende Vorlesungen für den Studenten vorgesehen: die Pflicht-Lehrveranstaltungen *IuG* (Informatik und Gesellschaft) und *CBau* (Übersetzerbau) sowie eine dritte Vorlesung aus dem Wahlpflichtbereich vorgesehen. Hier ist es aber wichtig aus welchem Wahlpflichtkatalog die beiden Vorlesungen, die bereits in Semester 5 gehört wurden, stammen. Waren dies beides Vorlesungen aus dem *Katalog A*, wird eine Vorlesung aus *Katalog B* noch benötigt. Stammen beide Vorlesung in Semester 5 aus *Katalog B*, muss nun eine Vorlesung aus *Katalog A* gewählt werden. Wurde in Semester 5 jeweils eine Vorlesung aus je einem der beiden Kataloge gewählt, kann nun noch einmal frei entschieden werden. Die Realisierung dieser Gegebenheiten erfolgt über den ODER-Konnektor. An ihm werden die Auswertungen für die beiden Variablen *Wahlpflicht1* und *Wahlpflicht2* gemacht. Enthalten beide Variablen den Wert *B*, so wird der oberste Pfad gewählt (der Student muss noch eine Vorlesung aus Katalog A hören), enthalten beide Variablen den Wert *A*, so wird der mittlere Pfad ausgewählt (der Student muss noch eine Vorlesung aus Katalog B hören), in allen anderen Fällen wird der dritte Pfad ausgewählt und der Student wählt, wie in Semester 5, frei eine Vorlesung aus den Katalogen A und B. Hier liegen wiederum die gleichen Wahrscheinlichkeiten für eine Auswahl zugrunde, wie in Semester 5. Zusätzlich muss noch eine Veranstaltung des gewählten Nebenfaches gehört werden. Die Vorlesung aus dem Wahlpflichtbereich muss wieder mit einer mündlichen Prüfung abgeschlossen werden. Zu diesem Zweck wird wieder der Dienst *pruefen* der Funktionseinheit *muendl_Pruefung* eingebunden und von den Prozesskettenelementen *Pruefung* genutzt.

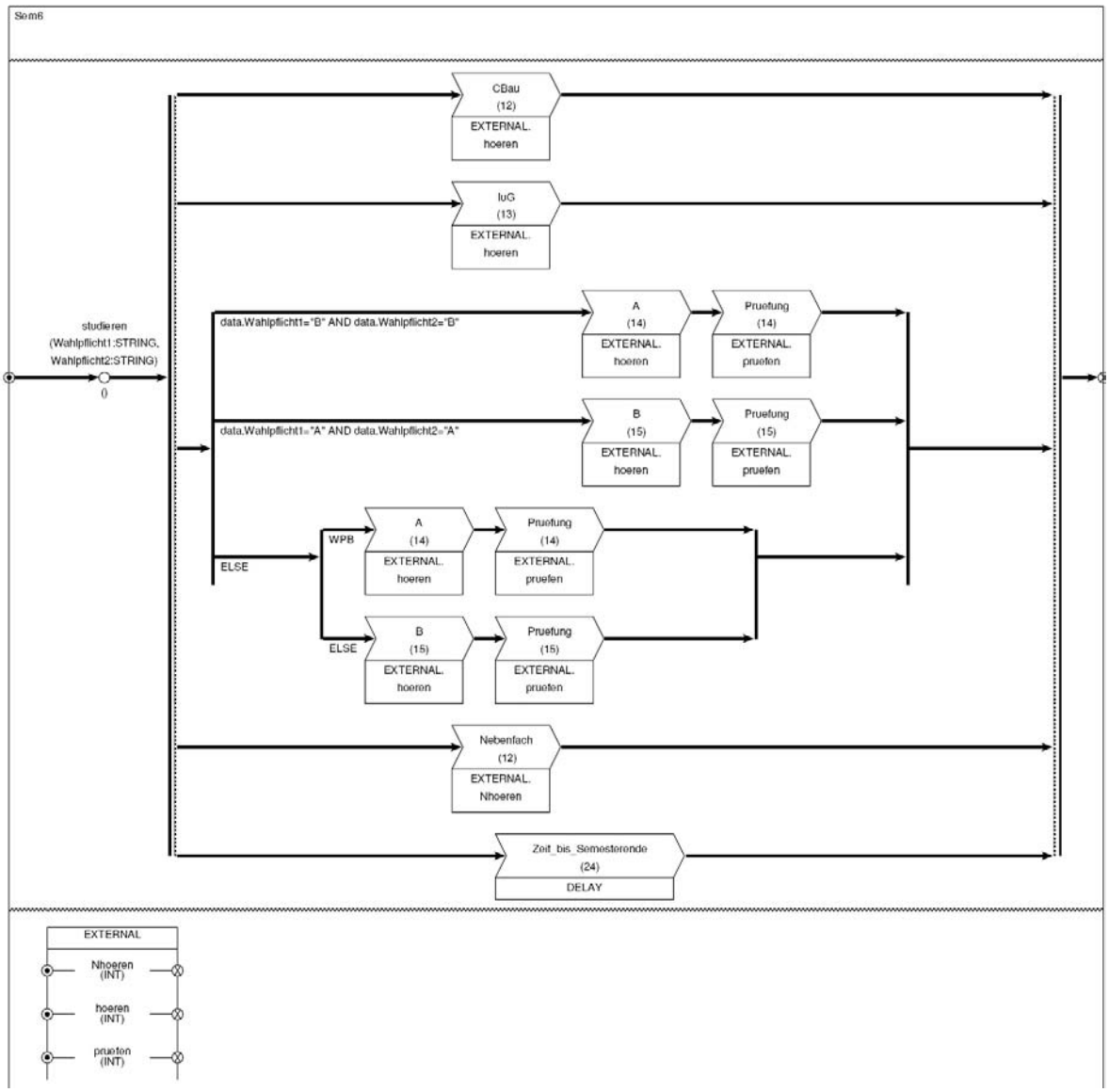


Abbildung 5.47: Funktionseinheit Sem6

5.6.22 Sem7

Im siebten Semester (siehe Abb. 5.48) sind folgende Vorlesungen für den Studenten vorgesehen: der erste Teil der *PG* (Projektgruppe), die sich insgesamt über zwei Semester erstreckt, ein Seminar und zwei Veranstaltungen aus dem Schwerpunktgebiet. Da die kompletten Anforderungen der Schwerpunktgebiete alle von einem extra dafür eingefügten Dummy-Lehrstuhl angefordert werden, sind hier die beiden Vorlesungen in einem Prozesskettenelement zusammengefasst. Weil für jede dieser beiden Vorlesungen je eine Prüfung erforderlich ist, wird zweimal ein Element *Pruefung* in die Prozesskette eingefügt, welches den externen Dienst *pruefen* der Funktionseinheit *muendl_Pruefung* nutzt. Zusätzlich müssen in diesem Semester auch noch Leistungen in *LNWs* (Leistungsnachweise) erbracht werden.

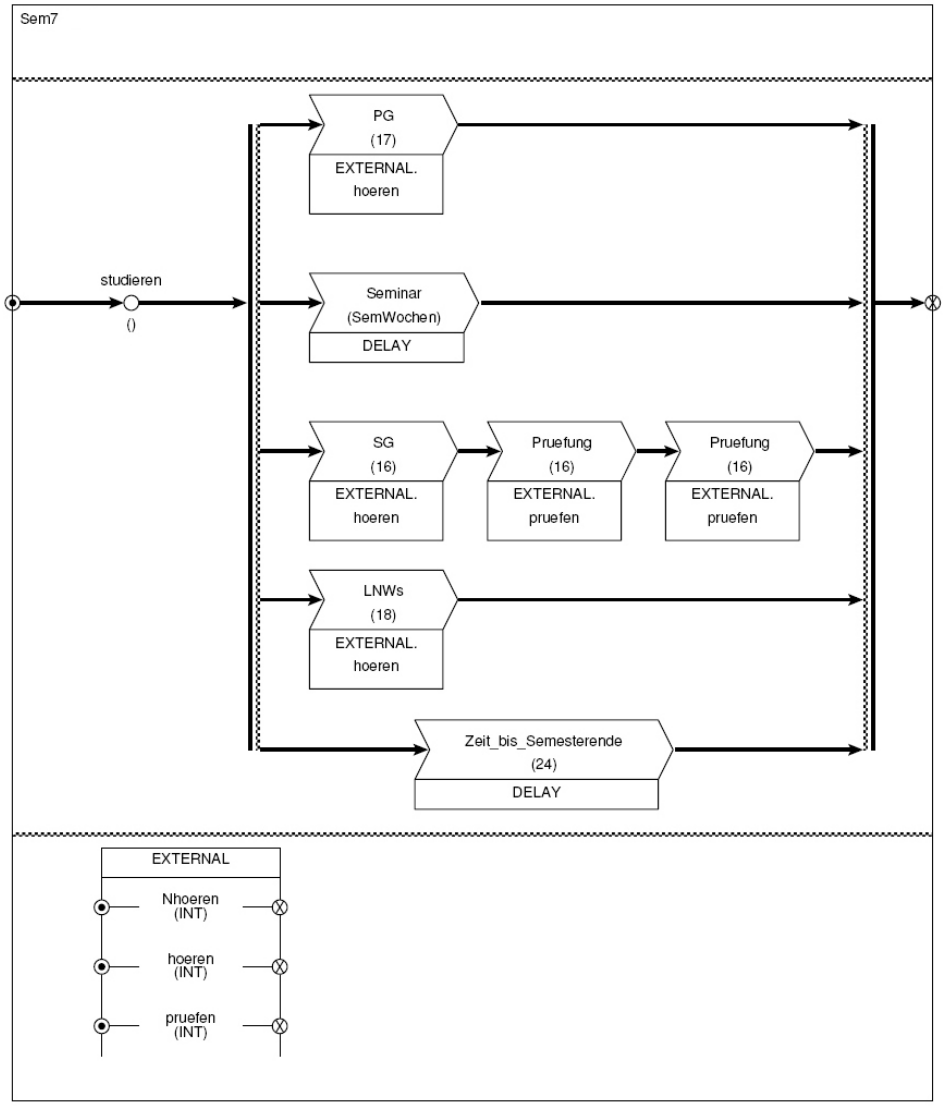


Abbildung 5.48: Funktionseinheit Sem7

5.6.23 Sem8

Im achten Semester (siehe Abb. 5.49) sind folgende Vorlesungen für den Studenten vorgesehen: Der zweite Teil der *PG* (Projektgruppe) und zwei Veranstaltungen aus dem Schwerpunktgebiet. Diese werden analog zu den beiden Vorlesungen aus dem Schwerpunktgebiet in Semester 7 realisiert. Zusätzlich müssen auch in diesem Semester noch einige Leistungsnachweise erbracht werden.

5.6.24 Semester9

Dieses Semester ist für die Diplomarbeit vorgesehen. Dies wird lediglich als Verzögerung über 26 Wochen, was einem Semester entspricht, modelliert, da im Idealfall bis zu diesem Zeitpunkt alle anderen Leistungen erbracht worden sind.

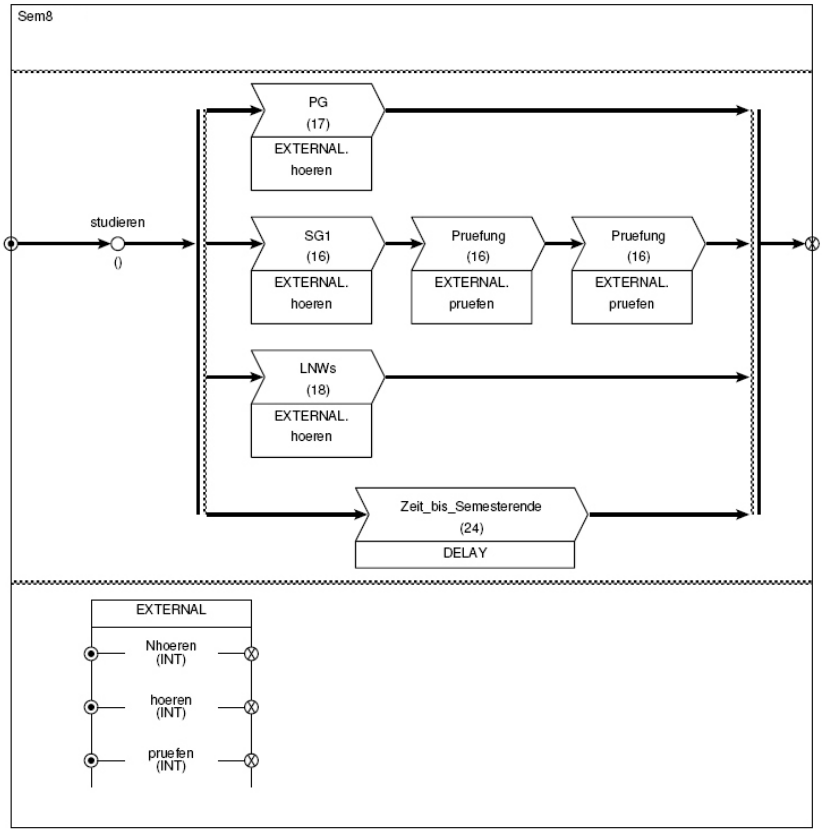


Abbildung 5.49: Funktionseinheit Sem8

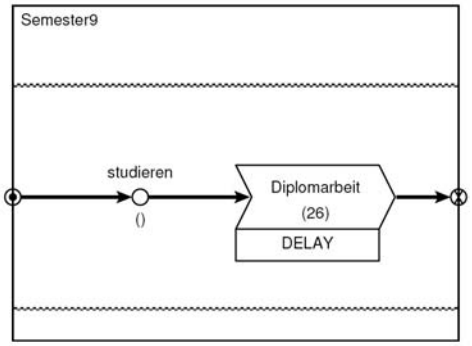


Abbildung 5.50: Funktionseinheit Semester9

5.6.25 Verifikation und Abbruch der ProC/B Analyse

Bei der Verifikation des ProC/B-Modells stellte sich leider heraus, dass es nicht fehlerfrei ist. Die Ausgabewerte der Population- und der einzelnen Turnaroundmesspunkte wichen zum Teil sehr stark von den Sollwerten ab. Zwischenzeitliche Tests zur Verifikation waren aufgrund verschiedener Fehler im Simulationstool ProC/B leider unmöglich. Eine Zusammenfassung der Tool-Fehler findet sich in Kapitel 5.9.

Aufgrund dieser Fehler im ProC/B-Tool konnte lange Zeit keine Simulation durchgeführt werden, um eventuelle Modellierungsfehler zu entdecken. Mit dreiwöchiger Verzögerung wurde das Modell nur teilweise getestet.
Mit kleineren Laufzeiten lief das Simulationsmodell einwandfrei und lieferte vernünftige Ergebnisse, jedoch mit kleineren Abweichungen, die im akzeptablen Bereich lagen. Diese Abweichungen hatten keine großen Auswirkungen auf die Simulationsergebnisse, was auch durch mehrere Durchläufe mit kleineren Laufzeiten bestätigt wurde. Für größere Laufzeiten musste die Simulation vorzeitig abgebrochen werden, da diese auch nach 27 Stunden immer noch im ersten Durchlauf war. Mittels der Zwischenergebnisse, die in der Konsole angezeigt wurden, konnte erkannt werden, dass die Simulationsergebnisse von den erwarteten Ergebnissen gewaltig abwichen. Da diese Abweichungen nicht am ProC/B-Tool lagen, wurde davon ausgegangen, dass der Fehler irgendwo in dem Modell selbst zu finden war. Wo genau dieser ausschlaggebende Fehler war, konnte in der kurzen Simulationszeit nicht lokalisiert, analysiert und behoben werden. Die Ergebnisanalyse des ProC/B-Modells konnte aufgrund der fehlenden Ergebnisse nicht durchgeführt werden.

5.7 Modellvalidierung

Bevor das Modell validiert werden kann, müssten gemäß den zehn Schritten einer Simulationsstudie Pilotläufe durchgeführt werden. Da das HIT-Modell sehr schnell implementiert worden ist, wurden keine Pilotläufe vorgenommen, stattdessen wurden für die Validierung die Ergebnisse der Produktionsläufe (siehe Abschnitt 5.8) zu Rate gezogen. Nun folgt die Vorgehensweise der ursprünglich angedachten Modellvalidierung.
Modellvalidierung sind alle Aktivitäten, die untersuchen, ob das richtige Modell entwickelt wurde. Dabei wird überprüft, ob das reale System in geeigneter Weise idealisiert und abstrahiert wurde. Die Validierung vergleicht das Modell mit dem realen System und untersucht, ob das erstellte Modell für den Anwendungszweck geeignet ist. Die Validierung kann erst durchgeführt werden, wenn die Basis dafür gelegt ist. Die Basis besteht aus:

- einem klar strukturierten Entwicklungsprozess des Modells, der Maßgaben für die Erstellung von Ergebnissen der jeweiligen Phase vorgibt,
- den detaillierten Kenntnissen des realen Systems und klaren Untersuchungszweck,
- der ausführlichen Dokumentation aller dieser Ergebnisse und der durchgeführten Tätigkeiten, die Vorraussetzung für eine entwicklungsbegleitende Validierung ist. Die Validierung muss entwicklungsbegleitend erfolgen, um die Eignung eines Modells frühzeitig zu erkennen oder zu widerlegen.

Während des gesamten Semesters wurde beim Entwicklungsprozess immer strukturiert vorgegangen. Zunächst hat die Projektgruppe ein Untersuchungsziel festgelegt, das erreicht werden soll (siehe Unterkapitel 5.1) und dafür benötigte Information gesammelt (siehe 5.2.2). Vor der eigentlichen Erzeugung des Modells wurde eine klare Ablaufstruktur entwickelt, die nur das nötigste enthielt (siehe dazu 5.2.1). Das reelle System wurde sehr detailliert betrachtet, dazu wurde die DPO 2001 und die bisherigen Erfahrungen der Projektgruppenteilnehmer zu Rate gezogen, so dass die erzeugte Struktur so gut wie möglich den realen Ablauf des Informatikstudiums widerspiegelt. Die Arbeit am Modell wurde ausführlich dokumentiert, so dass sie nachvollziehbar ist. Bei der Erstellung des eigentlichen Simulationsmodells wurde versucht, mit den zur Verfügung stehenden Tools, das Strukturmodell abzubilden. Jeder Schritt der Modellierung wurde ebenfalls dokumentiert. Leider konnte an dieser Stelle

keine entwicklungsbegleitende Validierung erzielt werden. Die Überprüfung der beschreibenden Dokumentation lieferte keine Unstimmigkeiten des Modells mit der entworfenen Struktur, aber sie zeigte, dass das Ausmaß des Modells zu groß geworden ist. Hinsichtlich dessen kann man leicht den Überblick verlieren. Die Validierung und das Testen der einzelnen Modellmodule war nicht realisierbar, weil das gesamte Modell über mehrere Schichten aufgebaut wurde und diese voneinander abhängig waren. Eine wichtige Aufgabe der Validierung ist, die Ausgabe des Modells mit der aus dem realen System (Universität Dortmund) zu vergleichen, und zu überprüfen, ob beide Ergebnisse miteinander übereinstimmen. Doch in diesem Fall konnte keine eindeutige Aussage erstellt werden. Dies wird in den folgenden Punkten dargelegt:

1. wegen der mangelnden Anzahl an Daten kann keine glaubwürdige Verteilungsfunktion für die Eingabedaten des Modells geschätzt werden.

2. es werden die Daten von 1989 bis 2004 verwendet. Die Modelle werden aber nach der DPO 2001 erstellt. Die Abläufe der beiden Modelle weichen von dem realen Ablauf der Universität Dortmund ab.

Es wurden zwei Modelle aufgebaut, eines mit dem HIT-Tool und das andere mit dem ProC/B-Tool. Es wird angenommen, dass beide Modelle das Geschehen an der Universität Dortmund repräsentieren. Das Ziel ist es zu überprüfen, ob beide Modelle mit den gleichen Eingaben die Ergebnisse liefern, die mit hoher Wahrscheinlichkeit übereinstimmen. Zur Überprüfung der Ergebnisse aus dem HIT- und ProC/B-Modell wird der KS-Test benutzt. Da der t-Test nur für kontinuierliche Werte benutzt werden kann, ist er in diesem diskreten Fall nicht anwendbar. Bei der Ergebnisvalidierung sollte entsprechend dem Kochrezept aus dem Kapitel 3.7 vorgegangen werden. Dazu müssen alle Simulationsergebnisse, die validiert werden sollen, im Tabellenform gespeichert, im R-Tool gescannt und mit dem KS-Test gegenübergestellt werden. Abweichend vom Kochrezept werden die Ergebnisse im HIT-Modell nicht durch die Messpunkte sondern durch die globalen Variablen erfasst und tabellarisch ausgegeben. Auf die ausführliche Beschreibung der vom HIT-Modell ausgegebenen Daten wird an dieser Stelle verzichtet und auf das Kapitel 5.8 hingewiesen. Das ProC/B-Modell gibt einen Teil der Ergebnisse (*POPULATION, TURNAROUNDTIME*) durch die Messpunkte aus. Die übrigen Ergebnisse (wie die Anzahl der Professoren und HiWis, verteilt nach der Zugehörigkeit der Lehrstühle) werden mittels globalen Variablen ausgegeben. Da ProC/B keine direkte Änderung des HI-SLANG-Codes unterstützt, müssen die Pfade zu den Dateien, wo die Ergebnisse gespeichert werden sollen, manuell in den von ProC/B erzeugten HI-SLANG-Code eingefügt werden. Neben den Pfaden müssen noch die Messpunkte für die *POPULATION* und *TURNAROUNDTIME* in den HI-SLANG-Code eingesetzt werden und weitere Zeilen, die zur Datenausgabe notwendig sind. Um diesen Vorgang und die Tipparbeit zu erleichtern, wurde der Zeichenstrom-Editor *sed* zur Hilfe genommen. Da der *sed*-Editor keine direkte Arbeit mit den Variablen unterstützt, musste er in einem *Shell*-Skript (in diesem Fall *sed.script*) gespeichert und mit dem Befehl

```
./sed.script
```

aufgerufen werden. Im weiteren wird das *Shell*-Skript gezeigt und die einzelnen Skriptzeilen mit ihren Funktionen dargestellt. Für die bessere Übersicht wurden alle Zeilen untereinander geschrieben und durchnummeriert [2].

[2] An dieser Stelle sollte erwähnt werden, dass zur Ausführung des Skriptes alle Zeilennummern entfernt und alle Zeilen der *SED-Prozedur* nacheinander mit einer Leerstelle dazwischen geschrieben werden müssen.

```
#!/bin/csh -f
#*************************************************************
#   sed.script
#
#
#*** Variablen Definition ************************************

1.set x = ‘echo $PWD‘
2.set input_file = "Best.E1.hit"
3.set output_file = "BestCase.E1.hit"
4.set findlineOpen = ‘sed -e "/hold(SemWochen+2);
                           L7893zurueckgeben/=" -n $input_file‘
5.set addLineNumberOpen = ‘expr $findlineOpen + 4‘
6.set findlineClose = ‘sed -e "/MaxHiWiLSLNW:=0;;/="
                                                -n $input_file‘
7.set addLineNumberClose = ‘expr $findlineClose[2] + 5‘
8.set MesspunktNameLine = ‘sed -e "/messgeraet VIA F1BestCase;/="
                                                -n $input_file‘
9.set n = ‘expr $MesspunktNameLine + 1‘
10.set MesspunktName = ‘sed -e "$n p" -n $input_file‘

#*** Die SED-Prozedur*****************************************

11.sed -e ’/^%PARM = WARN/a\%BIND "PROC_AUS" to EXTEND
                                                ’"$x/procb.data"’’
12.-e ’/^%PARM = WARN/a\%BIND "PROC_AUS1" to EXTEND
                                                ’"$x/sem.data"’’
13.-e ’/^%PARM = WARN/c\%PARM = WARN, minmax’
14.-e ’’"$addLineNumberOpen"’ a\OPEN f2,"PROC_AUS1" LENGTH 80;’
15.-e ’/^COMPONENT modellobserver : observer/i\
                                        f2 :        OUTFILE;’
16.-e ’/^EXPERIMENT versuch METHOD SIMULATIVE/a\ VARIABLE’
17.-e ’/^EXPERIMENT versuch METHOD SIMULATIVE/a\
                                        replic    : INTEGER;’
18.-e ’/^EXPERIMENT versuch METHOD SIMULATIVE/a\
                                          startseed : REAL;’
19.-e ’/^EXPERIMENT versuch METHOD SIMULATIVE/a\
                                          myseed    : REAL;’
20.-e ’/^EXPERIMENT versuch METHOD SIMULATIVE/a\
                                          pop       : REAL;’
21.-e ’/^EXPERIMENT versuch METHOD SIMULATIVE/a\
                                        n         : INTEGER;’
22.-e ’/^EXPERIMENT versuch METHOD SIMULATIVE/a\
                                        run       : INTEGER;’
23.-e ’/^EXPERIMENT versuch METHOD SIMULATIVE/a\
                                          turn      : REAL;’
24.-e ’/^EXPERIMENT versuch METHOD SIMULATIVE/a\
```

```
                                          f1        : OUTFILE;’
25.-e ’/^EVALUATE MODEL/i\OPEN f1, "PROC_AUS" LENGTH 80;’
26.-e ’/^EVALUATE MODEL/i\ replic :=10;’
27.-e ’/^EVALUATE MODEL/i\ n := 2**30// replic+1;’
28.-e ’/^EVALUATE MODEL/i\ startseed, myseed :=13;’
29.-e ’/^EVALUATE MODEL/i\     FOR run :=1 STEP 1 UNTIL replic’
30.-e ’/^EVALUATE MODEL/i\         LOOP’
31.-e "/^MEASURE THROUGHPUT AT messgeraet;/a\
           MEASURE TURNAROUNDTIME AT $MesspunktName[1] DUE TO ALL
           ABSCISSA run ESTIMATOR MEAN, STANDARDDEVIATION,
           CONFIDENCE LEVEL 95 GLOBALSTOP WIDTH 5;"
32.-e "/^MEASURE THROUGHPUT AT messgeraet;/a\
          MEASURE POPULATION AT $MesspunktName[1] DUE TO ALL
          ABSCISSA run ESTIMATOR MEAN, STANDARDDEVIATION,
          CONFIDENCE LEVEL 95 GLOBALSTOP WIDTH 5;"
33.-e ’/^END EVALUATE/a myseed    := get_seed (startseed, n);’
34.-e ’/^END EVALUATE/a\startseed := myseed;’
35.-e ’/^END EVALUATE/a\pop  := get_result("MEAN","POPULATION",
                        "’$MesspunktName[1]’","all","dump",run);’
36.-e ’/^END EVALUATE/a\turn := get_result("MEAN","TURNAROUND
                   TIME","’$MesspunktName[1]’","all","dump",run);’
37.-e ’/^END EVALUATE/a\WRITELN FILE f1,pop,turn;’
38.-e ’/^END EVALUATE/a\END LOOP;’
39.-e ’/^END EVALUATE/a\CLOSE f1;’
40.-e ’/^END EVALUATE/a\;’
41.-e ’/^MaxProfLS2:=0;;/i\WRITELN FILE f2, "Lehrstuhl       ",
                                     "MaxProf    ", "MaxHiWi ";’
42.-e ’/^MaxProfLS2:=0;;/i\WRITELN FILE f2,  "Lehrstuhl2",
                                        MaxProfLS2, MaxHiWiLS2;’
43.-e ’/^MaxProfLS3:=0;;/i\WRITELN FILE f2,  "Lehrstuhl3",
                                        MaxProfLS3, MaxHiWiLS3;’
44.-e ’/^MaxProfLS5:=0;;/i\WRITELN FILE f2,  "Lehrstuhl5",
                                        MaxProfLS5, MaxHiWiLS5;’
45.-e ’/^MaxProfLS6:=0;;/i\WRITELN FILE f2,  "Lehrstuhl6",
                                        MaxProfLS6, MaxHiWiLS6;’
46.-e ’/^MaxProfLS10:=0;;/i\WRITELN FILE f2, "Lehrstuhl10",
                                       MaxProfLS10, MaxHiWiLS10;’
47.-e ’/^MaxProfLS12:=0;;/i\WRITELN FILE f2, "Lehrstuhl12",
                                       MaxProfLS12, MaxHiWiLS12;’
48.-e ’/^MaxProfLSA:=0;;/i\WRITELN FILE f2,  "LehrstuhlLSA",
                                        MaxProfLSA, MaxHiWiLSA;’
49.-e ’/^MaxProfLSB:=0;;/i\WRITELN FILE f2,  "LehrstuhlLSB",
                                        MaxProfLSB, MaxHiWiLSB;’
50.-e ’/^MaxProfLSS:=0;;/i\WRITELN FILE f2,  "LehrstuhlLSS",
                                        MaxProfLSS, MaxHiWiLSS;’
51.-e ’/^MaxProfLSPG:=0;;/i\WRITELN FILE f2, "LehrstuhlLPG",
                                       MaxProfLSPG, MaxHiWiLSPG;’
```

```
52.-e '/^MaxProfLSLNW:=0;;/i\WRITELN FILE f2,"LehrstuhlLNW",
                                  MaxProfLSLNW, MaxHiWiLSLNW;'
53.-e '/^MaxProfLSLNW:=0;;/i\WRITELN FILE f2, "
                                                        ";'
54.-e ''"$findlineClose[2]"' c\MaxHiWiLSLNW:=0;;
                                          CLOSE f2;hold(9);'
55.$input_file > $output_file
```

In den ersten zehn Zeilen werden als erstes die Variablen in der Shell Umgebung definiert und initialisiert. Der Variable *x* in Zeile eins wird der aktuelle Pfad zugewiesen. Die Input- bzw. Outputfiles werden in den Zeilen zwei und drei bestimmt. In den Zeilen vier und fünf bzw. sechs und sieben werden Variablen definiert, die zur Lokalisierung und Erhöhung der Zeilennummer benötigt werden. Dadurch wird erreicht, dass die neue Zeile direkt nach dem Befehlsblock eingefügt werden kann. Die Zeilen acht und neun definieren wiederum die Variablen, die zur Bestimmung des Messpunktnamens dienen. In Zeile zehn wird der Variable *MesspunktName* die Zeile, in der der Messpunktname steht, zugewiesen. Die Variable *MesspunktName* wird in den Zeilen 31-32 aufgerufen, wobei nur das erste Wort der kompletten Zeile betrachtet wird. Ab Zeile elf beginnen die *sed*-Befehle. Das Skript beginnt mit dem Befehl *sed* und die darauf folgenden Befehle mit $-e/\text{\^{}} \ldots /$. Diese wiederum bedeuten eine Suche nach einem bestimmten Zeichensatz im Quellcode. Der gesuchte Zeichensatz oder die Zeichenfolge steht immer zwischen $/\text{\^{}}$ und $/$. Nach dem zweitem Slash kommt immer der auszuführende Befehl, der in diesem Fall durch die Buchstaben mit Backslash $a\backslash$, $i\backslash$ bzw. $c\backslash$, realisiert wird. Dabei steht der *sed*-Befehl $a\backslash$ für eine Texteinfügung nach einer Zeile, $i\backslash$ für eine Texteinfügung vor einer Zeile und $c\backslash$ für eine Textersetzung. Am Ende kommt der Text, der entweder eingefügt oder mit den gefundenen Zeilen ersetzt werden soll. An dieser Stelle sollte auch erwähnt werden, dass im *sed*-Editor deutlich mehr Befehle existieren, die aber für das Verstehen des Skripts irrelevant sind. Eine ausführlichere *sed* Beschreibung und eine Reihe von anderen Befehlen kann man in [Lüdi 04] nachschlagen. Es folgt die kurze Beschreibung der einzelnen Skriptzeilen. In den Zeilen 11-13 werden nach der Quellcodezeile „%PARM = WARN“ die Pfade zu den Dateien angelegt, wo die Ergebnisdaten gespeichert werden sollen und ein Parameter *minmax* eingefügt, der für die Berechnung des minimalen und maximalen Wertes verantwortlich ist. Die Zeilen von 15-24 dienen der Variablendeklaration. Hier werden die Variablen für die Ausgabedaten, Schleifenzähler und Filedaten angelegt. Mit den Zeilen 14 und 25 werden an den entsprechenden Stellen Zeilen eingefügt, die die Ausgabefiles f1 und f2 öffnen. Die Zeilen 26-30 sind für das Einfügen der Schleife aus dem Kochrezept notwendig und sind für die zehn Ausführungen der Simulation verantwortlich. In den Zeilen 31-40 werden die Messpunkte für *POPULATION* und *TURNAROUNDTIME* festgelegt, Ausgabevariablen dieser Messpunkte initiiert und Zeilen in den Quellcode eingefügt, die die Ergebnisdaten in *procb.data* schreiben. Der Quellcode bekommt nach der Ausführung der Zeilen 41-53 des Skriptes neue Zeilen, die für die Ausgabe der benötigten Professoren und HiWis bestimmt sind. Die Zeilen 41 und 53 sind nicht relevant für die Simulation, dienen aber zur übersichtlichen Datenausgabe. Mit der Zeile 54 wird der Befehl “CLOSE f2;“ eingefügt, um das File f2 zu schließen. Die letzte Zeile dient der Erzeugung der Ausgabedatei (*$output_file*). Da der ursprüngliche Quellcode über 14000 Zeilen enthält, ist es hier nicht möglich, den durch die Anwendung des Skriptes erzeugten Endcode darzustellen. Wegen des Fehlers im ProC/B-Simulationsmodell ist es zur Zeit nicht möglich die Dateien *procb.data* und *sem.data* mit den aussagekräftigen Ergebnissen zu erzeugen. Um die Funktionalität des Skriptes zu überprüfen, wurde das Skript *sed.script* auf den zu diesem Zeitpunkt vorhandenen Quellcode angewendet und die Simulation mit geringerer Laufzeit (235 Wochen $\approx$ 9 Semester) durchgeführt. Die Ausgaben in *procb.data* und in *sem.data* sehen nach der Skriptausführung wie folgt aus:

procb.data:

```
1.323134E+003 2.335312E+002
```

wobei die erste Spalte der *POPULATION* entspricht und die zweite Spalte der *TURNAROUNDTIME*. Bei *sem.data* werden drei Spalten ausgegeben. Die erste Spalte zeigt die angeforderte Lehrstühle. In der zweiten und dritten Spalte werden die maximale Anzahl der angeforderten Professoren bzw. Hi-Wis in SWS angezeigt.

sem.data:

```
Lehrstuhl         MaxProf    MaxHiWi

Lehrstuhl2          0          0
Lehrstuhl3          0          0
Lehrstuhl5          4         56
Lehrstuhl6          0          0
Lehrstuhl10         0          0
Lehrstuhl12         4         56
LehrstuhlLSA        0          0
LehrstuhlLSB        0          0
LehrstuhlLSS        0          0
LehrstuhlLPG        0          0
LehrstuhlLNW        0          0
      .             .          .
      .             .          .
      .             .          .
Lehrstuhl         MaxProf    MaxHiWi
Lehrstuhl2          0          0
Lehrstuhl3          2         27
Lehrstuhl5          4         52
Lehrstuhl6          2         26
Lehrstuhl10         4         53
Lehrstuhl12         4        122
LehrstuhlLSA        4         52
LehrstuhlLSB        4         52
LehrstuhlLSS        4         36
LehrstuhlLPG        4         72
LehrstuhlLNW        3         18
```

Die Validierung der beiden Modelle war nicht möglich, da eines der beiden Modelle (ProC/B) noch erhebliche Fehler enthielt. Diese konnten bis zur Fertigstellung dieses Berichtes nicht behoben werden.
Die Validierung wird im zweiten PG-Semester fortgesetzt und im Anhang des zu erstellenden Endberichtes konkretisiert.

5.8 Produktionsläufe, Analyse und Interpretation der Daten (HIT)

Um einen Experimentenentwurf durchführen zu können, müssten in nur einem Modell durch Parametervariation die unterschiedlichen Simulationsziele erreicht werden. Aus diesem Grund entfällt der Experimentenentwurf, da das Best- und das Worstcase Modell zwei eigenständige Modelle sind und keine Parameter variiert werden.
Der folgende Abschnitt beschäftigt sich mit den Produktionsläufen und der Auswertung der dadurch gewonnnen Simulationsergebnisse. Leider liegen nur die Ergebnisse des HIT-Modells vor, da aus oben genannten Gründen (5.6.25) das ProC/B Modell keine verwertbaren Ergebnisse liefert.
Bei der Ergebnisanalyse wird nur das Kerninformatikstudium berücksichtigt. Die Produktionsläufe, aus denen die benötigten Daten gewonnen werden, bestehen in beiden Fällen aus 10 Replikationen mit einer Modellzeit von je 5200 Semesterwochen. Dies entspricht einem Zeitraum von 100 Jahren (200 Semestern). Die Realisierung dieser Experimentenserie wird im nächsten Unterkapitel beschrieben.

5.8.1 Bestcase

Für die Bestcaseanalyse werden als Messströme die *Population* betrachtet und die *Turnaroundtime* der Studenten der Kerninformatik, die letztendlich von dem Auswertungsobjekt *ki* (Kerninformatik) ausgelesen werden. Die gewonnen Daten werden in einer Datei festgehalten. In der folgenden Abbildung ist zu sehen, wie die Replikationen, das Abfangen der Messströme und das Anlegen der Ausgabedateien realisiert werden.

```
OPEN f2, "kerninfo" LENGTH 80;

replic := 10 ;
n      := 2**30//replic+1;
startseed, myseed := 13;
FOR run :=1 STEP 1 UNTIL replic
   LOOP

    EVALUATE test;
    myseed := get_seed (startseed, n);
    startseed := myseed;

    pop  := get_result("MEAN","POPULATION","ki","all","DUMP",0.0);
    turn := get_result("MEAN","TURNAROUNDTIME","ki","all","DUMP",0.0);

    WRITELN FILE f2, pop,turn;

   END LOOP;
   CLOSE f2;
```

Das Vorgehen orientiert sich an dem Kochrezept (s.Abschnitt 3.7).

Nach der Simulation der 10 Replikationen ergibt sich folgende Ausgabe.

```
Population          Turnaroundtime
-------------       --------------
1.684595E+003       2.340000E+002
1.774525E+003       2.340000E+002
1.769940E+003       2.340000E+002
1.813210E+003       2.340000E+002
1.793870E+003       2.340000E+002
1.858005E+003       2.340000E+002
1.734660E+003       2.340000E+002
1.768925E+003       2.340000E+002
1.830780E+003       2.340000E+002
1.791010E+003       2.340000E+002
-------------       --------------
 Durchschnitt        Durchschnitt
  ca. 1800               234
```

Wie man anhand der Turnaroundtime sehen kann benötigen die Studenten immer 234 Semesterwochen, sprich 9 Semester, um ihr Studium erfolgreich zu beenden. Somit kann man festhalten, dass unter der Bedingung uneingeschränkter Ressourcen der Studienverlauf gemäß der DPO 2001 in der Regelstudienzeit zu bewerkstelligen ist.

Da sich unter der Annahme, dass die Regelstudienzeit eingehalten wird, im Schnitt fünf Jahrgänge im System befinden müssen, bedeutet das bei einem durchschnittlichen Wert der Population von 1800 (siehe Durchschnitt Population) sich im System befindenden Prozessen, dass jährlich 360 Absolventen die Universität verlassen.

Von besonderem Interesse bei der Simulationsstudie ist die Menge der benötigten Ressourcen bezogen auf ein Semester, als da wären die Summe der angeforderten SWS der Professoren, WiMis und Hörsäle. Die Räume werden bei dieser Betrachtung ausgespart, weil sie gleich der Anzahl der SWS der WiMis sind. Wie schon bei den Messströmen werden auch diese Werte in eine separate Datei geschrieben. Allerdings mussten an dieser Stelle auf der obersten Ebene des Modells globale Variablen eingeführt werden, da HIT es einem nicht erlaubt, die Anzahl der in einem Counter vorhandenen Ressourcen direkt auszulesen. Diese Variablen werden bei jeder Ressourcenanforderung/-freigabe entsprechend aktualisiert. So können jederzeit die beanspruchten Ressourcen ausgelesen werden.

Im folgenden ist sowohl die Deklaration als auch die Initialisierung der Variablen zu erkennen:

```
VARIABLE prof : INTEGER DEFAULT 1000000;
VARIABLE wimi : INTEGER DEFAULT 1000000;
VARIABLE hs   : INTEGER DEFAULT 1000000;
VARIABLE raum : INTEGER DEFAULT 1000000;

VARIABLE prof_akt : INTEGER DEFAULT 0;
VARIABLE wimi_akt : INTEGER DEFAULT 0;
VARIABLE hs_akt   : INTEGER DEFAULT 0;
VARIABLE raum_akt : INTEGER DEFAULT 0;
```

Die ersten vier Variablen sind Zählvariablen mit einer Startinitialisierung von 1000000. Diese Zählvariablen werden benötigt, da der Komponententyp *Counter* in HIT nicht die Möglichkeit bietet, seinen aktuellen Wert auszulesen. Die nächsten vier Variablen dienen der eigentlichen Feststellung der aktuell benötigten Ressourcen. Zu diesem Zweck wurden in der Komponente *uni* eigens zwei Dienste kreiert, die zum einen die aktuellen Werte pro Semester berechnen und zum anderen diese Werte in eine Datei schreiben.

Der nachstehende Code berechnet die aktuell beanspruchten Ressourcen:

```
spend(10);

prof_akt := 1000000-prof-34;

WRITELN("Prof-Anzahl:");
WRITELN(prof_akt);

wimi_akt := 1000000-wimi;

WRITELN("Wimi-Raum-Anzahl:");
WRITELN(wimi_akt);

hs_akt := 1000000-hs-34;

WRITELN("HS-Anzahl:");
WRITELN(hs_akt);
```

Die spend(10) Anweisung sorgt dafür, dass jede Vorlesung ausreichend Zeit zur Anforderung all ihrer Ressourcen hat. Die aktuelle SWS Anzahl der angeforderten Professoren setzt sich aus der Differenz der Maximalkapazität des imitierten Counters und seiner derzeit noch verfügbaren Kapazität und einem zusätzlichen Korrektur-Subtrahenden zusammen. Dieser Subtrahend wird benötigt, da einige Vorlesungen sowohl im Wintersemester als auch im Sommersemester angeboten werden, und es so in einem Semester zu einer Doppelanforderung der Ressourcen kommt. In der Summe ergeben 9 SWS Nebenfach, 16 SWS A-B Katalog, 3 SWS Leistungsnachweise und 6 SWS Schwerpunktsgebiet den Korrektur-Subtrahenden 34. Analog wird die Anzahl der zurzeit benötigten SWS für die Hörsäle berechnet. Da die Anzahl der WiMis einzig von der Anzahl der zustande kommenden Übungsgruppen abhängt, wird an dieser Stelle der Korrektur-Subtrahend nicht benötigt.

Der nun folgende Code schreibt die gewonnenen Werte in eine Datei:

```
OPEN f1, "bestcase" LENGTH 80;
FOR i := 0 STEP 26 UNTIL 5200
LOOP
spend(10);
WRITELN FILE f1, prof_akt,wimi_akt,hs_akt;
spend(16);
END LOOP;
CLOSE f1;
```

Die FOR-Schleife und die beiden spend Befehle sorgen dafür, dass die Daten in jedem Semester genau einmal erfasst werden.
Exemplarisch werden nun die ersten zehn Datensätze der Ausgabedatei abgebildet:

prof_akt	wimi_akt	hs_akt
55	161	55
56	161	56
55	405	55
56	422	56
55	583	55
56	528	56
211	808	55
212	797	56
147	779	55
148	726	56

Diese Ausgabe stimmt mit den theoretischen Werten überein, die man nach Summation der benötigten SWS der Veranstaltungen gemäß der DPO 2001 erhält. Danach werden in den ersten Wintersemestern 55 SWS und Sommersemestern 56 SWS Professoren (und ebenso viele SWS Hörsäle) benötigt. Die Zahl der Professoren-SWS steigt erst im siebten Semester signifikant an, da in diesem Semester die Studierenden erstmalig eine PG belegen. Im Gegensatz dazu bleibt die Anzahl der genutzten Hörsäle für Winter- und Sommersemester konstant, da die Projektgruppen keine Hörsäle belegen. Ebenso lässt sich an der Anzahl der WiMi-SWS ablesen, dass ab dem dritten Semester mit dem Beginn des Hardwarepraktikums verstärkt Ressourcen gebündelt werden. Es lassen sich allerdings keine genauen Aussagen über die Anzahl der benötigten WiMis und HiWis treffen, da sich die Arbeitsverträge dieser sowohl in den aufzubringenden Arbeitsstunden sowie den Aufgabenbereichen, in welchen sie eingesetzt werden, unterscheiden. Unabhängig von der Aufteilung des Arbeitsaufkommens auf die WiMis, lässt sich feststellen, dass im Durchschnitt (ermittelt aus allen 2000 Daten) 766 SWS WiMis benötigt werden.
Unter der Annahme, dass ein Professor einer Lehrverpflichtung von neun SWS nachkommen muss, kann man nun aus den Werten der zehn Replikationen ableiten, dass man, um optimale Bedingungen gewährleisten zu können, 28 Professoren benötigt. Dazu wurde der Maximalwert aller auftretenden Ergebnisse betrachtet, welcher 244 SWS betrug. Diese Zahl wurde durch die angenommene Lehrverpflichtung geteilt und das Ergebnis dieser Division (27,1111) aufgerundet, da es nur ganze Professoren gibt. Leider lassen sich diese Werte nur schlecht mit den tatsächlich vorliegenden Ist-Werten vergleichen, da einige der getroffenen Annahmen, z.B. es existiert nur ein Schwerpunktsgebiet mit vier Vorlesungen, Leistungsnachweise werden von allen Studenten in nur zwei unterschiedlichen Veranstaltungen erworben usw., von der Realität abweichen. Tatsächlich existieren nämlich sieben Schwerpunktsgebiete mit mindestens einem Dutzend Vorlesungen, und es sind zahlreiche Möglichkeiten gegeben, Leistungsnachweise zu erwerben. Dies führt zu einer Komplexität, welche im Best- und Worstcasemodell außer Acht gelassen wurde, und erst im Realcasemodell Einzug in die Modellierung findet.

5.8.2 Worstcase

Beim Worstcase werden ebenfalls die beiden Messströme *Population* und *Turnaroundtime* des oben bereits erwähnten *ki* Auswertungsobjektes betrachtet. Die Codesegmente stimmen mit denen aus dem

Bestcase bis auf Dateinamen überein, allerdings ist nicht mehr die benötigte Menge der gleichen Ressourcen interessant, da aus den Worstcaseannahmen ersichtlich ist, dass ein Engpass lediglich bei dem SoPra und bei der PG auftreten kann. Die Ressource Professor ist an dieser Stelle nicht von Belang, da davon ausgegangen wird, dass die PG keine Professoren anfordert und dieser Wert somit konstant gehalten wird. Zu diesem Zweck werden nur zwei zusätzliche globale Variablen eingefügt, die die jeweils auf einen SoPra- oder PG-Platz wartenden Studenten zählen.

```
VARIABLE anz_pg    : INTEGER DEFAULT 0;
VARIABLE anz_sopra : INTEGER DEFAULT 0;
```

Die Werte dieser globalen Variablen werden auf die gleiche Art und Weise wie die Variablen im Bestcase in zwei separate Dateien geschrieben.
Exemplarisch werden nun die ersten zehn Datensätze gezeigt:

SoPra-Anwärter	PG-Anwärter
0	0
126	0
373	0
657	48
1000	96
1230	144
1387	192
1653	240
1997	288
2316	336

Die Anzahl der aktuell wartenden Studenten für das SoPra wird nur jedes Sommersemester ausgegeben, für die PG in jedem Wintersemester. An den Werten ist leicht abzulesen, dass das SoPra erst im vierten Semester beginnt und die PG im siebten Semester. Bedingt durch die Tatsache, dass das SoPra jeweils nur 192 Studenten aufnehmen kann, erhöht sich die Anzahl der auf eine PG wartenden Studenten kontinuierlich um 48, was der Differenz zwischen angebotenen SoPra und PG Plätzen entspricht. Die Zahl der SoPra-Anwärter steigt um keinen konstanten Summanden, da diese nur lediglich von der jeweiligen zufällig verteilten Ankunftsrate der Studenten abhängig ist. Da beide Veranstaltungen nicht in der Lage sind, den ankommenden Strom von Studenten zu bedienen, steigt die Zahl der in der Universität verweilenden Studenten stetig an. Die nun folgenden Werte für die *TURNAROUNDTIME* und die *POPULATION* belegen, dass sich im Gegensatz zum Bestcase die durchschnittliche Anzahl und Verweildauer der Studierenden immer weiter vergrößert.

POPULATION	TURNAROUNDTIME
1.403077E+004	1.447769E+003
1.405865E+004	1.432054E+003
1.375678E+004	1.407942E+003
1.402861E+004	1.421743E+003
1.379080E+004	1.433904E+003
1.385056E+004	1.422589E+003
1.374567E+004	1.430011E+003
1.390985E+004	1.424493E+003
1.314834E+004	1.376849E+003
1.368543E+004	1.406875E+003

Anhand des gemittelten Wertes für die *TURNAROUNDTIME* lässt sich ablesen, dass die Studenten im Durchschnitt aufgerundet 55 Semester ($Durchschnittlicher\,Wert\,der\,TURNAROUNDTIME \approx 1420.6\,/\,(26\,Wochen = 1\,Semester) \approx 54.64$) benötigen, um das Studium zu absolvieren.

5.9 Probleme bei der Modellierung

Leider haben wir es aufgrund einiger Probleme mit dem ProC/B-Tool nicht mehr geschafft ein laufendes Modell zu erstellen. Trotz der schnellen Reaktion des Entwicklungsteams haben wir leider immer wieder enorme Zeit verloren. Die ersten Probleme betrafen die Weiterleitung des Dienstes *change* der vorgefertigten Funktionseinheiten *Counter* und *Storage* an externe Funktionseinheiten, bei denen ein Parameter *Prio* mit weitergeleitet werden musste, der leider nirgendwo dokumentiert war. Benutzte man die Funktionseinheiten *direkt*, so wurde der Standard-Wert dieses Parameters benutzt, so dass man ihn nicht angeben musste. Bei der Weiterleitung klappte dies leider nicht mehr.
Die Fehlermeldung, dass die virtuellen Dienste einer Funktionseinheit nicht zu *echten* Diensten passten, war/ist leider nicht sehr aussagekräftig, da sie anfangs auch auftauchte, obwohl die Anbindung letztendlich doch funktionierte. Nachdem die Entwickler an diesem Problem gearbeitet haben, haben wir nun den Eindruck, den wir aber nicht weiter verifiziert haben, dass die Meldung teilweise immer noch erscheint obwohl kein Fehler vorliegt, nun aber in einigen Fällen auch NICHT erscheint, obwohl die Anbindung fehlerhaft ist. Auch das kostete uns vor allem zu Anfang sehr viel Zeit, da wir nach einem Fehler im Modell gesucht haben, der nicht vorhanden war. Da diese Fehlermeldung teilweise öfter nacheinander erscheint, wir vermuten für jede angeblich falsche Anbindung einmal, wäre es wünschenswert, wenn die Fehlermeldung auch einen Hinweis enthalten würde, bei welchem virtuellen Dienst der Fehler auftritt.
Um die Funktionseinheit *Storage* überhaupt benutzen zu können, mussten wir aufgrund eines Fehlers von HIT auf eine alternative Variante der MOBASE warten, die noch getestet wurde. Daher konnten wir erstmal eine zeitlang keine Test-Läufe mit dem Compiler durchführen. Schließlich wurde uns dann die neue MOBASE zur Verfügung gestellt, allerdings war diese noch nicht in das Tool eingebunden, so dass wir erstmal manuell eine Änderungen am HIT-Code durchführen musste, damit es funktionierte.
Der wohl schönste *Fehler* war der, dass wir an einem Abend keine Fehler hatten. Wir waren gerade dabei, das was wir bis dato modelliert hatten, zu testen, um erstmal zu wissen, ob der Compiler es soweit akzeptiert. Morgens in der Sitzung hatte Jürgen uns noch mitgeteilt, er hoffe, dass wir möglichst bald die neue MOBASE erhalten würden, und Abends bekamen wir zu unserer Überraschung auch keinen Fehler mehr dazu angezeigt, bevor wir die HIT-Datei entsprechend modifizierten. Also sind wir davon ausgegangen, dass jetzt alles fehlerfrei war. Auch sonst wurde unser Modell, wie es bis dahin war, in keinster Weise mehr vom Compiler beanstandet. Am nächsten Morgen allerdings war das nicht mehr so. Wir erhielten haufenweise Fehlermeldungen. Im Gespräch mit Jürgen stellte sich dann heraus, dass am Vortag nichts an den Tools, weder an HIT noch an ProC/B geändert worden war. Wir hätten also die Fehlermeldungen bekommen müssen. Warum das nicht der Fall war, konnte auch bis jetzt nicht geklärt werden.
Das nächste Problem betraf wieder die Funktionseinheit *Storage*. Der angebotene Dienst *content* wurde in HIT als *Procedure* deklariert und nicht, wie z.B. der Dienst *change* als *Service*. In ProC/B existiert allerdings keine Unterscheidung bei der Anbindung. Alles wird als *Service* angebunden, was somit zu falschem Quellcode in HIT führte.
Und auch danach machte uns die Funktionseinheit *Storage* wieder Probleme. Der Dienst *content* arbeitete nicht. Es wurde lediglich immer der, spätestens bei der Übersetzung nach Simula automatisch eingesetzte, Standardwert der Zielvariablen zurückgeliefert, nicht aber der Inhalt des Storage. Die ers-

te Lösung der Entwickler, doch statt Prozessvariablen globale Variablen zu benutzen, denn laut ihren Tests würde es dann klappen, war für uns leider nicht umsetzbar, da wir zwingend Prozessvariablen brauchten. Der Fehler wurde dann für eine neuere Version des ProC/B-Tools behoben, welche wir aber noch nicht direkt nutzen konnten, so dass wir uns mit einem insgesamt doch recht zeitaufwendigen Umweg über manuelle Aufrufe der einzelnen Compiler begnügen mussten.
Zu guter Letzt machte dann auch noch der GCC-Compiler Ärger. Bei der Größe unseres Modells forderte er aufgrund eines Fehlers auf den Linux-Rechnern der Uni über 2GB an Arbeitsspeicher für die Übersetzung an, was die Systeme nicht leisten konnten. Die Folge war, dass die Übersetzung nach über einer Stunde einfach abbrach. Da dieser Fehler auch nicht schnell behoben werden konnte, mussten wir die eigentliche Simulation also auf den Solaris-Maschinen der Uni ausführen, wo ohne GCC-Compiler gearbeitet wird.
All das hat uns insgesamt so weit zurückgeworfen, dass wir zeitlich nicht mehr in der Lage waren, ein Worstcase-Modell zu erstellen.
Desweiteren sind uns noch einige Funktionen aufgefallen, welche die Arbeit mit ProC/B deutlich vereinfachen und beschleunigen würden oder die nicht korrekt arbeiten. Diese möchten wir an dieser Stelle noch kurz aufführen:

- Die Möglichkeit bestehende Verbindungen (Pfeile) von einem Prozessketten Element auf ein anderes umzulegen wäre sehr hilfreich, speziell wenn die Kanten (längere) Beschriftungen oder Bedingungen enthalten.

- Fügt man mehrmals hintereinander die gleichen zuvor kopierten Elemente ein, so kann es passieren, dass die Anbindungen an Dienste einen irreparablen Schaden erleiden. Das Tool erkennt, dass es den Dienst nicht gibt, man kann ihn aber in den Attributen des Prozesskettenelements nicht ändern und somit das Fenster nicht mehr schließen. Das Tool ist blockiert.

- Es wäre, gerade wenn einzelne Funktionseinheiten sehr oft eingefügt werden müssen, sehr viel besser, wenn die Namensänderung nicht nur einen .1 anfügen würde, sondern z.B. von .1 bis .9 mitzählen würde. So wie es jetzt ist, werden die Namen sehr schnell sehr lang und unübersichtlich

- Die Automatische Anpassung der Größe von Konnektoren scheint nicht richtig zu arbeiten. Besonders wenn eine Prozesskette der Übersicht halber „zeilenweise“ dargestellt wird, versagt die Größenberechnung.

- Eine Option, die ProC/B-Modelle ähnlich wie HIT direkt als Batch-Job ausführbar macht, wäre wünschenswert.

- Die Möglichkeit Minimal- und Maximalwerte der Messpunkte auszugeben, sowie den jeweils aktuellen Messwert zum Messzeitpunkt.

6 Weitere Vorgehensweise

Zu Beginn des nächsten Semesters müssen die bestehenden Modelle noch validiert und deren Annahmen überdacht werden. So ist zum Beispiel die strikte Trennung zwischen den einzelnen Semestern nicht mehr aufrecht zu erhalten, da es dem Studenten in der Regel freigestellt ist, in welchem Semester er welche Veranstaltung besuchen und wann er die dazugehörige Prüfung ablegen möchte. Selbst die Trennung zwischen Grund- und Hauptstudium muss nicht immer eingehalten werden, da Leistungsnachweise des Hauptstudiums schon während des Grundstudiums erlangt werden können. Außerdem sind im Realcase die Durchfallquoten zu berücksichtigen, was zur Folge hat, dass Studenten nach dreimaligem Nichtbestehen einer Fachprüfung das Studium endgültig beenden müssen. Weiterhin sind die real vorhandenen Ressourcen festzustellen und entsprechend in dem Modell zu implementieren, als da wären, die Anzahl der Professoren und wissenschaftlicher Mitarbeiter (zwischen HiWis und WiMis wird nicht unterschieden, da die entsprechenden Daten nicht vorliegen), die Anzahl und Größe der Räume und Hörsäle, Anzahl der Seminare etc. Auch der Detaillierungsgrad der Modellierung muss im vorhinein überdacht werden. So ist beispielsweise zu überprüfen, ob alle Vorlesungen des Schwerpunktgebietes einzeln zu modellieren sind oder weiterhin von einer Dummyklasse repräsentiert werden sollen.
Die Nebenfächer sollen feinkörniger betrachtet werden, da der Studienverlauf durch die Wahl des Nebenfaches beeinflusst werden kann. Ebenfalls ist geplant die Prüfungsordnung der *Angewandten Informatik* sowie des *Lehramtes Informatik* zu berücksichtigen.
Eines der Resultate der Modellierungsarbeit im ersten Semester besteht in der Erkenntnis, dass moderne graphische Modellierungstools durch ihre einfachen copy/paste-Funktionen dazu verleiten, Modelle mit unnötig großem Codeumfang zu erstellen. Ziel im zweiten Semester wird es also auch sein, die Performanz der Modelle bei der Implementierung zu berücksichtigen.

Literaturverzeichnis

[Aren 04] *Arena Basic Edition User's Guide*. http://literature.rockwellautomation.com/idc/groups/literature/documents/um/arenab-um001_-en-p.pdf, März 2006

[Balc 95] BALCI, O.: *Principles and techniques of simulation validation, verification, and testing*. Proceedings of the 1995 Winter Simulation Conference ed. C. Alexopoulos, K.Kang, W.R.Lilegdon, and D.Goldsman, 1995

[Baus 02] BAUSE, F.; BEILNER, H.; FISCHER, M.; KEMPER, P.; VÖLKER, M.: *The ProC/Btoolset for the Modelling and Analysis of Process Chains.* TOOLS 2002, LNCS 2324, pp.51-70. Springer-Berlin Heilderberg, 2002

[Beil 89] BEILNER, H.; MÄTER, J.; WEISSENBERG, N.: *Towards a Performance Modeling Environment: News on HIT*. Proceedings 4th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Plenum Publishing, 1989

[Beil 99] BEILNER, H.; BAUSE, F.; TATLITÜRK, H.; VAN ALMSICK, A.; VÖLKER, M.: *Zum B-Modellformalismus - Version B1 - zur Vorbereitung automatischer Analysen von Modellen logischer Systeme hinsichtlich technischer, ökonomischer und ökologischer Ziele*. SFB 559; Bericht Nr. 99002, 1999

[Beil 04] BEILNER, H.; MÄTER, J.; SCZITTNICK, M.; WYSOCKI, CH.: *Hands on Hit*. Universität Dortmund, Informatik IV, July 2004

[Berg 99] BERGER, A.; BREGMAN, L.; KOGAN, Y.: *Bottleneck analysis in multiclass closed queueing networks and its application*. Queueing Systems 31(1999) 217-237, 1999

[Blak 95] BLAKE, R.; BRESSE, J.S.: *Automatic Bottleneck Detection* . Microsoft Research; Technical Report MSR-TR-95-10, 1995

[Bütt 96] BÜTTNER, M.; HECK, E.; LANGE, R.; SCZITTNICK, M.; STRÜWER, M.; VERWOHLT, M.; WYSOCKI, C.: *HITGRAPHIC User's Guide*. Universität Dortmund, Informatik IV, 1996

[Bütt 00] BÜTTNER, M.; FRICKE, B.; KLAASSEN, O.; NOLTE, S.; SCZITTNICK, M.; STAHL, H.; WEISSENBERG, N.: *HI-SLANG REFERENCE MANUAL*. Universität Dortmund, Informatik IV, 2000

[Burg 00] BURGHARDT, M.: *Projektmanagement, Leitfaden für die Planung, Überwachung und Steuerung von Entwicklungsprojekten*. Publicis MCD Verlag, 2000

[DPO 01] *Diplomprüfungsordnung für den Studiengang Informatik an der Universität Dortmund vom* 07.12.2001 *(Stand:* 02.02.2004*)* . http://dekanat.cs.uni-dortmund.de/web/de/content /ordnungen/ki/DPO_KI2001_Stand_Feb2004.pdf, März 2006

[Eick 02] EICKHOFF, M.; HIERWECK, M.; SCHWENKE, M.: *Hands On ProC/B-Tools*. SFB 559, Technical Report 02004, 2002

[Gibs 01] GIBSON, J.P.: *Formal requirements models: simulation, validation and verification*. http://www.cs.may.ie/~pgibson/Research/Publications/E-Copies/nuim-cs-tr-2001-02.pdf, März 2006

[Harm 03] HARMON, S.Y.; YOUNGBLOOD, S.M.: *A Proposed Model for Simulation Validation Proces Maturity*. https://www.dmso.mil/public/library/projects/vva/evolvingconcepts /03s_siw_12.doc, März 2006

[Hier 05] HIERWECK, M.; FINZEL, J.; VAN ALMSICK, A.; KRIEGE, J.; SCHWENKE, M.: *ProC/B-Editor-Handbuch*. SFB 559, interner Bericht, 2005

[Jain 91] JAIN, R.: *The Art Of Computer Systems Performance Analysis*. New York: John Wiley & Sons, Inc., 1991

[Law 91] LAW, A.M.; KELTON, W.D.: *Simulation Modeling & Analysis; Second Edition*. McGraw-Hill, Inc., 1991

[Leng 00] LENGEWITZ, P.; WEISSENBERG, N.: *HIT User's Guide for UNIX Systems*. Universität Dortmund, Informatik IV, 2000

[Käpp 02] KÄPPNER, M.; LAAKMANN, F.; STRACKE, N.: *DortmunderProzesskettenparadigma - Grundlagen*. SFB 559, Technical Report 02005, 2002

[Kopp 05] KOPP, F.A.: *Projektmanagement*. http://www.fhbb.ch./tools/documente/doc/0003_1197_ PM-Skript.pdf, März 2006

[Lüdi 04] LÜDIGER, T.: *Messung und Charakterisierung von Ethernet-Netzwerklasten*. Diplomarbeit, Universität Dortmund, 2004

[Mäte 04] MÄTER, J.: *Hands on Tools-Hands on HIT*. Universität Dortmund, Informatik IV, 2004

[Müll 01] MÜLLER, N.TH.: *Einführung in die ereignisgesteuerte Simulation*. Vorlesung im WS 2001/2002; http://www.informatik.uni-trier.de/ mueller/Lehre/simu01.pdf, März 2006

[Muno 02] MUÑOZ, J.C.: *Moving Bottlenecks: A Theory Grounded on Experimental Observation* . Proceedings 12th ISTTT, (M.A.P. Taylor, editor) pp.441-462, Pergamon, 2002

[Pete 01] PETERS, B.A.; SMITH, J.S.; MEDEIROS, D.J.; ROHRER, M.W.: *A Practical Bottleneck Detection Method*. in Proceedings of the 2001 Winter Simulation Conference, 1999

[TNG 99] THE NUMBERS GROUP: *Project Management*. http://www.snc.edu/socsci/chair/333/numbers.html, März 2006

[Vena 05] VENABLES, W.N.; SMITH D.M. AND THE R DEVELOPMENT CORE TEAM: *An Introduction to R. 2.2.0*. http://cran.r-project.org/doc/manuals/R-intro.pdf, März 2006

[Weis 99] WEISSENBERG, N.; WILDE, A.: *HIT and HI-SLANG An Introduction*. Universität Dortmund, Informatik IV, 1999

[Wien 02] WIENDAHL, H.P.; HEGENSCHEIDT, M.: *Engpassanalyse von Produktionssystemem mit Kennlinien*. In Werkstattstechnik online Jahrgang 92(2002) H.9, 2002

[Zing 04] ZINGEL, H.: *Grundzüge des Projektmanagements*. http://www.zingel.de, März 2006