UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

PG 476

Bottleneckanalyse des Informatikstudiums an der Universität Dortmund

Endbericht

Erstellt 11. Oktober 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. Projektmanagement **2**

3. Bottleneckanalyse der Studiensituation an der Universität Dortmund **4**
3.1. Problemstellung und Ziele 4
3.2. Strukturmodell der Universität Dortmund 4
3.3. Aggregierung 6
3.3.1. Allgemeine Informationen 6
3.3.2. Mögliche Umsetzung 7
3.4. Parallele Simulation 7
3.4.1. Verteilte Simulation 8
3.4.2. Parallele Simulation 8
3.4.3. Entwurf einer parallelen Simulation 8

4. Realcase-Studie des Studiums im Fachbereich Informatik **11**
4.1. Problemstellung und Planung der Studie 11
4.2. Datenerhebung und Modellentwurf 11
4.2.1. Struktur- und Ablaufdiagramme 11
4.2.2. Annahmen 19
4.2.3. Zusammenfassung der Annahmen 20
4.2.4. Datenerhebung/-analyse 21
4.3. Modellierungsrichtlinien 23
4.4. Implementierung und Verifikation des HIT- Modells für den Realcase 25
4.4.1. Realcase 26
4.4.2. kerninfo 28
4.4.3. grundstudiumki 30
4.4.4. hauptstudiumki 38
4.4.5. v_gti 43
4.4.6. v_dap1 43
4.4.7. v_logik 44
4.4.8. p_hapra 44
4.4.9. p_pg 45
4.4.10. diplom_arbeit 46
4.5. Implementierung und Verifikation des ProC/B- Modells für den Realcase 48
4.5.1. Modell 48
4.5.2. Realcase 49
4.5.3. KI 52
4.5.4. GS 53

4.5.5. HS 58
4.6. Experimentendesign 66
4.7. Modellvalidierung 66
4.8. Produktionsläufe, Analyse und Interpretation der Daten 73

5. Fazit der Projektgruppe **78**
5.1. Toolkritik 78
5.2. Fazit der PG-Mitglieder 79

Anhang **91**

A. Skripte für die parallele Simulation **91**
A.1. parsimscript 91
A.2. parascript 99

B. Validierung der Bestcasemodelle **101**

C. Vorschlag zu Abschlussberichten **116**

Literatur **118**

1. Einleitung

Die Projektgruppe (PG) 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.
Im ersten Semester der Projektgruppe wurden als erste Teile der Bottleneckanalyse des Informatikstudiums an der Universität Dortmund die Best- und der Worstcase-Szenarien untersucht (siehe [ZB 06]). Das Ziel des zweiten Semesters ist die Betrachtung und Analyse des Realcases.
Zu Beginn diesen zweiten Semesters wird eine zeitliche Planung der Projektgruppe aufgestellt, um sicherzustellen, dass die gesetzten Fristen eingehalten werden (siehe Kapitel 2).
Danach wird die grobe Struktur der Universität Dortmund beschrieben. Da eine vollständige Untersuchung der Universität Dortmund zu zeitaufwändig wäre, wird ein Mittel (*Aggregierung*) zur Realisierung von Simulationsstudien bei umfangreichen Systemen erläutert und eine mögliche Umsetzung für die konkrete Anwendung dieser Technik auf den Fall der Bottleneckanalyse der Universität Dortmund entwickelt (siehe Kapitel 3).
Daran schließt sich mit der Simulationsstudie des Fachbereichs Informatik (Kapitel 4) 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 HIT [Bütt 96] und ProC/B [Baus 02] umgesetzt. Zeitgleich mit der Implementierung und der einhergehenden Verifikation der Modelle 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.
Das Kapitel 5 enthält eine abschließende Kritik der Bewertungstools und die Einzelkritiken der Gruppenmitglieder sowie des Veranstalters zur PG.
Zusätzlich wird im Anhang dieses Endberichts die Validierung der Modelle aus dem vergangenen Semester nachgereicht (siehe Anhang B).
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/ Kosten- und Leistungsrechnung und bei 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-Editors unterstützt haben.

2. Projektmanagement

Der Zwischenbericht der Projektgruppe liegt zum Start des zweiten Semesters noch nicht in seiner endgültigen Version vor und das obwohl die Gruppe entgegen ihrer ursprünglichen Planung ihre Arbeit an einigen Terminen in den Semesterferien fortgesetzt hat. Trotzdem oder gerade deswegen wird das Projektmanagement weiterhin betrieben. Denn die Verzögerung liegt nicht an einem fehlerhaften Projektmanagement, sondern an Fehlern, die im Zwischenbericht [ZB 06] erläutert werden, und an einer mangelnden Qualität des ersten Entwurfs des Zwischenberichts.
Die erste und auch wichtigste Entscheidung des Projektmanagements ist somit, fertige und korrigierte Texte sofort an den Betreuer Jürgen Mäter weiterzuleiten. Auf diese Weise soll eine Quelle der Verzögerung eliminiert werden. Die Dokumentation findet also wieder semesterbegleitend statt.
Die andere Quelle, nämlich potentielle Fehler in den Modellierungstools, kann nicht im Vorfeld ausgeschaltet werden. Daher soll eine straffe Zeitplanung ein gewisses Zeitpolster am Semesterende schaffen, um somit einen eventuellen Verzug zu kompensieren.
Eine Einarbeitungsphase, wie sie im ersten Semester nötig war, kann im zweiten Semester entfallen. Die Projektgruppe startet ihre Arbeit infolgedessen unmittelbar mit der Konzeptionsphase. In dieser Phase wird mit Hilfe der Erfahrungen aus dem Best- und Worstcase das Modell für den Realcase und die entsprechenden Annahmen aufgestellt. Für diese Prozedur ist eine Deadline auf Ende April gesetzt, was einem Bearbeitungsaufwand von circa einem Monat entspricht. Falls es notwendig ist weitere Daten zu erheben, geschieht dies ebenfalls in dieser Phase.
Das Ende der darauf folgenden Durchführungsphase wird auf Ende Juni gesetzt. Folglich ist es geplant, das HIT- und das ProC/B-Modell bis Mitte Juni fertig zu stellen, um genügend Zeit für die Validierung, Verifikation und natürlich die Fehlersuche zu haben.
Aufgrund dieser Deadlines sollte ausreichend Zeit zur Verfügung stehen, um die Endpräsentation vorzubereiten, welche für den 10. Juli 2006 angesetzt ist.
Da die Projektgruppe im vergangenen Semester gute Erfahrung mit dem Tool ProMa FREEWARE 4.08 gemacht hat, wird dieses auch in diesem Semester erneut eingesetzt. Die Aufstellung der gesetzten Deadlines in dem Tool ist in Abbildung 2.1 zu sehen.
Die Verantwortlichkeiten und Termine für kleinere Aufgaben, die sich während des Semesters ergeben, werden wie gehabt direkt beim Auftreten geregelt.

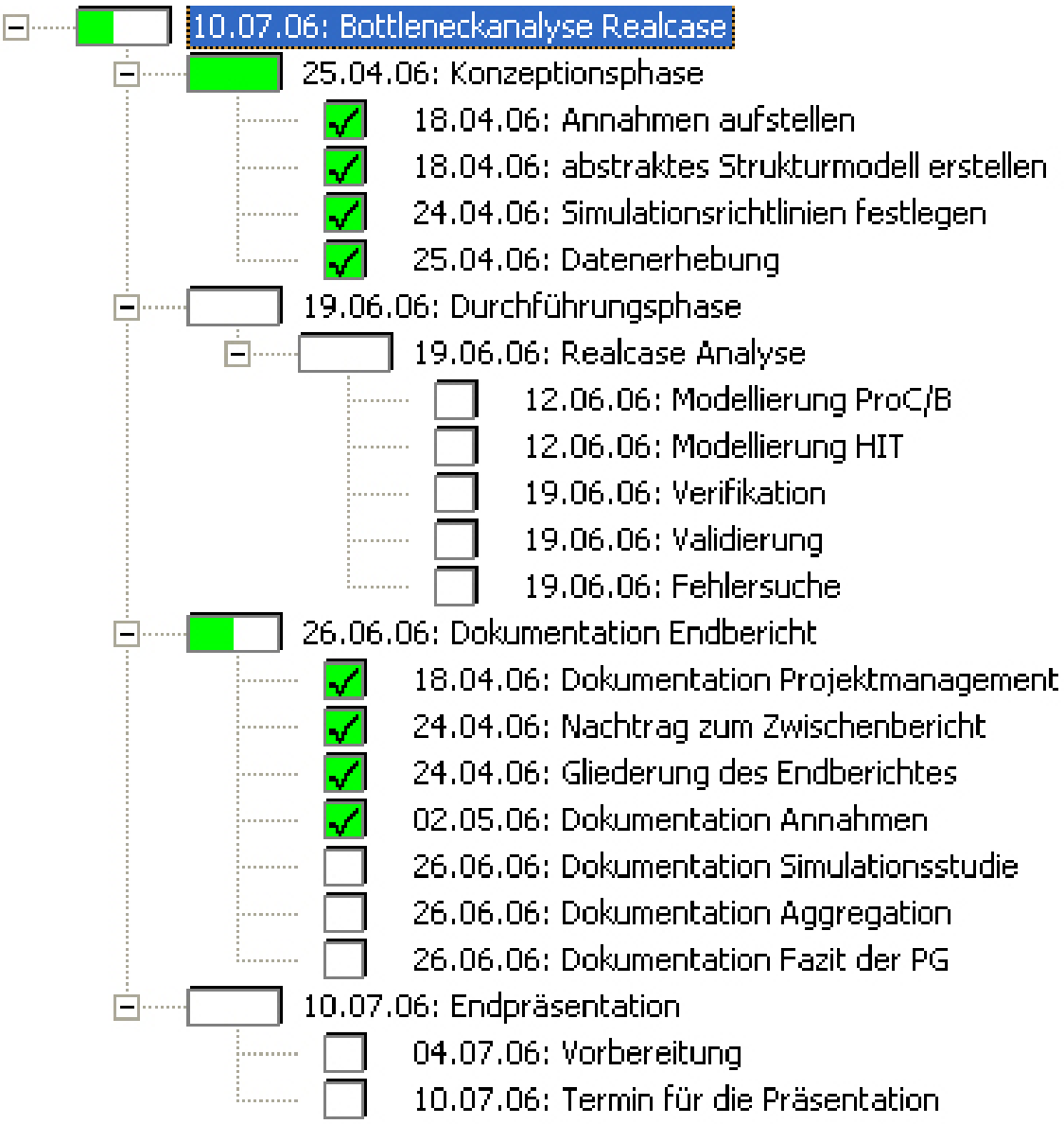


Abbildung 2.1.: Projektplan

3. Bottleneckanalyse der Studiensituation an der Universität Dortmund

Im folgenden Abschnitt wird zunächst die Problemstellung der Studie beschrieben.
Weiterhin wird auf die Struktur der Universität Dortmund eingegangen. Da aus dieser Struktur bei einer entsprechenden Bottleneckanalyse mittels Simulation ein sehr großer und komplexer Umfang resultieren würde, wird weiterhin ein Mittel (*Aggregierung*) zur Realisierung von Simulationsstudien bei solch umfangreichen Systemen erläutert.
Anschließend wird auf Basis des Aggregierungsansatzes eine mögliche Umsetzung für die konkrete Anwendung dieser Technik auf den Fall der Bottleneckanalyse der Universität Dortmund entwickelt.
Um die Simulationszeit zu verkürzen, stellt der letzte Abschnitt dieses Kapitels ein Prinzip der *parallelen Simulation* vor.

3.1. Problemstellung und Ziele

Die Projektgruppe beschäftigt sich mit dem so genannten Realcase zur Bottleneckanalyse der Studiensituation an der Universität Dortmund. Das angestrebte Auswertungsziel ist es, die Anzahl an zu erwartenden Diplomanden, deren Studiendauer und die für die Studiendauer relevanten Faktoren zu bestimmen.

3.2. Strukturmodell der Universität Dortmund

Bevor das abstrakte Modell erstellt werden kann, müssen eine einheitliche Syntax und Semantik festgelegt werden. Diese werden in der Abbildung 3.1 dargestellt.
Das *Strukturmodell* der Universität Dortmund (Abbildung 3.2) 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.
Auf der obersten Ebene steht die Universität *Uni* selbst. Darunter gibt es 16 weitere Ebenen für die einzelnen Fachbereiche der Universität, die alle die Ressource *Vorlesung* besitzen. Diese Fachbereiche bestehen aus Lehrstühlen, welche die Ressourcen *Personal* zur Verfügung stellen, wobei *Personal* die Menge an Professoren und Privat-Dozenten umfasst. Auf die Aufnahme der Ressourcen *Hörsäle*, bzw. *Räume* wird verzichtet, da sie für die Ziele der Studie, unter den getroffenen Annahmen (Kapitel 4.2.2), nicht relevant sind.

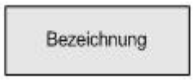
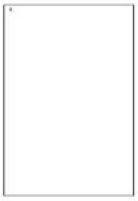
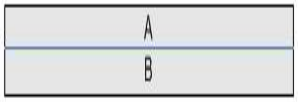
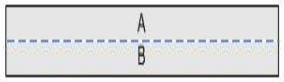
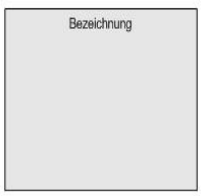
Symbol	Beschreibung
	Der Abhängigkeitspfeil wird zur Darstellung der Abhängigkeit zwischen zwei oder mehreren Vorlesungen benutzt. Vorlesungen, die durch diesen Pfeil verbunden sind, werden nur der Reihe nach „abgearbeitet“ und die Reihenfolge darf nicht geändert werden.
	Der Überganspfeil zeigt die Ablauffolge des Datenflusses bzw. den 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 denen nur eine ausgewählt werden muss, entweder A oder B.
	Die Ellipse mit der Bezeichnung in der Mitte repräsentiert die benötigten Ressourcen, zum Beispiel Personal.
	Das Rechteck mit der Überschrift wird zur Darstellung des allgemeinen Konzeptes oder globalen Einheiten wie *Universität* oder *Fachbereich* angewendet.

Abbildung 3.1.: Legende

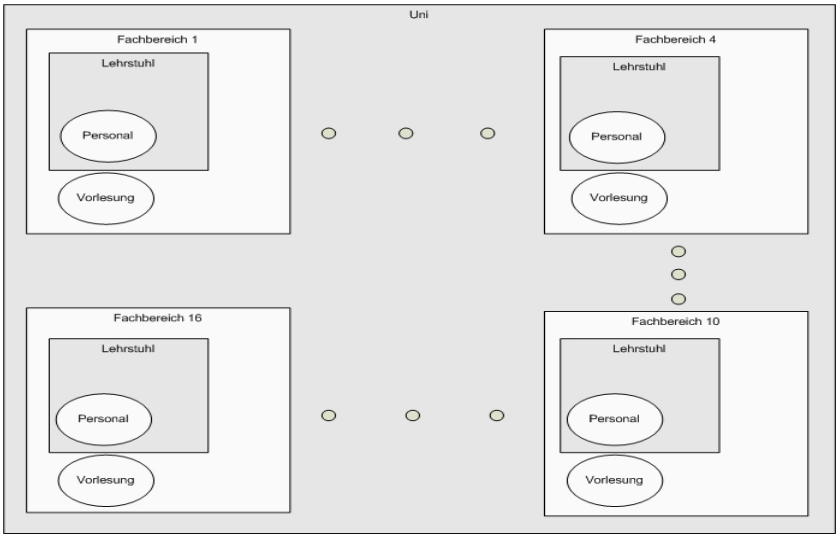


Abbildung 3.2.: Strukturmodell der Universität Dortmund

3.3. Aggregierung

Der folgende Abschnitt befasst sich mit den Grundlagen der Aggregierung. Anschließend wird eine Grundidee zur möglichen Umsetzung dieser Technik auf den Kontext der Bottleneckanalyse der Universität Dortmund diskutiert.

3.3.1. Allgemeine Informationen

Das Hauptziel der Aggregierung ist es, die Dauer eines Simulationslaufes zu reduzieren, da die Simulation eines großen Systems zu zeitaufwendig ist. Dies geschieht indem die detaillierten Teilmodelle zuerst analysiert werden, um sie danach in einfachere, aber verhaltensäquivalente Ersatzdarstellungen zu überführen. Um ein komplexes Modell aggregieren zu können, muss es somit möglich sein, es in disjunkte Teilmodelle zu zerlegen.
Die Voranalyse kann mit Hilfe von analytisch-algebraischen, numerischen und simulativen Verfahren durchgeführt werden.
Die Aggregierung besteht aus fünf Phasen:

- Auswahl/Bestimmung der zu aggregierenden Subsysteme/Teilmodelle
- Wahl der Struktur des Aggregats
- Konstruktion und Parametrisierung des Aggregats
- Integration des Aggregats in das Gesamtmodell
- Analyse des aggregierten Simulationsmodells oder weitere Aggregierung durch Wiederholung der ersten vier Phasen

Aggregate und Teilmodelle besitzen dieselben Schnittstellen und sind für die Umgebung nicht zu unterscheiden. Die Umgebung kennt nur die Schnittstelle und merkt den Zeitverbrauch, den eine Dienstausführung benötigt. Der Unterschied zwischen einem Aggregat und dem Teilmodell ist jedoch, dass

das Aggregat eine einfachere interne Struktur besitzt und dadurch die Simulationsdauer reduziert. Dadurch, dass das Aggregat, unter Umständen, nur eine Approximation des Teilmodells darstellt, wird zwar der Simulationsaufwand verringert, allerdings können Aggregierungsfehler entstehen.
Eine wünschenswerte Eigenschaft von Teilmodellen ist die Wiederverwendbarkeit, sprich die Möglichkeit sie mehrmals in einem Modell zu verwenden.

3.3.2. Mögliche Umsetzung

Für eine mögliche Erweiterung der Bottleneckanalyse auf den Gesamtkontext der Universität Dortmund, ist eine Einteilung in einzelne Teilsysteme anhand der existierenden Fachbereiche sinnvoll. Das Verhalten dieser Teilsysteme muss in Voranalysen bestimmt werden, um eine möglichst realitätsnahe Verhaltensweise bei der Umsetzung in Aggregate des Gesamtsystems zu gewährleisten. In der Realität interagieren einige Fachbereiche miteinander und sind folglich nicht disjunkt. Zum Beispiel halten die Professoren vom Fachbereich Mathematik auch Vorlesungen für die Studenten des Fachbereiches Informatik. Dies widerspricht der Voraussetzung für die Aggregierung hinsichtlich der Disjunktheit der Teilmodelle. Aufgrund der Erfahrungen mit dem Best- und Worstcase wurden neue Ziele getroffen, was zu Folge hat, dass auf die Modellierung der Ressourcen ganz verzichtet wird. Somit sind die Fachbereiche bzw. die Teilmodelle disjunkt.
Die im vierten Kapitel folgende Simulationsstudie stellt die notwendigen Analysen für die Erstellung des Aggregats für den Fachbereich Informatik zur Verfügung.
Weiterhin sind für eine Zusammenfassung der einzelnen Aggregate zu einem Gesamtsystem einheitliche Schnittstellen zu definieren. Diese sind in Abhängigkeit der gewünschten Ergebnisse bzw. Analysehintergründe zu wählen.
Unter der Annahme, dass das Modell der Universität Dortmund die in Kapitel 3.1 genannten Ziele verfolgt, ergibt sich eine einfache Schnittstellendefinition: Die insgesamt auf Uni-Ebene erzeugten Prozesse (Studenten) lassen sich aufgrund zu ermittelnder Einschreibezahlen auf die einzelnen Fachbereiche verteilen. Hierbei ist anzumerken, dass an der Universität Dortmund nicht alle Studiengänge mit einem Diplom abgeschlossen werden. Für die Betrachtung dieser Studiengänge ohne uni-interne Abschluss-Arbeiten, ist eine Umformulierung der Ziele (vgl. Kapitel 3.1) bzw. eine Erweiterung des Untersuchungsgegenstandes notwendig.
Weitere Informationen zur Aggregierung sind in [Tepp 03] zu finden.

3.4. Parallele Simulation

Die parallele bzw. verteilte Simulation bezieht sich auf das Ausführen der Simulation in einem Multiprozessorsystem oder in einem Netzwerk. Im Weiteren werden diese Simulationsarten kurz beschrieben und dann die in der Arbeit angewendete Vorgehensweise näher erläutert. Für ausführlichere Beschreibungen der parallelen bzw. verteilten Simulation gibt es sehr viele Literaturquellen, wie zum Beispiel [Ward 96] und [Schu 02].
In der sequentiellen, ereignisorientierten Simulation bewährten sich Konzepte, die wenig hilfreich beim Entwurf paralleler Simulationsmethoden waren. Die zu simulierenden Ereignisse der sequentiellen ereignisorientierten Simulation werden nach Ereignispunkten total geordnet. Parallele Simulationen basieren in der Regel auf Ereignissen, die durch das Kausalitätsprinzip lediglich partiell geordnet sind. Dabei sollten diese Ereignisse so angeordnet werden, dass sie in keinem kausalen Zusammenhang stehen und die Möglichkeit besteht, sie parallel simulieren zu können. Allgemein kann man die Simulationen in konservative und optimistische Verfahren unterteilen. Konservative Verfahren bewahren das Kausalitätsprinzip, während optimistische Verfahren Verletzungen des Kausalitätsprinzips

zulassen, erkennen und beheben (vgl. [Ward 96]).

3.4.1. Verteilte Simulation

Die *Verteilte Simulation* besteht aus mehreren Teilmodellen, die zeitparallel auf mehreren Computern eines Computernetzwerks ausgeführt werden. Das Simulationsmodell besteht aus mehreren disjunkten Teilmodellen, die auch physisch getrennt vorliegen. Der Entwickler des Simulationsmodells hat die Möglichkeit das Modell explizit in mehrere Teilmodelle zu zerlegen oder mehrere getrennt vorliegende Simulationsmodelle über eine gemeinsame Infrastruktur zu einem Gesamtsimulationsmodell zu verbinden. Demnach werden die Teilmodelle auf verschiedenen Computern gestartet und gleichzeitig abgearbeitet. Um die Abhängigkeiten der Teilmodelle zu wahren, kommunizieren diese innerhalb des Computernetzwerks untereinander.

3.4.2. Parallele Simulation

Die *Parallele Simulation* besteht dagegen aus einem physisch einheitlich vorliegenden Simulationsmodell. Der Entwickler des Modells beschreibt explizit die parallel ausführbaren Teilmodelle. Dabei wird die parallele Programmierung verwendet, welche durch die Programmumgebung (*Software des Parallelen Simulators, Laufzeitumgebung der Programmiersprache, Betriebssystem oder Hardware*) den einzelnen Prozessoren bzw. Computern zur Verarbeitung zugewiesen wird. Die Kommunikation der Teilmodelle erfolgt hier über einen gemeinsamen Speicher (Mehrprozessorrechner) oder ein Computernetzwerk.

3.4.3. Entwurf einer parallelen Simulation

Das Hauptziel der parallelen und verteilten Simulation ist die Laufzeitbeschleunigung, d.h. die Erhöhung der Performance der Simulation. Die Vielzahl von Simulationsläufen mit veränderten Parametern sollte in kürzester Zeit abgeschlossen werden. In der Arbeit von P. Heidelberger [Heid 86] wurden zwei Verfahren der parallelen Simulation analysiert. Das erste Verfahren zerlegt das Modell in mehrere Teilmodelle und führt diese auf verschiedenen Rechnern aus. Im zweiten Verfahren werden die einzelnen Replikationen auf verschiedene Rechner verteilt und ausgeführt. Diese beiden Verfahren wurden anhand von Speed-Up's [1] α_P ausgewertet. Dabei wurden auch folgende Faktoren betrachtet: die Anzahl von Prozessoren *P*, die Initialphase und die inhärente Variabilität [2]. Die Analyse zeigt, dass bei einer langen Initialphase oder wenn eine große Anzahl von Prozessoren zur Verfügung steht und der Speed-Up $\alpha_P \rightarrow \infty$ wenn $P \rightarrow \infty$, das erste Verfahren zu wählen ist. Falls die Initialphase im Vergleich zur Simulationslaufzeit kurz ist, *P* gering ist oder den Speed-Up Faktor nicht stark beeinflusst, wird das zweite Verfahren verwendet.

Für das ProC/B- bzw. HIT-Modell wird das zweite Verfahren verwendet, weil die Initialphase, im Vergleich zur Gesamtlaufzeit, kurz und die Anzahl der zur Verfügung stehenden Rechner gering ist. Im Rahmen der Laufzeitbeschleunigung wurde das Ziel gesetzt, einzelne Modellausführungen/Replikationen parallel auf verschiedenen Rechnern zu simulieren. Das dazu entwickelte Konzept wird in

[1] Speed-Up ist definiert als $\alpha_P = t_1/t_P$, wobei t_1 die Simulationslaufzeit auf einem Prozessor und t_P die Simulationslaufzeit auf *P* Prozessoren ist.

[2] Hiermit sind Modellteile gemeint, deren Ausführungszeiten mit der parallelen Simulation nicht beschleunigt werden können.

der Abbildung 3.3 dargestellt und im Weiteren beschrieben. Bei diesem Konzept werden alle Modellreplikationen auf die zur Verfügung stehenden Simulationsrechner aufgeteilt. Basis für die parallele Simulation ist die *Remote Shell* (rsh), die in der folgenden Syntax benutzt wird.

```
rsh rechner_name "[kommando_1];[kommando_2];...;[kommando_n]"
```

Mit diesem Aufruf werden dem Simulationsrechner (*rechner_name*) die Anweisungen (*kommando_1, . . . ,kommando_n*) zugewiesen, was zu Folge hat, dass diese auf dem Simulationsrechner ausgeführt werden.
Es wurden zwei Skripte erstellt, das Hauptskript *parsimscript* (siehe Anhang A.1) und das Hilfsskript *parascript* (siehe Anhang A.2), die in der Abbildung als dicke Rechtecke dargestellt sind. Beide Skripte sind unter den Solaris-Rechnern ausführbar. Um die Skripte ausführen zu können müssen einige Voraussetzungen erfüllt werden. Eine dieser Vorraussetzungen ist eine Datei *.rhost* mit Angabe sämtlicher Rechner, die für die Simulation zur Verfügung stehen, im *Home-Verzeichnis* anzulegen. Der Eintrag in die *.rhost* Datei ist wie folgt zu realisieren.

```
rechner_1 benutzername
rechner_2 benutzername
        .
        .
        .
rechner_n benutzername
```

Die *.rhost* Datei ermöglicht den in ihr enthaltenen Rechnern den nicht authentifizierten Zugriff. Dies erfolgt nur in einem lokalen Netzwerk. Des Weiteren muss eine Datei *rechner.txt* erzeugt werden, in der die Simulationsrechner aufgelistet sind, die bei der Simulation tatsächlich gebraucht werden. An dieser Stelle sollte auch erwähnt werden, dass die Rechner in *rechner.txt* auch in *.rhost* enthalten sein müssen. Damit sind die Voraussetzungen für die parallele Simulation erfüllt und das Skript *parsimscript* ausführbar.
Mit dem folgenden Befehl

```
./parsimscript rechner.txt hit-datei
```

wird die parallele Simulation gestartet. Bei diesem Befehl wird die HIT-Datei als zweiter Parameter übergeben.
Im weiteren Ablauf fordert das Skript den Benutzer auf, die Eingabe eines Startseed-Wertes (*startseed*) und die Anzahl der Replikationen (*#replikationen*) anzugeben. Des Weiteren erzeugt *parsimscript* die Datei *newstartseed.txt*, in der alle Seed-Werte für die jeweiligen Replikationen aufgelistet sind und splittet sie danach, je nach Anzahl der Simulationsrechner, in kleinere Seedlisten (*teil<Rechner_1>, . . . , teil<Rechner_n>*) auf. Weiterhin generiert *parsimscript* mit Hilfe von *parascript* für jeden Simulationsrechner ein eigenes Skript (*parascript _<Rechner_1>*, . . . , *parascript _<Rechner_n>*). Die erzeugten Skripte benutzen die nach dem Splitten entstandenen Seedlisten, um den *rsh*-Aufruf zu starten. Mit diesen *rsh*-Aufrufen bekommen die Simulationsrechner ihre Instruktionen und erstellen für jeden Rechner ein eigenes Ergebnis (*ergebnis_<Rechner_1>, . . . , ergebnis_<Rechner_n>*). *parsimscript* benutzt die einzeln erstellten Ergebnisse, um sie zu einem Endergebnis (*Ergebnis.txt*) zu vereinen. Am Ende werden die erstellten Dateien und Skriptdateien, die als temporäre Dateien dienten, gelöscht.

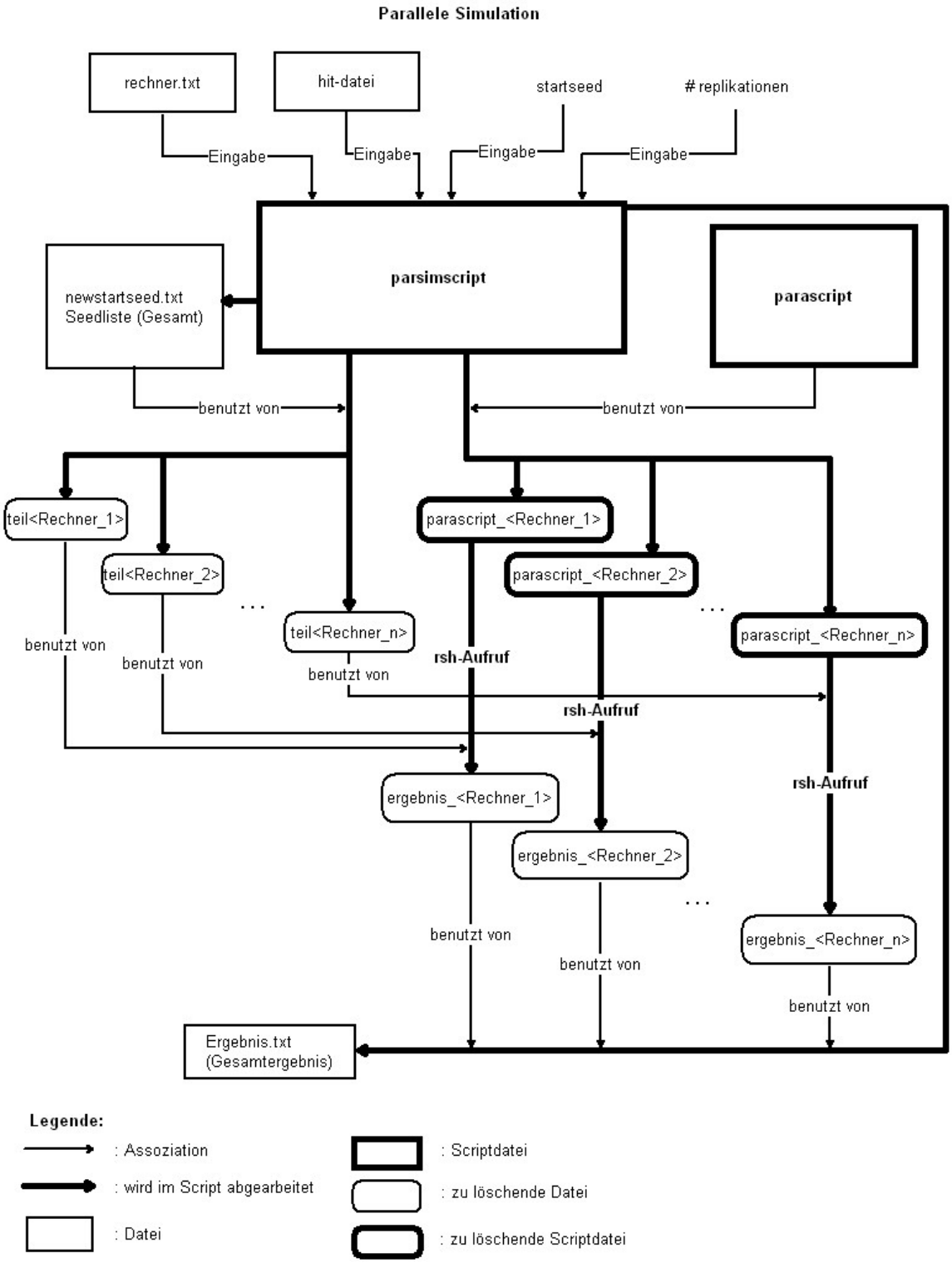


Abbildung 3.3.: Parallele Simulation

4. Realcase-Studie des Studiums im Fachbereich Informatik

Das vierte Kapitel widmet sich der eigentlichen Simulation. Aufgrund des großen Umfangs der Bottleneckanalyse der gesamten Universität Dortmund, beschränkt sich die Arbeit der Projektgruppe im Folgenden auf die Voranalyse zur Bildung eines Aggregates des Fachbereiches Informatik.
Dabei wird versucht sich an den zehn 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.

4.1. Problemstellung und Planung der Studie

Die Projektgruppe beschäftigt sich mit dem so genannten *Realcase* der Studiensituation. Es wird versucht, die Ereignisse und Gegebenheiten, die Einfluss auf den Studienablauf haben können, in ihren Auswirkungen so real wie möglich umzusetzen, um zu garantieren, dass die Ergebnisse der Simulation mit denen des realen Systems übereinstimmen. Aus diesem Grund werden Durchfallquoten eingeführt. Um den Umfang der Analyse im kontrollierbaren Rahmen zu halten, wird von Ereignissen, die keine Auswirkungen auf die Ergebnisse haben, abstrahiert. Das von der Projektgruppe gesetzte Ziel ist es, die Anzahl der Diplomanden und deren durchschnittliche Studiendauer zu ermitteln. Auf die Modellierung der Ressourcen wird verzichtet (siehe Kapitel 4.2.2).

4.2. Datenerhebung und Modellentwurf

Dieses Kapitel enthält die im Vorfeld aufgestellten Struktur- und Ablaufdiagramme, die für die Modelle getroffenen Annahmen und die durchgeführte Datenerhebung bzw. -Analyse.

4.2.1. Struktur- und Ablaufdiagramme

Die für die Struktur und Ablaufdiagramme geltende Syntax und Semantik sind weiterhin in der Abbildung 3.1 dargestellt. Die gesamte Struktur des abstrakten Modells wurde in zwei Teile aufgeteilt: *Strukturmodell des Grundstudiums Kerninformatik bzw. Angewandte Informatik* und *Strukturmodell des Hauptstudiums Kerninformatik bzw. Angewandte Informatik*.

Strukturmodell des Grundstudiums Kerninformatik Das Grundstudium des Kerninformatikstudiums an der Universität Dortmund hat einen bestimmten vorgegeben Ablaufplan, der durch den Fachbereich Informatik vorgeschlagen wird. 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 4.1 zeigen an, dass die Semester nacheinander durchlaufen werden.

Allerdings ist dieser explizite Studienablauf für das Grundstudium keine Pflicht, sondern nur ein möglicher Ablaufplan des Grundstudiums. Jeder Student kann für sich selbst entscheiden, in welcher Reihenfolge er die Vorlesungen besucht und eventuell eine Prüfung in der jeweiligen Vorlesung ablegt.
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 4.1 dargestellt.
Für die gesamte Studienzeit wählt der Studierende eines von insgesamt 22 angebotenen Nebenfächern aus. Diese haben im Grundstudium jeweils einen Umfang von 12 bis 19 Semesterwochenstunden (SWS). Die am häufigsten gewählten Nebenfächer sind *BWL*, *Theoretische Medizin*, *Elektrotechnik* und *Mathematik*. Wobei für Studenten mit den Nebenfächern *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. In der Abbildung 4.1 wird nur

ein Nebenfach aufgeführt, da es für das Modell nicht relevant ist, für welches Nebenfach sich der Student entscheidet, denn die Wahl des Nebenfachs hat keinen Einfluss auf die Ressourcen, die der Fachbereich Informatik zur Verfügung stellen muss.

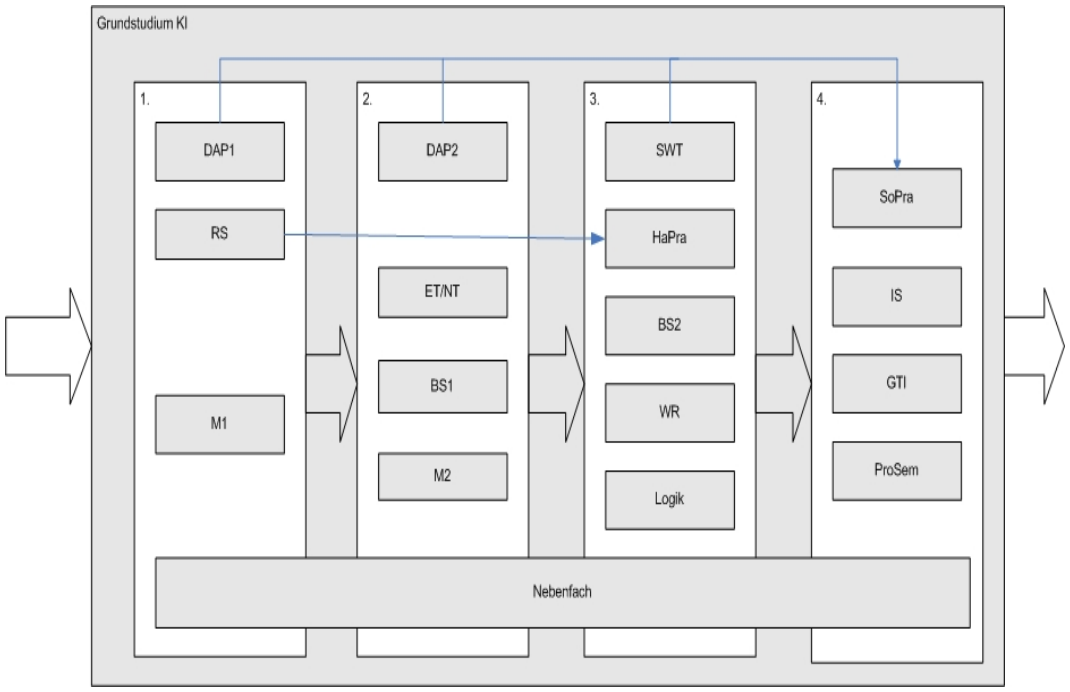


Abbildung 4.1.: Ablaufplan des Grundstudiums für Kerninformatik

Strukturmodell des Grundstudiums Angewandte Informatik Als Alternative zu der Kerninformatik wird an der Universität Dortmund die 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)

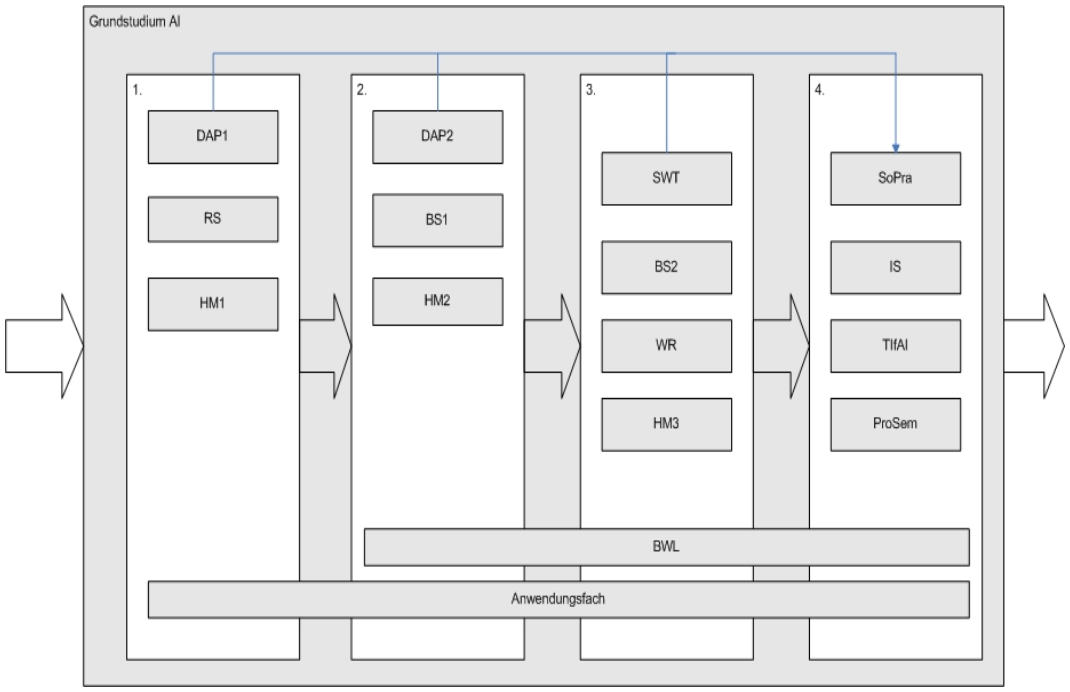


Abbildung 4.2.: Ablaufplan des Grundstudiums für Angewandte Informatik

Die Übergangspfeile in der Abbildung 4.2 zeigen an, dass die Semester nacheinander durchlaufen werden.
Auch dieser Studienablauf wird vom Fachbereich Informatik vorgeschlagen, wobei er jedoch nicht eingehalten werden muss. Nur das *SoPra* setzt die erfolgreiche Teilnahme an *DAP1*, *DAP2* und *SWT* voraus. Dies wird durch die Abhängigkeitspfeile in der Abbildung 4.2 dargestellt.
Für die gesamte Studienzeit wählt der Studierende eines von insgesamt acht (Stand Februar 2004) angebotenen Anwendungsfächern aus. Diese haben im Grundstudium jeweils einen Umfang von 12

bis 19 Semesterwochenstunden (SWS).
Für Studenten mit den Anwendungsfächern *Architektur* und *Logistik* gilt es, anstatt des Leistungsnachweises über *Höhere Mathematik I*, einen 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 einen Leistungsnachweis über *Logik* zu erwerben.

Strukturmodell des Hauptstudiums Kerninformatik Der folgende Studienablauf ist vom Fachbereich Informatik vorgeschlagen: 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- und 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 mit 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:

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

Katalog 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 4.3 gemäß Diplomprüfungsordnung auf das fünfte und sechste Semester aufgeteilt. Außerdem wird die Wahl der drei Wahlpflicht-Lehrveranstaltungen in der Abbildung 4.3 mit Hilfe des UND- und ODER-Konnektors über das fünfte und sechste Semester dargestellt. Das Nebenfach wird in der Abbildung 4.3 als ein Block 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 ergeben. 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 4.3 analog zum ersten Abschnitt repräsentiert.

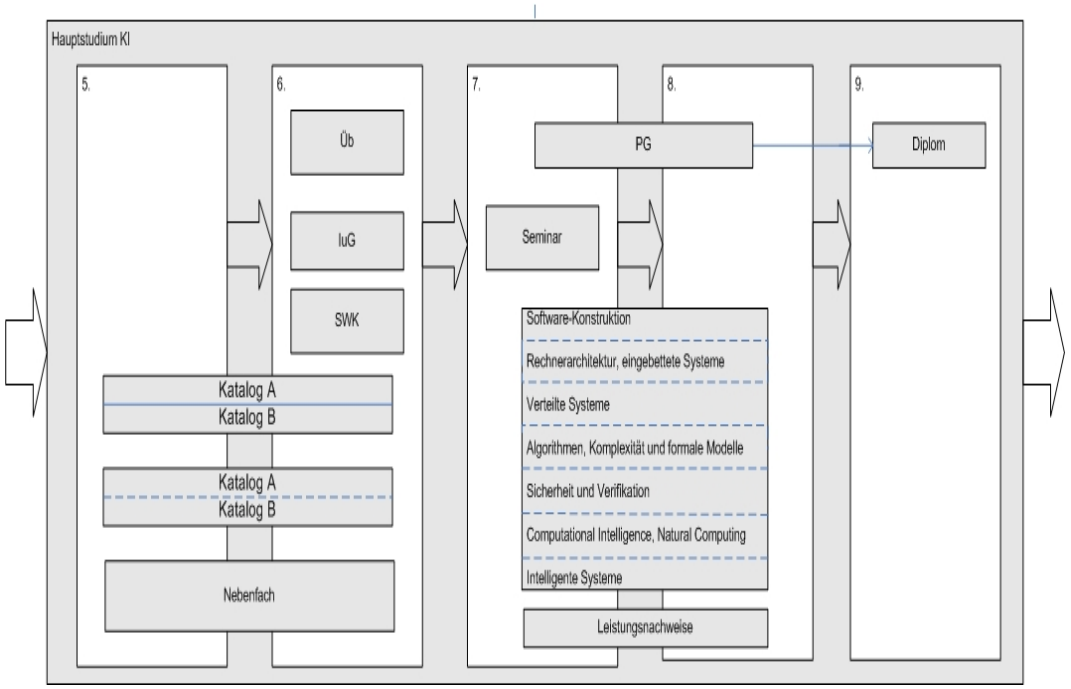


Abbildung 4.3.: Ablaufplan des Hauptstudiums der Kerninformatik

Wie schon im Grundstudium muss sich der Student nicht an diese vorgegebene Reihenfolge halten. Zu jedem beliebigen Zeitpunkt kann der Student im Hauptstudium eine Prüfung seiner Wahl machen. Die Diplomarbeit wird jedoch gesondert behandelt. Hier wird die Vereinbarung getroffen (siehe Kapitel 4.2.2), dass eine erfolgreiche Absolvierung der PG Voraussetzung für die Diplomarbeit ist, da laut DPO nach der PG noch mindestens ein Teil der Diplomprüfung abzulegen ist.

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 4.4 dargestellt.
Im fünften und sechsten Semester sind Leistungspunkte der Lehrveranstaltungen zu erwerben, die in den Katalogen des Wahlpflicht- und nichttechnischen Wahlpflichtbereich angeboten werden. Wobei Lehrveranstaltungen aus dem Katalog *Nichttechnische Wahlpflicht-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 nachstehend 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. Zusätzlich ist 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.

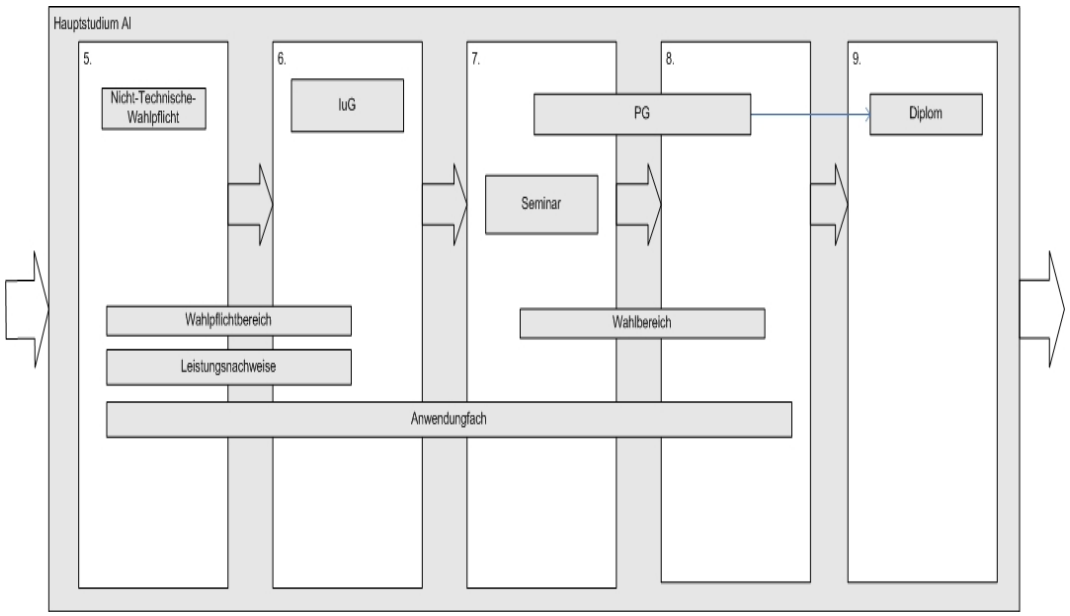


Abbildung 4.4.: Ablaufplan des Hauptstudiums der Angewandten Informatik

Auch bei der Angewandten Informatik muss sich der Student nicht an diese Reihenfolge halten. Die einzige Ausnahme, wie auch bei der Kerninformatik, ist, dass eine erfolgreiche Teilnahme an der PG Voraussetzung für die Diplomarbeit ist.

4.2.2. Annahmen

Für die Umsetzung des Realcase-Modells werden folgende Annahmen getroffen:
Es werden keine Ressourcen modelliert, da davon ausgegangen wird, dass diese in ausreichendem Maße vorhanden sind. Unter der Annahme, dass alle Vorlesungen stattfinden, wird die explizite Modellierung der Professoren vernachlässigt. Auf die Aufnahme von wissenschaftlichen Mitarbeitern (*WiMis*) und studentischen Hilfskräften (*HiWis*) in das Modell wird ebenfalls verzichtet, weil diese nicht ausschlaggebend für auftretende Studiumsverzögerungen sein können, da sie nur als Beisitzer (*WiMis*) in mündlichen Prüfungen und als Übungsgruppenleiter (*WiMis* und *HiWis*) tätig werden. Im Falle der Übungen wird angenommen, dass Übungen immer stattfinden, wobei die Größe und Anzahl der Übungen nicht relevant für die Studienzeit ist. Im Folgeschluss dieser Annahme wird ebenfalls auf eine Modellierung von räumlichen Kapazitäten verzichtet.
Im Weiteren werden Vorlesungen ebenfalls nicht explizit, sondern nur durch angebotene Prüfungen inklusive entsprechender Durchfallquoten modelliert. Für mündliche Prüfungen wird eine pauschale Durchfallquote von 1% angenommen. Die Durchfallquoten für die schriftlichen Prüfungen sind in Kapitel 4.2.4 zu finden. Hinsichtlich der Prüfungstermine wird angenommen, dass die Studenten alle angebotenen Termine wahrnehmen. Daraus ergibt sich, dass bei schriftlichen Prüfungen bei Bedarf beide Termine in dem jeweiligen Semester wahrgenommen werden und bei mündlichen Prüfungen analog alle drei Termine. Es wird weiterhin angenommen, dass Studenten die Universität nur dann vorzeitig verlassen, wenn sie dreimal durch eine Fachprüfung gefallen sind. Außerdem wird eine Schranke von 20 Semestern für das Grundstudium gesetzt. Sollte ein Student länger als 20 Semester für sein Vordiplom benötigen, so wird dieser exmatrikuliert.
Die Pflichtpraktika (*HaPra*, *SoPra*) und die *Projektgruppe* werden hingegen explizit modelliert, da diese durch ihre beschränkte Größe, bzw. Anzahl der Plätze einen möglichen Engpass darstellen und somit eine deutliche Auswirkung auf die Studiendauer haben können. Für das *HaPra* und *SoPra* werden die Kapazitäten im Modell mittels Parametern vorgegeben. Für diese Parameter sind später entsprechende Modell-Experimente geplant, um ihren Einfluss auf die Ergebnisse der Analyse zu testen. Bei den beiden Praktika und der Projektgruppe wird weiterhin die Annahme getroffen, dass der Erhalt eines Praktikumsplatzes mit dem Scheinerwerb gleichzusetzen ist. Ein Nicht-Bestehen der Praktika ist somit ausgeschlossen.
Da die Wahl des *Nebenfaches* weder einen Einfluss auf den zeitlichen Ablauf, noch auf die benötigten Ressourcen des Informatikstudiums hat, werden die Nebenfächer nicht betrachtet. Die Nebenfächer haben keinen Einfluss auf die Studienzeit, da für ihre Prüfungen keine Durchfallquoten vorliegen. Diese zu erheben, wäre zu zeitaufwendig gewesen. Die einzige Ausnahme bildet hier das Nebenfach *Elektrotechnik*, da hier statt des *HaPra* das *EPra* vorgesehen ist. Um dieses im Modell dennoch zu berücksichtigen, wird für das *HaPra* die Anzahl der Plätze im Modell um 20 Prozent im Vergleich zur Realität reduziert, da dies dem Anteil der Informatikstudenten mit Nebenfach *Elektrotechnik* entspricht.
Für die *Projektgruppe* wird angenommen, dass pro Semester eine zufällige Anzahl an PGs angeboten wird. Die Datenerhebung hat ergeben, dass diese zwischen acht und zwölf liegt, so dass die zufällig gewählte Anzahl an PGs in diesem Rahmen liegt. Somit wird der potentielle Ausfall von PGs berücksichtigt. Für die Zuteilung eines PG-Platzes an den Studenten wird angenommen, dass dieser den Platz sofort zu Beginn seines Hauptstudiums erstmalig beantragt und bei Nichtzuteilung jedes Semester erneut anfragt. Dies weicht von der Reihenfolge, die in der DPO vorgesehen und im Strukturmodell abgebildet ist, ab. Aufgrund der Tatsache, dass der Studienablauf der Studenten in diesem Punkt erfahrungsgemäß nicht mit dem in der DPO vorgesehenen Ablauf übereinstimmt, wurde die Modellierungsannahme entsprechend angepasst.

Die Vorlesungen aus den Schwerpunktsgebieten und aus den Katalogen A und B werden nicht explizit modelliert, sondern als eine bestimmte Anzahl an mündlichen Prüfungen dargestellt. Für die Kataloge A und B ergeben sich in der Summe drei Prüfungen, wobei zwischen den Katalogen aufgrund fehlender Ressourcenzuteilung nicht mehr unterschieden wird. Für die Schwerpunktsgebiete sind insgesamt 18 LP zu erbringen, wofür zwei bis sechs Prüfungen anzusetzen sind. Hierbei wird die Annahme getroffen, dass im Durchschnitt vier Prüfungen abgelegt werden, so dass nur diese Anzahl modelliert wird. Auf eine Unterteilung auf die sieben Schwerpunktsgebiete wird ebenfalls verzichtet.
Es wird im Weiteren die Annahme getroffen, dass sich *Seminare* und *Leistungsnachweise*, die in mündlichen Prüfungen erworben werden, nicht ursächlich auf eine Studienzeitverzögerung auswirken, da *Seminare* und Lehrveranstaltungen, in denen man *Leistungsnachweise* erwerben kann, zu jeder Zeit in ausreichendem Umfang angeboten werden. Außerdem gibt es keine maximale Anzahl an Prüfungsversuchen. Daher werden diese Veranstaltungen nicht modelliert.
Die Diplomarbeit kann theoretisch direkt nach dem Erreichen des Hauptstudiums begonnen werden. Laut DPO muss nach der PG noch mindestens ein Teil der Diplomprüfung, d.h. eine Fachprüfung oder die Diplomarbeit, ausstehen. Zu diesem Zweck wurde vereinbart, dass der Beginn der Diplomarbeit an eine erfolgreiche Absolvierung der Projektgruppe gebunden ist. Zusätzlich wurde festgelegt, dass die Diplomarbeit frühestens im fünften Semester des Hauptstudiums liegen darf. Analog zu den Praktika und der Projektgruppe wird die Annahme getroffen, dass die Diplomarbeit in jedem Fall erfolgreich abgeschlossen wird. Die Anzahl an *Diplomarbeiten* wird nicht beschränkt, da sie als Ergebnis der Simulationsstudie gemessen werden soll.
Für die Generierung der Prozesse wird davon ausgegangen, dass sich die Einschreibezahlen gleichverteilt im Intervall von [192,780] befinden (siehe Kapitel 5.2.2 im Zwischenbericht [ZB 06]).
Die Studenten der Fachrichtung *Angewandte Informatik* und *Lehramt Informatik* werden nicht in Betracht gezogen, da sie als *DELAY* modelliert keinen Einfluss auf die Ergebnisse haben. Deshalb werden nur Studenten der *Kern-Informatik* modelliert.
Die Ressourcen, die die *Angewandten Informatiker* in PG und SoPra benötigen, werden aus den gegebenen Kapazitäten entsprechend ihrem Anteil herausgerechnet. Für die Ergebnisinterpretation ist zu beachten, dass die Diplomanden des Studiengangs *Angewandte Informatik* nicht berücksichtigt worden sind.
Eine Zeiteinheit im Modell entspricht einem Semester. Außerdem stellt ein Prozess einen Cluster von Studenten mit der Größe zwölf dar, was der Anzahl der Teilnehmer einer PG entspricht.

4.2.3. Zusammenfassung der Annahmen

- keine Modellierung von Ressourcen (Hörsäle, Professoren, etc.)
- Vorlesungen werden durch Prüfungen mit Durchfallquoten modelliert
- Durchfallquoten mündlicher Prüfungen pauschal 1%
- Bei Bedarf werden beide (schriftlich) bzw. alle drei (mündlich) Prüfungstermine pro Vorlesung in dem jeweiligen Semester wahrgenommen.
- Exmatrikulation nur bei dreimaligem Durchfallen durch eine Fachprüfung
- Exmatrikulation bei Überschreitung einer Studiendauer von 20 Semestern im Grundstudium
- explizite Modellierung von HaPra, SoPra und PG
- Kapazitäten von HaPra und SoPra als Experiment-Parameter

- Erhalt eines Praktikumsplatzes und PG-Platzes wird gleichgesetzt mit dem Scheinerwerb
- keine Modellierung des Nebenfachs
- Reduzierung der Plätze für das HaPra um 20% im Vergleich zur Realität wegen dem Nebenfach Elektrotechnik
- zufälliges Angebot von acht bis zwölf Projektgruppen pro Semester
- Beantragung des PG-Platzes direkt beim Erreichen des Hauptstudiums
- keine Unterscheidung der Kataloge A und B - Modellierung als drei mündliche Prüfungen
- keine Unterscheidung der Schwerpunktsgebiete - Modellierung als vier mündliche Prüfungen
- keine Modellierung von Seminaren und Leistungsnachweisen, die mit mündlichen Prüfungen erlangt werden
- Diplomarbeit erst nach Absolvierung der PG
- Diplomarbeit frühestens im fünften Semester des Hauptstudiums
- Diplomarbeiten werden immer erfolgreich abgeschlossen
- keine Beschränkung der Anzahl an Diplomarbeiten
- Neu-Einschreibungen pro Jahr gleichverteilt im Intervall [192, 780]
- ausschließliche Modellierung der Kern-Informatik
- Semester als Zeiteinheit
- Clusterung von zwölf Studenten zu einem Prozess

4.2.4. Datenerhebung/-analyse

Nachdem die Annahmen für das Realcase-Modell getroffen wurden (siehe Kapitel 4.2.2), ist es noch notwendig, folgende Daten zu ermitteln bzw. festzulegen, um alle Prozessläufe des Modells zu ermöglichen:

- die Verteilung der Einschreibungszahlen,
- die Aufteilung der Studenten in Kerninformatik und Angewandte Informatik,
- Angebote für Projektgruppen,
- deren Aufteilung unter allen Lehrstühlen des Fachbereichs Informatik,
- Durchfallquoten und
- Anzahl Professoren/Privatdozenten sowie deren Zugehörigkeit zu den Lehrstühlen

Verteilung der Einschreibungszahlen Für die Verteilung der Einschreibungszahlen der Informatikstudenten wird die diskrete Gleichverteilung mit dem Intervall [192,780] genommen (siehe [ZB 06] Kapitel 5.2.2). Die gemäß [Law 91] durchzuführende Validierung dieser Verteilung wurde in diesem Kapitel bereits vorgenommen und kann somit an dieser Stelle entfallen.

Aufteilung der Studenten in Kerninformatik und Angewandte Informatik Der prozentuale Anteil an Studenten der Kerninformatik und Studenten der Angewandten Informatik bleibt äquivalent zu den Anteilen, die bei den Best- bzw. Worstcase-Modellen benutzt worden sind (vgl. [ZB 06]) und beträgt 85% für Kerninformatik und 15% für Angewandte Informatik.

Angebote für Projektgruppen Aus den zur Verfügung stehenden Projektgruppenstatistiken wurde ermittelt, dass vom Jahr 1989 bis 2006 durchschnittlich zehn Projektgruppen pro Semester angeboten wurden. Davon fallen 20% aus, also zwei Projektgruppen pro Semester. Da in den letzten sechs Jahren überwiegend mehr als zehn Projektgruppen angeboten wurden, werden zwischen acht und zwölf Projektgruppen modelliert. Die genaue Anzahl wird im Modell zufällig bestimmt.

Aufteilung der PGs unter allen Lehrstühlen des Fachbereichs Informatik Mit der Auswertung der vorliegenden Statistiken konnte auch die Zuordnung der einzelnen Projektgruppen zu den bestimmten Lehrstühlen vorgenommen werden, was in der Tabelle 4.1 gezeigt wird.

Lehrstühle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
PG Anzahl	75	12	11	63	25	27	20	7	0	28	21	28
Prozentuale Anteil	24%	4%	3%	20%	8%	9%	6%	2%	0%	9%	7%	9%

Tabelle 4.1.: Zuordnung Projektgruppen zu den Lehrstühlen von 1989 bis 2006

Es sollte allerdings erwähnt werden, dass acht Prozent der gesamten PGs von externen Fachbereichen oder von der Informatikrechner-Betriebsgruppe (IRB) angeboten wurden und somit nicht berücksichtigt werden, da das Modell keine anderen Fachbereiche betrachtet bzw. die IRB kein Lehrstuhl ist.

Durchfallquoten Es existieren kaum Statistiken und Informationen über die Durchfallquoten der Vorlesungen aus dem Hauptstudium. Deswegen werden nur die Durchfallquoten für die Vorlesungen des Grundstudiums und die Leistungsnachweise aus dem Hauptstudium, bei denen anschließend eine schriftliche Klausur angeboten wird, betrachtet. Die Durchfallquoten für diese Vorlesungen sind einzeln in der Tabelle 4.2 aufgelistet. Diese Ergebnisse wurden anhand von Klausuren der letzten Jahre ermittelt.

Sommersemester			
Vorlesung	**Teilnehmeranzahl**	**Anzahl durchgefallener Teilnehmer**	**Durchfallquoten**
BS1	419	174	41,53%
DAP2	283	162	57,24%
IS	204	77	37,75%
M2	127	54	42,52%
SWT	199	68	34,17%
Wintersemester			
Vorlesung	**Teilnehmeranzahl**	**Anzahl durchgefallener Teilnehmer**	**Durchfallquoten**
BS2	145	39	26,90%
DAP1	197	95	48,22%
IuG	236	25	10,59%
Logik	185	98	52,97%
M1	222	163	73,42%
RS	238	84	35,29%
SWK	251	88	35,06%
WR	277	133	48,01%

Tabelle 4.2.: Durchfallquoten

Die getroffenen Annahmen für die Hauptstudiumsvorlesungen werden im Kapitel 4.2.2 ausführlich beschrieben.

Anzahl Professoren und deren Zugehörigkeit zu den Lehrstühlen Jeder Lehrstuhl besitzt mindestens einen Professor, der die Vorlesungen aus dem zugehörigen Bereich bzw. Schwerpunkt hält. Die Lehrstühle mit der Anzahl zugehöriger Mitarbeiter (ohne HiWis und WiMis) sind in der Tabelle 4.3 dargestellt, wobei die Abkürzung (i.R.) für einen emeritierten Professor steht. Diese Daten werden benötigt, um sie mit der Anzahl der Diplomanden zu vergleichen.

Lehrstühle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Mitarbeiteranzahl	3+1(i.R.)	7	1	2+1(i.R.)	4	2	1	1	0	1	3	2

Tabelle 4.3.: Zuordnung von Lehrkräften zu den Lehrstühlen

4.3. Modellierungsrichtlinien

Vor Beginn der eigentlichen Modellierung wurden einige Absprachen getroffen, um sicherzustellen, dass beide Modelle den gleichen Sachverhalt darstellen.
Bei der Implementierung wird auf eine explizite Unterscheidung der einzelnen Semester verzichtet und lediglich zwischen Sommer- und Wintersemester unterschieden.
Der erste Implementierungsansatz enthielt in beiden Modellen Konstrukte um Blöcke parallel abzuarbeiten (CONCURRENT-Statement bei HIT beziehungsweise UND-Konnektoren bei ProC/B). Dieser Ansatz führte allerdings zu Synchronisationsproblemen, da die Blöcke trotz ihrer Parallelität

im Modell sequentiell abgearbeitet wurden. Somit wurde beschlossen, dass die Prozesse sequentiell in Nullzeit überprüfen, ob die einzelnen Veranstaltungen besucht werden. Dieses Konstrukt zu entwickeln und die Fehler im ersten Implementierungsansatz zu entdecken stellte sich als eine der zentralen Herausforderungen im Umgang mit den Modellierungswerkzeugen bei der Umsetzung des Realcase-Modells dar. Die genaue Umsetzung findet sich in den Kapiteln 4.4 und 4.5.
Aus Performancegründen wurde beschlossen, dass ein Prozess nicht nur einen Studenten repräsentiert, sondern eine Gruppe von zwölf Studenten. Diese Clustergröße ergibt sich durch die Anzahl der Studenten, die in einer einzelnen Projektgruppe Platz finden.
Um sicherzustellen, dass alle Studenten eine echt positive Wahrscheinlickeit haben einen Praktikumsplatz zu erhalten, wird ein Konstrukt verwendet, welches mit Zufallszahlen arbeitet. Dieses Konstrukt wird in den jeweiligen Dokumentationen der Modelle genauer beschrieben (siehe Kapitel 4.4 und 4.5).
Beiden Modellen liegt das gleiche Infile zugrunde, in dem die Werte für die globalen Variablen festgelegt sind.

```
0.48  dap1
0.73  m1
0.35  rs
0.57  dap2
0.42  bs1
0.20  et_nt
0.42  m2
0.53  logik
0.48  wr
0.34  swt
0.27  bs2
0.38  is
0.11  iug
0.00  uebau
0.35  swk
0.01  mund_pr
18    hapra
16    sopra
```

Die in dem Kommentar des Infile vorkommenden und dort kleingeschriebenen Abkürzungen haben die folgende Bedeutung:

DAP1:	Datenstrukturen, Algorithmen und Programmierung 1
M1:	Mathematik für Informatiker 1
RS:	Rechnerstrukturen
DAP2:	Datenstrukturen, Algorithmen und Programmierung 2
BS1:	Betriebssysteme, Rechnernetze und verteilte Systeme 1
ETNT:	Elektrotechnik und Nachrichtentechnik
M2:	Mathematik für Informatiker 2
Logik:	Logik für Informatiker
WR:	Wahrscheinlichkeitsrechnung und mathematische Statistik für Informatiker
SWT:	Softwaretechnik

BS2:	Betriebssysteme, Rechnernetze und verteilte Systeme 2
IS:	Informationssysteme
IUG:	Informatik und Gesellschaft
UEBAU:	Übersetzerbau
SWK:	Softwarekonstruktion
MUENDL:	global für alle mündliche Prüfungen
HAPRA:	Hardwarepraktikum
SOPRA:	Softwarepraktikum

Diese Bezeichnungen werden genau so in beiden Modellen verwendet. Die letzten beiden Werte (hapra und sopra) stehen für die Kapazitäten in den beiden Praktika. Alle anderen Werte sind die Durchfallquoten in den jeweiligen Veranstaltungen.

4.4. Implementierung und Verifikation des HIT- Modells für den Realcase

Die Struktur des HIT-Modells ist in der Abbildung 4.5 zu sehen (Object Structure von HIT). Die Abbildung stellt aus Gründen der Übersicht nur Ausschnitte des Modells dar, enthält aber genug Komponenten, um den hierarchischen Aufbau zu veranschaulichen.

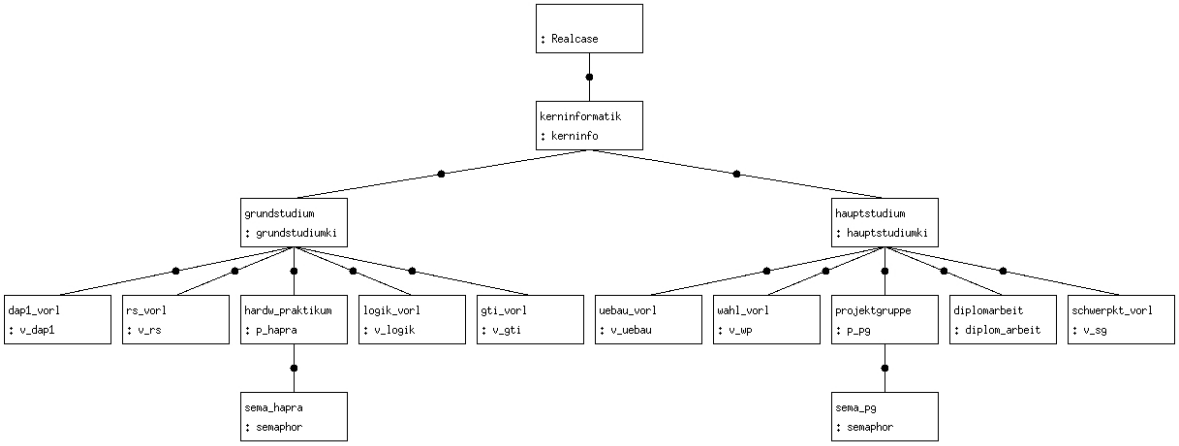


Abbildung 4.5.: Struktur des HIT-Modells

Auf der obersten Hierarchieebene befindet sich das Modell vom Typ *Realcase*. Als hierarchisch oberster Komponententyp fungiert die *kerninfo*, welche dem Studierenden die Lehre anbietet. Gemäß den Annahmen (Kapitel 4.2.2) werden die anderen beiden Studiengänge Angewandte Informatik und Lehramtinformatik ausgespart. Deshalb werden im Folgenden die Begriffe Kerninformatik und Informatik synonym verwendet.
Bei der Kerninformatik unterteilt sich das Studium in Grund- und Hauptstudium (*grundstudiumki*, *hauptstudiumki*). In diesen beiden Komponententypen werden alle in der DPO 2001 vorgesehenen Veranstaltungen durch eine Komponente repräsentiert. Aufgrund des großen Umfangs der angebotenen Veranstaltungen ist nur eine stellvertretende Auswahl in der Abbildung 4.5 zu sehen.

Nun wird das Modell vom Typ *Realcase* und die einzelnen Komponententypen beschrieben. Diese wären *kerninfo*, *grundstudiumki*, *hauptstudiumki*, *v_gti* als Beispiel für eine Vorlesung mit einer mündlichen Fachprüfung, *v_dap1* als Beispiel für eine Vorlesung mit einer schriftlichen Fachprüfung, *v_logik* als Beispiel für eine Vorlesung, in der ein Leistungsnachweis zu erwerben ist, *p_hapra* als Beispiel für die beiden Praktika HaPra und SoPra, *p_pg* (Projektgruppe) und abschließend der die Diplomarbeit repräsentierende Komponententyp *diplom_arbeit*.
Die Komponententypen für Vorlesungen, Praktika und Diplomarbeit wurden aus Gründen der Übersichtlichkeit erstellt. Man hätte ihren Code in den Komponententypen *grundstudiumki* und *hauptstudiumki* unterbringen können. Die Performanz des Gesamtmodells leidet nur geringfügig unter dieser Entscheidung.

4.4.1. Realcase

Auf dieser Ebene befindet sich der Student, der im Studiengang Informatik studiert und sein Abschluss zu erwerben sucht.

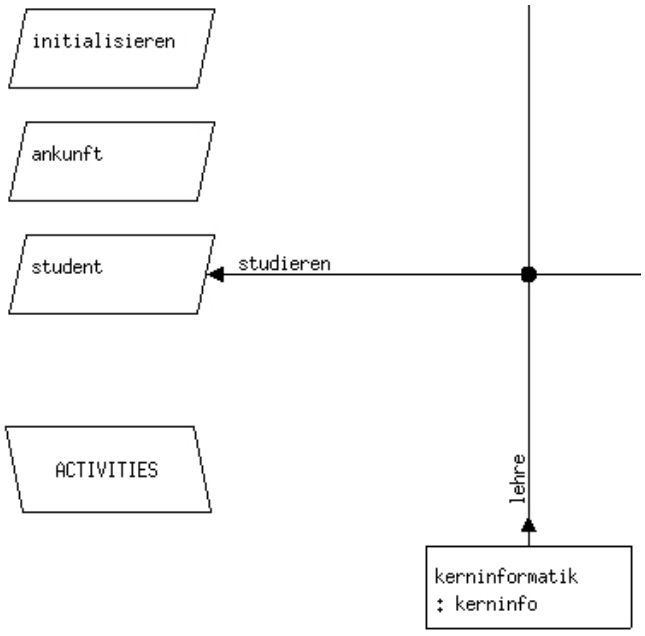


Abbildung 4.6.: Realcase

Dazu ruft der Service *student* den used-service *studieren* auf, welcher mit dem provided-service *lehre* der Komponente *kerninformatik* verknüpft ist. Dies wird in Abbildung 4.6 dargestellt. Außerdem werden hier die globalen Variablen deklariert, welche die für alle Prozesse geltenden Durchfallwahrscheinlichkeiten verwalten. Der Code sieht wie folgt aus :

```
{Durchfallquoten}
VARIABLE dap1  : REAL  DEFAULT 0.0;
VARIABLE m1    : REAL  DEFAULT 0.0;
VARIABLE rs    : REAL  DEFAULT 0.0 ;

VARIABLE dap2  : REAL  DEFAULT 0.0 ;
```

```
VARIABLE bs1   : REAL  DEFAULT 0.0 ;
VARIABLE et_nt : REAL  DEFAULT 0.0 ;
VARIABLE m2    : REAL  DEFAULT 0.0 ;

VARIABLE logik : REAL  DEFAULT 0.0 ;
VARIABLE wr    : REAL  DEFAULT 0.0 ;
VARIABLE swt   : REAL  DEFAULT 0.0 ;
VARIABLE bs2   : REAL  DEFAULT 0.0 ;

VARIABLE is    : REAL  DEFAULT 0.0 ;

VARIABLE iug   : REAL  DEFAULT 0.0 ;
VARIABLE uebau : REAL  DEFAULT 0.0 ;
VARIABLE swk   : REAL  DEFAULT 0.0 ;

VARIABLE mund_pr : REAL DEFAULT 0.0 ;

{Praktikaplaetze}
VARIABLE hapra : INTEGER ;
VARIABLE sopra : INTEGER ;

VARIABLE pg_anwaerter    : INTEGER ;
VARIABLE hapra_anwaerter : INTEGER ;
VARIABLE sopra_anwaerter : INTEGER ;

{I/O Dateien}
VARIABLE in  : INFILE  ;
VARIABLE sem : OUTFILE ;

{Hilfskonstrukte}
VARIABLE diplomanden     : INTEGER ;
VARIABLE swd             : INTEGER ;
```

Es werden die Variablen für die Durchfallquoten der einzelnen Vorlesungen, die Kapazitäten der einzelnen Praktika, Hilfskonstrukte und I/O Dateien deklariert. Mit dem in den *Activities* einmalig aufgerufenen Dienst *initialisieren* (CREATE 1 PROCESS initialisieren ;) werden die Durchfallquoten und die Anzahl der Praktikaplätze aus dem Infile (siehe Kapitel 4.3) eingelesen. Der Code sieht wie folgt aus :

```
OPEN in , "WERTE" LENGTH 80;

READ FILE in, dap1 ; READLN FILE in;
READ FILE in, m1   ; READLN FILE in;
READ FILE in, rs   ; READLN FILE in;
READ FILE in, dap2 ; READLN FILE in;
READ FILE in, bs1  ; READLN FILE in;
READ FILE in, et_nt; READLN FILE in;
READ FILE in, m2   ; READLN FILE in;
```

```
READ FILE in, logik; READLN FILE in;
READ FILE in, wr   ; READLN FILE in;
READ FILE in, swt  ; READLN FILE in;
READ FILE in, bs2  ; READLN FILE in;
READ FILE in, is   ; READLN FILE in;
READ FILE in, iug  ; READLN FILE in;
READ FILE in, uebau; READLN FILE in;
READ FILE in, swk  ; READLN FILE in;
READ FILE in, mund_pr; READLN FILE in;
READ FILE in, hapra; READLN FILE in;
READ FILE in, sopra; READLN FILE in;

sopra_anwaerter := 0 ;
hapra_anwaerter := 0 ;
pg_anwaerter    := 0 ;

CLOSE in;
```

Nachdem die Datei mit dem *OPEN* Befehl geöffnet wurde, wird die Datei zeilenweise eingelesen, wobei die Kommentare, die der Übersichtlichkeit halber angeben, um welche Durchfallquote es sich speziell handelt, mit dem *READLN* Befehl übersprungen werden. Die drei globalen Variablen *sopra_anwaerter*, *hapra_anwaerter* und *pg_anwaerter* werden zudem mit dem Wert Null initialisiert, um sicherzustellen, dass bei mehreren Experimentläufen ihre Anzahl korrekt berechnet wird. Abschließend wird die Datei mit dem *CLOSE* Befehl geschlossen.
Jedes Jahr schreiben sich zwischen 16 und 65 Studentencluster ein, was mit Hilfe der diskreten Gleichverteilung (randint) realisiert wird. Somit wird alle zwei Semester 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(16,65) PROCESS student;
  spend(2);
END LOOP;
```

Diese Schleife erzeugt alle zwei Semester eine Anzahl an Prozessen *student*, welche gleichverteilt in dem Intervall [16,65] liegt. Die Grenzen dieses Intervalls wurden nicht als Modellparameter spezifiziert, da davon ausgegangen wird, dass sich diese nicht ändern.

4.4.2. kerninfo

Der in der Kerninformatik angebotene Dienst *lehre* ruft mit den beiden used-services *grundstudium* und *hauptstudium* die gleichnamigen provided-services der Komponenten *grundstudium* und *hauptstudium* (vom Typ *grundstudiumki* und *hauptstudiumki*) auf. Dieser Sachverhalt ist in der Abbildung 4.7 dargestellt. Weiterhin sind die selbstdefinierten Ströme *absolventen* und *schwund* vom Typ *Event* zu sehen. Der Dienst *update_schwund* dient nur dem Update des Stroms *schwund* und wird jedes Semester einmal aufgerufen. Der Strom *absolventen* misst die durchschnittliche Studiendauer der Prozesse und der Strom *schwund* misst die Anzahl der Prozesse, die wegen Exmatrikulation das Systems verlassen.

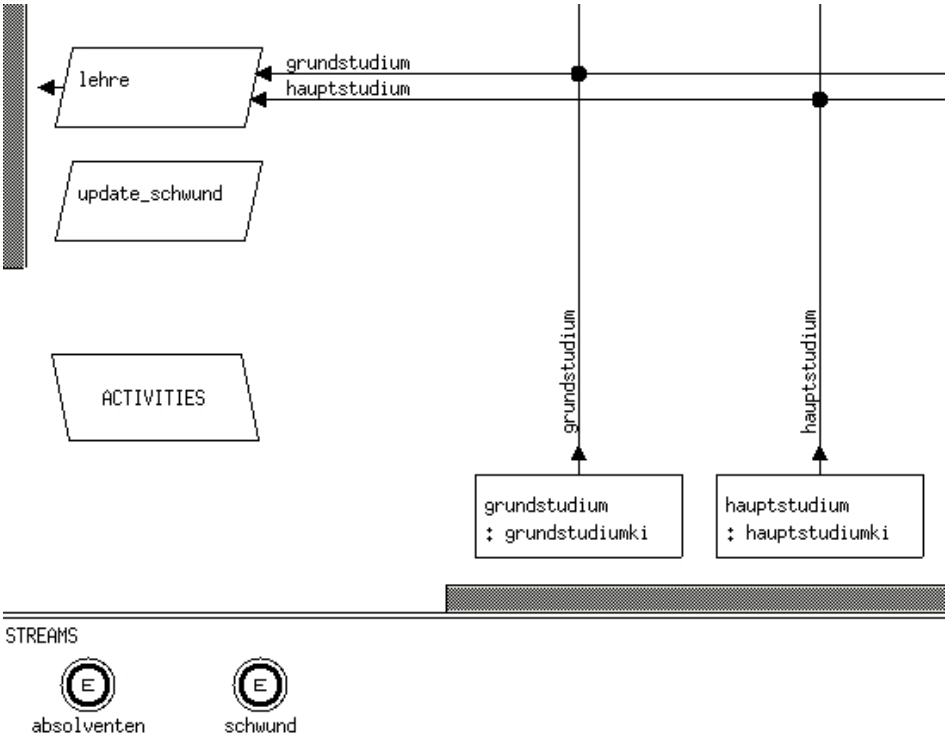


Abbildung 4.7.: kerninfo

Der Dienst *lehre* ist folgendermaßen implementiert:

```
(mat,zeit) := grundstudium;
IF NOT mat THEN
swd := swd +1;
END IF;
IF mat THEN
 (mat,zeit) := hauptstudium(zeit);
IF NOT mat THEN
swd := swd +1;
END IF;
END IF;
IF mat THEN
  UPDATE absolventen BY (zeit);
END IF;
```

Die lokal definierten Variablen *mat* vom Typ Boolean und *zeit* vom Typ Integer sind Hilfskonstrukte, die zum einen den Matrikulationsstatus und zum anderen die Studiendauer in Semestern angeben. In der ersten Zeile werden beide Variablen durch die Rückgabe des Dienstaufrufes *grundstudium* gesetzt. Nur wenn der Student noch immatrikuliert ist, wird der Dienst *hauptstudium* mit dem Parameter *zeit*, welcher die Grundstudiumsdauer übergibt, aufgerufen. Wenn der Student nach diesem Aufruf weiterhin immatrikuliert ist, und folglich sein Diplom mit Erfolg erhalten hat, wird dem Stream *absolventen* die Studiendauer des Prozesses übergeben. Die globale Variable *swd* zählt alle exmatrikulierten Prozesse. Die Variable *mat* muss zweimal überprüft werden, da Prozesse sowohl im Grundstudium als auch im Hauptstudium exmatrikuliert werden können.

4.4.3. grundstudiumki

Die in der Abbildung 4.8 zu sehenden used-services des Dienstes *grundstudium dap1_v* bis *is_v* stellen die im Grundstudium angebotenen und mit den Annahmen (Kapitel 4.2.2) konform gehenden Veranstaltungen gemäß DPO 2001 ([DPO 01]) dar, welche durch je eine eigene Komponente vom entsprechendem Typ repräsentiert werden, die aber nicht alle im Ausschnitt der Abbildung 4.8 zu sehen sind. So ruft zum Beispiel der used-service *dap1_v* den provided-service *pruefung* der Komponente *dap1_vorl* vom Typ *v_dap1* auf.

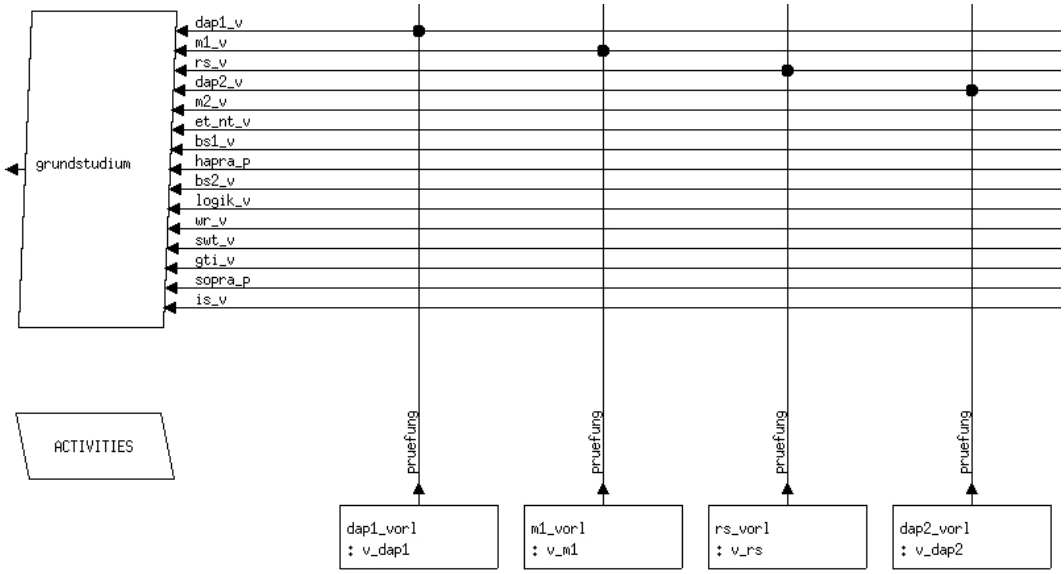


Abbildung 4.8.: grundstudiumki

Der Code und die dafür benötigten Variablendeklaration für den Dienst *grundstudium* werden nun vollständig dargestellt, um anschließend in Teilen detailliert erläutert zu werden.

Variablendeklaration:

```
{Hilfskonstrukte}
 VARIABLE winter   : BOOLEAN DEFAULT TRUE ;
 VARIABLE vordipl_b: BOOLEAN DEFAULT FALSE ;

{Variablen zur Bestimmung bestandener Prüfungen}
 VARIABLE dap1_b  : BOOLEAN DEFAULT FALSE ;
 VARIABLE dap2_b  : BOOLEAN DEFAULT FALSE ;
 VARIABLE swt_b   : BOOLEAN DEFAULT FALSE ;
 VARIABLE rs_b    : BOOLEAN DEFAULT FALSE ;
 VARIABLE m1_b    : BOOLEAN DEFAULT FALSE ;
 VARIABLE m2_b    : BOOLEAN DEFAULT FALSE ;
 VARIABLE logik_b : BOOLEAN DEFAULT FALSE ;
 VARIABLE wr_b    : BOOLEAN DEFAULT FALSE ;
```

```
 VARIABLE et_nt_b : BOOLEAN DEFAULT FALSE ;
 VARIABLE bs1_b   : BOOLEAN DEFAULT FALSE ;
 VARIABLE bs2_b   : BOOLEAN DEFAULT FALSE ;
 VARIABLE is_b    : BOOLEAN DEFAULT FALSE ;
 VARIABLE gti_b   : BOOLEAN DEFAULT FALSE ;
 VARIABLE hapra_b : BOOLEAN DEFAULT FALSE ;
 VARIABLE sopra_b : BOOLEAN DEFAULT FALSE ;

{Variable, die überprüft, ob der Student eingeschrieben ist.}
 VARIABLE mat  : BOOLEAN DEFAULT TRUE ;

{Variable, die die Studiendauer speichert.}
 VARIABLE zeit : INTEGER DEFAULT 1 ;
```

Die boolesche Variable *winter* bestimmt für jeden Prozess, ob er sich im Winter- oder Sommersemester befindet. Diese Variable wurde nicht global definiert, um potentiell auftretende Synchronisationsprobleme zu vermeiden. Die boolesche Variable *vordipl_b* gibt an, ob das Vordiplom erfolgreich bestanden wurde. Die nächsten 15 Variablen geben an, ob die jeweilige Veranstaltung bestanden wurde. Zuletzt werden die beiden bereits erwähnten Variablen *mat* und *zeit* deklariert.

Code:

```
BRANCH
PROB 0.2 : hapra_b := TRUE ;
ELSE     : hapra_b := FALSE;
END BRANCH;

WHILE mat AND NOT vordipl_b
  LOOP

      spend(0.1);

      IF NOT hapra_b AND  rs_b AND zeit > 2 THEN
        hapra_anwaerter :=  hapra_anwaerter + 1 ;
      END IF;


      IF NOT sopra_b  AND  dap1_b AND dap2_b AND swt_bgrund
         AND ( zeit MOD 2 = 0 ) AND  (zeit > 3) THEN
         sopra_anwaerter :=  sopra_anwaerter + 1 ;
      END IF;

      spend(0.9);

  IF winter THEN

        IF NOT dap1_b  THEN
```

```
  IF zeit = 1 THEN
    dap1_b := dap1_v(0);
    ELSE
     dap1_b := dap1_v(1);
    IF NOT dap1_b THEN
        mat := FALSE ;
     END IF;
   END  IF;
  END IF;


IF NOT hapra_b AND rs_b AND(zeit >=3 ) THEN
hapra_b := hapra_p;
END IF;

 IF NOT rs_b  THEN
  IF zeit = 1 THEN
    rs_b := rs_v(0);
    ELSE
     rs_b := rs_v(1);
        IF NOT rs_b THEN
        mat := FALSE ;
     END IF;
   END  IF;
  END IF;


 IF NOT m1_b  THEN
    m1_b := m1_v;
 END IF;


 IF NOT bs2_b  THEN
   IF zeit = 3 THEN
      bs2_b := bs2_v(0);
   ELSE
      IF zeit = 5 THEN
         bs2_b := bs2_v(1);
           IF NOT bs2_b THEN
           mat := FALSE ;
           END IF;
      END IF;
   END IF;
 END IF ;


 IF NOT logik_b AND (zeit >=3)  THEN
```

```
         logik_b := logik_v;
       END IF;


       IF NOT wr_b AND (zeit >=3) THEN
          wr_b := wr_v;
       END IF;


       IF NOT swt_b AND (zeit >=3) THEN
          swt_b := swt_v;
       END IF;


ELSE    {Sommersemester}

     IF NOT hapra_b AND rs_b AND (zeit >= 3) THEN
          hapra_b := hapra_p;
     END IF;


     IF NOT sopra_b AND dap1_b AND dap2_b AND swt_b
        AND (zeit >= 4 ) THEN
         sopra_b := sopra_p ;
     END IF;


     IF NOT dap2_b  THEN
       IF zeit = 2 THEN
         dap2_b := dap2_v(0);
         ELSE
          dap2_b := dap2_v(1);
          IF NOT dap2_b THEN
            mat := FALSE ;
          END IF;
        END  IF;
     END IF;


     IF NOT m2_b  THEN
       IF zeit = 2 THEN
         m2_b := m2_v(0);
         ELSE
          m2_b := m2_v(1);
          IF NOT m2_b THEN
            mat := FALSE ;
          END IF;
```

```
        END  IF;
     END IF;


     IF NOT is_b THEN
        IF zeit = 4 THEN
             is_b := is_v(0);
        ELSE
          IF zeit = 6 THEN
             is_b := is_v(1);
             IF NOT is_b THEN
               mat := FALSE ;
            END IF;
          END IF;grund
        END IF;
     END IF;


     IF NOT gti_b AND zeit = 4 THEN
         gti_b := gti_v;
         IF NOT gti_b THEN
         mat := FALSE ;
         END IF ;
     END IF;


    IF NOT bs1_b AND (zeit >=2) THEN
         bs1_b := bs1_v;
    END IF;


    IF NOT et_nt_b AND (zeit >=2) THEN
         et_nt_b := et_nt_v;
    END IF;


END IF;

IF dap1_b AND  dap2_b AND swt_b AND rs_b AND  m1_b AND m2_b
   AND logik_b AND  wr_b AND et_nt_b AND bs1_b AND bs2_b
   AND is_b AND  gti_b  AND hapra_b AND sopra_b THEN
vordipl_b := TRUE;
END  IF;

IF NOT vordipl_b THEN
  zeit := zeit + 1 ;
```

```
    IF zeit MOD 2 = 0 THEN
      winter := FALSE ;
    ELSE
      winter := TRUE ;
    END IF;
  END IF;

 IF zeit > 20 THEN
   mat := FALSE ;
 END IF;

 END LOOP;

RESULT mat,zeit ;
```

Gemäß den Annahmen (siehe Kapitel 4.2.2) nehmen 20% der Studenten aufgrund ihrer Nebenfachwahl nicht am HaPra teil, so dass 20% der Prozesse im Modell das HaPra zu Beginn als bestanden angerechnet wird - zu sehen im nachstehenden Code:

```
BRANCH
PROB 0.2 : hapra_b := TRUE ;
ELSE     : hapra_b := FALSE;
END BRANCH;
```

Danach tritt der Prozess in eine WHILE-Schleife, die den Ablauf des Grundstudiums abbildet, ein, welche er nur mit einem bestandenen Vordiplom oder durch Exmatrikulation verlassen kann.

```
WHILE mat AND NOT vordipl_b
  LOOP
   .
   .
  END LOOP;
```

Zu Beginn der Schleife wird die Anzahl, der auf Praktika wartenden Prozesse, gezählt. Dabei muss der Prozess die jeweiligen Voraussetzungen erbracht, das Praktikum noch nicht bestanden und eine bestimmte Zeit studiert haben. So muss der Prozess zum Beispiel für das HaPra die Vorlesung Rechnerstrukturen (rs) bestanden haben und sich mindestens im dritten Semester befinden.
Um sicher zu stellen, dass alle anfordernden Prozesse angekommen und gezählt worden sind, geschieht das Zählen erst nach einer minimalen Verzögerung (spend (0.1);), welche nach dem Zählen zu einem ganzen Semester vervollständigt wird.

```
      spend(0.1);

      IF NOT hapra_b AND  rs_b AND zeit > 2 THEN grund
        hapra_anwaerter :=  hapra_anwaerter + 1 ;
      END IF;


      IF NOT sopra_b  AND  dap1_b AND dap2_b AND swt_b
```

```
         AND NOT (winter) AND  (zeit > 3) THEN
         sopra_anwaerter :=  sopra_anwaerter + 1 ;
      END IF;
   grund
      spend(0.9);

      IF winter THEN
         .
         .
      ELSE {sommersemester}
         .
         .
      END IF;
```

Anschließend überprüft eine IF-Anweisung anhand der Variable *winter*, ob sich der Prozess im Winter- oder Sommersemester befindet. In beiden Blöcken werden die entsprechenden Veranstaltungen sequentiell mit je einem IF-Block abgearbeitet. Die einzelnen IF-Blöcke bestehen aus Nullzeitanweisungen, so dass trotz der sequentiellen Abarbeitung die Veranstaltungen in dem jeweiligen Semester parallel ablaufen. Durch die Nullzeit ergibt sich ein Problem, wenn sich Veranstaltungen im gleichen Semesterblock (Winter- bzw. Sommersemester) befinden und voneinander abhängen (Beispiel: RS und HaPra). Es muss auf die richtige Reihenfolge in der Sequenz geachtet werden, um zu verhindern, dass beide Veranstaltungen im gleichen Semester ablaufen. Das bedeutet, dass Veranstaltungen vor ihren Voraussetzungen in der Sequenz zu finden sind (im Wintersemester HaPra vor RS, im Sommersemester SoPra vor DAP2).
Wie bereits erwähnt, stehen die einzelnen IF-Blöcke für je eine Veranstaltung. Im Folgenden werden vier verschiedene repräsentative IF-Blöcke behandelt, die stellvertretend für eine schriftliche Fachprüfung (DAP1), eine mündliche Fachprüfung (GTI), einen Leistungsnachweis (Logik) und ein Praktikum (HaPra) sind.

mündliche Fachprüfung Hier werden zuerst die Voraussetzungen überprüft. Diese sind zum einen die Kontrolle, ob GTI schon bestanden wurde und zum anderen, ob der Prozess sich im vierten Semester befindet. Es gibt hierbei keine Unterscheidung zwischen den drei möglichen Prüfungsversuchen, da alle drei Versuche in einem Semester absolviert werden können. Die Prüfungsversuche werden durch den Aufruf *gti_v* angefordert, welcher der Variablen *gti_b* einen Wahrheitswert abhängig von dem Bestehen zurückliefert. Falls der Prozess durch alle drei Versuche durchfällt, wird er exmatrikuliert (mat := FALSE ;).

```
IF NOT gti_b AND zeit = 4 THEN
       gti_b := gti_v;
       IF NOT gti_b THEN
          mat := FALSE ;
       END IF ;
END IF;
```

schriftliche Fachprüfung Auch hier wird am Anfang überprüft, ob DAP1 bereits bestanden wurde. Für den Fall, dass es noch nicht bestanden wurde, wird der Semesterfortschritt kontrolliert. Wenn

sich der Prozess im ersten Semester befindet, nimmt er seinen ersten und gegebenenfalls zweiten Prüfungsversuch wahr, indem der Dienst dap1_v(0) aufgerufen wird. Ansonsten absolviert er seinen dritten Versuch (dap1_v(1)). Abhängig von dem Übergabeparameter 0 oder 1, erkennt die entsprechende Komponente, ob es sich um die ersten zwei Prüfungsversuche oder den dritten Versuch handelt. In beiden Fällen weisen die Aufrufe der Variable *dap1_b* einen Wahrheitswert zu, ob die Prüfung bestanden wurde oder nicht. Falls der Prozess durch den dritten Versuch durchfällt, wird er analog zu der mündlichen Fachprüfung exmatrikuliert.

```
IF NOT dap1_b  THEN
       IF zeit = 1 THEN
          dap1_b := dap1_v(0);
        ELSE
          dap1_b := dap1_v(1);
             IF NOT dap1_b THEN
                mat := FALSE ;
             END IF;
        END IF;
END IF;
```

Leistungsnachweis Die Überprüfung der Voraussetzungen geschieht wie bereits beschrieben. Wenn diese erfüllt sind, wird der entsprechende Dienst aufgerufen. Der Unterschied zu den Fachprüfungen ist der, dass der Prozess in dem jeweiligen Semester nur zwei Prüfungsversuche hat und bei Nicht-Bestehen keine Exmatrikulation stattfindet.

```
IF NOT logik_b AND (zeit >=3)  THEN
       logik_b := logik_v;
END IF;
```

Praktikum Das Praktikum ist vom Code her analog zu einem Leistungsnachweis aufgebaut. Jedoch handelt es sich bei dem entsprechenden Dienstaufruf um das Beantragen eines Praktikumsplatzes.

```
IF NOT hapra_b AND rs_b AND (zeit >= 3) THEN
       hapra_b := hapra_p;
END IF;
```

Nach der Abarbeitung der Blöcke für das Winter- und Sommersemester wird überprüft, ob alle Voraussetzungen für das Vordiplom erfüllt sind. Wenn dies der Fall ist, so sind die booleschen Variablen auf TRUE gesetzt, so dass die *vordipl_b* Variable ebenfalls auf TRUE gesetzt wird.
Anschließend überprüft der Prozess, ob er sein Vordiplom bestanden hat. Für den Fall, dass er es nicht bestanden hat, wird die Semesteranzahl um eins inkrementiert und abhängig von der aktuellen Semesteranzahl wird entschieden, ob er sich im Winter- oder Sommersemester befindet.
Die letzte IF-Abfrage exmatrikuliert den Prozess gemäß den Annahmen (Kapitel 4.2.2), sobald er mehr als 20 Semester studiert hat.

```
IF dap1_b AND  dap2_b AND swt_b AND rs_b AND  m1_b AND m2_b
   AND logik_b AND  wr_b AND et_nt_b AND bs1_b AND bs2_b
   AND is_b AND  gti_b  AND hapra_b AND sopra_b THEN
   vordipl_b := TRUE;
```

```
END IF;

IF NOT vordipl_b THEN
   zeit := zeit + 1 ;

   IF zeit MOD 2 = 0 THEN
      winter := FALSE ;
   ELSE
      winter := TRUE ;
   END IF;
END IF;

IF zeit > 20 THEN
   mat := FALSE ;
END IF;
```

Sobald der Prozess die Schleife verlässt, übergibt er den Status der Matrikulation und die Semesteranzahl durch den Ausdruck *RESULT mat, zeit* an den Dienst *lehre* in der Komponente *kerninformatik*.

4.4.4. hauptstudiumki

Der Aufbau des Hauptstudiums (Abbildung: 4.9) ist analog zu dem des Grundstudiums. Wie auch beim *grundstudiumki* wird nur ein Ausschnitt in der Abbildung 4.9 dargestellt.

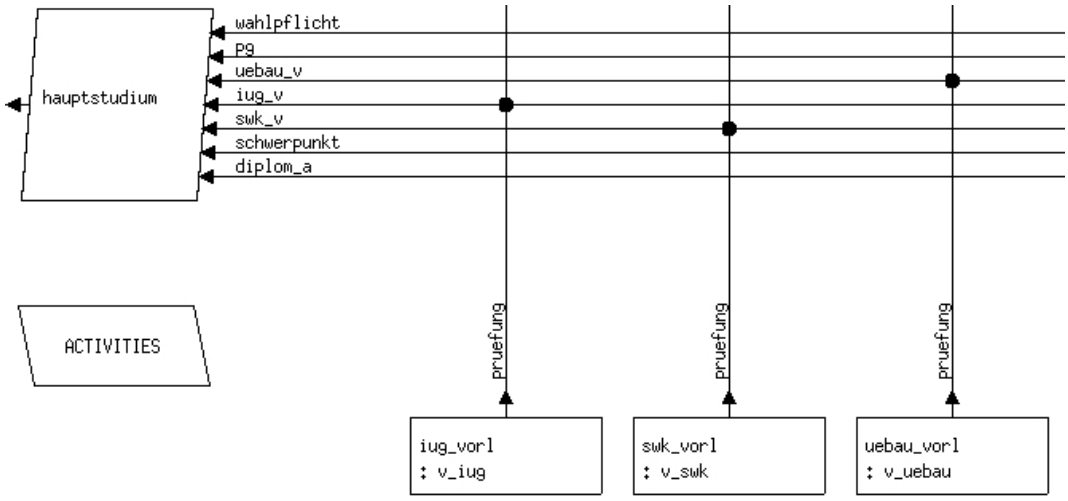


Abbildung 4.9.: hauptstudiumki

Dies gilt in großen Teilen auch für den nun folgenden Code.

Variablendeklaration:

```
{Hilfskonstrukte}
 VARIABLE winter   : BOOLEAN DEFAULT TRUE;
```

```
 VARIABLE diplom   : BOOLEAN DEFAULT FALSE;
 VARIABLE vd       : INTEGER ;

{Variablen zur Bestimmung bestandener Prüfungen}
 VARIABLE pg_b   : BOOLEAN DEFAULT FALSE ;
 VARIABLE pg_b1  : BOOLEAN DEFAULT FALSE ;
 VARIABLE wp_b1  : BOOLEAN DEFAULT FALSE ;
 VARIABLE wp_b2  : BOOLEAN DEFAULT FALSE ;
 VARIABLE wp_b3  : BOOLEAN DEFAULT FALSE ;
 VARIABLE sg_b1  : BOOLEAN DEFAULT FALSE ;
 VARIABLE sg_b2  : BOOLEAN DEFAULT FALSE ;
 VARIABLE sg_b3  : BOOLEAN DEFAULT FALSE ;
 VARIABLE sg_b4  : BOOLEAN DEFAULT FALSE ;
 VARIABLE swk_b  : BOOLEAN DEFAULT FALSE ;
 VARIABLE iug_b  : BOOLEAN DEFAULT FALSE ;
 VARIABLE ueb_b  : BOOLEAN DEFAULT FALSE ;
 VARIABLE diplomarbeit_b   : BOOLEAN DEFAULT FALSE ;

{Variable, die überprüft, ob der Student eingeschrieben ist.}
 VARIABLE mat    : BOOLEAN DEFAULT TRUE ;
```

Die Deklarationen der Hauptstudiumsveranstaltungen sind analog zu denen im Grundstudium. Die Unterschiede bestehen darin, dass die Variable *zeit* als formaler Parameter deklariert ist (VARIABLE zeit : INTEGER;), da sie übergeben wurde. Zusätzlich wurde eine neue Integer Variable *vd* deklariert, die angibt, in welcher Zeit das Vordiplom bestanden wurde.

Code:

```
vd := zeit ;

WHILE mat AND NOT diplom

LOOP
     zeit :=  zeit + 1 ;

     IF zeit MOD 2 = 0 THEN

     winter := FALSE ;
     ELSE
     winter := TRUE ;
     END IF;


     spend(0.1);

     IF NOT pg_b1 THEN
        pg_anwaerter :=  pg_anwaerter + 1;
```

```
     END IF;

     spend(0.9);

IF winter THEN

    IF pg_b AND (zeit > (vd + 4) ) THEN
     diplomarbeit_b := diplom_a ;
    END IF;


    IF pg_b1 THEN
     pg_b := TRUE;
    END IF;


    IF NOT pg_b1 THEN
     pg_b1 := pg ;
    END IF;


    IF NOT wp_b1  THEN
       wp_b1 := wahlpflicht;
           IF NOT wp_b1 THEN
             mat := FALSE;
           END IF;grund
     END IF;


    IF NOT wp_b2  THEN
       wp_b2 := wahlpflicht;
          IF NOT wp_b2 THEN
            mat := FALSE;
          END IF;
    END IF;


    IF NOT sg_b1 AND ( zeit > (vd +2) ) THEN
     sg_b1 := schwerpunkt;
      IF NOT sg_b1 THEN
         mat := FALSE;grund
      END IF;
    END IF;


    IF NOT sg_b2 AND ( zeit > (vd +2) ) THEN
      sg_b2 := schwerpunkt;
```

```
        IF NOT sg_b2 THEN
          mat := FALSE;
        END IF;
    END IF;

    IF NOT swk_b THEN
     swk_b := swk_v;
    END IF;
grund
ELSE

    IF pg_b AND (zeit > (vd + 4) ) THEN
     diplomarbeit_b := diplom_a ;
    END IF;


    IF pg_b1 THEN
     pg_b := TRUE;
    END IF;


    IF NOT pg_b1 THEN
     pg_b1 := pg ;
    END IF;


    IF NOT ueb_b THEN
     ueb_b := uebau_v ;
    END IF;


    IF NOT iug_b THEN
     iug_b := iug_v ;
    END IF;


    IF NOT wp_b3  THEN
       wp_b3 := wahlpflicht;
         IF NOT wp_b3 THEN
           mat := FALSE;
         END IF;
    END IF;


    IF NOT sg_b3 AND ( zeit > (vd +2) ) THEN
       sg_b3 := schwerpunkt;
        IF NOT sg_b3 THEN
```

```
            mat := FALSE;
         END IF;
      END IF;


      IF NOT sg_b4 AND ( zeit > (vd +2) ) THEN
         sg_b4 := schwerpunkt;
          IF NOT sg_b4 THEN
             mat := FALSE;
          END IF;
      END IF;

END IF;

IF pg_b AND swk_b AND ueb_b AND iug_b AND wp_b1 AND wp_b2 AND wp_b3
   AND sg_b1 AND sg_b2 AND sg_b3 AND sg_b4 AND diplomarbeit_b THEN
   diplom := TRUE ;
END IF;

END LOOP;

RESULT mat,zeit ;
```

Zu Beginn wird der *vd* Variable die Zeit zugewiesen, die der Prozess für das Grundstudium benötigt hat (vd := zeit;). Der restliche Code besteht aus einer WHILE-Schleife, die ebenfalls einen Block für das Winter- und Sommersemester enthält, in denen die jeweiligen Veranstaltungen des Hauptstudiums sequentiell (mit den richtigen Abhängigkeiten) aufgeführt sind. Lediglich die Passage zu Beginn der Schleife weicht von der des Grundstudiums ab. Im Gegensatz zum Grundstudium wird die Semesteranzahl bereits anfangs um eins erhöht und davon abhängig das Winter- oder Sommersemester ermittelt. Danach wird analog zu den Praktika im Grundstudium, die Anzahl der, auf eine Projektgruppe wartenden, Prozesse bestimmt.

```
WHILE mat AND NOT diplom

LOOP
     zeit :=  zeit + 1 ;

     IF zeit MOD 2 = 0 THEN

     winter := FALSE ;
     ELSE
     winter := TRUE ;
     END IF;

     spend(0.1);

     IF NOT pg_b1 THEN
```

```
        pg_anwaerter :=  pg_anwaerter + 1;
    END IF;

    spend(0.9);
    .
    .
    .
END LOOP;
```

Im Anschluss werden nun die Komponententypen der bereits erwähnten repräsentativen Beispiel-Veranstaltungen dargestellt.

4.4.5. v_gti

Der Komponententyp enthält nur den Dienst *pruefung* (siehe Abbildung 4.10).

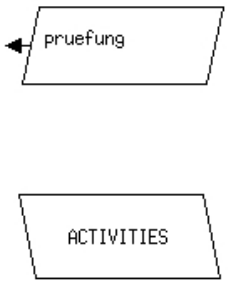


Abbildung 4.10.: v_gti

Der Code sieht wie folgt aus:

```
BRANCH
PROB mund_pr*mund_pr*mund_pr : RESULT FALSE;
ELSE                         : RESULT TRUE;
END BRANCH;
```

Mit der Annahme, dass man die drei möglichen Prüfungsversuche in einem Semester absolviert, ergibt sich die endgültige Durchfallwahrscheinlichkeit durch den Ausdruck *mund_pr*mund_pr*mund_pr*, wobei *mund_pr* die Durchfallwahrscheinlichkeit für eine einzelne mündliche Prüfung darstellt. Mit der Gegenwahrscheinlichkeit wird die Prüfung bestanden. Durch den entsprechende RESULT-Befehl wird das Ergebnis der Prüfung an den aufrufenden Dienst übergeben.

4.4.6. v_dap1

Der Aufbau des Komponententyps ist analog zu *v_gti*. Im Unterschied zu einer mündlichen Fachprüfung können nicht alle drei Versuche in einem Semester abgelegt werden. Deshalb wird ein Parameter i vom Typ Integer übergeben, anhand dessen unterschieden werden kann, ob die ersten zwei Versuche (Parameter = 0) oder der finale letzte Versuch (Parameter = 1) bestritten wird. Der Code sieht wie folgt aus:

```
CASE i
WHEN 0 : BRANCH
         PROB dap1*dap1 : RESULT FALSE ;
         ELSE           : RESULT TRUE  ;
         END BRANCH;
WHEN 1 : BRANCH
         PROB dap1    : RESULT FALSE ;
         ELSE         : RESULT TRUE  ;
         END BRANCH;
END CASE;
```

4.4.7. v_logik

Der Aufbau ist wiederum analog zu *v_gti*. Auch die Ermittlung der Durchfallwahrscheinlichkeit und der Gegenwahrscheinlichkeit ist im Code analog geregelt. Wie bereits beschrieben gibt es nur zwei Prüfungstermine (siehe Kapitel 4.4.3 Leistungsnachweis).

```
BRANCH
PROB logik*logik : RESULT FALSE ;
ELSE             : RESULT TRUE ;
END BRANCH;
```

4.4.8. p_hapra

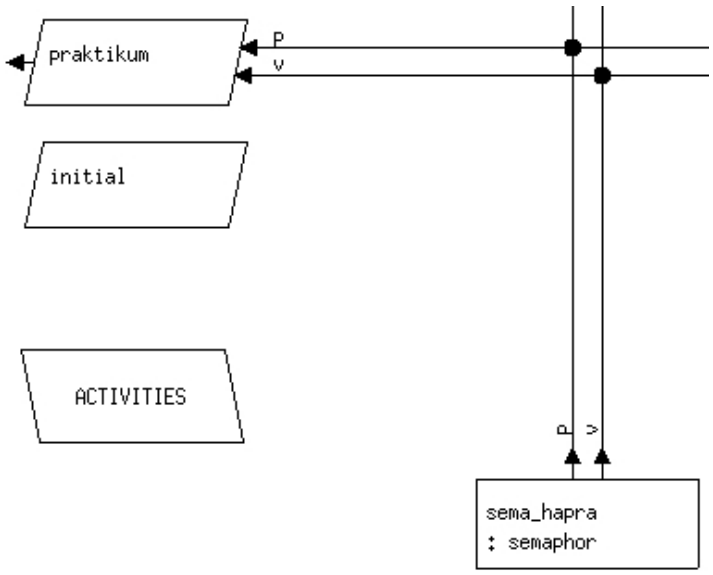


Abbildung 4.11.: p_hapra

Der Komponententyp besteht aus zwei Diensten, welche in der Abbildung 4.11 zu sehen sind, nämlich *initial* und *praktikum*, welcher durch eine Semaphore sicherstellt, dass zu jedem Zeitpunkt nur genau ein Prozess den Code der Komponente abarbeiten kann. Der Dienst *initial* wird in den Activities durch den Aufruf *CREATE 1 PROCESS initial EVERY 1 ;* jedes Semester einmal aufgerufen

und weist zur Hälfte des Semesters der lokalen Variable *h_platz* vom Typ Integer die Kapazität der globalen Variablen *hapra* zu.

Dies ist der dazugehörige Code:

```
spend(0.5);
h_platz := hapra ;
```

Jeder in dem Dienst *praktikum* ankommende Prozess, reserviert zuerst die Semaphore durch den Aufruf *p;*. Sollte die Anzahl der *hapra_anwaerter* kleiner oder gleich der Anzahl zur Verfügung stehender Plätze sein, erhält der Prozess einen Platz und die Anzahl der *hapra_anwaerter* und die Kapazität der HaPra-Plätze werden dekrementiert. Sollten mehr Plätze angefordert als angeboten werden, so wird eine Zufallszahl aus dem Intervall eins bis *hapra_anwaerter* gleichverteilt gezogen und der lokal definierten Variablen *h_zahl* vom Typ Integer zugewiesen. Sollte die gezogene Zufallszahl kleiner oder gleich der Kapazität der HaPra-Plätze sein, so erhält der Prozess wie zuvor beschrieben einen Platz. Andernfalls gilt das Praktikum als nicht bestanden und die Zahl der Anwärter wird um eins dekrementiert.

```
p;

IF ( hapra_anwaerter <= h_platz ) THEN
    hapra_anwaerter := hapra_anwaerter - 1;
    h_platz := h_platz - 1;
    RESULT TRUE;

ELSE

  h_zahl := randint(1, hapra_anwaerter);

    IF ( h_zahl <= h_platz ) THEN
       hapra_anwaerter := hapra_anwaerter - 1;
       h_platz := h_platz - 1 ;
       RESULT TRUE ;

    ELSE

    hapra_anwaerter := hapra_anwaerter - 1;
    RESULT FALSE;

    END IF;

END IF;

v;
```

4.4.9. p_pg

Der Aufbau des Komponententyps *p_pg* ist identisch zu dem der Praktika. Gemäß den Annahmen (Kapitel 4.2.2) werden pro Semester zwischen acht und zwölf Projektgruppen angeboten, was in dem

Code des Dienstes *initial* wie folgt implementiert ist:

```
spend(0.5);
pg_platz := 0 ;
pg_platz :=  randint(8,12);
```

4.4.10. diplom_arbeit

Der in der Abbildung 4.12 zu sehende Dienst *diplom* ist für die Inkrementierung der Zählvariablen *diplomanden* und die Rückgabe des Wertes TRUE verantwortlich. Der Dienst *loeschen* wird einmalig in den Activities angestoßen (CREATE 1 PROCESS loeschen;) und aktualisiert in einer Endlos-Schleife jedes Semester den Stream *diplomarbeiten* und setzt den Wert der Variablen anschließend wieder auf Null. Um sicher zu stellen, dass alle anfordernden Prozesse angekommen und gezählt worden sind, geschieht das Schreiben erst nach einer Verzögerung von einer halben Zeiteinheit.

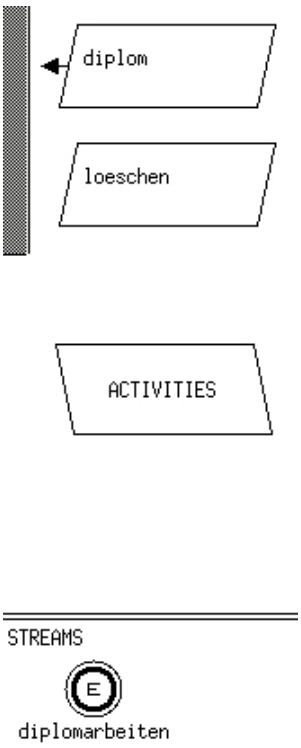


Abbildung 4.12.: diplom_arbeit

Der dazugehörige Code lautet:

```
LOOP
spend(0.5);
UPDATE diplomarbeiten BY diplomanden ;
diplomanden := 0 ;
spend(0.5);
END LOOP;
```

Verifikation des HIT-Modells

Auf eine Verifikation der oberen Ebenen und deren Dienste wird aufgrund ihrer einfachen Struktur verzichtet. Ebenso wird die unterste Ebene, welche alle Vorlesungs-Komponententypen enthält, an dieser Stelle nicht ausführlich behandelt. Sie geht mit der Verifikation des Durchfallverhaltens einher, bei welcher die Durchfallquoten einzelner Vorlesungen mittels des *infiles* so variiert wurden, dass ihr Einfluss auf den selbst definierten Stream *absolventen*, der die TURNAROUNDTIME der Prozesse misst, offenbar wurde. So wird auch die Korrektheit der die Prüfungsversuche unterscheidenden IF-Blöcke des Komponententyps *grundstudiumki* gezeigt. Auch die WHILE-Schleife funktioniert richtig, da diese nach Exmatrikulation oder bestandenem Vordiplom ordnungsgemäß verlassen wird.
Überhaupt beschränkt sich der wesentliche Kern der Verifikation auf den Komponententyp *grundstudiumki*, da dieser den mit Abstand größten Codeumfang aufweist. Der Komponententyp *hauptstudiumki* ist analog aufgebaut, so dass es möglich ist, die Verifikationsergebnisse des Grundstudiums auf das Hauptstudium zu übertragen.
Folgende Szenarien wurden durchgespielt:

1. Bedingung: Durchfallquote jeder Vorlesung 0%; ausreichend Praktika- und PG-Plätze
   TURNAROUNDTIME absolventen: `9.000000`

2. Bedingung: Durchfallquote 100%, für jede Vorlesung einzeln betrachtet
   TURNAROUNDTIME absolventen: `undefined`

3. Bedingung: Durchfallquote beliebig; kein Angebot an Praktikumsplätzen
   TURNAROUNDTIME absolventen: `undefined`

Es ist also ersichtlich, dass bei optimalen Bedingungen eine ideale Durchlaufzeit von neun Semestern erreicht wird, was der Regelstudienzeit entspricht (erste Bedingung), und dass bei all zu beschränkten Ressourcen kein Prozess sein Diplom erhält.
Die Fälle, die hinsichtlich der Durchfallquoten und des Angebotes der Praktika- und PG-Plätze zwischen den Extremen angesiedelt sind, wurden ebenfalls untersucht und subjektiv für plausibel befunden. Eine genauere Untersuchung inklusive Ergebnisse befindet sich in Kapitel 4.7.
Weiterhin wurden auch Spezialfälle getestet, in denen die Durchfallquote einer Fachprüfung in den ersten beiden Versuchen auf 100% und im dritten Versuch auf 0% gesetzt wurde. Ist die Fachprüfung beispielsweise im dritten Fachsemester angesetzt, so wird diese Prüfung erst im fünften Semester bestanden und das Studium entsprechend verzögert.
Unabhängig von den Durchfallquoten gilt es, die Komponententypen für die beiden Praktika und die Projektgruppe separat zu untersuchen. Bei der Verifikation wird an dieser Stelle aus Analogiegründen exemplarisch nur der Komponententyp *p_sopra* betrachtet. Zum einen sieht man, dass bei Nicht-Erbringung der Voraussetzung (Durchfallquote einer benötigten Vorlesung auf 100%) keine Anwärter für das Praktikum gezählt werden. Zum anderen werden speziell beim SoPra, welches nur im Sommersemester angeboten wird, die Anwärter auch nur alle zwei Semester gezählt. Ebenso bekommen, bei zu Testzwecken konstant gesetzter Ankunftsrate und ausreichend bereitgestellten Kapazitäten, alle Prozesse einen Platz, wie aus obigem Bestcase-Szenario ersichtlich ist. Im entgegengesetzten Fall fehlender Ressourcen (Szenario drei) steigt die Anwärterzahl kontinuierlich an.
Der in Kapitel 4.4 bereits beschriebene Mechanismus zur Bereitstellung der Praktikaplätze funktioniert also auch wie gewünscht.

4.5. Implementierung und Verifikation des ProC/B- Modells für den Realcase

Im folgenden Abschnitt wird die Modellierung und Umsetzung des Realcase 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 4.2.2 erläutert worden. Die Grundlagen für die gesamte Modellierung bilden die im Kapitel 4.2.1 gezeigten Struktur- und Ablaufdiagramme.
Der Aufbau des gesamten Simulationsmodells erfolgt hierarchisch, siehe Abbildung 4.13. Die Wurzel *Modell* bildet nur die Umgebung für die Modellierung, in die dann alle weiteren Konstrukte eingebunden werden. Die nächste Ebene besteht aus dem Knoten *Realcase*. In diesem liegt der nächst tiefere Knoten *Uni*, welcher auf der nächsten Ebene den Knoten *FBI* enthält, der den Fachbereich Informatik repräsentiert. Stellvertretend für den Bereich Kerninformatik besteht die nächste Ebene aus dem Knoten *KI*, der das Studium der Kerninformatik modelliert. Dieser verzweigt sich in der nächsten Hierarchieebene in die beiden Knoten *GS* und *HS*, in denen jeweils die Veranstaltungen und Abläufe für das Grund- und Hauptstudium realisiert werden.

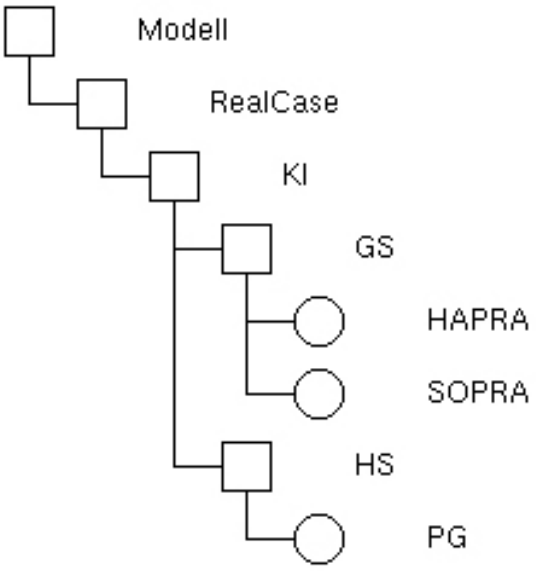


Abbildung 4.13.: Struktur des ProC/B-Modells

4.5.1. Modell

Die Ebene *Modell* wird benötigt, um die auf der Ebene *Realcase* definierten Rewards (selbst definierte Messströme) ausgeben zu können, da dies der ProC/B-Philosophie folgend eine Ebene höher geschehen muss. Aus diesem Grund existiert diese Ebene im HIT-Modell nicht.
Die Realisierung der eigentlichen Modellierung erfolgt dann in der Ressource *Realcase*.
Die Definition des Infiles muss ebenfalls nach ProC/B-Philosophie auf der obersten Ebene geschehen, daher erfolgt dies hier, obwohl das Infile erst in der Ressource *Realcase* ausgelesen wird.

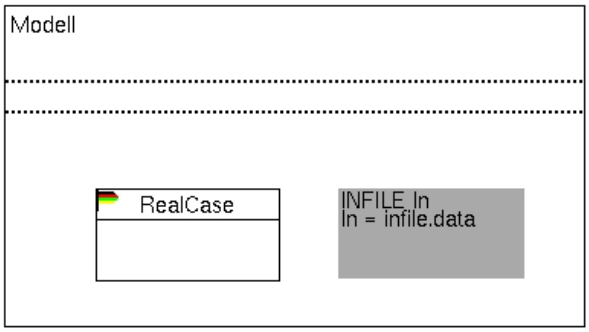


Abbildung 4.14.: Funktionseinheit Modell

4.5.2. Realcase

Die Funktionseinheit *Realcase* (siehe Abb. 4.15) besteht aus insgesamt fünf Prozessketten.

Die erste Prozesskette *studieren* stellt den eigentlichen Studiumsablauf dar. Es werden gemäß der Datenerhebungen in Kapitel 4.2.4 und den getroffenen Modellierungsannahmen in Kapitel 4.2.2 jedes Jahr zwischen 16 und 65 Studentencluster (ein Cluster entspricht 12 Studenten) neu ins System aufgenommen. Dieses wird mit Hilfe der diskreten Gleichverteilung (*randint(16,65)*) realisiert. Zu Beginn bekommt jeder Prozess zwei eigene Prozessvariablen *ex* vom Typ *Boolean* und *Zeit* von Typ *Real*. Initialisiert werden die Variablen wie folgt: *ex* = false und *Zeit* = 0.
Die Variable *ex* dient zur späteren Überprüfung ob der Cluster, dem dieser Prozess entspricht, noch immatrikuliert ist, oder bereits aufgrund von gescheiterten Prüfungen exmatrikuliert wurde. In diesem Fall hat die Variable *ex* den Wert *TRUE*. Die Variable *Zeit* gibt die Anzahl der bereits vom Prozess absolvierten Semester an. Die Variable wird nach dem Ablauf jedes Semester inkrementiert und dient so zur späteren Berechnung der durchschnittlichen Studiendauer der Diplomanden.
Die Prozesskette besitzt ein Prozesskettenelement *studieren*, welches den Dienst *studieren* der Funktionseinheit *KI* aufruft. Nachdem das Prozesskettenelement abgearbeitet wurde, werden die Prozesse am *Oder-Konnektor* aufgeteilt. Die Prozesse, für die die Variable *ex* den Wert *TRUE* besitzt, wird in den oberen Pfad weitergeleitet zum Updateelement *Schwund*. Hier wird das Update für den Reward *Schwund* durchgeführt, der Reward wird um den Wert 1.0 erhöht.

```
UpdateElement Schwund
     UPDATE Schwund 1.0
```

Alle anderen Prozesse durchlaufen den unteren Pfad und das Updateelement *RewardUpdate*, hier werden die Werte der beiden definierten Messströme aktualisiert.

```
UpdateElement RewardUpdate
     UPDATE Durchlaufzeit data.Zeit
     UPDATE Diplome 1.0
```

Die Prozesskette *infile_einlesen* dient zum Einlesen der Eingabeparameter aus dem Infile, in welchem diese gespeichert sind. Das Einlesen der Parameter findet einmalig zum Zeitpunkt 0 statt. Bedingt durch die Tatsache, dass die eingelesenen Parameter frühestens zum Zeitpunkt 1 oder später benötigt werden, wird sichergestellt, dass das Einlesen des Infiles rechtzeitig stattgefunden hat. Das eigentliche

Einlesen der Parameter geschieht durch das Codeelement *Einlesen*. Hier werden die Werte aus dem Infile in die globalen Variablen des Modells eingelesen.

```
CodeElement Einlesen
     OPEN In, "In" LENGTH 80;
     READ FILE In, DAP1;
     READLN FILE In;
     READ FILE In, M1;
     READLN FILE In;
     READ FILE In, RS;
     READLN FILE In;
     READ FILE In, DAP2;
     READLN FILE In;
     READ FILE In, BS1;
     READLN FILE In;
     READ FILE In, ETNT;
     READLN FILE In;
     READ FILE In, M2;
     READLN FILE In;
     READ FILE In, LOGIK;
     READLN FILE In;
     READ FILE In, WR;
     READLN FILE In;
     READ FILE In, SWT;
     READLN FILE In;
     READ FILE In, BS2;
     READLN FILE In;
     READ FILE In, IS;
     READLN FILE In;
     READ FILE In, IUG;
     READLN FILE In;
     READ FILE In, UEBAU;
     READLN FILE In;
     READ FILE In, SWK;
     READLN FILE In;
     READ FILE In, MUENDL;
     READLN FILE In;
     READ FILE In, HAPRA;
     READLN FILE In;
     READ FILE In, SOPRA;
     CLOSE In;
```

Die dritte Prozesskette *HapraPlaetze* die jedes Semester einmal gestartet wird, dient zur neuen Bereitstellung der Hapra Plätze in jedem Semester. Das *DelayElement* mit dem Wert *0.5* sichert die rechtzeitige Bereitstellung, bevor einzelne Prozesse freie Plätze anfragen, da diese Anfragen nur zu ganzen Zeiteinheiten stattfinden, die Bereitstellung jedoch schon 0,5 Zeiteinheiten vorher erfolgt.

```
CodeElement setzen
     HAPRA_AKT:=HAPRA;
```

Die Prozesskette *SopraPlaetze*, die alle zwei Semester einmal gestartet wird, dient zur neuen Bereitstellung der Sopra Plätze ab dem zweiten Semester. Das *DelayElement* mit dem Wert *1.5* sichert die rechtzeitige Bereitstellung dieser Plätze, bevor einzelne Prozesse freie Plätze anfragen, da diese Anfragen nur zu ganzen Zeiteinheiten stattfinden, die Bereitstellung jedoch schon 0,5 Zeiteinheiten vorher erfolgt.

```
CodeElement setzen
     SOPRA_AKT:=SOPRA;
```

Die Prozesskette *PGPlaetze*, die jedes Semester einmal gestartet wird, dient zur Berechnung der zur Verfügung stehenden *PG Plätze*. Die Anzahl der PGs wird gemäß den Annahmen (vgl. Kapitel 4.2.2) mittels diskreter Gleichverteilung *randint(8,12)* generiert. Die Berechnung erfolgt jedes Semester neu. Durch das *DelayElement* wird wieder gesichert das die Berechnung erfolgt, bevor die Anfragen für die Plätze gestellt werden.

```
CodeElement setzen
     PG_AKT:=randint(8,12);
```

Die der Funktionseinheit *Realcase* zu Verfügung stehenden Ressourcen sind zum einen die Funktionseinheit *KI* und zum anderen eine Reihe von globalen Variablen. Diese sind wie folgt definiert:
DAP1 (Datenstrukturen, Algorithmen und Programmierung 1), *M1* (Mathematik für Informatiker 1), *RS* (Rechnerstrukturen), *DAP2* (Datenstrukturen, Algorithmen und Programmierung 2), *BS1* (Betriebssysteme 1), *ETNT* (Elektrotechnik und Nachrichtentechnik), *M2* (Mathematik für Informatiker 2), *Logik*, *WR* (Wahrscheinlichkeitsrechnung und mathematische Stochastik), *SWT* (Softwaretechnik), *BS2* (Betriebssysteme 2), *IS* (Informationssysteme), *IUG* (Informatik und Gesellschaft), *UEBAU* (Übersetzerbau), *SWK* (Softwarekonstruktion), *MUENDL* (global für alle mündliche Prüfungen), welche alle vom Typ *Real* sind. Diese Variablen speichern die jeweilige Durchfallquote zu den Prüfungen der entsprechenden Vorlesungen. Alle Variablen werden zur Sicherheit mit dem Wert 1.0, was einer Durchfallquote von 100% entspricht, initialisiert.
Die Variablen *HAPRA* (Hardwarepraktikum) und *SOPRA* (Softwarepraktikum) vom Typ *Integer* speichern die Anzahl der Plätze für die beiden Pflichtpraktika. Die Variablen *SOPRA_AKT* und *HAPRA_AKT* des Typs *Integer* speichern im späteren Verlauf des Modells die im aktuellen Semester noch zur Verfügung stehenden freien Plätze der Praktika.
Gleiches gilt für die Variable *PG_AKT*, in ihr werden die jeweils noch freien Plätze für die Projektgruppen eines Semesters verwaltet.
Die Variable *Dipl* vom Typ *Integer* verwaltet die Anzahl der erreichten Diplome und wird mit dem Wert 0 initialisiert.
Zusätzlich werden die Messströme *Durchlaufzeit*, *Schwund* und *Diplome* definiert. Hier wird die Studiendauer und die Anzahl der Diplome bzw. Studienabbrecher gezählt, die beiden letzteren gemittelt über die Dauer der Simulationszeit.

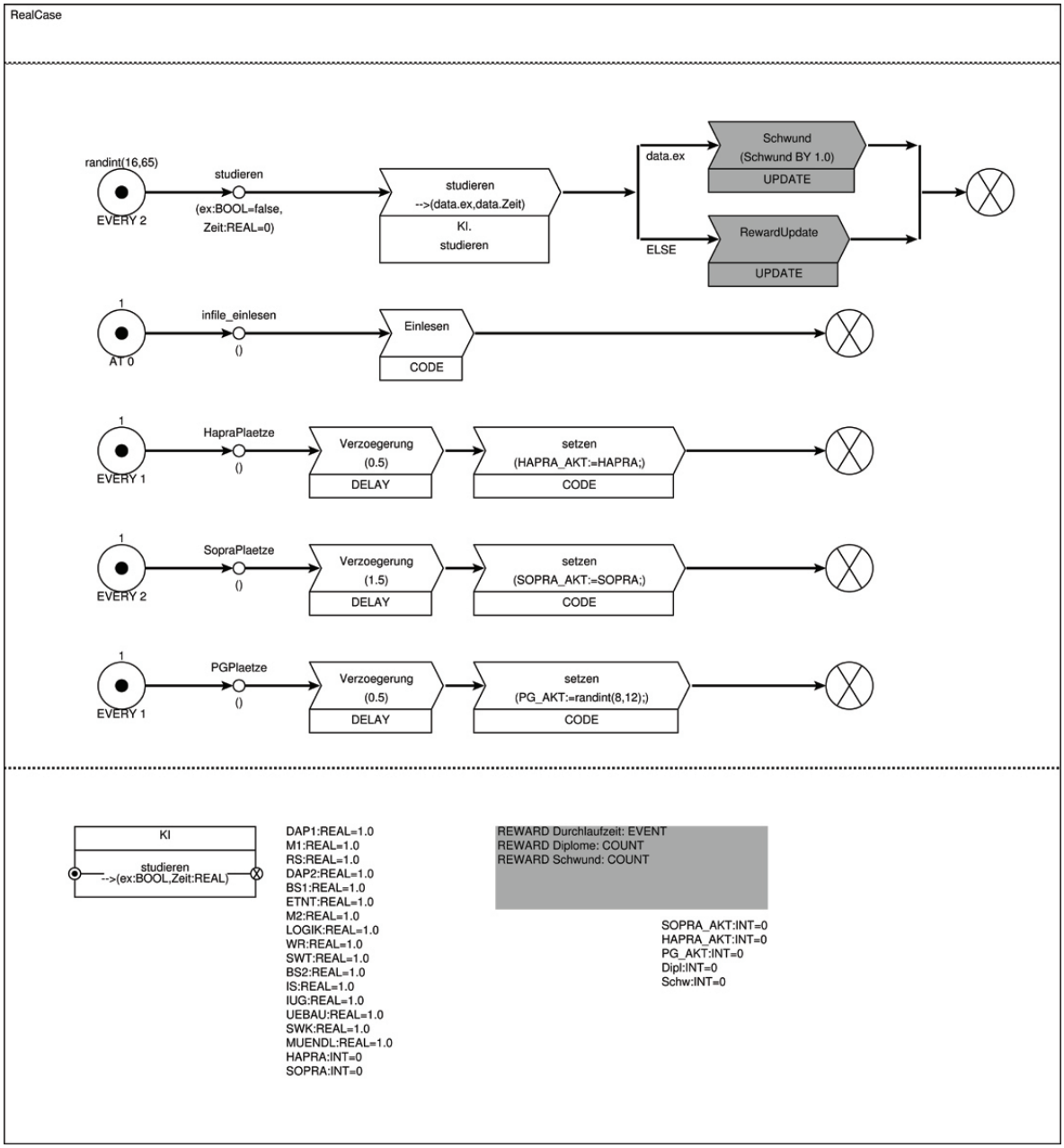


Abbildung 4.15.: Funktionseinheit Realcase

4.5.3. KI

Die Funktionseinheit *KI* (siehe Abb. 4.16) besteht aus einer Prozesskette *studieren*, zwei Ausgabeparametern *ex* und *Zeit* und einer Variable *Winter* vom Typ *Boolean*, die als *TRUE* initialisiert wird, um anzugeben, dass das Studium im Wintersemester beginnt.
Die Prozesskette enthält zwei Prozesskettenelemente. Das Prozesskettenelement *Grundstudium* mit den Ausgabeparametern *ex* und *Zeit* ruft den Dienst *studierenGS* der Funktionseinheit *GS* auf. Anschließend wird überprüft ob der Prozess, z.B. aufgrund mehrerer nicht bestandener Prüfungen, bereits exmatrikuliert ist. Trifft diese Bedingung zu, wird die Prozesskette verlassen. Ansonsten erreicht

der Prozess das Hauptstudium. Dazu werden dem Prozesskettenelement *Hauptstudium* die Parameter *Zeit* und *Winter* übergeben und der Dienst *studierenHS* der Funktionseinheit *HS* aufgerufen. Das *Hauptstudium* enthält zwei Ausgabeparameter *ex* und *Zeit*.
Als Ressourcen stehen die Funktionseinheiten *GS* und *HS* zur Verfügung.

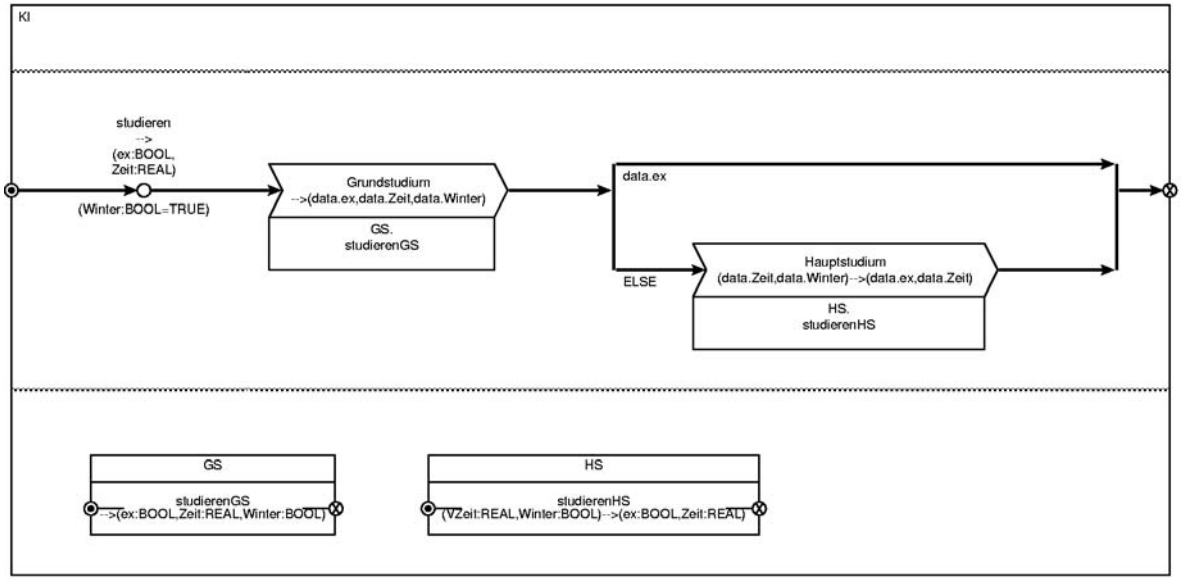


Abbildung 4.16.: Funktionseinheit KI

4.5.4. GS

Die Funktionseinheit *GS* besteht aus einer Prozesskette *studierenGS*.

Zu Beginn wird für jede Grundstudiumsvorlesung eine eigene Prozessvariable vom Typ *Boolean* angelegt, die angibt, ob die jeweilige Vorlesung bestanden worden ist. Diese werden mit *FALSE* initialisiert. Zusätzlich gibt es zwei weitere Prozessvariablen *HAPRAZahl* und *SOPRAZahl* vom Typ *Integer*, die später für die Zufallszahl benutzt werden, um festzustellen, ob der Prozess einen Praktikumsplatz bekommt. *Winter*, *ex* und *Zeit* sind Ausgabeparameter (siehe Abbildung 4.18).

Zwanzig Prozent aller Studenten haben Elektrotechnik als Nebenfach und müssen deshalb kein Hardware-Praktikum machen. Aus diesem Grund wird am Anfang der Prozesskette bei jedem Prozess, mit einer Wahrscheinlichkeit von 0,2, die Prozessvariable *HAPRA_B* auf *TRUE* gesetzt. Dadurch wird verhindert, dass dieser Student unnötig einen Praktikumsplatz belegt.
Der restliche Block der Prozesskette ist eine Schleife, die im Folgenden beschrieben wird. Zunächst wird in dem Codeelement *AnwaerterZaehlen* die Anzahl der Anwärter für die Praktika berechnet. Ein Prozess ist dann ein Anwärter, wenn er die jeweiligen Voraussetzungen für ein Praktikum erfüllt hat. Für das Hardware-Praktikum ist die einzige Voraussetzung die Vorlesung Rechnerstrukturen, die erfolgreich absolviert werden muss, bevor man am Hardware-Praktikum teilnehmen darf, während für das Software-Praktikum die Voraussetzungen DAP1, DAP2 und Softwaretechnik sind. Die zwei DELAY-Elemente vor und nach dem Prozesskettenelement *AnwaerterZaehlen* sorgen dafür, dass die Anzahl der Anwärter bereits vor Beginn des Semesters feststeht, da ansonsten das Risiko bestehen

würde, dass diese nicht richtig berechnet werden würden. Anschließend folgt ein Oder-Konnektor und zwei an ihm angeschlossene Blöcke, die durch je einen öffnenden und einen schließenden Und-Konnektor gebildet werden und in Abhängigkeit vom aktuellen Semester (Winter oder Sommer), abgearbeitet werden.
Diese beiden Blöcke unterteilen sich weiter in jeweils acht kleinere Blöcke, wobei jeder Block entweder eine Vorlesung oder ein Praktikum repräsentiert. Zunächst wird der Block für die Vorlesung DAP1 beschrieben (siehe Abbildung 4.19). Da die Blöcke für die Vorlesungen DAP2, BS2, M2, IS, GTI und RS vom Aufbau äquivalent zum Block DAP1 sind, wird an dieser Stelle auf eine Beschreibung dieser Blöcke verzichtet.

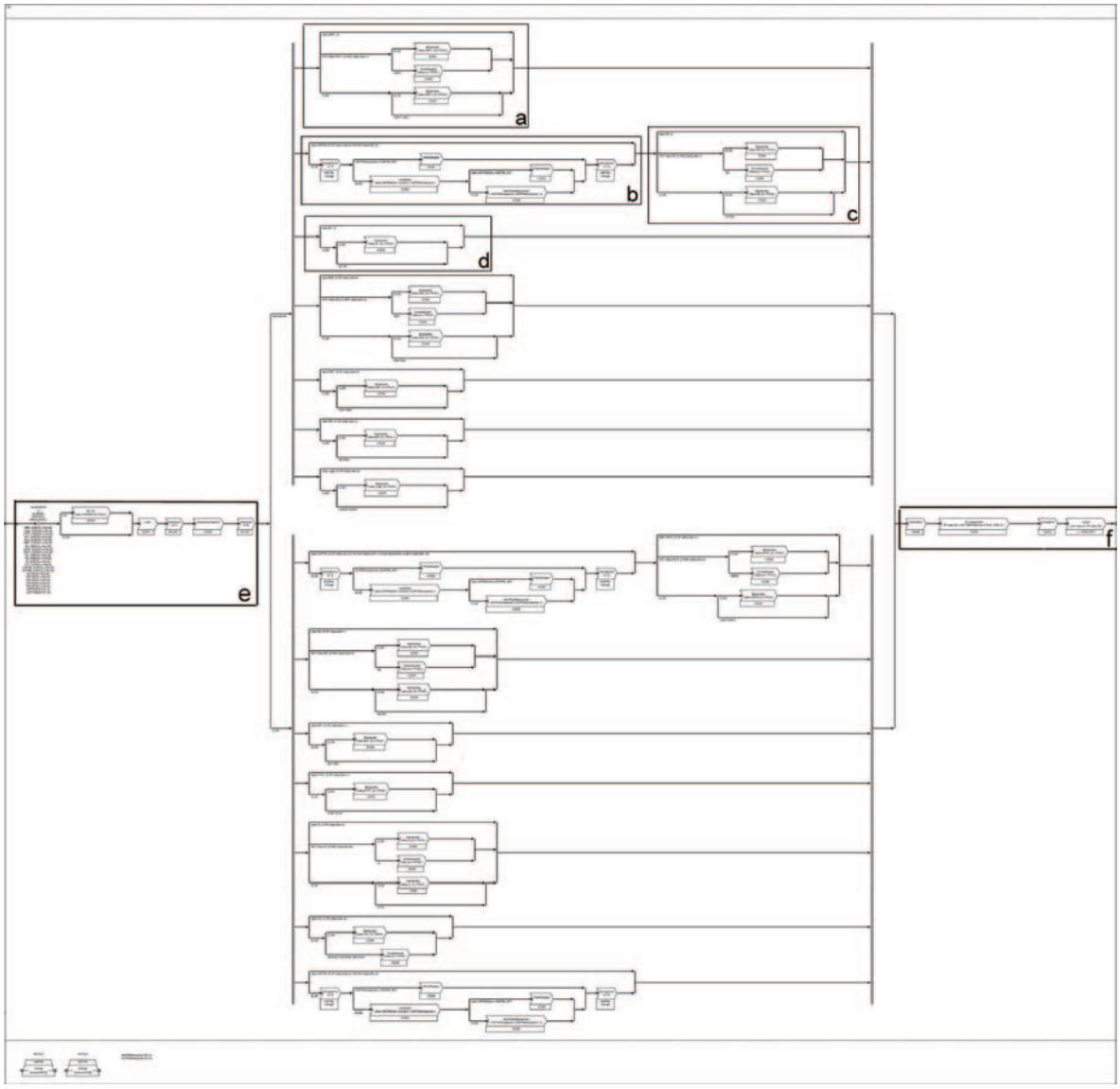


Abbildung 4.17.: Strukturübersicht Funktionseinheit GS

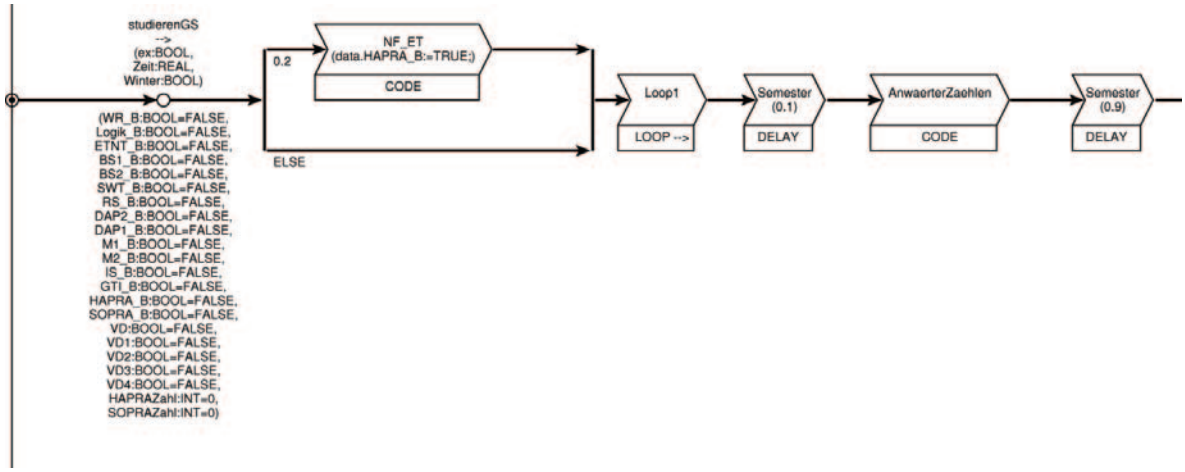


Abbildung 4.18.: Abschnitt e aus Abb. 4.17

Der DAP1-Block lässt sich nun wie folgt beschreiben. Zu Beginn laufen alle Prozesse zunächst auf einen öffnenden Oder-Konnektor auf, welcher drei ausgehende Pfade hat. Beim obersten Pfad wird überprüft, ob ein Prozess die Prüfung DAP1 bereits bestanden hat, d.h., es wird überprüft, ob seine Variable *DAP1_B* gleich *TRUE* ist. Da eine bereits erbrachte Leistung nicht noch einmal erbracht werden muss, würde es an dieser Stelle keinen Sinn machen, dass der Prozess den DAP1-Block erneut durchläuft und daher wird dieser durch den obigen Pfad direkt in den schließenden Oder-Konnektor geleitet, ohne das Prüfungskonstrukt noch einmal zu durchlaufen. Der mittlere Pfad beschreibt den Fall, dass der Prozess die DAP1-Prüfung noch nicht bestanden hat und sich mindestens im dritten Semester befindet. Da, wie bereits in den Annahmen beschrieben, bei einer Fachprüfung die ersten beiden Versuche direkt auf einmal unternommen werden und DAP1 eine Prüfung im ersten Semester ist, kann es sich hierbei nur um den dritten und letzten Versuch handeln. Der Prozess, für den dieser Fall zutrifft, wird dann diesen dritten Versuch mit einer festgelegten Wahrscheinlichkeit von *DAP1* nicht bestehen, mit der Konsequenz, dass dieser exmatrikuliert wird, d.h., seine Variable *ex* wird auf *TRUE* gesetzt, was im Prozesskettenelement *Exmatrikuliert* realisiert wird. Ansonsten wird die Prüfung im Prozesskettenelement *Bestanden* als bestanden markiert, d.h., seine Variable *DAP1_B* wird auf *TRUE* gesetzt. Der unterste Pfad schließlich beschreibt den Fall, dass der Prozess die Prüfung noch nicht bestanden hat und erst im ersten Semester ist. Er unternimmt dann zwei aufeinanderfolgende Prüfungsversuche, die er mit einer Wahrscheinlichkeit *DAP1 * DAP1* nicht besteht. Ansonsten wird die Prüfung im Prozesskettenelement *Bestanden* als bestanden markiert.

Das Hardware-Praktikum findet sowohl im Winter-, als auch im Sommersemester statt. Das Software-Praktikum nur im Sommersemester. Diese werden am Beispiel des HaPra im Wintersemester beschrieben und haben den folgenden Aufbau (siehe Abbildung 4.20).
Hat der Student HaPra bereits bestanden, oder ist noch im ersten Semester, oder hat RS noch nicht bestanden, verlässt er den HaPra-Block und nimmt nicht an der Verteilung der HaPra-Plätze teil. In allen anderen Fällen wird der untere Pfad durchlaufen in dem die Vergabe der Hapra-Plätze realisiert wird. Die beiden Semaphore-Elemente zu Beginn und am Ende dieses Abschnitts, stellen sicher, dass in diesem Abschnitt der Prozesskette immer nur ein einziger Prozess gleichzeitig verarbeitet wird. Dadurch wird eine korrekte Verteilung der HaPra-Plätze gesichert. Ist die Anzahl der HaPra-Anwärter kleiner als die zur Verfügung stehenden HaPra-Plätze (HAPRAAnwaerter<=HAPRA_AKT), wird durch das Codeelement *PlatzBelegen* die Vergabe eines HaPra-Platzes umgesetzt:

```
CodeElement PlatzBelegen
     HAPRA_AKT:=HAPRA_AKT-1;
     data.HAPRA_B:=TRUE;
     HAPRAAnwaerter:=HAPRAAnwaerter-1;
```

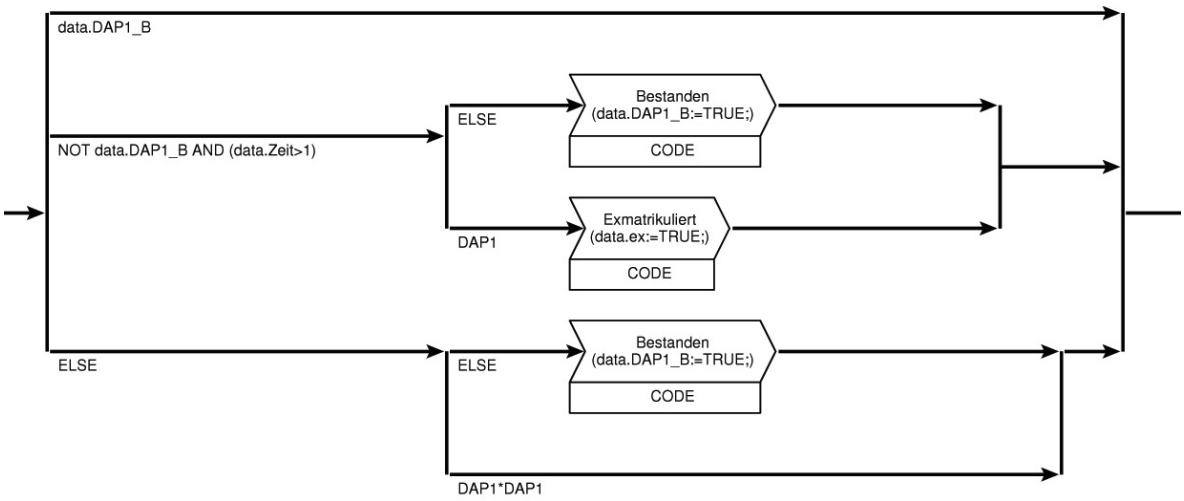


Abbildung 4.19.: Abschnitt a aus Abb. 4.17

Falls weniger HaPra-Plätze als Anwärter zur Verfügung stehen, wird der untere Pfad durchlaufen. Hier wird mit Hilfe des Codeelements *LosZiehen* eine Zufallszahl aus dem Intervall [1, *HAPRAAnwaerter*] gezogen, in der Prozessvariable *HAPRAZahl* gespeichert und entschieden ob der jeweilige Prozess einen Platz zugeteilt bekommt oder nicht. Ist die gezogene Zufallszahl kleiner oder gleich der Anzahl der HaPra-Plätze, wird dem Prozess ein HaPra-Platz zugeteilt, der Prozess durchläuft den oberen Pfad am Oder-Konnektor und das Codeelement *PlatzBelegen* sorgt dann für die Zuweisung. Für den Fall, dass die gezogene Zahl größer als die Anzahl der verfügbaren Plätze ist, wird der untere Pfad durchlaufen und kein HaPra-Platz zugewiesen. Der Prozess muss im nächsten Semester erneut anfragen. Das Codeelement *KeinPlatzBekommen* sorgt dafür, dass die Anzahl der HaPra-Anwärter um den eben bearbeiteten Prozess reduziert wird, so dass irgendwann nur noch so viele Anwärter übrig sind wie es Plätze gibt. Diese Technik sorgt dafür, dass die Vergabe der HaPra-Plätze nicht von der Reihenfolge, in der die Prozesse anfragen, abhängig ist und bei jedem Durchlauf immer nur die ersten oder letzten anfragenden Prozesse einen Platz zugewiesen bekommen.
Da Rechnerstrukturen eine Voraussetzung für das Hardware-Praktikum ist und beide Veranstaltungen somit nicht im gleichen Semester absolviert werden können, folgt der RS-Block nach dem HaPra-Block (siehe Abbildung 4.17 Block b und c).

Die Blöcke der Vorlesungen Mathematik 1 für Informatiker, Softwaretechnik, Wahrscheinlichkeitsrechnung und Stochastik, Logik und Elektrotechnik und Nachrichtentechnik haben den gleichen Aufbau. Deshalb wird im Folgendem nur Mathematik 1 für Informatiker beschrieben (siehe Abbildung 4.21).
Hat der Student die Vorlesung bereits bestanden, wird der Block direkt verlassen. Ansonsten wird die Vorlesung mit einer Wahrscheinlichkeit von *M1*M1* ebenfalls verlassen. In diesem Fall muss der

Student diese Vorlesung noch einmal besuchen, da diese noch nicht bestanden worden ist. In allen anderen Fällen wird die Vorlesung als bestanden markiert und der Prozess verlässt den Block.

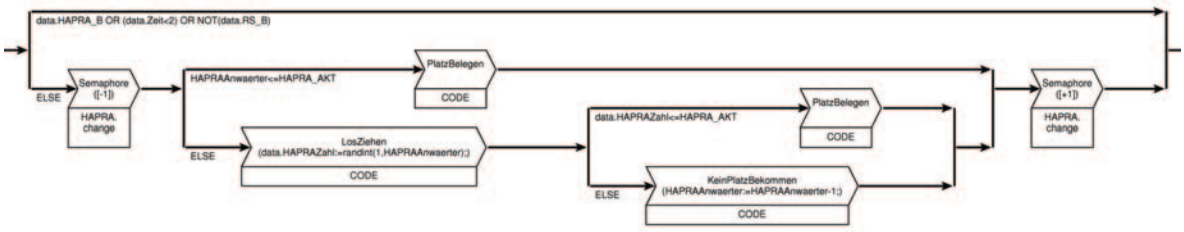


Abbildung 4.20.: Abschnitt b aus Abb. 4.17

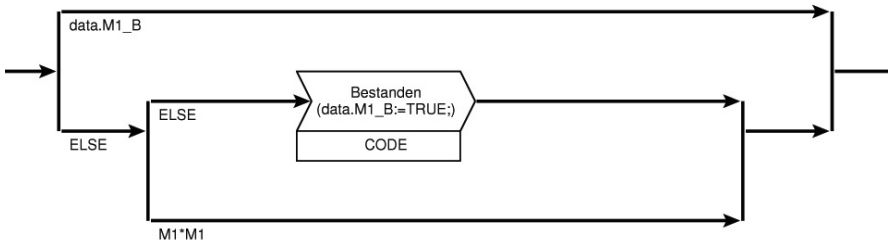


Abbildung 4.21.: Abschnitt d aus Abb. 4.17

Am Ende der Prozesskette wird durch das Codeelement *ZeitSetzen* die Prozessvariable *Zeit*, die die Anzahl der absolvierten Semester zählt, um eins erhöht und die Prozessvariable *Winter* entsprechend angepasst, je nachdem ob der Prozess sich dann im Winter- oder Sommersemester befindet. Das Codeelement *ZuLangeDabei* sorgt dafür, dass Prozesse die sich länger als 20 Semester im Grundstudium befinden entsprechend der Annahmen (siehe Kapitel 4.2.2) automatisch exmatrikuliert werden. Im Codelement *Vordiplom* wird überprüft, ob alle Vorlesungen und Praktika aus dem Grundstudium erfolgreich absolviert wurden. Wenn das der Fall ist, hat der Prozess das Vordiplom erreicht, was durch die Variable *data.VD*, die auf *TRUE* gesetzt wird, gekennzeichnet wird. Anschließend wird überprüft, ob das Vordiplom erfolgreich absolviert worden ist oder der Student exmatrikuliert worden ist. In diesem Fall wird die Schleife und somit auch die Funktionseinheit *GS* verlassen (siehe Abb. 4.17 Abschnitt f und Abb. 4.22.

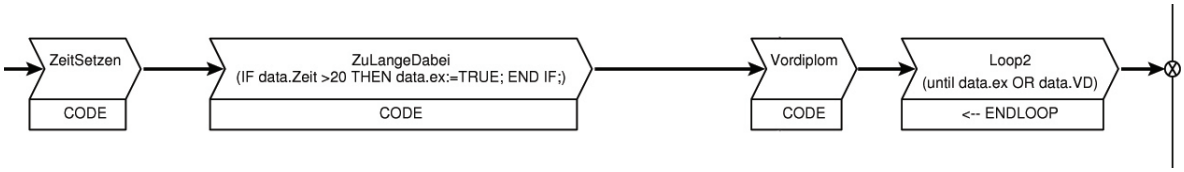


Abbildung 4.22.: Abschnitt f aus Abb. 4.17

Nun folgen die in den beschriebenen Abschnitten benutzten Codeelemente der Funktionseinheit *GS*:

```
CodeElement AnwaerterZaehlen
  IF NOT(data.HAPRA_B) AND (data.Zeit>=2) AND data.RS_B
                       THEN HAPRAAnwaerter:=HAPRAAnwaerter+1;
                       END IF;
  IF NOT(data.SOPRA_B) AND (data.Zeit>=3) AND data.DAP1_B
                       AND data.DAP2_B AND data.SWT_B
                       AND NOT(data.Winter)
                       THEN SOPRAAnwaerter:=SOPRAAnwaerter+1;
                       END IF;

CodeElement Vordiplom
 IF (data.DAP1_B AND data.M1_B AND data.RS_B)
                     THEN data.VD1:= TRUE; END IF;
 IF (data.DAP2_B AND data.M2_B AND data.BS1_B AND data.ETNT_B)
                     THEN data.VD2:= TRUE; END IF;
 IF (data.BS2_B AND data.HAPRA_B AND data.WR_B
                     AND data.Logik_B AND data.SWT_B)
                     THEN data.VD3:= TRUE; END IF;
 IF (data.GTI_B AND data.IS_B AND data.SOPRA_B)
                     THEN data.VD4:= TRUE; END IF;
 IF (data.VD1 AND data.VD2 AND data.VD3 AND data.VD4)
                     THEN data.VD:=TRUE; END IF;
```

4.5.5. HS

Die Funktionseinheit *HS* besitzt eine Prozesskette *studieren*. In der Funktionseinheit wird der Ablauf des Hauptstudiums modelliert. Einen Überblick des Aufbaus der Funktionseinheit liefert die Abb. 4.23. Die mit a bis f markierten Ausschnitte werden im Folgenden vergrößert gezeigt und näher beschrieben.
Der Prozesskette *studieren* werden die Prozessvariablen *VZeit* und *Winter* übergeben (siehe Abb. 4.23 Abschnitt a und Abb. 4.24). Die Ausgabeparameter der Prozesskette sind *ex* und *Zeit*. Für die Prozesskette werden außerdem noch die folgenden lokalen Prozessvariablen definiert: *SWK_B*, *UeBAU_B*, *IuG_B*, *AB1_B* (erste Prüfung aus AB-Katalog), *AB2_B* (zweite Prüfung aus AB-Katalog), *AB3_B* (dritte Prüfung aus AB-Katalog), *SP1_B* (erste Prüfung Schwerpunktgebiet), *SP2_B* (zweite Prüfung Schwerpunktgebiet), *SP3_B* (dritte Prüfung Schwerpunktgebiet), *SP4_B* (vierte Prüfung Schwerpunktgebiet), *Diplom_B* (Diplomarbeit), *DP* (Konjunktion der einzelnen Diplom Prüfungen), *DP1* (erste Diplomprüfung), *DP2* (zweite Diplomprüfung), *DP3* (dritte Diplomprüfung), *DP4* (vierte Diplomprüfung), alle Variablen sind vom Typ *Boolean* und werden mit dem Initialisierungswert *False* belegt. Die Variablen werden zur Verwaltung der abgelegten Prüfungen der entsprechenden Vorlesung verwendet. Legt ein Student eine Prüfung erfolgreich ab, so wird der Wert der entsprechenden Variable auf *TRUE* gesetzt. Die weiteren Variablen *PG* und *PGZahl* sind vom Typ *Integer* und bekommen den Initialisierungswert *0* zugewiesen. Die Variable *PG* dient zur Speicherung einer erfolgreich absolvierten PG, bzw. zur Überprüfung ob der Prozess bereits einen PG-Platz bekommen hat. Die Variable *PGZahl* dient zur Speicherung der für die Verteilung der PG-Plätze zu ziehenden Zufallszahl. Zu Beginn der Prozesskette *studierenHS* wird in dem Prozesskettenelement *Vordiplomszeit* das Semester, in

dem der Student sein Vordiplom erworben hat, gespeichert. Dies ist notwendig um den Ablauf des Hauptstudiums gemäß den getroffenen Annahmen (Kapitel 4.2.2) sicher zu stellen. Andernfalls wäre es möglich, dass der Student bereits im ersten und zweiten Hauptdiplomssemester alle Prüfungen ablegt. Die Variable dient also zur Überprüfung der bereits im Hauptstudium absolvierten Semester. Das folgende Loop-Element *Loop1* stellt den Einstieg in das Hauptstudium dar. Der Prozess verbleibt so lange innerhalb der Schleife, bis im Loop-Element *Loop2*, ganz am Ende der Prozesskette *studieren-HS* eine der Abbruchbedingungen des Loop-Elementes *Loop2* erfüllt wird (siehe Abb. 4.23 Abschnitt f, und Abb. 4.29). Also bis entweder der Prozess aufgrund von nicht bestandenen Prüfungen exmatrikuliert worden ist, oder alle einzelnen Diplomprüfungsteile bestanden hat und somit die Variable *DP* den Wert *TRUE* besitzt.

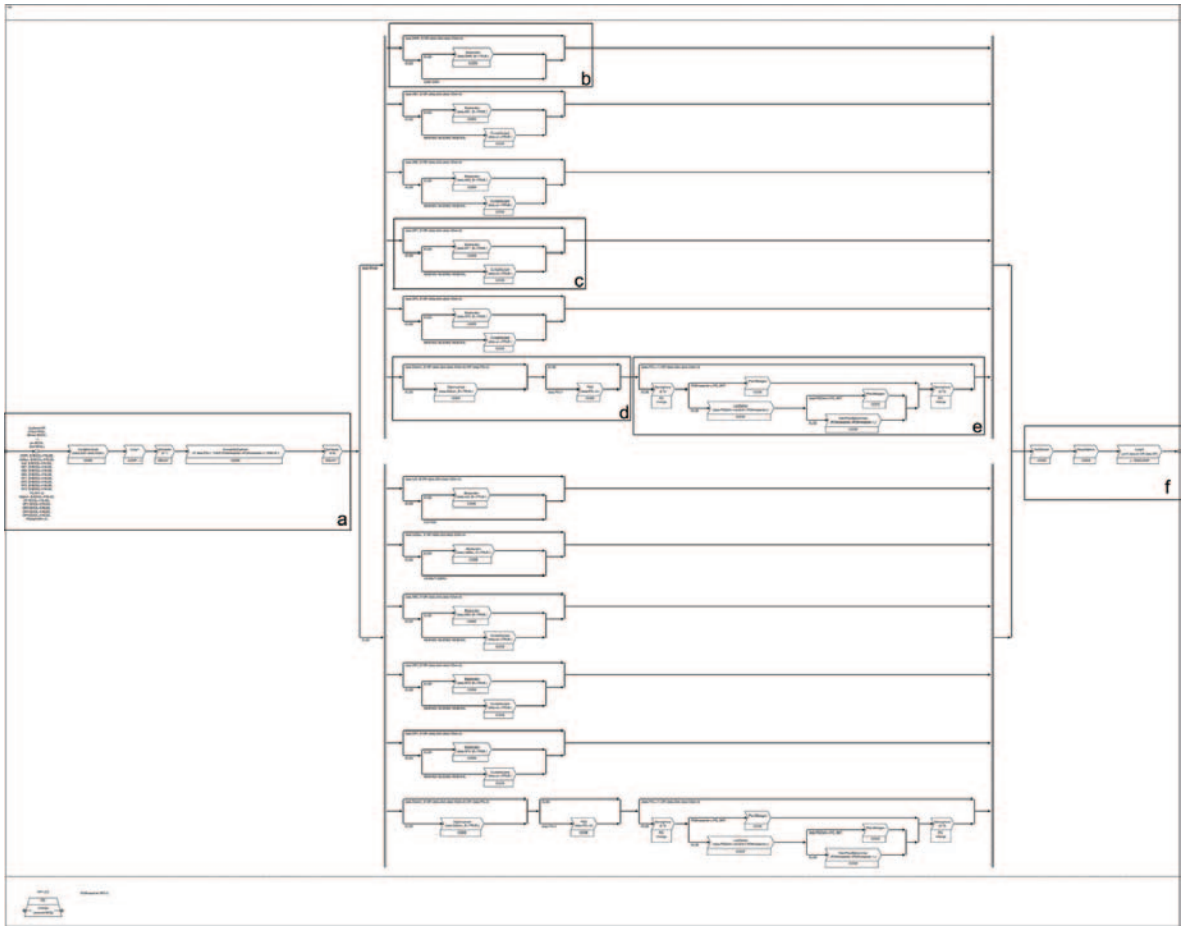


Abbildung 4.23.: Strukturübersicht Funktionseinheit HS

Im Folgenden wird der Aufbau und die Funktion des Schleifeninneren beschrieben. Im Anschluss an den Schleifenbeginn durchläuft der Prozess ein Delay-Element *Semester* mit dem Wert *0,1*. In Kombination mit dem zwei Elemente späteren Delay-Element *Semester* mit dem Wert *0,9* stellt es die Dauer eines Semesters dar. Das Codeelement *AnwaerterZaehlen* berechnet die aktuelle Anzahl an Prozessen, die einen PG-Platz anfordern können. Dazu wird zunächst der Wert der Variable *PG* überprüft, denn nur Prozesse deren Variable den Wert *0* haben sind noch mögliche Anwärter auf einen PG-Platz. Für

den Fall einer gültigen Überprüfung der Variable wird die lokale Variable *PGAnwaerter* um den Wert *1* erhöht. Die Berechnung innerhalb der beiden Delay-Elemente erfolgt, um sicher zustellen, dass die Zählung stattgefunden hat, bevor die Vergabe der Plätze beginnt und somit die aktuelle Anwärterzahl korrekt berücksichtigt werden kann.

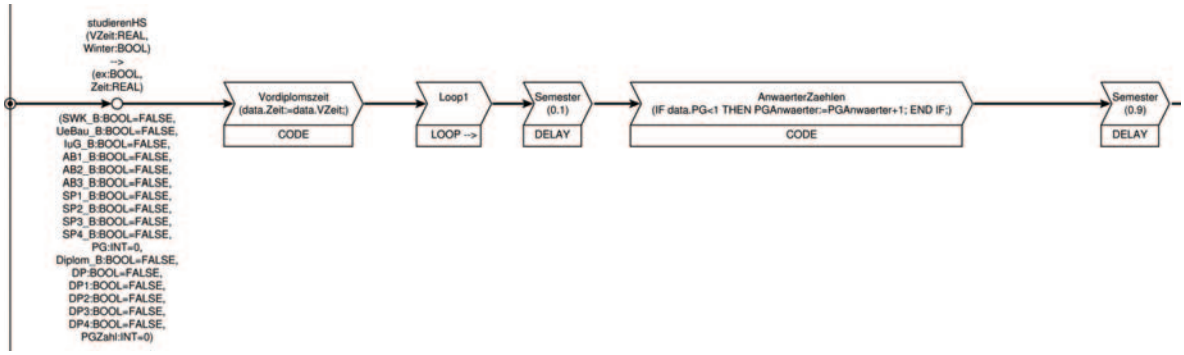


Abbildung 4.24.: Abschnitt a aus Abb. 4.23

Der folgende Oder-Konnektor trennt die Veranstaltungen des Hauptstudiums gemäß ihrer Zugehörigkeit in Winter- und Sommersemester auf. Zu diesem Zweck wird die Prozessvariable *Winter* ausgewertet. Für den Fall, dass die Variable den Wert *TRUE* besitzt, wird der Prozess in den Block mit den Veranstaltungen des Wintersemesters geleitet, andernfalls in den mit denen des Sommersemesters. An dem folgenden Und-Konnektor werden die Prozesse in die einzelnen Veranstaltungen, die absolviert werden müssen, zerlegt. Dabei ist die jeweilige Realisierung der Veranstaltung im Modell nur abhängig vom Typ der Prüfung, bzw. Veranstaltung, so dass sich gleiche Typen nur in den Bezeichnungen der Variablen unterscheiden. Daher wird im Folgenden jeweils für einen Typ einer Veranstaltung exemplarisch ein Teil des gesamten Und-Konnektors beschrieben.
Als Beispiel für eine Prüfung durch die ein Leistungsnachweis erworben wird, wird die Vorlesung *SWK* gewählt (siehe Abb. 4.23 Abschnitt b und 4.25). Zunächst wird an dem Oder-Konnektor die Zulassung für die Vorlesung geprüft. Für den Fall das der Prozess bereits die Vorlesung *SWK* erfolgreich absolviert hat, hat der zugehörige Prozess in der Variablen *SKW_B* den Wert *TRUE* gespeichert und der Prozess durchläuft den oberen Pfad am Oder-Konnektor. Auch für den Fall, dass der aktuelle Wert der Variable *Zeit* kleiner ist als *VZeit*+0, wird eine Änderung der Variablen *SWK_B* verhindert. Der zweite Summand dieser Summe gibt an ab welchem Hauptstudiumssemester diese Vorlesung besucht werden kann und stellt somit den gewünschten Ablauf des Hauptstudiums sicher. Es gibt Vorlesungen die z.B. erst im dritten Semester des Hauptstudiums möglich sein sollen, hier besitzt der Summand demnach den Wert *2*. In beiden Fällen wird der Prozess direkt an den schließenden Und-Konnektor weitergeleitet. In allen anderen Fällen hat der Prozess noch keinen gültigen Leistungsnachweis in *SWK* und wird an den zweiten Pfad des Oder-Konnektors weitergeleitet. Hier wird durch einen weiteren Oder-Konnektor ein mögliches Durchfallen durch die Prüfung, bzw. ein Nicht-Bestehen des Leistungsnachweises realisiert. Mit der Wahrscheinlichkeit *SWK*SWK*, *SWK* ist hier die Durchfallquote für die Prüfung der Vorlesung *SWK*, besteht der Student beide in einem Semester angebotenen Leistungsnachweisprüfungen nicht und durchläuft die Vorlesung im nächsten Semester (Schleifendurchlauf) erneut. Dies kann beliebig oft wiederholt werden. In allen anderen Fällen durchläuft der Prozess den oberen Pfad und damit das Prozesskettenelement *Bestanden*. Hier wird der Wert der Variable *SWK_B* auf den Wert *TRUE* geändert was einem erfolgreichen Erwerb des Leistungsnachweises entspricht.

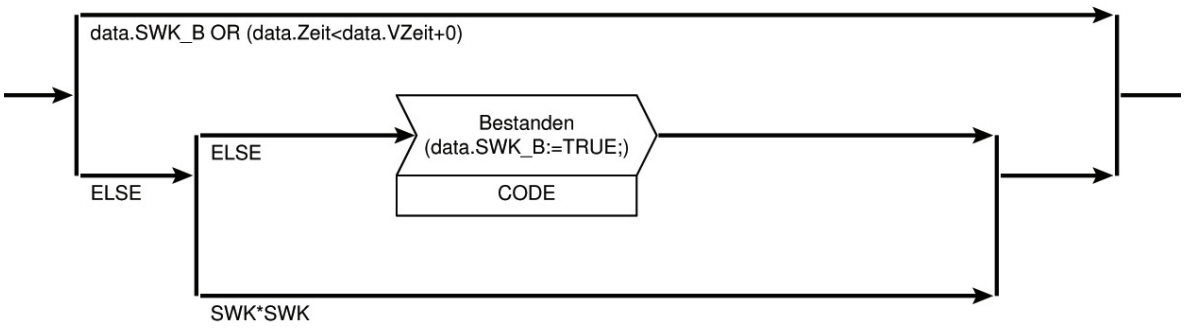


Abbildung 4.25.: Abschnitt b aus Abb. 4.23

Der Aufbau einer normalen Prüfung wird anhand des Beispiels der ersten Schwerpunktgebietsprüfung (SP1) erläutert (siehe Abb. 4.23 Abschnitt c und Abb. 4.26). Der Aufbau dieses Modellteils gleicht dem Aufbau der eben beschriebenen Scheinprüfung bis auf eine Änderung in einem Punkt. Die Tatsache, dass man normale (Nicht-Schein-) Prüfungen maximal drei mal absolvieren darf, wird durch ein zusätzliches Prozesskettenelement realisiert. Es wird in den Pfad des zweiten Oder-Konnektors eingefügt welches der Prozess mit einer Wahrscheinlichkeit von *MUENDL*MUENDL*MUENDL* durchläuft. Die Variable *MUENDL* hat den Wert der Wahrscheinlichkeit mit der ein Prozess durch eine mündliche Prüfung durchfällt gespeichert. Das Prozesskettenelement *Exmatrikuliert* setzt die Variable *ex* auf den Wert *TRUE*. Dadurch wird erreicht, dass der Prozess zum Ende des Semesters exmatrikuliert wird und die Schleife und die Funktionseinheit *HS* verlässt. Ein gültiger Abschluss des Studiums wird somit nicht mehr möglich.

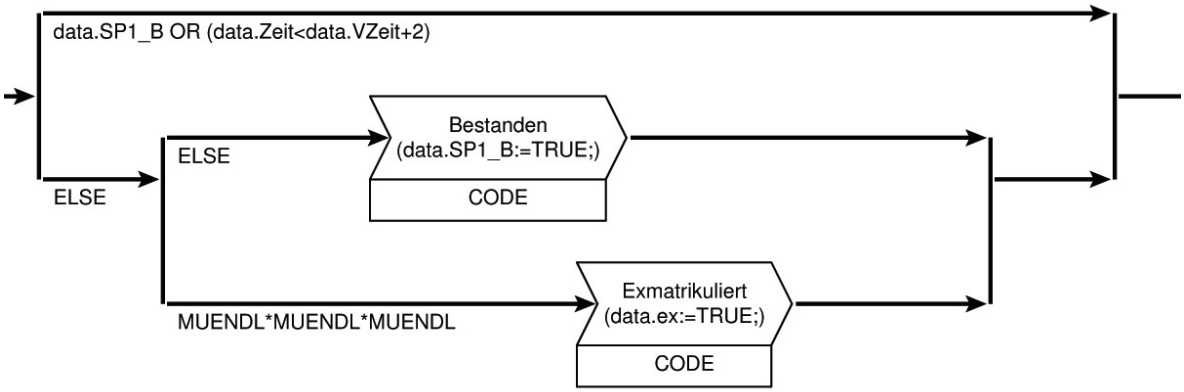


Abbildung 4.26.: Abschnitt c aus Abb. 4.23

Die Abbildungen 4.27 und 4.28 zeigen die Realisierung der PG-Platzvergabe und der Diplomarbeit. Am Beginn des Prozesskettenteils steht wieder ein Oder-Konnektor, hier wird über den Zugang zur Diplomarbeit entschieden. Da gemäß den Annahmen (Kapitel 4.2.2) die Diplomarbeit erst im Anschluss an die PG absolviert werden darf, durchläuft der Prozess den oberen Pfad des Oder-Konnektors in dem Fall, in dem die Diplomarbeit bereits bestanden ist, also der Wert der Variable *Diplom_B* den Wert *TRUE* besitzt, wenn die Anzahl der Semester, die seit dem Erreichen des Vordiploms vergangen

sind, weniger als vier sind (data.Zeit<data.Zeit+4), oder wenn die PG noch nicht abgeschlossen ist (data.PG<2). In allen anderen Fällen sind alle Voraussetzungen für die Diplomarbeit erfüllt und der Prozess durchläuft den unteren Pfad am Oder-Konnektor und der Wert der Variable *Diplom_B* wird durch das Codeelement *Diplomarbeit* auf den Wert *TRUE* gesetzt. Im weiteren Verlauf des Prozesskettenteils folgt ein weiterer Oder-Konnektor, hier wird die Variable *PG* ausgewertet. Im Fall, dass die Variable den Wert *1* besitzt, also bereits ein Semester der PG absolviert wurde und kein PG-Platz mehr benötigt wird, durchläuft der Prozess den unteren Pfad und das Codeelement *PG2* setzt den Wert der Variable *PG* auf den Wert *2*, was gleichbedeutend mit einer absolvierten PG ist. Den weiteren Verlauf zeigt die Abbildung 4.28. Hier wird zunächst am Oder-Konnektor ausgewertet, ob der Prozess als Anwärter auf einen PG-Platz zu werten ist. Für den Fall, dass bereits im letzten Semester ein PG-Platz erhalten worden ist, oder die PG bereits komplett absolviert wurde (data.PG>=1) durchläuft der Prozess den oberen Pfad und nimmt nicht an der Verteilung der PG-Plätze teil. In allen anderen Fällen wird der untere Pfad durchlaufen in dem die Vergabe der PG-Plätze realisiert wird. Die beiden Semaphore-Elemente zu Beginn und Ende dieses Abschnitts stellen sicher, dass in diesem Abschnitt der Prozesskette immer nur ein einziger Prozess gleichzeitig verarbeitet werden kann. Dadurch wird eine korrekte Verteilung der PG-Plätze gesichert. Falls die Anzahl der PG-Anwärter kleiner ist als die zur Verfügung stehenden PG-Plätze (PGAnwaerter<=PG_AKT), wird durch das Codeelement *PlatzBelegen* die Vergabe eines PG-Platzes umgesetzt.

```
CodeElement PlatzBelegen
     PG_AKT:=PG_AKT-1;
     data.PG:=1;
     PGAnwaerter:=PGAnwaerter-1;
```

Falls insgesamt weniger PG-Plätze verfügbar sind als Anwärter wird der untere Pfad durchlaufen. Hier wird durch Ziehen einer Zufallszahl entschieden, ob der jeweilige Prozess einen Platz zugeteilt bekommt. Für das Ziehen der Zufallszahl sorgt das Codeelement *LosZiehen*, hier wird eine Zufallszahl aus dem Intervall [1, PGAnwaerter] gezogen und in der Prozessvariable *PGZahl* gespeichert. Liegt die gezogene Zufallszahl in dem Intervall [1, Anzahl der PG-Plätze], so wird dem Prozess ein PG-Platz zugeteilt, der Prozess durchläuft den oberen Pfad am Oder-Konnektor und das Codeelement *PlatzBelegen* sorgt für die Zuweisung. Für den Fall, dass die gezogene Zahl größer ist als die Anzahl der verfügbaren Plätze, wird der untere Pfad durchlaufen und kein PG-Platz zugewiesen, der Prozess muss also im nächsten Semester erneut anfragen. Das Codeelement *KeinPlatzBekommen* sorgt dafür, dass die Anzahl der PG-Anwärter um den eben bearbeiteten Prozess reduziert wird, bis irgendwann nur noch so viele Anwärter übrig sind wie es Plätze gibt. Diese Technik sorgt dafür, dass die Vergabe der PG-Plätze nicht von der Reihenfolge, in der die Prozesse anfragen, abhängig ist und bei jedem Durchlauf immer nur die ersten oder letzten anfragenden Prozesse einen Platz zugewiesen bekommen.

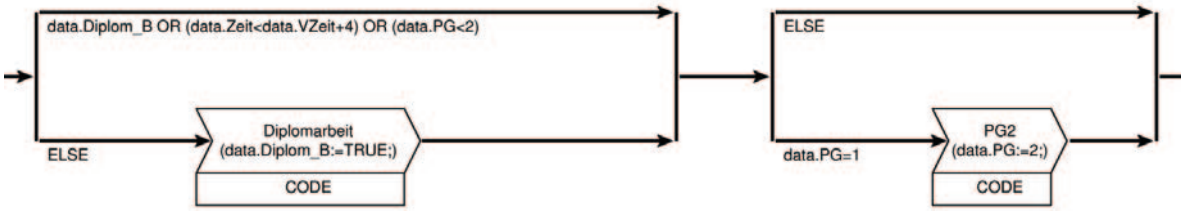


Abbildung 4.27.: Abschnitt d aus Abb. 4.23

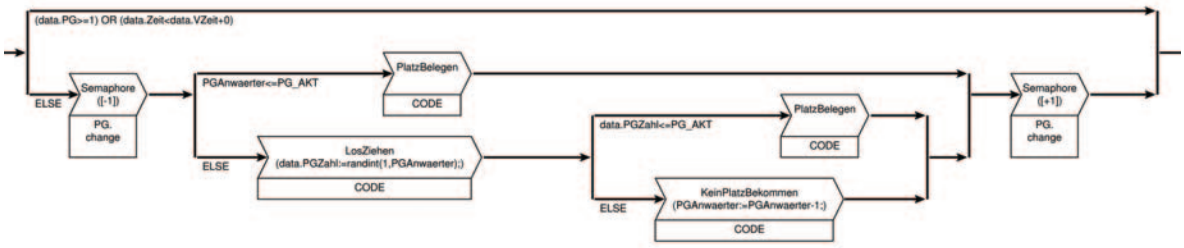


Abbildung 4.28.: Abschnitt e aus Abb. 4.23

Am Ende der Prozesskette (siehe Abschnitt f 4.23 und 4.29) wird durch das Codeelement *ZeitSetzen* die Prozessvariable *Zeit*, die die Anzahl der absolvierten Semester zählt, um eins erhöht. Und es wird die Prozessvariable *Winter* entsprechend angepasst, um im nächsten Durchlauf der Schleife die entsprechenden Veranstaltungen anbieten zu können, je nachdem ob sich der Prozess dann im Winter- oder Sommersemester befindet.

```
CodeElement ZeitSetzen
     data.Zeit:=data.Zeit+1;
     IF data.Zeit MOD 2 = 1 THEN data.Winter:=FALSE; END IF;
     IF data.Zeit MOD 2 = 0 THEN data.Winter:=TRUE; END IF;
```

Im Anschluss wird durch das Codeelement Hauptdiplom überprüft, ob alle Bestandteile des Hauptstudiums erfüllt worden sind. Wenn alle einzelnen Teile (DP1 bis DP4) bestanden wurden und alle zugehörigen Variable durch das Codeelement auf *TRUE* gesetzt worden sind, wird abschließend die Variable *DP* auf *TRUE* gesetzt, was bedeutet, dass das Studium inklusive Diplomarbeit vollständig abgeschlossen wurde.

```
CodeElement Hauptdiplom
     IF (data.SWK_B AND data.AB1_B AND data.AB2_B)
                    THEN data.DP1:= TRUE; END IF;
     IF (data.UeBau_B AND data.IuG_B AND data.AB3_B)
                    THEN data.DP2:= TRUE; END IF;
     IF (data.SP1_B AND data.SP2_B)
                    THEN data.DP3:= TRUE; END IF;
     IF (data.SP3_B AND data.SP4_B AND (data.PG=2)
                    AND data.Diplom_B)
                    THEN data.DP4:= TRUE; END IF;
     IF (data.DP1 AND data.DP2 AND data.DP3 AND data.DP4)
                    THEN data.DP:=TRUE; END IF;
```

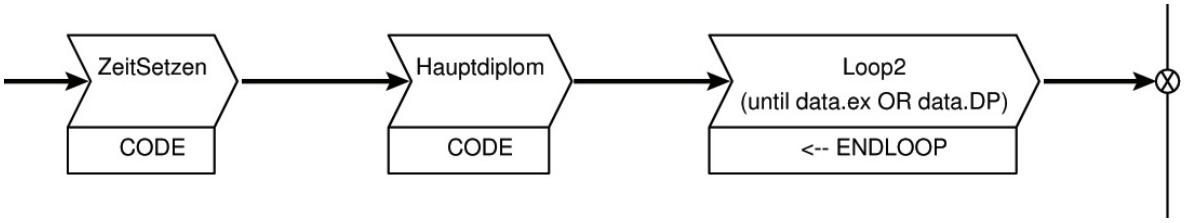


Abbildung 4.29.: Abschnitt f aus Abb. 4.23

Die in den beschriebenen Abschnitten benutzten Codeelemente der Funktionseinheit *HS* sehen wie folgt aus:

```
CodeElement Vordiplomszeit
     data.Zeit:=data.VZeit;

CodeElement AnwaerterZaehlen
     IF data.PG<1 THEN PGAnwaerter:=PGAnwaerter+1; END IF;

CodeElement Bestanden
     data.SWK_B:=TRUE;

CodeElement Bestanden
     data.SP1_B:=TRUE;

CodeElement Exmatrikuliert
     data.ex:=TRUE;

CodeElement Diplomarbeit
     data.Diplom_B:=TRUE;

CodeElement PG2
     data.PG:=2;

CodeElement PlatzBelegen
     PG_AKT:=PG_AKT-1;
     data.PG:=1;
     PGAnwaerter:=PGAnwaerter-1;

CodeElement LosZiehen
        data.PGZahl:=randint(1,PGAnwaerter);

CodeElement KeinPlatzBekommen
        PGAnwaerter:=PGAnwaerter-1;

CodeElement ZeitSetzen
     data.Zeit:=data.Zeit+1;
     IF data.Zeit MOD 2 = 1 THEN data.Winter:=FALSE; END IF;
     IF data.Zeit MOD 2 = 0 THEN data.Winter:=TRUE; END IF;

CodeElement Hauptdiplom
     IF (data.SWK_B AND data.AB1_B AND data.AB2_B)
                    THEN data.DP1:= TRUE; END IF;
     IF (data.UeBau_B AND data.IuG_B AND data.AB3_B)
                    THEN data.DP2:= TRUE; END IF;
     IF (data.SP1_B AND data.SP2_B)
                    THEN data.DP3:= TRUE; END IF;
```

```
IF (data.SP3_B AND data.SP4_B AND (data.PG=2)
               AND data.Diplom_B)
               THEN data.DP4:= TRUE; END IF;
IF (data.DP1 AND data.DP2 AND data.DP3 AND data.DP4)
               THEN data.DP:=TRUE; END IF;
```

Verifikation des ProC/B-Modells

Aus Trivialitätsgründen wird auf eine Verifikation der oberen Ebenen verzichtet. Eine explizite Verifikation erfolgt nur für die Funktionseinheiten *GS* und *HS*. Diese erfolgt über die Verifikation des Durchfallverhaltens bei Variation einzelner Durchfallquoten im *infile*. Gemessen wurde die TURNAROUNDTIME aller Prozesse.
Folgende Szenarien wurden untersucht:

1. Bedingung: Durchfallquote aller Vorlesungen 0%; ausreichende Praktikums- und PG-Plätze
   TURNAROUNDTIME aller Prozesse: 9.000000

2. Bedingung: Durchfallquote 100%; für DAP1 oder RS
   TURNAROUNDTIME aller Prozesse: 3.000000

3. Bedingung: Durchfallquote 100%; für DAP2 oder M2
   TURNAROUNDTIME aller Prozesse: 4.000000

4. Bedingung: Durchfallquote 100%; für BS2
   TURNAROUNDTIME aller Prozesse: 5.000000

5. Bedingung: Durchfallquote 100%; für IS
   TURNAROUNDTIME aller Prozesse: 6.000000

6. Bedingung: Durchfallquote 100%; für GTI
   TURNAROUNDTIME aller Prozesse: 4.000000

7. Bedingung: Durchfallquote 100%; für M1, ETNT, BS1, SWT, WR,
   Logik, SWK, UeBau oder IUG
   TURNAROUNDTIME aller Prozesse: 20.000000

8. Bedingung: Durchfallquote 0%; kein Angebot an Praktikumsplätzen
   TURNAROUNDTIME aller Prozesse: 20.000000

Erwartungsgemäß liegt die TURNAROUNDTIME aller Prozesse bei garantiertem Durchfallen durch die einzelnen Prüfungen exakt in dem Semester, in dem die zugehörige Prüfung das dritte Mal nicht bestanden wurde und somit eine Exmatrikulation erfolgt. Den Erwartungen entsprechend liegt auch die TURNAROUNDTIME bei mangelnden Praktikumsplätzen oder garantiertem Durchfallen in einer der Scheinprüfungen bei 20, da die Anfrage eines Platzes bzw. das Ablegen der Prüfung eigentlich beliebig oft erfolgen kann und die Zeit nur durch den eingebauten „freiwilligen“ Abbruch nach genau diesen 20 Zeiteinheiten begrenzt wird. Der Wert bei ausreichenden Plätzen und Durchfallquoten von 0% ist auch wie erwartet, da dies dem in der DPO vorgesehenen Studienverlauf entspricht.
Zusätzlich wurde das Konstrukt für die Vergabe der Praktikums- und PG-Plätze genauer verifiziert.

Dies geschah über die Messung der auflaufenden Anwärter bei konstanter Ankunftsrate und Durchfallquoten von 0%, so dass alle Voraussetzungen auf jeden Fall erfüllt sind. Hier wurden folgende Fälle unterschieden:

1. Bedingung: weniger Prozesse als Praktikumsplätze
   -> alle Anwärter erhalten einen Platz

2. Bedingung: genauso viele Prozesse wie Praktikumsplätze
   -> alle Anwärter erhalten einen Platz

3. Bedingung: mehr Prozesse als Praktikumsplätze
   -> stetig steigende Anzahl an Anwärtern, auch ältere Anwärter bekommen einen Platz.

Somit zeigt sich, dass jeder Prozess eine Chance größer Null hat, einen Platz zu bekommen, egal wie lange er bereits wartet.

4.6. Experimentendesign

Beim Experimentendesign geht es um Optimierungstechniken für das Ausführen von Experimenten. Es stellt einen Weg dar, zu entscheiden, welche Konfigurationen vor dem Experimentenstart benutzt werden sollen, damit die gewünschten Informationen mit der kleinsten Anzahl an Simulationen gewonnen werden können. Meist gibt es eine Vielzahl von Parametern, die man jedoch aus Zeitgründen nicht alle analysieren kann. Deshalb ist es wichtig, nur die für die Simulation wichtigen Parameter zu wählen und die anderen nicht mehr zu betrachten.
Für die gewählten Parameter ist es möglich verschiedene Kombinationen für eine Experimentenserie auszuprobieren, um dann die Ergebnisse vergleichen zu können, vorausgesetzt diese kombinierten Parameter haben Einfluss aufeinander.
Für die vorliegenden Modelle werden als Parameter die Cluster für das HaPra und das SoPra betrachtet. Es werden verschiedene Kombinationen dieser Werte ausprobiert, da es keine feste Anzahl an Praktikumsplätzen gibt und diese dadurch variabel sind. Die Projektgruppe hat sich bei der Realcase-Analyse auf eine bestimmte Größe der Praktikumsplätze (HaPra 18 und SoPra 16) geeinigt, und führt zusätzlich die Simulation für das HaPra und SoPra zum einen mit einer Erhöhung und Verringerung der Praktikumsplätze um zwei Cluster, und zum anderen mit einer Erhöhung/ Verringerung der Praktikumsplätze um neun beziehungsweise acht Cluster durch.
Die Kombination der Parameter wird ausgelassen, da sie keinen Einfluss aufeinander haben, weil diese weder gemeinsame Voraussetzungen, noch die vorausgesetzten Vorlesungen der Praktika in irgendeiner Weise voneinander abhängig sind, und so die Anzahl an Simulationsläufen nur unnötig groß werden würde.
Die genauere Analyse dieser Experimentläufe wird in dem Unterkapitel 4.8 durchgeführt.

4.7. Modellvalidierung

In diesem Unterkapitel wird die Modellvalidierung der beiden Modelle durchgeführt. Die Verifizierung der beiden Modelle zeigt, dass die Modelle entsprechend den Annahmen modelliert sind. Das Ziel der folgenden Validierung ist es, zu zeigen, dass die Ergebnisse dieser Modelle Ähnlichkeiten aufweisen. Durch die Annahmen sind die Realcase-Modelle von HIT und ProC/B deutlich vereinfacht, so dass keine Anpassung der Modelle, wie für die Bestcase- und Worstcase-Modelle, für die Validierung durchgeführt werden muss.

Um die Tests durchführen zu können, müssen zuvor mehrere Replikationen vorgenommen werden, damit eine genügende Anzahl an Ergebnissen zur Verfügung steht. Im Folgenden werden 100 Replikationen für jedes Modell durchgeführt. Die Ergebnisse der Replikationen werden im R-Tool [Vena 05] eingescannt und mit dem KS-Test verglichen. Die Resultate vom KS-Test für die *Studiendauer*, *Anzahl der Diplomanden* und *Schwund* werden im Weiteren aufgeführt.
KS-Test für die *Studiendauer* [1]:

```
> ks.test(hit100$Studiendauer,procb100$Studiendauer)

        Two-sample Kolmogorov-Smirnov test

data:  hit100$Studiendauer and procb100$Studiendauer
D = 0.08, p-value = 0.9062
alternative hypothesis: two.sided
```

Die Wahrscheinlichkeit, dass die beiden Modelle die gleiche Studiendauer ermitteln, beträgt ca. 91%.

KS-Test für die Anzahl der *Diplomanden* [1]:

```
> ks.test(hit100$Diplomanden,procb100$Diplomanden)

        Two-sample Kolmogorov-Smirnov test

data:  hit100$Diplomanden and procb100$Diplomanden
D = 0.07, p-value = 0.967
alternative hypothesis: two.sided

Warning message:
cannot compute correct p-values with ties in:
ks.test(hit100$Diplomanden, procb100$Diplomanden)
```

Die Wahrscheinlichkeit, dass die beiden Modelle die gleiche Anzahl an Diplomanden liefern, beträgt ca. 97%. Allerdings gibt der KS-Test eine Warnung aus. Die Begründung liegt darin, dass gleiche Ergebnisdaten als Ausgangspunkt mit in den Test einfließen.

KS-Test für den *Schwund* [1]:

```
> ks.test(hit100$Schwund,procb100$Schwund)

        Two-sample Kolmogorov-Smirnov test

data:  hit100$Schwund and procb100$Schwund
D = 0.07, p-value = 0.967
alternative hypothesis: two.sided

Warning message:
cannot compute correct p-values with ties in:
ks.test(hit100$Schwund, procb100$Schwund)
```

[1] jeweils 100 Werte

Die Wahrscheinlichkeit, dass die beiden Modelle den gleichen Schwund liefern, beträgt ca. 97%. Auch dieser Test ist mit einer Warnung behaftet.

Obwohl der KS-Test für alle drei Fälle sehr gute Ergebnisse liefert, sind sie jedoch durch die Warnungen nicht zufrieden stellend. Sicherheitshalber wurden noch weitere Tests zur Hilfe genommen. Das sind der *t-Test* und der *wilcox-Test*. Der *t-Test* ist bei einer normal verteilten Stichprobe anwendbar. Um zu überprüfen, ob die Stichproben normalverteilt sind, wird der *shapiro-Test* angewendet. Falls der *shapiro-Test* für beide Stichprobenmenge eine Normalverteilung zeigt, wird der *t-Test* benutzt, sonst der *wilcox-Test*.

shapiro-Test für die *Studiendauer* [1]:

```
> shapiro.test(hit$Studiendauer)

        Shapiro-Wilk normality test

data:  hit$Studiendauer
W = 0.9901, p-value = 0.6736

> shapiro.test(procb$Studiendauer)

        Shapiro-Wilk normality test

data:  procb$Studiendauer
W = 0.9879, p-value = 0.4987
```

Der *shapiro-Test* gibt kein eindeutiges Ergebnis aus, so dass beide Tests verwendet werden können.

t-Test für die *Studiendauer* [1]:

```
> t.test(hit$Studiendauer,procb$Studiendauer)

        Welch Two Sample t-test

data:  hit$Studiendauer and procb$Studiendauer
t = -0.2342, df = 197.384, p-value = 0.8151
alternative hypothesis: true difference in
means is not equal to 0
95 percent confidence interval:
 -0.05480209  0.04316889
sample estimates:
mean of x mean of y
 15.77178  15.77760
```

Die Wahrscheinlichkeit, dass die beiden Modelle die gleiche Studiendauer liefern, beträgt ca. 82%.

[1] jeweils 100 Werte

wilcox-Test für die *Studiendauer* [1]:

```
> wilcox.test(hit$Studiendauer,procb$Studiendauer)

        Wilcoxon rank sum test with continuity correction

data:  hit$Studiendauer and procb$Studiendauer
W = 4934, p-value = 0.8728
alternative hypothesis: true mu is not equal to 0
```

Die Wahrscheinlichkeit, dass die beiden Modelle die gleiche Studiendauer aufweisen, beträgt ca. 88%.

shapiro-Test für die *Diplomanden* [1]:

```
> shapiro.test(procb$Diplomanden)

        Shapiro-Wilk normality test

data:  procb$Diplomanden
W = 0.991, p-value = 0.7428

> shapiro.test(hit$Diplomanden)

        Shapiro-Wilk normality test

data:  hit$Diplomanden
W = 0.9913, p-value = 0.7666
```

Das Ergebnis des *shapiro-Tests* weist eine Normalverteilung auf, so dass die Ergebnisse des *t-Tests* glaubwürdiger sind als die vom *wilcox-Test*.

t-Test für die *Diplomanden* [1]:

```
> t.test(hit$Diplomanden,procb$Diplomanden)

        Welch Two Sample t-test

data:  hit$Diplomanden and procb$Diplomanden
t = 0.2293, df = 197.916, p-value = 0.8188
alternative hypothesis: true difference in
means is not equal to 0
95 percent confidence interval:
 -0.01063869  0.01343869
sample estimates:
mean of x mean of y
   7.3692    7.3678
```

[1] jeweils 100 Werte

Die Wahrscheinlichkeit, dass die beiden Modelle die gleiche Anzahl an Diplomanden liefern, beträgt ca. 82%.

wilcox-Test für die Anzahl an *Diplomanden* [1]:

```
> wilcox.test(hit$Diplomanden,procb$Diplomanden)

        Wilcoxon rank sum test with continuity correction

data:  hit$Diplomanden and procb$Diplomanden
W = 5120.5, p-value = 0.7692
alternative hypothesis: true mu is not equal to 0
```

Die Wahrscheinlichkeit, dass die beiden Modelle die gleiche Anzahl an Diplomanden ermitteln, beträgt ca. 77%.

shapiro-Test für den *Schwund* [1]:

```
> shapiro.test(hit$Schwund)

        Shapiro-Wilk normality test

data:  hit$Schwund
W = 0.9864, p-value = 0.3961

> shapiro.test(procb$Schwund)

        Shapiro-Wilk normality test

data:  procb$Schwund
W = 0.9846, p-value = 0.2969
```

Der *shapiro-Test* liefert sehr schlechte Ergebnisse, so dass der *wilcox-Test* ein glaubwürdigeres Ergebnis ausgibt.

t-Test für den *Schwund* [1]:

```
> t.test(hit$Schwund,procb$Schwund)

        Welch Two Sample t-test

data:  hit$Schwund and procb$Schwund
t = 0.4499, df = 197.941, p-value = 0.6533
alternative hypothesis: true difference in
means is not equal to 0
95 percent confidence interval:
 -0.1405924  0.2236924
```

[1] jeweils 100 Werte

```
sample estimates:
mean of x mean of y
 11.68885  11.64730
```

Die Wahrscheinlichkeit, dass die beiden Modelle den gleichen Schwund zeigen, beträgt ca. 66%.

wilcox-Test für den Schwund [1]:

```
> wilcox.test(hit$Schwund,procb$Schwund)

        Wilcoxon rank sum test with continuity correction

data:  hit$Schwund and procb$Schwund
W = 5039.5, p-value = 0.924
alternative hypothesis: true mu is not equal to 0
```

Die Wahrscheinlichkeit, dass die beiden Modelle ähnlichen Schwund ermitteln, beträgt ca. 93%.

Die durchgeführten Tests haben noch einmal bestätigt, dass die Ergebnisse der beiden Modelle mit hoher Wahrscheinlichkeit übereinstimmen. Daraus lässt sich schließen, dass beide Modelle das gleiche Verhaltensmuster aufweisen. Im weiteren Verlauf werden die Ergebnisse graphisch dargestellt. In den Abbildungen 4.30, 4.31 und 4.32 werden die Verteilungsfunktionen für die Studiendauer, Anzahl der Diplomanden und den Schwund gezeigt. In den Abbildungen werden die Ergebnisse für das HIT-Modell in rot und für das ProC/B-Modell in schwarz dargestellt.

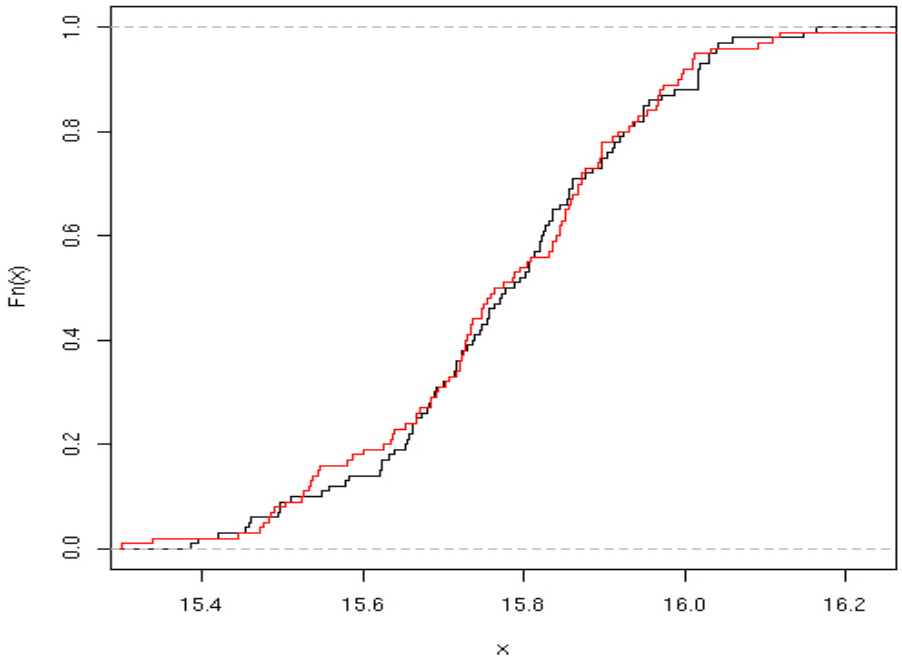


Abbildung 4.30.: Verteilungsfunktionen für die Studiendauer

[1] jeweils 100 Werte

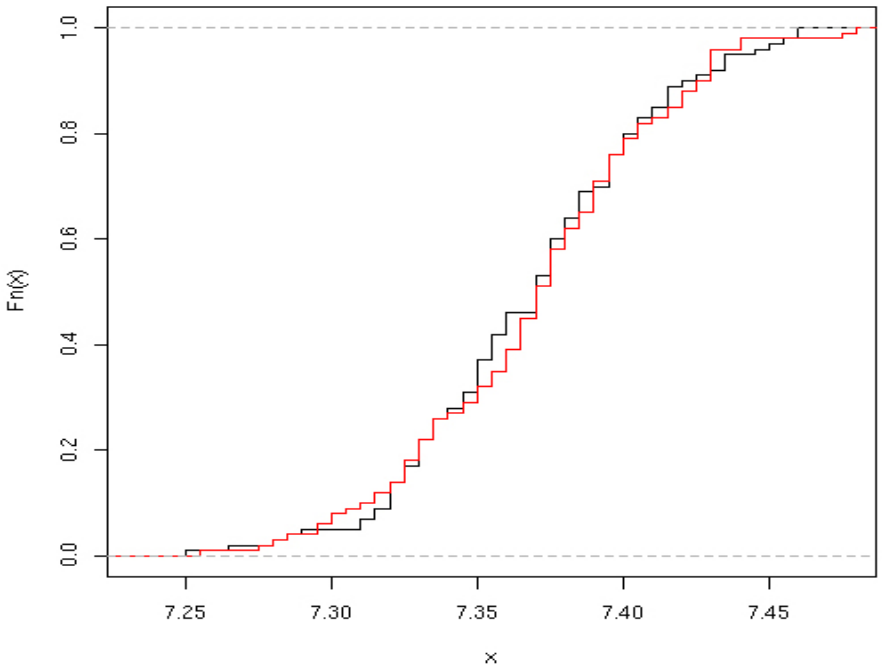


Abbildung 4.31.: Verteilungsfunktionen für die Anzahl der Diplomanden

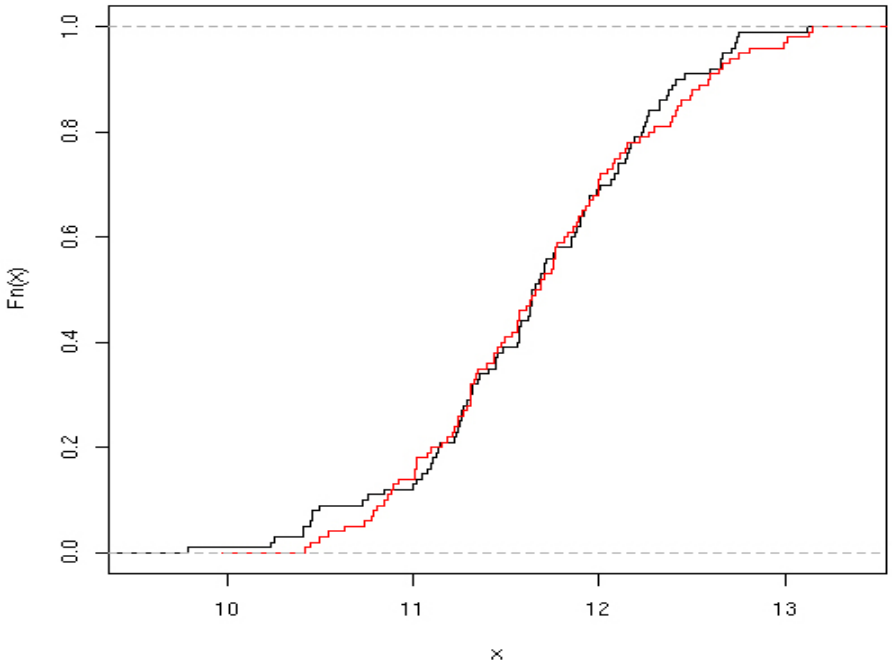


Abbildung 4.32.: Verteilungsfunktionen für den Schwund

Alle drei Abbildungen bestätigen visuell die Testergebnisse der ausgeführten Tests. Das wird dadurch gezeigt, dass die beiden Verteilungsfunktionen in den jeweiligen Abbildungen sehr nah beieinander liegen.

Die durchgeführte Validierung zeigte, dass beide Modelle an allen festgelegten Messpunkten (*Studiendauer*, *Anzahl der Diplomanden* und *Schwund*) mit hoher Wahrscheinlichkeit die gleichen Ergebnisse liefern.

4.8. Produktionsläufe, Analyse und Interpretation der Daten

Die Validierung der beiden Modellen hat gezeigt, dass die Modelle das gleiche Verhalten aufweisen und fast gleiche Ergebnisse liefern (siehe Tabelle 4.4; alle Werte normal). Diese Ergebnisse wurden mit einem 90-prozentigen Konfidenzintervall nachgewiesen. Somit kann eine Analyse der durchgeführten Experimentläufe vorgenommen werden.

Experimentläufe						
Parameter	**HIT-Modell**			**ProC/B-Modell**		
	Studien-dauer	**Diplomanden-anzahl**	**Schwund**	**Studien-dauer**	**Diplomanden-anzahl**	**Schwund**
alle Werte normal	15.973720 ±1.73 %	7.420000 ±7.67%	12.380000 ±10.22%	15.924375 ±1.37%	7.430000 ±7.70%	12.020000 ±6.82%
HaPra zwei Cluster weniger	15.692150 ±1.67%	7.325000 ±8.07%	11.420000 ±22.08%	15.675712 ±1.38%	7.375000 ±7.66%	11.695000 ±6.12%
HaPra zwei Cluster mehr	15.730200 ±1.90%	7.450000 ±7.74%	10.725000 ±11.44%	15.825793 ±1.58%	7.410000 ±7.73%	11.290000 ±6.71%
HaPra neun Cluster weniger	17.294720 ±1.94%	6.345000 ±8.17%	13.405000 ±7.93%	17.247191 ±1.84%	6.235000 ±8.55%	13.675000 ±6.76%
HaPra neun Cluster mehr	15.885890 ±1.63%	7.405000 ±8.08%	12.275000 ±15.17%	16.216016 ±1.41%	7.435000 ±7.97%	12.525000 ±6.87%
SoPra zwei Cluster weniger	16.265940 ±1.65%	6.430000 ±10.09%	12.980000 ±12.29%	16.037867 ±1.49%	6.475000 ±7.03%	12.915000 ±6.66%
SoPra zwei Cluster mehr	15.391780 ±1.74%	8.270000 ±8.81%	11.265000 ±9.83%	15.348274 ±1.50%	8.260000 ±8.79%	10.705000 ±5.88%
SoPra acht Cluster weniger	16.689100 ±2.65%	3.715000 ±9.58%	14.925000 ±9.19%	16.732167 ±2.77%	3.720000 ±9.94%	16.465000 ±7.87%
SoPra acht Cluster mehr	22.195097 ±9.67%	9.585000 ±7.96%	8.705000 ±6.54%	23.694995 ±7.97%	9.595000 ±8.15%	9.670000 ±6.75%

Tabelle 4.4.: Ergebnisse der Experimentläufe (Mittelwert und 90%-iges Konfidenzintervall)

Zunächst wird jeweils ein Simulationslauf mit normal eingestellten Parametern (*SoPra = 16 Cluster = 192 Plätze pro Semester, HaPra = 18 Cluster = 216 Plätze*), die während der Datenerhebung ermittelt bzw. bei den Annahmen getroffen wurden, durchgeführt.
Beide Modelle liefern sehr ähnliche Ergebnisse: Die ermittelte Studiendauer ist bei beiden Modellen gleich und entspricht 16 Semestern, die Anzahl der Diplomanden ist ebenfalls gleich und beträgt 90 Studenten pro Semester (HIT: $7.42 * 12$ Studenten $= 89.04 \approx 90$; ProC/B: $7.43 * 12$ Studenten $= 89.16 \approx 90$). Nur der Schwund unterscheidet sich um vier Studenten pro Semester (HIT: 149, ProC/B: 145). Dieser Simulationslauf zeigt, dass die durch die DPO 2001 [DPO 01] festgelegte Studiendauer von neun Semestern nicht eingehalten werden kann. Die Hauptursachen dafür sind die

unbegrenzte Anzahl von möglichen Versuchen bei dem Erwerb der Leistungsnachweise und die hohe Durchfallquote bei den Prüfungen, die Voraussetzungen für das HaPra bzw. SoPra sind.
Der Simulationslauf mit normal eingestellten Parametern ist für den Vergleich mit den anderen Experimenten notwendig.
Die ersten Experimentläufe werden mit den HaPra-Plätzen als Parameter durchgeführt. Das Ziel dieser Experimente ist die Untersuchung der Auswirkung des HaPra's auf die, in den Modellen gemessenen, Größen. Wie schon im Abschnitt 4.6 erwähnt wurde, wird bei den Experimentversuchen die Anzahl von HaPra-Plätzen zuerst um zwei und dann um neun Cluster (je 12 Plätze) reduziert bzw. erhöht. Die Experimente mit zwei Clustern Unterschied hatten nur geringe Auswirkungen auf die Studiendauer, die Diplomandenanzahl und den Schwund, so dass an dieser Stelle kein Engpass zu erkennen ist.
Die Erhöhung der HaPra-Plätze um neun Cluster liefert ebenfalls Ergebnisse, die sich kaum von den Ergebnissen des Normallaufs unterscheiden. Das bedeutet, dass eine Erhöhung der HaPra-Kapazitäten keinen Sinn macht, da dadurch keine Verbesserung hinsichtlich der Studiendauer erreicht werden kann.
Die Platzreduzierung um neun Cluster hat zur Folge, dass die Studiendauer steigt, die Diplomandenanzahl sinkt und der Schwund ebenfalls ansteigt. Dieser Anstieg wird dadurch verursacht, weil das Vordiplom nun vermehrt nicht mehr in 20 Semestern absolviert werden kann, was laut den Annahmen (Kapitel 4.2.2) dann automatisch zur Exmatrikulation führt. Die Experimentserie mit den HaPra-Plätzen zeigt, dass das HaPra keinen Engpass darstellt und die von der Universität angebotene Anzahl an HaPra-Plätzen, keinen Einfluss auf einen reibungslosen Studienablauf hat.

Bei der zweiten Experimentserie wurden die SoPra-Plätze variiert. Werden diese um zwei Cluster (24 Plätze) erhöht, steigt die Diplomandenanzahl (HIT: 100, ProC/B: 100), der Schwund sinkt (HIT: 136, ProC/B: 129) und die Studiendauer bleibt unverändert. Es kommt also zu einer Verbesserung der Ergebnisse im Vergleich zum Normalfall.
Die Reduzierung der Plätze um zwei Cluster erhöht die Studiendauer um ein Semester und verringert bei beiden Modellen die Diplomandenanzahl von 90 auf 78 Studenten. Der Schwund steigt, da sich die Wahrscheinlichkeit erhöht das Vordiplom nicht in 20 Semestern zu erwerben. Dieser Experimentlauf verdeutlicht, dass das SoPra einen Engpass darstellt. Weitere Experimentläufe mit einer Änderung der zur Verfügung stehenden SoPra-Plätze um $\pm 50\%$ bestätigen dieses Resultat.
Bei einer Verringerung der Plätze um 50% erhöht sich die Studiendauer bei beiden Modellen um ein Semester im Vergleich zum Normalfall. Die Diplomandenanzahl sinkt entsprechend auf 45 Studenten pro Semester und der Schwund steigt auf 180 für das HIT- bzw. 198 Studenten für das ProC/B-Modell. Aus diesen Ergebnissen wird deutlich, dass die Reduzierung der SoPra-Plätze um 50%, einen größeren Einfluss auf die Anzahl der Diplomanden und den Schwund hat als auf die Studiendauer.
Ein unerwarteter Effekt wird bei der Erhöhung der SoPra-Plätze um 50% beobachtet. Hier steigt die Studiendauer deutlich an (HIT: 23 Semester, ProC/B: 24 Semester). Der Grund dafür ist, dass die Erhöhung der SoPra-Plätze um 50% von 192 auf 288 zur Erhöhung der Studentenanzahl mit Vordiplom führt, die dann, entsprechend den Annahmen, sofort einen PG-Platz beantragen dürfen. Da aber jedes Semester nur 8 bis 12 Projektgruppen mit je 12 Plätzen zur Verfügung stehen und es im Vergleich dazu wesentlich mehr Anwärter auf einen PG-Platz gibt, führt die Erhöhung der SoPra-Plätze um 50% zu einem noch größeren Engpass bei den Projektgruppen.
Diese Experimentläufe zeigen, dass das SoPra mit den an der Universität angebotenen 192 Plätzen eine Engstelle ist. Die Variierung der SoPra-Plätze führt entweder zu einem größeren Engpass, oder verursacht den Engpass an einer anderen Stelle.
Um die Analyseergebnisse besser nachvollziehen zu können, sind die Experimentergebnisse aus der Tabelle 4.4 in den Abbildungen 4.33, 4.34 und 4.35 dargestellt. Die Abbildung 4.33 zeigt alle Ex-

perimentergebnisse für die Studiendauer (mit dazugehörigen unteren und oberen Grenzen), die auf Studiendauer bei einem normalen Lauf des HIT-Modells normiert sind.

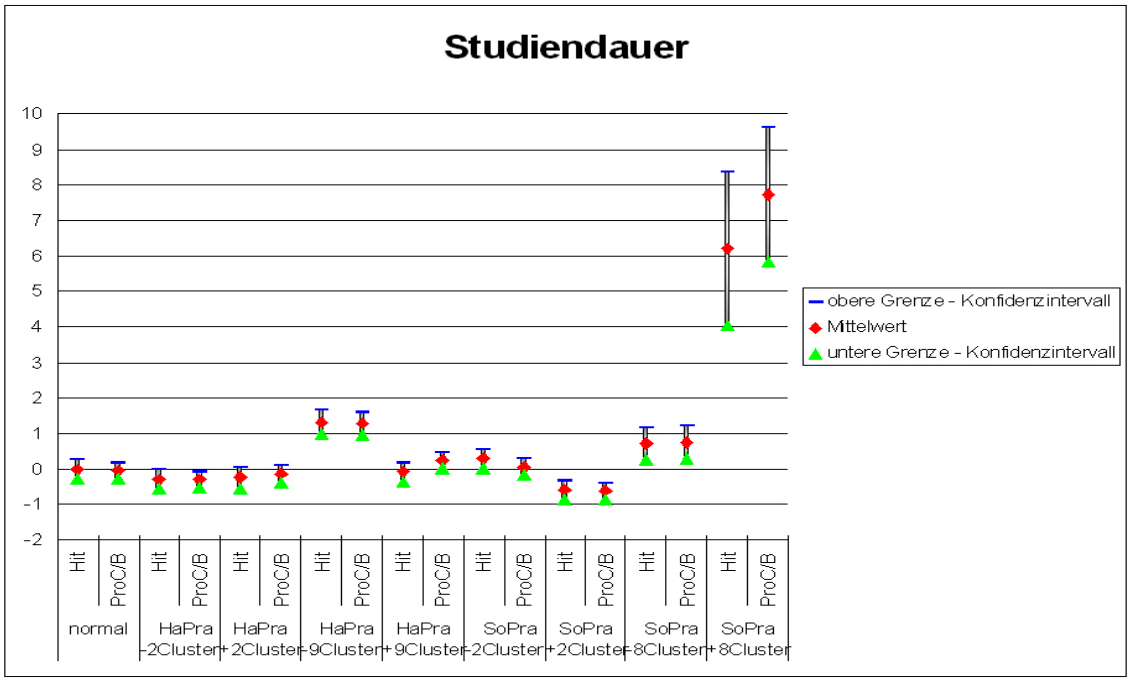


Abbildung 4.33.: Experimentläufe für die Studiendauer (normiert auf 15,973720 Semester)

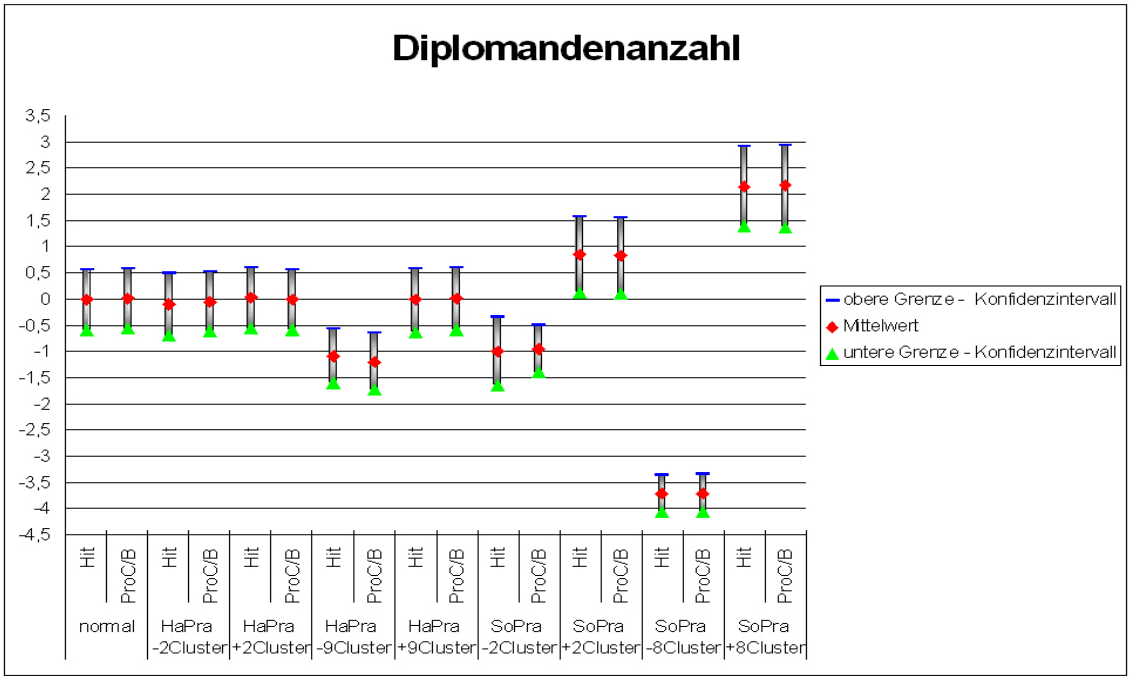


Abbildung 4.34.: Experimentläufe für die Diplomandenanzahl (normiert auf 7,42 Cluster)

Die Abbildung 4.34 stellt alle Experimentergebnisse für die Diplomandenanzahl (mit dazugehörigen unteren und oberen Grenzen) dar, die auf Diplomandenanzahl bei normalem Lauf des HIT-Modells normiert sind. Die Abbildung 4.35 präsentiert normierte Experimentergebnisse für den Schwund (mit dazugehörigen unteren und oberen Grenzen). Die Normierungsgröße ist für diesen Fall der Schwund bei einem Normalexperiment des HIT-Modells.

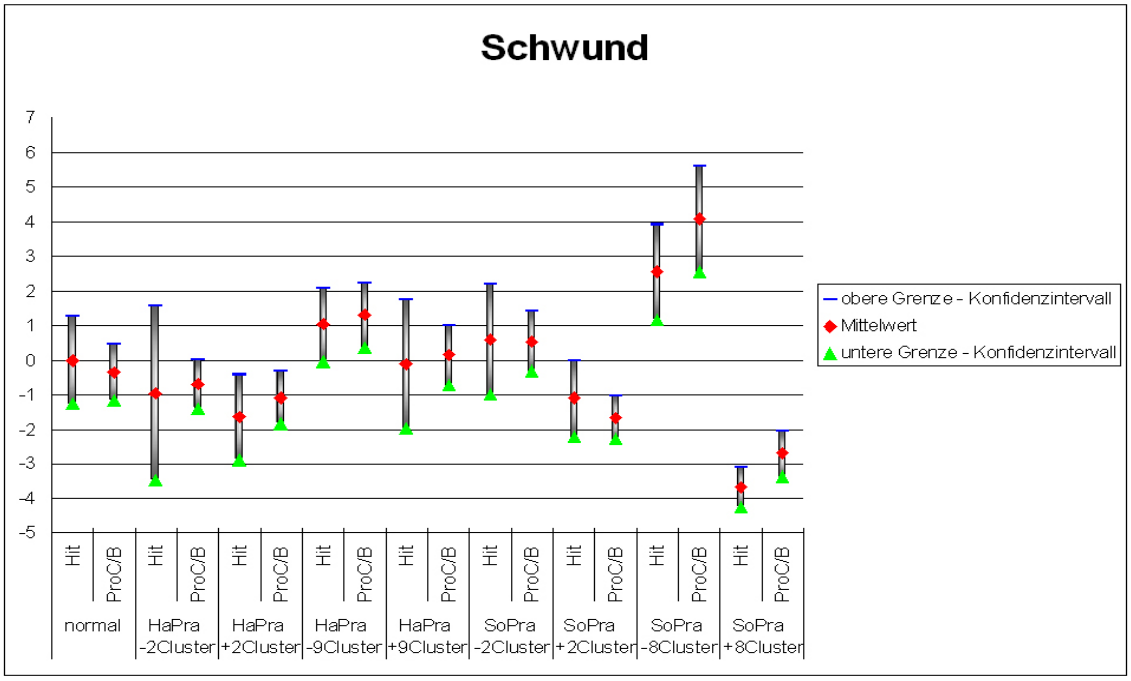


Abbildung 4.35.: Experimentläufe für den Schwund (normiert auf 12,38 Cluster)

Ein weiteres Ziel war die Untersuchung der Kapazitätsplanung für den Fachbereich Informatik. Die von der Verwaltung Dezernat 2 zur Verfügung gestellten Daten zeigen, dass für den Normalverlauf des Informatikstudiums 339,16 SWS pro Semester vorgesehen sind. Die geschätzte SWS-Anzahl pro Semester ist in der Tabelle 4.5 dargestellt, wobei jeder Lehrveranstaltung eine entsprechende Anzahl an SWS (entsprechend der DPO 2001) zugeordnet wird. Die SWS-Anzal für die PG, Seminare, Proseminare, Kataloge A-B und Schwerpunktgebiete wird für die gesamt angebotene Veranstaltungen grob geschätz. So wird angenommen, dass durchschnittlich 20 Projektgruppen (je 8 SWS), 13 Seminare (je 2 SWS) und so weiter (siehe Tabelle 4.5) pro Semester stattfinden. Aufgrund dieser Schätzung liegt die notwendige SWS-Anzahl deutlich über den vorgesehenen 339,16 SWS. Das hat zur Folge, dass bei der betrachteten Studentenanzahl die SWS-Kapazität nicht ausreicht und die Erhöhung der Studentenanzahl eine weitere Gefahr im Bezug auf die Auslastung der Lehrpersonal darstellen würde.

Sommersemester		Wintersemester	
Lehrveranstaltung	**SWS-Anzahl**	**Lehrveranstaltung**	**SWS-Azahl**
BS1	3	**DAP1**	6
DAP2	6	**RS**	5
IS	3	**SWT**	3
SoPra	4	**HaPra**	4
Projektgruppe ($\approx$ 20)	160	**Projektgruppe ($\approx$ 20)**	160
GTI	6	**BS2**	3
Proseminar ($\approx$ 12)	24	**Proseminar ($\approx$ 3)**	6
Seminar ($\approx$ 13)	26	**Seminar ($\approx$ 13)**	26
Katalog A-B	18	**Katalog A-B**	24
Schwerpunktgebiet	107	**Schwerpunktgebiet**	134
Üb	3		
IuG	3		
SWK	3		
Total:	366	**Total:**	371
Durchschnitt	**368.5**		

Tabelle 4.5.: Geschätzte SWS-Anzahl pro Semester

Mit Hilfe dieser Experimentläufe konnten die Engpässe in den Modellen ermittelt und gezeigt werden. Die daraus resultierende Erkenntnis ist, dass trotz der getroffenen Annahmen, welche die beiden Modelle deutlich vereinfacht haben, die durchschnittliche Studiendauer 16 Semester beträgt. Die Diplomandenanzahl liegt bei 90 Studenten pro Semester. Somit haben die 27 Professoren des Fachbereichs Informatik (siehe 4.2.4) bei bereits, laut Kapazitätsberechnung, fast voll ausgelegtem Lehrveranstaltungsplan durchschnittlich drei Diplomanden zusätzlich zu ihren Lehrverpflichtungen zu betreuen. Durch die Studiengänge Angewandte Informatik und Lehramt Informatik, welche, wie bereits erwähnt, in der Analyse nicht berücksichtigt wurden, kommt für den Studienablauf und die Professoren noch eine zusätzliche Belastung hinzu.

5. Fazit der Projektgruppe

Dieses Kapitel enthält das Fazit der Projektgruppe. Es teilt sich auf in eine Kritik an den Bewertungstools HIT und ProC/B im ersten Teil und einer persönlichen Stellungnahme aller Teilnehmer im zweiten Abschnitt.

5.1. Toolkritik

An dieser Stelle soll eine Kritik der Bewertungstools HIT und ProC/B erfolgen. Zunächst das Fazit über die Arbeit mit ProC/B.
Das erste Semester war vor allem durch Fehler geprägt, die durch einzelne Modellierungskonstrukte von ProC/B verursacht wurden. Diese Fehler sollen im Folgenden kurz aufgeführt werden:

- Die Weiterleitung des Dienstes *change* der vorgefertigten Funktionseinheiten *Counter* und *Storage* an externe Funktionseinheiten funktionierte nicht, da 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.

- Der bei der Funktionseinheit *Storage* 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.

- Der in der Funktionseinheit *Storage* definierte 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.

Glücklicher Weise wurden Fehler dieser Art, nach Rücksprache mit den Entwicklern des Tools ProC/B, bereits im ersten Semester umgehend behoben, so dass im zweiten Semester dadurch keine Verzögerungen der Arbeit bei der Modellierung des Modells mehr entstanden. Währenddessen bestanden die Fehler im zweiten Semester hauptsächlich in den Grundfunktionen des ProC/B Bewertungstools, die die Arbeit meist doch recht umständlich gestallteten. Diese stellten sich wie folgt dar:

- Die Funktion *Rückgängig* bereitete einige Probleme, da nach Betätigung dieser entweder das Tool abgestürzt ist oder beim Wiedereinfügen zuvor gelöschter Konstruktverbände, diese unvollständig wiederhergestellt wurden.

- Ähnliches wie bei der Funktion *Rückgängig*, gilt auch für die Funktion *Wiederherstellen*.

Leider ließ sich der Arbeitsablauf, bei dem diese Fehler hervorgerufen wurden, nicht reproduzieren, so dass die Entwickler bislang keine Chance hatten diese angemessen zu beseitigen.

Abschließend lässt sich festhalten, dass das ProC/B Bewertungstool gut einsetzbar ist, d.h., die Bedienung ist bis auf ein paar kleine Details recht intuitiv und durch die graphische Darstellung anschaulicher als die HIT-Modelle.
Dennoch halten wir folgende Verbesserungsvorschläge für recht sinnvoll:

- Es sollte möglich sein, eine Verbindung zwischen zwei Prozessskettenelementen auf ein drittes Prozesskettenelement umzuleiten. Das Problem ist, dass eine bestehende Verbindung nur gelöscht und an einer anderer Stelle wieder neu gezogen werden kann. Dies ist besonders dann unvorteilhaft, wenn die Verbindung eine längere Beschriftung hatte, die man dann für die neu gezogene Verbindung erneut eintragen muss.
- Die Möglichkeit Minimal- und Maximalwerte messen zu lassen auch direkt über den ProC/B-Analysator auswählen zu lassen wäre sehr praktisch, da dies im Moment nur über direkte Änderungen im HIT-Code möglich ist.
- Vor allem für große Modelle wäre es wünschenswert, die Batch-Funktion die in HIT-Graphic implementiert ist, auch in den ProC/B-Analysator zu übertragen.

Im Gegensatz zu ProC/B traten im Umgang mit HIT keine wirklichen Toolprobleme auf, die von den HIT-Entwicklern hätten behoben werden müssen. Der einzige Verbesserungsvorschlag, der sich einem schon bei dem ersten Kontakt mit dem Werkzeug aufdrängt, ist eine benutzerfreundlichere und moderne Oberfläche. Einige Bedienkonzepte sind gewöhnungsbedürftig:

- Das Schließen oder Speichern geht nur über das Kontextmenü. Ein unachtsames Schließen des Fensters über die Kreuzschaltfläche im rechten oberen Eck desselbigen hat sogar zur Folge, dass sich eine neue Instanz von HITGRAPHIC nur nach vorherigem manuellen Zurücksetzen des sog „busy-bits“ öffnen lässt.
- Das Scrollen ist nur mit gehaltener mittlerer Maustaste möglich.

Auch wenn diese ungewohnten Bedienkonzepte nach kurzer Zeit verstanden sind, so erlauben sie doch keinen intuitiven Umgang mit dem Tool, da man auch nach längerer Arbeit immer wieder in über Jahre erworbene Benutzungsschemata zurückfällt, und z.B. doch wieder die Kreuzschaltfläche zum Schließen des Fensters benutzt. Unweigerlich geht damit, gerade in Zeiten erhöhten Stresses, welcher durch Zeitdruck oder einen zu langen Arbeitstag entstehen kann, ein hohes Frustrationspotenzial einher, das so indirekt die eigentliche Simulationstätigkeit erschwert.
Die zwei größten Probleme bei der Modellierung betreffen nicht die Bewertungswerkzeuge HIT und ProC/B direkt, sondern die Modellwelt, auf der sie aufbauen. Zum Einen ist es leider nicht möglich Prozesse innerhalb einer Prozesskette zu löschen oder auch nur zu einer anderen Senke zu leiten, denn alle Prozesse aus einer Quelle müssen auch in eine Senke gehen. Dies macht die Modellierung von alternativen Abläufen und die Messungen an diesen recht umständlich. Zum Anderen ist die Modellierung von parallelen Abläufen in Nullzeit sehr schwierig gewesen, da die Modellwelt eigentlich stochastische Ereigniszeiten vorsieht und das gleichzeitige Auftreten von Ereignissen als Ausnahme betrachtet wird. Hierfür eine - wenn auch umständliche und komplizierte Lösung zu finden, hat den Hauptteil der Modellierungszeit in Anspruch genommen.

5.2. Fazit der PG-Mitglieder

Dieser Abschnitt gewährt allen Teilnehmern der Projektgruppe die Möglichkeit, ihre persönliche Meinung zu dem vergangenen Jahr zu äußern. Die Sortierung der Fazits erfolgt in alphabetischer Rei-

henfolge der Nachnamen der Projektgruppen-Teilnehmer. Für den Inhalt ist jeder Teilnehmer selbst verantwortlich.

Manuel Beer

Hier soll nun ein persönliches PG-Fazit aus meiner Sicht stehen. Nun ja, um es mal mit den Worten unseres scheidenden Bundestrainers auszudrücken: „Des sin Gefühle, wo man schwer beschreibe kann."
Begonnen hat das ganze mit dem Seminar in Schwerte, was aus meiner Sicht ein voller Erfolg war. Wir hatten zwar jede Menge Stoff in Form von Vorträgen, aber die abendliche Party samt großer, lautstarker Tabu-Runde und Rainers bleicher Gesichtsausdruck am nächsten Morgen machten die zwei Tage zu einer gelungenen Veranstaltung, man lernte sich gegenseitig kennen und schaffte auch noch eine gute fachliche Basis für die kommenden Aufgaben. Der erste Eindruck von den Kollegen mit denen man sich nun folgenden zwei Semester lang auseinander setzen musste war also schon mal durchweg positiv und die schlimmsten Befürchtungen wurden nicht erfüllt.
Das Motto welches das folgende erste Semester für meine Arbeit am besten wieder gibt ist dann wohl frei nach Sepp Herberger: „Nach dem Screenshot ist vor dem Screenshot." Zunächst begann die Arbeit allerseits mit mehr oder weniger großer Begeisterung, aber zumindest mit einer ordentlichen Portion Motivation, so dass eigentlich alles recht gut lief. Das Highlight aus geselliger Sicht war dann ganz klar der gemeinsame Weihnachtsmarktbesuch, im Nachhinein hätte man so was wohl öfter machen sollen.
Leider blieb es nicht bei der allgemein guten Stimmung. Im Laufe der zweiten Semesterhälfte hatten einige den Eindruck doch deutlich mehr zu arbeiten als andere. Ich für meinen Teil muss sagen das das auch in den letzten beiden Semesterwochen und den Ferien zu dem für mich frustrierensten Abschnitt der PG geführt hat. Im Rückblick kann man sagen das Jürgens „Ich verwarne Ihnen" in Form von gelben Karten, zum rechten Zeitpunkt kam und bereits in der Ankündigung bei einigen (auch nicht Betroffenen) für gehörig Wirbel sorgte. Hätte ich zu diesem Zeitpunkt eine Prognose für den weiteren Verlauf der PG abgeben müssen, hätte ich wohl gesagt: „Es könnte so oder so ausgehen."
Aber eine „PG dauert 90Minuten... ähm 2 Semester", so dass es in der zweiten Halbzeit, ähm Semester, es dann doch spürbar besser lief. Ich muss sagen das zweite Semester hat mir durchweg gut gefallen, was sicherlich auch daran lag dass wir bis zum Ende gut im Zeitplan lagen und die eigentliche Arbeite rechtzeitig zum Semesterende fertig hatten. Bleibt nun noch diesen Endbericht fertig zu stellen und dann war's das.
Der Abschluss der PG gestaltet sich für mich auf jeden Fall recht versöhnlich. Einerseits bin ich froh es geschafft zu haben, andererseits wird mir die gemeinsame Arbeit auch ein Stück weit fehlen. Rückblickend denke ich, wir haben es alle ganz gut getroffen und hatten auch ne Menge Spaß bei der Sache. Vielleicht sind die zwischenzeitlichen Verstimmungen ja auch einfach etwas „hoch sterilisiert" worden.
Fachlich denke ich das die PG mir doch sehr viel gebracht hat. Wie man an den Ergebnissen des ersten Semesters gesehen hat, ist die Einarbeitung in den fachlichen Hintergrund nicht so schnell zu schaffen gewesen, wie wir uns das alle gedacht haben. Um so besser sind dann aber, aus meiner Sicht, die Arbeit und auch die Ergebnisse des zweiten Semesters ausgefallen. Für alle die eine Projektgruppe noch vor sich haben, kann ich sagen, es ist zwar eine Menge Arbeit, aber bei weiten nicht so schlimm wie von vielen dargestellt und wie schon gesagt, Spaß macht es durchaus auch.
Damit bleibt mir nur noch zu sagen: „Ich danke Sie!", meinen PG-Kollegen für die interessanten und abwechslungsreichen letzten 2 Semester, Jürgen für seine kritische Haltung zu unserer Arbeit, speziell zu den Dokumentationen, hat sicherlich zu deren Qualität beigetragen, allen für die Unterstützung bei

unserer Arbeit und allen Lesern dieses Berichtes dafür, das Sie bis hier hin durchgehalten haben.
„Ich habe fertig!“

Eren Boncuklu

Die Bottleneckanalyse war mein PG-Erstwunsch. Und es ging mir wie den anderen Teilnehmern auch: Ich wusste nicht, was mich erwartet. Zu anfangs war die Motivation bei allen gleich hoch und dementsprechend das Arbeitsklima auch sehr gut. Dass sich das Bild im Laufe der Arbeit etwas gewandelt hat ist ganz klar. Unterschiedliche Gruppen, das bedeutet auch unterschiedliche Ansprüche an die einzelnen Teilnehmer, unterschiedliche Interessen, und diverse Abstimmungsschwierigkeiten; Reiberein zwischen den Gruppen und Teilnehmern sind eine logische, fast notwendige eigendynamische Konsequenz. Wenn allerdings die Vorwürfe zu persönlichen Fehden ausufern, ist das eigentliche Ziel einer solchen Projektgruppe, die sich ja durch Gruppenarbeit auszeichnet, verfehlt. Die Gruppentreffen im 2. Semester brachten da ein wenig mehr Klarheit in die Arbeitsabläufe hinein. Die Koordination funktionierte besser, die Modelle, die zu anfangs noch viel zu kompliziert und komplex waren, wurden jetzt kleiner und arbeiteten wesentlich effizienter. Sicher gab es noch Abstimmungsschwierigkeiten und Missverständnisse, aber die einzelnen Gruppenmitglieder verhielten sich professionell und konnten ihre Emotionen im Zaum halten.
Ein Vorwurf meinerseits geht auch in Richtung Gruppenleiter: Die Kommunikation zwischen der Gruppe und ihm funktionierte nicht immer einwandfrei. In den Plenumssitzungen hat er selten Stellung zu den Vorschlägen der Gruppen bezogen. In den Einzelsitzungen dagegen leistete er solide Hilfestellung und trug sehr konstruktiv zum Fortschritt der Gruppe bei. Allerdings hatte er Änderungen, die wir bereits vorher mit ihm abgesprochen hatten, gedanklich nicht mehr parat; Missverständnisse waren häufig die Folge. Ähnliches gilt auch für unseren Zwischenbericht. Teilweise schon abgesegnete Textpassagen wurden in den nächsten Versionen wieder bemängelt. Traten Rechtschreibfehler auf, sprach er der Ausarbeitung die gesamte Qualität ab, wurde ein Fachwort am Anfang in großen Lettern und am Ende anders geschrieben, sprach er im Ganzen von einer inkonsistenten Arbeit. Sagen will ich damit nicht, dass die Anmerkungen des Gruppenleiters falsch waren, sie waren allesamt richtig. Nur die Mittel und Formulierungen, von denen er Gebrauch machte, um seinen Anmerkungen Nachdruck zu verleihen, waren oft zu drastisch. Immer wieder musste die Arbeit eingereicht werden und seine Formulierungen wirkten oft erdrückend und demotivierend. Aber vielleicht ist es eben die penible akribische Kleinarbeit des Gruppenleiters, die den Bericht auf ein hohes Qualitätsniveau bringt.
Bei so viel Kritik möchte ich schließlich auch noch ein paar kritische Worte zu mir selbst niederlegen. Gelegentlich, und diesen Vorwurf muss ich gelten lassen, war ich zu vorschnell und viel zu sehr überzeugt von der Richtigkeit unserer Modelle. Zu selten habe ich dann auch Kritik angenommen, selbst wenn sie begründet richtig war. Ich führe das teilweise auf mein eigenes Verschulden zurück, aber auch auf das MoPra, dass in meinen Augen viel zu kurz war. Zu wenig Erfahrung bedeutet auch, dass sich Modellierungsfehler einschleichen und unter Umständen nicht gefunden werden.
Letztlich bleibt mir die Erfahrung, dass es eine Herausforderung war mit elf anderen Mitgliedern ein Gruppenziel zu realisieren. Trotz gelegentlicher Reiberein in kritischen Phasen, war das Arbeitsklima angenehm. Insofern war die Projektgruppe und der Kontakt und Erfahrungsaustausch mit ihren Mitgliedern ein persönlicher Erfahrungsgewinn für mich.

Menderes Deniz

Am Anfang des ersten Semesters habe ich mich bemüht, Kontakte mit meinem Komilitonen aufzubauen. Ich denke, dass das mir gut gelungen ist. Die Seminarphase hat das Gruppengefühl der ein-

zelnen Gruppenmitgliedern gestärkt. Nach der Seminarphase wurde die Gruppe in drei Teilgruppen aufgeteilt. Ich kam in die Statistik Gruppe mit Oktay, Genadi und Yongfeng. Die Statistik Gruppe hat gemerkt, dass die anderen beiden Gruppen, Dokumentation und Modellierer, uns als den PG Engpass gesehen haben. Dies war aber jedoch nie der Fall. Die Fehler hatten sich die Modellierer selber eingebaut und somit uns die arbeit erschwert, so dass wir mehrmals unsere Ansätze erneuern mussten. Dieses mehrmalige erneuern der Ansätze hat uns sehr viel Zeit gekostet. Dies sahen die anderen PG Mitglieder als Engpass, weil sie die Statistik Gruppe in den Sitzungen ignoriert haben. Wenn mich nach Jahren Jemand nach der PG Fragen sollte, werde ich mich nur an bestimmte Personen erinnern. Ich habe zu dem auch bemerkt, dass jeder nur seinen Part erledigen wollte und dem anderen nicht unterstützen wollte. Dies änderte sich aber.
Mit dem Betreuer der PG war ich, und vorallem die Statistik Gruppe, sehr zufrieden. Jürgen Mäter stand zu allen unseren Problemen zu Verfügung. Ich bedanke mich bei Jürgen Mäter für seine Gedult und Hilfsbereitschaft. Ich wünsche allen meinem PG-Komilitonen, die alle eine starke Persöhnlichkeit haben, einen erfolgreichen weiteren Studienverlauf und im Leben viel Erfolg.

Viktor Faber

In eine PG zu kommen, bei der man fast keine der geforderten Voraussetzungen angekreuzt hat und somit kaum nützliches Vorwissen aus dem Bereich Simulation mitbringen konnte und am Ende doch einiges gelernt zu haben, würde ich diese PG für mich persönlich schon als erfolgreich bezeichnen und halte diese auch für eine sehr sinnvolle Veranstaltung.
Das Einführungsseminar im Haus Villigst fand ich persönlich sehr gut und ist für mich somit das Highlight dieses Seminars. Die Seminarthemen waren recht interessant und teilweise auch sehr nützlich für die weitere Vorgehensweise der PG. Ich fand es auch sehr interessant die beiden Modellierungstools kennenzulernen. Zur Einführung dazu diente das MoPra, was meiner Meinung nach sehr hilfreich war und den Einstieg sehr erleichtert hat.
Eine der wichtigsten Erfahrungen, die man aus dieser PG mitnehmen sollte, ist, dass die Simulationen, auch bei vergleichbar kleinen Modellen, sehr, sehr lange dauern kann. Was das eigentliche Thema und das angestrebte Ziel unserer Projektgruppe betrifft (Bottlenecksuche des Informatikstudiums), bin ich mit den Ergebnissen etwas unzufrieden. Besonders mit der Analyse des Realcases. Der von uns entdeckte Engpass ist zwar in unserem Modell der Engpass, meiner Meinung nach aber nicht in der Realität. Vielmehr sind es andere Faktoren, die das Studium beeinflussen. Diese konnten wir im Modell natürlich nicht umsetzten, wie z.B. die Qualität der Lehre, Raumplanung, etc. Oft fehlten uns auch wichtige und notwendige Daten und Statistiken der letzten Jahre. Das hängt sicherlich auch damit zusammen, dass die von uns untersuchte DPO relativ neu ist. Insofern sind unsere Ergebnisse nicht wirklich aussagekräftig und können einer Verbesserung und Beschleunigung des Studiums nicht wirklich beitragen. Dieses aber als Ziel anzusetzten wäre für eine PG aber etwas zuviel verlangt, denke ich.
Die Stimmung und die Motivation unter allen PG-Teilnehmern war am Anfang natürlich am besten und hat mit der Zeit leider nachgelassen. Das ist aber ganz normal und hat weder was mit dem Thema, noch mit den Teilnehmern selbst was zu tun. Apropos Teilnehmer, hier die „Spieler“ in der Einzelkritik (bitte nicht alles wörtlich nehmen):
Beer, Manuel: Mein zuverlässiger und sehr engagierter Doku-Partner.
Boncuklu, Eren: Der HIT-Man unter uns - War immer davon überzeugt, dass sein Modell das schnellste und natürlich absolut fehlerfrei ist
Deniz, Menderes: Paralleller Simulant, Linux-Helfer, Strafkasseneinzahler

Fels, Rainer: Er ist, wegen seinen späten Korrekturen, für alle nachträglich eingebauten Fehler im ZB und EB verantwortlich
Glasirin, Genadi: Eine der fleißigsten Bienen unter allen PG-Teilnehmern!
Ketteltasche, Dominic: Unser vorbildlicher Projekt-Manager
Lohmeier, Darius: Hatte das beste Seminar der letzte 5 Jahre (Zitat Jürgen).
Mai, Yongfeng: Chef der Statistiker
Özdemir, Oktay: Hat als Mitglied der Statistiker viele lange Tage im Keller verbracht.
Sereflioglu, Akansel: So ruhig und unauffällig, dass ich ihn fast vergessen habe zu erwähnen.
Thüner, Michael: Hat die ProCB-Entwickler zum verzweifeln gebracht.

Alles in allem bin ich mit der Projektgruppe und mit den Teilnehmern sehr zufrieden. Hätte ja auch schlimmer kommen können :)

Rainer Fels

Das Thema Bottleneckanalyse hatte mich zu Beginn der PG-Anmeldungsphase eigentlich gar nicht so recht interessiert. Um ehrlich zu sein, war diese PG meine letzte vergebene Priorität, wodurch sich meine Begeisterung anfangs doch recht in Grenzen hielt. Diese anfängliche Skepzis war spätestens nach der einführenden Seminarphase verschwunden, die dazu beigetragen hatte, dass man sich besser kennengelernt hat und auch die Analyse eines Bottlenecks der Universität Dortmund besser motiviert wurde.
Was die Arbeit innerhalb der PG angeht, so denke ich, dass diese zwar nicht immer so verlief wie geplant, aber dadurch auch keine Probleme entstanden sind, die man nicht mit den entsprechenden Leuten kurz hätte klären können. Unterm Strich lässt sich sagen, dass es zwar Leute gibt, die sicherlich mehr als Andere gearbeitet haben, aber doch alle was zum Gesamtziel beigetragen haben. Des Weiteren hat Jürgen eigentlich auch immer darauf geachtet, dass sich keiner vor der Arbeit gedrückt hat und wenn es nötig war, durch die Ansprache einer möglichen gelben Karte, die betreffenden Personen wieder an ihre Arbeit erinnert, was auch gut geklappt hat.
Was den Spassfaktor innerhalb der PG angeht, kam dieser auch nie zu kurz. Was zu lachen gab es eigentlich immer und wenn es nur mal wieder die üblichen Verdächtigen waren, die zu spät zur PG-Sitzung kamen und in die Strafkasse einzahlen mussten (ich schließe mich da keinesfalls aus, gerade im ersten Semester habe ich gut eingezahlt). Aber auch ansonsten waren wir schon ein recht lustiger Haufen.
Jetzt wo sich die PG dem Ende neigt, ist keine Spur von Skepzis mehr vorhanden. Wir haben uns doch zu einer recht guten Gruppe gemausert und es ist schon irgendwie schade, dass sich diese jetzt auflöst. Naja ich denke aus den Augen wird man sich nicht verlieren und es bleibt eigentlich nur zu sagen: *Bis die Tage Jungs!*

Genadi Glasirin

In dem folgenden Text will ich meine persönliche Meinung zu der Arbeit der PG-476 äußern. Als Erstes will ich sagen, dass ich hier keinen PG-Teilnehmer in irgendwelcher Weise kritisieren oder beleidigen will, da ich sonst mit mir selbst anfangen müsste. Wenn sich jemand von mir angegriffen fühlt, bitte ich um Entschuldigung. Zweitens will ich allen PG-Mitglieder danken für die Zusammenarbeit und die, im größten Teil, gute Atmosphäre während dieser Zeit. Einerseits bin ich froh, dass die PG vorbei ist und man sich mit etwas anderem und Wichtigerem beschäftigen kann, andererseits werde ich unsere Projektgruppe vermissen, da ich:

- mich an die Leute, mit denen man fast ein Jahr gearbeitet hat, gewöhnen habe
- in der PG-Zeit Verantwortung für einige Teile der Arbeit ubernahm und ein tolles Gefühl bekam, wenn diese Arbeit erfolgreich erledigt wurde und zum Erfolg des gesamten Projektes ein Stückchen beigetragen hatte.

Was habe ich bei der PG gelernt:

- bei einer Teamarbeit ist es nicht immer leicht, auf einen gemeinsamen Nenner zu kommen, da es sehr viele Meinungen und viele unterschiedliche Persönlichkeiten in einer Gruppe gibt,
- es ist oft von Vorteil, allein zu arbeiten, wenn man weiß, was und wie man es machen muss, da sonst zu viel Zeit für Diskussionen verloren geht,
- andererseits kann man bei der Zusammenarbeit mit anderen Teammitglieder sehr viel lernen, durch die Meinungsunterschiede oft bessere Problemlösungen entwickeln und natürlich auch amüsant die Zeit verbringen.

Was mir persönlich fehlte oder mich ärgerte:

- es gab kaum einen Spaßfaktor, die PG war die meiste Zeit zu seriös. Es wäre mir lieber, wenn solche Momente, wie unsere Seminarphase in Schwerte und der gemeinsame Ausflug zum Weihnachtsmarkt öfter vorkämen. Leider wurde fast jeder Vorschlag etwas gemeinsam zu unternehmen wegen einer oder zwei Personen, die zu dem vorgeschlagenen Zeitpunkt keine Zeit hatten, sofort abgelehnt, ohne irgendwelche andere Alternative oder Vorschläge zu betrachten.
- Die Kommunikation zwischen den Untergruppen (bei uns gab es drei (Modellierer, Dokumentierer, Statistiker)) war sehr schlecht. Fragen, die man per Mail an die Gruppenmitglieder geschickt hatte, wurden oft ignoriert oder mit größerer Zeitverzögerung beantwortet. Schlechte Kommunikation führte auch zur unnötigen und überflüssigen Arbeit (besonders bei der Datenerhebung und Modellvalidierung).
- Die Arbeit bzw. Arbeitsaufwand waren nicht gleichmäßig verteilt. Leider kann ich an dieser Stelle nicht genau beurteilen, ob das an den Fähigkeiten der einzelnen Teilnehmer lag, alles einfach schnell zu erledigen und die gewonnene Zeit für die Prüfungsvorbereitungen zu nutzen oder an irrationaler Arbeitweise einzelner Untergruppen, die in diesem Jahr sich nur mit der PG-Arbeit beschäftigt hatten und kaum Zeit für etwas anderes gefunden hatten.
- Da die Modellierer fast jede PG-Sitzung irgendwas vorgetragen hatten und weil die beiden Modelle die Basis für die anderen Untergruppen waren, hatte ich oft das Gefühl, dass die Meinung der Modellierer im Vergleich zu der Meinung der anderen Mitglieder mehr Gewicht hatte. Das hat man auch während der Kritik vom PG-Veranstalter nach dem ersten PG-Semester gesehen. Die Kritik war hauptsächlich an die Dokumentierer und Statistiker gerichtet, was meiner Meinung nach, nicht ganz korrekt war, da diese Untergruppen auch sehr große Arbeit geleistet hatten. Diese Kritik hat einen Frust unter einigen PG-Mitglieder verursacht und die Stimmung am Anfang des zweiten Semesters verdorben.

Was mir bei der Projektgruppe gefallen hat:

- Die schon früher erwähnte Seminar-Phase: die PG-Seminar-Phase war ein sehr guter Einstieg in das PG-Arbeitsjahr. Zwei Tage dieser Phase beinhalteten sowohl die Arbeit als auch den Spaß. Alle waren nett und freundlich, man wünschte sich, dass es die ganze Zeit so bleibt.

- PG-Mitglieder waren sehr nett und freundlich über die ganze Zeit und auch die Hilfsbereitschaft war fast immer vorhanden, wenn jemand was nicht verstand oder Unterstützung brauchte.

- Da ich zur Statistik-Untergruppe gehörte, habe ich entsprechend mehr Zeit mit den Statistikern verbracht. Obwohl wir oft Meinungsdifferenzen hatten, was zu den Verzögerungen bei der Arbeit führte, hat die Zusammenarbeit sehr viel Spaß gemacht und man lernte auch andere Kulturen kennen.

- Der Ausflug zum Weihnachtsmarkt war der Höhepunkt des ersten Semesters in Sachen der Freizeitgestaltung. Die Stimmung war gut und es war sehr lustig dort zusammen die Zeit zu verbringen und mit dieser guten Laune in die Weihnachtsferien zu starten.

- Unserer PG-Veranstalter war auch sehr nett, obwohl er uns oft kritisiert hat, muss man an dieser Stelle sagen, dass diese Kritik nicht immer, aber oft berechtigt war. Andererseits, wenn man die Hilfe brauchte oder nicht weiter kam, könnte man jeder Zeit (von 7.30 bis 15.30:)) zu ihm kommen und alles klären und besprechen.

Abschließend will ich noch sagen, dass ich am Ende der Projektgruppe mit der Zusammenarbeit und den erreichten Ergebnissen zufrieden bin. Ich bin auch froh, dass ich dieser Projektgruppe zugeordnet war und mit diesen Leuten ein Jahr, das sehr schnell vorbeiging, arbeiten dürfte.

Dominic Ketteltasche

Das Thema Bottleneckanalyse hat bei der Projektgruppenvorstellung vor einem Jahr mein Interesse geweckt, so dass ich mich damals dafür entschied hierfür meine erste Priorität zu vergeben. Und nach (fast) getaner Arbeit denke ich, dass ich mich heute wohl wieder so entscheiden würde.
Ich persönlich bin mit der geleisteten Arbeit zufrieden, zumal diese Art von Gruppenarbeit eine neue und wichtige Erfahrung war. Sicherlich hat man vorher schon in Gruppen zusammen gearbeitet oder gelernt, aber noch nicht in diesem Umfang über so einen langen Zeitraum. Zu Beginn war ich ein wenig skeptisch, da ich niemanden aus der Gruppe im Vorfeld wirklich richtig kannte. Meine Befürchtungen haben sich aber schnell verflüchtigt. Ich denke, dass die Gruppe relativ gut zusammen gearbeitet hat, wobei vor allen Dingen im zweiten Semester die Erkenntnis gewonnen wurde, dass sich nicht alle Teilnehmer zu gleichen Teilen in die Arbeit eingebracht haben. Darunter hat bei einigen leider gegen Ende die anfangs gute Laune gelitten. Sicherlich kann man die Arbeit aller schlecht beurteilen, da nicht immer alle Gruppen gleichzeitig im Pool waren. Der genannte Eindruck wurde aber auch bei mir geweckt, da es mich persönlich betrübte, dass sich die Resonanz auf mails doch sehr zurückhielt und es eigentlich außerhalb der offiziellen Sitzungen kaum möglich war, Freiwillige für jegliche Arbeiten zu finden.
Allerdings ist unsere Präsentation nun gehalten und die Arbeit so gut wie fertig. Was nun noch fehlt ist die finale Version des Endberichts, wobei ich zuversichtlich bin, dass wir hier schneller und mehr Erfolg haben werden als beim Zwischenbericht.
Dank unseres Betreuers Jürgen Mäter, der mit seinen Adleraugen sämtliche Fehler entdeckt, wird unser Endbericht am Schluss in einer guten Fassung vorliegen. Sicherlich kamen uns die entdeckten Fehler gelegentlich nicht so schlimm vor, wie Jürgen sie uns in der Sitzung schilderte. Die regelmäßigen Standpauken haben uns zwar hin und wieder etwas demotiviert, allerdings glaube ich, dass der Endbericht dadurch durchaus an Qualität gewonnen hat.
Zum Schluss möchte ich mich von allen verabschieden und besonders bei denen bedanken, die mit mir oft Leid (Arbeit) und Freud (Arbeit) geteilt haben.

Darius Lohmeier

Da es viele Studenten höheren Semesters nicht versäumen, einem seit Studienbeginn Angst und Bange in Bezug auf den Arbeitsaufwand der Projektgruppe zu machen, war ich natürlich sehr gespannt wie arbeitsintensiv und ressourcenbindend die PG nun tatsächlich werden würde. Glücklicherweise wurde mir meine favorisierte PG zugeteilt und auch einige vertraute Gesichter erleichterten den Start in das einjährige Projekt.
In der einführenden Seminarphase schien die Motivation der Gruppenmitglieder ebenso groß zu sein wie eine konkrete Vorstellung über die Umsetzung unserer Pläne gering. Nach dem ersten Kontakt mit den Modellierungstools bildete sich aber schnell eine Idee, wie die Universität in einem Modell umzusetzen sei. Leider schoss diese Idee im ersten Semester derart weit über das Ziel hinaus, dass die Gruppe länger als geplant mit den entstandenen Modellen zu hadern hatte. Die Tatsache, dass der gesamte Arbeitsfluss im zweiten Semester viel flüssiger von statten ging, ist mit Sicherheit auf einen nicht zu unterschätzenden Lernfortschritt zurückzuführen, welcher sich nicht nur auf fachliches Wissen bezieht sondern auch auf das Erlernen der sozialen *Spielregeln*, die bei einer so langen Gruppenarbeit zu beachten sind.
Selbstverständlich war die intrinsische Leistungsmotivation bei den einzelnen PG-Teilnehmern unterschiedlich stark ausgeprägt, aber das darf bei einer Gruppengröße von zwölf Personen nicht überraschen und schlug durch eine strikte Aufteilung in kleinere Gruppen mit eigenem Verantwortungsbereich nicht verzögernd zu Buche. Alles in Allem war ich mit der PG zufrieden und werde mich hüten, anderen Studenten *Gruselgeschichten* á la „Du wirst ein Jahr jedes Wochenende durcharbeiten“ zu erzählen. Ich wünsche allen PG-Kameraden einen erfolgreichen weiteren Studienverlauf.

Yongfeng Mai

Obwohl ich die deutsche Sprache nicht Perfekt behersche, kamen die anderen PG mitglieder mir entgegen. Ich beginne mit dem Fazit des ersten Semester meines PGs. In der Mopra Phase kam ich in die Gruppe 2. Ich lernte dabei Menderes, Oktay und Akansel dadurch näher kennen. Ich habe mich bemüht, die mir gestellten Aufgaben, ordentlich zu erledigen. Für mich war das erste Semester eine Kennenlern Phase gewesen, die PG mitglieder und die Simulationswerkzeuge. Vergleich mit dem Werkzeug *Arena*, welche in Vorlesung MAO vorgestellt wird, die HIT und ProC/B haben auf mich sehr gut Eindruck gamacht. Mit der habe ich tiefer Verstand über Simulation gemacht. Im zweiten Semester kam ich in die Statistik Gruppe mit Oktay, Menderes und Genadi. Wir haben uns regelmäßig getroffen. Jeder ist für die geteilte Aufgabe sehr verantwortlich. Für die Negativ hatte ich das Gefühl gehabt, dass sich die Einzelnen Gruppen entfremden. Ich hatte dann fast nur noch mit der Statistik-Gruppe Kontakt. Die Athmosphäre im ersten Semester war nicht Vorhanden. Allerdings muss ich mich hier die PG mitglieder, insbesondere die von Gruppe2, statistiker Gruppe und unserem Betreuer Jürgen Mäter bedanken, dass sie sehr viel Gedult mit mir zu kommunizieren hatten und bei Problem halfen.

Oktay Özdemir

Bevor die PG anfing habe ich gehofft, dass ich in eine Gruppe reinkomme, in der Teamgeist, Disziplin und gegensetiger Respekt herscht. Dies sah ich auch in der ersten Phase, die Seminarphase, im ersten Semester. In der zweiten Hälfte des ersten Semesters wurde ich in die Statistik Gruppe eingeteilt. Zum Abschluss des ersten Semesters gab es eine Krisensitzung, wo die Statistik Gruppe von der PG als Engpass angesehen wurde. Dies war jedoch nicht der Fall, denn der Engpass waren die Modellierer, weil die richtigen Ergebnisse der Modellierer an unsere Gruppe spät übermittelt wurde. Für die

statistische Analyse, Modellvalidierung u.s.w. waren wir auf die Modellergebnisse der Modellierer angewiesen. Wir mussten unsere Aufgaben mehrmals wiederholen, da die Modellierer immer andere Ergebnisse für richtig gehalten haben. Noch enttäuschter wurde ich am Anfang des zweiten Semesters. Ich sah wie andere Teilnehmer der PG ihren Stundenplan so voll hatten, dass ich mir die Frage gestellt habe, ob sie überhaupt Zeit für die PG haben. Ich kann mit Sicherheit sagen, dass die Statistikgruppe unter den anderen Gruppen viel erleiden und einstecken musste. Ich habe bis zum Ende der PG die Aufgaben der Statistikgruppe bearbeitet und noch viele Dokumentationen korrigiert. Meine einzigste Danksagung gilt eigentlich für alle, aber besonders für die Statistikgruppe und an Jürgen. Jürgen war das ganze Jahr hart aber fair, hat uns immer vorangetrieben und bei den Aufgaben unterstützt. Trotz der negativen Erfahrungen in der PG, habe ich auch positive Erfahrungen gesammelt. Zuletzt möchte ich noch hinzufügen, dass ich mit der Arbeit der PG sehr zufrieden bin. Ich wünsche allen PG-Mitgliedern einen erfolgreichen Studienverlauf.

Jürgen Mäter

Als „alter“ Veranstalter von Projektgruppen, mittlerweile sind es 19 Projektgruppen, bin ich noch geprägt von der hochschuldidaktischen Gesprächsrunde, die 1979 in [HDG 79] sich mit dem Projektstudium an der ehemaligen Abteilung Informatik der Universität Dortmund befasst und neben einem Sachbericht auch einen Erfahrungsbericht über den Ablauf der Projektgruppe vorgeschlagen hat, siehe auch Anhang C an dem ich mich grob orientieren möchte.
Aus aktuellem Anlass, nämlich die inzwischen beschlossene Einführung von Studiengebühren für Studierende an der Universität Dortmund, habe ich das Thema der Projektgruppe vorgegeben. Mit dem Hintergedanken, dass einerseits das Thema nicht nur für Studierende sondern auch für die Universität, bzw. den Fachbereich Informatik von Interesse sein könnte und andererseits der Einsatz am Lehrstuhl Informatik IV entwickelter Modellierungstools getestet werden könnten.
Als fachliche Lernziele, die in dieser Projektgruppe erreicht werden sollten, seien hier genannt:

- die Erarbeitung von Grundkenntnissen in ereignisdiskreter Simulation,
- die Modellierung komplexer Vorgänge,
- der Einsatz von Modellierungswerkzeugen und
- die Anwendung statistischer Testverfahren an einem realen Problem.

Zu den ausserfachlichen Lernzielen gehörten:

- das Arbeiten im Team,
- die Koordination der einzelnen Teams sowie
- das Projektmanagement, insbesondere die Projektzeitplanung.

Im Laufe der Einarbeitungsphase in das Projektthema hat sich herausgestellt, dass die ursprüngliche Aufgabenstellung, die gesamte Universität zu analysieren, den Zeitrahmen von zwei Semestern aufgrund des nachzuholenden Wissensstandes der Projektmitglieder sprengen würde. Um zumindest partiell Aussagen bezüglich potentieller Engpässe im Studienverlauf machen zu können, haben wir uns auf die Analyse des Fachbereichs Informatik beschränkt. Wie sich später herausstellte, mussten auch hier noch weitere Einschränkungen getroffen werden, z.B. im wesentlichen nur den Studiengang Kerninformatik zu analysieren.

Als Initiator der Projektgruppe war mir von vornherein klar, dass ich keine besonderen Kenntnisse aus dem Arbeitsgebiet Modellierung und Simulation von den Projektgruppenmitgliedern erwarten konnte. Deshalb habe ich ein einführendes Kompaktseminar derart gestaltet, dass ein erster grober Überblick in dieses Gebiet erarbeitet werden konnte. Wegen des enormen Umfangs des Arbeitsgebietes habe ich die Basisliteratur vorgegeben, um Irrwege von vornherein auszuschliessen. Es blieb den Teilnehmern jedoch überlassen, weiterführende Literaturstellen selbst zu recherchieren wovon aber nur wenig Gebrauch gemacht wurde. 75 % der Vorträge waren überraschend gut bis sehr gut. Die anderen 25 Prozent mussten überarbeitet und innerhalb der folgenden Projektgruppensitzungen nochmals vorgetragen werden.
Das Seminar fand extern an zwei aufeinander folgenden Tagen im Haus Villigst statt. Ein externer Ort wurde deshalb gewählt, um das Kennenlernen der Gruppenmitglieder untereinander zu fördern und zu versuchen eine „corporate identity“ mit der Gruppe und dem Thema zu manifestieren, was zumindest für das erste Projektsemester gelungen ist.
Die der Projektgruppe zur Verfügung gestellten Ressourcen sind im Rechnerbereich gut bis sehr gut gewesen. Der Projektgruppenarbeitsraum, in dem auch die Projektpruppensitzungen abgehalten wurden, bedarf einer dringenden Renovierung und Zusatzausstattung. Das Fehlen einer Leinwand, um mit einem Beamer oder Overheadprojektor Ergebnisse zu projizieren, wurde mittels eines weissen Bettlakens, das von einer Vorgänger-PG „vererbt“ wurde -hoch lebe der Pioniergeist - kompensiert.
Zum Verlauf der Projektgruppe ist anzumerken, dass sich die Projektgruppe in drei feste und nahezu selbständige Untergruppen organisiert hatte, deren gruppenübergreifende Kommunikation hauptsächlich in den Plenumssitzungen stattfand, was zur Folge hatte, dass in der Regel nur untergruppenlokal kommuniziert wurde. Als Stabsstelle wurde zusätzlich eine Projektmanagementgruppe gebildet, die einen Projektzeitplan entwarf und den Projektfortschritt kontrollierte. Die Sitzungsleitung und Protokollführung fand rollierend statt. Als Veranstalter habe ich mich im Hintergrund gehalten und nur beratend eingegriffen.
Als fortführende Einarbeitung, d.h. zur Vertiefung der im Kompaktseminar erworbenen Kenntnisse, folgte ein Modellierungspraktikum. Während der erste Teil des Praktikums so ausgelegt war, dass anhand einfacher Modelle die Grundlagen der Modellierung und Simulation erlernt werden sollten, war im zweiten Teil die Anwendung der im ersten Teil erworbenen Kenntnisse und Fähigkeiten an ausgewählten größeren Modellen zu beweisen. Es hat sich bereits hier gezeigt, dass die Projektgruppenmitglieder wohl dieses Gebiet unterschätzt haben und sie ziemlich blauäugig an die Aufgabenstellungen gegangen sind. Selten wurde vertiefende Literatur ”freiwillig” durchgearbeitet, so dass das Praktikum nur mit vielen Hilfestellungen absolviert werden konnte.
Bei der im ersten Projektsemester angefertigten Simulationsstudie der Best- und Worstcase-Situation des Studiums am Fachbereich Informatik hat sich gezeigt, dass ein geplantes Vorgehen für die Gruppenmitglieder noch Probleme bereitete. Zum einen war der Plan nur bei einigen Mitgliedern im Kopf und nicht dokumentiert und zum andern sind die Modellierer ziemlich blauäugig an die Probleme herangegangen, d.h. es wurden gleich zu Anfang viel zu detaillierte Modelle entworfen deren Laufzeiten bis zu zwei Tage waren. Dass derart große Modelle schwer zu verifizieren sind, haben die Teilnehmer wohl inzwischen auch erkannt. Zur Validierung der ProC/B-Modelle wurde aufgrund fehlenden Datenmaterials für den Best- und Worstcase, ausgehend von einem konzeptionellen Modell, ein HIT-Modell erstellt, um Vergleichsdaten zu generieren und als Nebeneffekt auch das Modellierungswerkzeug HIT mit zu bewerten. Mangelhafte Absprachen der Modellierungsgruppen resultierten zuerst in nicht vergleichbaren Modellen, so dass die Statistikgruppe, die eine Validierung vornehmen sollte, erst mit Hilfe von zusätzlich erstellten Hilfsskripten Datentransformationen vornehmen mussten.
Bei der Erstellung des Zwischenberichtes, obwohl frühzeitig initiiert, ergaben sich erhebliche Organisationsprobleme, d.h. dass Teile nicht rechtzeitig fertig wurden, bzw. deren Qualität zu wünschen üb-

rig ließ. Es weckte den Anschein, dass sorgfältiges Arbeiten ungewohnt war und sich zu sehr auf Korrekturen meinerseits verlassen wurde. Letztendlich wurde der Zwischenbericht erst mitten im zweiten Projektsemester fertig gestellt.
Nun sollte man eigentlich meinen, dass aus den Erfahrungen des ersten Projektsemesters ein besseres, gruppenorientiertes Arbeiten im zweiten Semester an den Tag gelegt worden wäre. Das Gegenteil war eher der Fall, die Modellierer dominierten zwar in den Plenumssitzungen, arbeiteten aber noch isolierter als im ersten Semester, so dass Modellierungsfehler erst zu späten Zeitpunkten erkannt werden konnten. Das hatte zur Folge, dass, obwohl rechtzeitig geplant, eine Kapazitätsberechnung zur Ermittlung von Planzahlen, wie sie das Dezernat 2 der Universität Dortmund für den Fachbereich Informatik erstellt, erst nachträglich den Ergebnissen der Experimente gegenübergestellt werden konnte.
Die Umsetzung des konzeptionellen Modells in das ProC/B-, bzw. HIT-Modell erfolgte, meines Erachtens, mehr wie eine Online-Programmieraufgabe als eine, die Eigenschaften der beiden Modellierungswerkzeuge ausnutzende Modellierungsaktivität, was zur Folge hatte, dass die Modellierer prompt in die so genannte Nullzeitfalle stochastischer Simulationsmodelle getappt sind. Lösungsansätze aus der Literatur wurden nicht intensiv gesucht. Ich werde den Eindruck nicht los, dass man nur schnellst möglichst das Projekt mit minimalem Einsatz zu Ende bringen wollte, um nebenbei noch Seminare und Prüfungen zu absolvieren, was einer intensiven Projektgruppenarbeit widerspricht. Die einzigen, die sich mit „Neuland“ befassen mussten, war die Statistikgruppe, die zum einen statistische Testverfahren analysieren, einsetzen und zum andern ein ganz anderes Gebiet, nämlich Verfahren zu Parallelisierung von Simulationen, erarbeiten und prototypisch implementieren musste. Weil die Statistikgruppe der deutschen Sprache nicht ganz so mächtig war, hätte sie mehr Unterstützung hinsichtlich der zu erstellenden Dokumentationsteile benötigt. Hier überwog wohl mehr die o.a. minimalistische Vorgehensweise der anderen Teilnehmer, so dass sich die Qualität der gesamten Gruppe an der der Statistikgruppe messen lassen muss.
Äußerst positiv möchte ich den unbedarften Umgang mit den eingesetzten Modellierungswerkzeugen bewerten. Dank der Gruppe wurden noch wesentliche Fehler und Unschönheiten der Werkzeuge entdeckt. Insbesondere haben sich die Modellierer „gutmütig“ gegenüber entdeckter Fehler in den Werkzeugen gezeigt und die von den Entwicklern vorgeschlagenen Umgehungen, die oft aufwendig waren, akzeptiert.
Das vorgeschriebene Fachgespräch wurde termingerecht und erfolgreich in erfrischend kurzer Form absolviert.
Nach einem gemeinsamen Besuch des Dortmunder Weihnachtsmarktes im ersten Projektsemester war ich eigentlich guter Dinge und hoffte auf ein Zusammenwachsen der Gruppe. Externe Treffen fanden während der Vorlesungszeit im zweiten Projektsemester leider nicht statt, worunter, meines Erachtens nach, auch das Gruppenklima gelitten hat.
Für den Endbericht, in der vorliegenden Form, waren leider wieder zu viele Korrekturdurchläufe erforderlich, so dass sich gegenüber dem Zwischenbericht keine Fortschritte erkennen lassen und der gesteckte Zeitrahmen mal wieder nicht eingehalten werden konnte.
Die Projektgruppe hat das abgespeckte Minimalziel erreicht. Ich hoffe, dass die Teilnehmer für ihr weiteres Studium, insbesondere für ihre Diplomarbeiten, etwas über wissenschaftliche Vorgehensweisen gelernt haben.
Ich wünsche allen Teilnehmern für die Zukunft alles Gute und dass sie ihr Studium gut abschließen.

Akansel Serefliogl u

Die zweisemestrige PG ist nun vorbei. Trotz Kopfzerreißen, sehr sehr viel Stress und Zeitinvestition hat es die PG nun endlich geschafft, all die geforderten Aufgaben zu erfüllen. Die gesamte Atmo-

sphäre während dem Praktikum würde ich als gut bezeichnen. Ich fand es sehr von Vorteil, dass wir zu Beginn der Projektgruppe einen Ausflug, für die Präsentationen und den Vorträge der einzelnen PG-Mitglieder, gemacht hatten. Dadurch konnte sich die Gruppe sehr schnell kennenlernen und es enstand eine positive Atmosphäre in der Gruppe. Obwohl ich schon einige Personen vor Beginn der Projektgruppe kannte, bemühte ich mich, auch die anderen Teilnehmer besser kennenzulernen. Im Verlaufe des gesamten Praktikums gab es meines Angesichts keine großen Probleme. Ich finde, dass alle Teilnehmer ihre Aufgaben zufriedenstellend erledigt haben und sich nun freuen, ihren Schein endlich ausgehändigt zu bekommen. Denn das eine Jahr war wirklich sehr stressig. Auch ich persönlich bin sehr froh darüber, dass es endlich so gut wie vorbei ist. Ich möchte hier auch keine Einzelkritik ausüben, oder einzeln auf Personen eingehen, da ich im großen und ganzen finde, dass jeder Teilnehmer die von ihm geforderten Aufgaben gut bewältigt hat. Zum Schluss möchte ich mich noch bei unserem Betreuer, Jürgen Mäter, herzlichst bedanken, der die Gruppe sehr gut betreut hat. Er war bei allen Fragen, die wir hatten, und bei allen Unklarheiten für uns da, und hat versucht uns sofort zu helfen. Vielen Dank

Michael Thüner

Als ich nach der Verteilung der PG-Plätze gesehen habe, dass ich in dieser PG gelandet bin, war ich erst etwas unsicher, ob mir das gefallen sollte, zumal mir meiner Ansicht nach etliche Vorraussetzungen fehlten. Wie sich herausstellte, fehlten diese aber den meisten PG-Teilnehmern, und dies wurde in der einleitenden Seminarphase aufgeholt.

Während ich der Zusammenarbeit anfangs mit gemischten Gefühlen entgegen sah, hat sich dies durch die Seminarphase erst gänzlich geändert. Wir haben uns alle sofort gut verstanden, und auch viel Spaß gehabt. Leider muss ich zugeben, dass sich das im Laufe des letzten Jahres wieder geändert hat - und ich denke auch nicht nur bei mir. Insgesamt ist bei mir im Laufe der Zeit, und - trotz einer vom PG-Leiter einberufenen Krisensitzung im Februar - vorallem im zweiten Semester, der Eindruck entstanden, dass die Arbeit sehr unterschiedlich verteilt bzw. auch die Arbeitsmoral sehr verschieden war. Dies hat meiner Ansicht nach dazu geführt, dass es gerade jetzt gegen Ende der PG bei einigen Leuten stark brodelte und die Moral deutlich sank. Für mich klares Zeichen dafür ist auch die Tatsache, dass zu Beginn des ersten Semesters oft und gerne von gemeinsamen Treffen und Aktionen in der Freizeit gesprochen wurde - wie z.B. der gemeinsame Weihnachtsmarktbesuch, oder das damals schon verkündete gemeinsame WM schauen diesen Sommer. Letzteres hat schon nicht mehr stattgefunden.

Die Betreuung durch Jürgen war eigentlich auch sehr gut. Er stand für Fragen und Probleme jeder Zeit zur Verfügung und die äußerst kritische Haltung zu unserem Zwischenbericht hat uns für den Endbericht sehr geholfen, so dass wir diesen wohl aller Voraussicht nach nicht mit 3 oder 4 Monaten Verspätung in endgültiger Fassung haben werden. Ich hätte mir nur im Bezug auf die oben angesprochenen Probleme gewünscht, dass die sogenannten Gelben Karten wirklich personenbezogen geäußert worden wären. So eher allgemein wie es damals formuliert war, hatte ich nachher den Eindruck, das sich eher die falschen Leute angesprochen und angespornt gefühlt haben.

Zum Thema der PG bleibt mir nur zu sagen, dass ich nach anfänglicher Skepsis sehr viel Spaß daran gefunden habe, auch wenn es eben auch viel Arbeit bedeutete.

A. Skripte für die parallele Simulation

Im folgenden werden die Skripte für die parallele Simulation vorgestellt. Zuerst wird das Skript *parsimscript* abgebildet, welches die Hauptaufgaben der parallelen Simulation ausführt. Im Anschluss daran folgt das Skript *parascript*, das vom *parsimscript* benötigt wird, um die Simulationsrechner zu steuern.

A.1. parsimscript

```
#!/bin/csh -f
#*********************************************************
#*  Aufruf:
#*  =======
#*   parsimscript [rechner.txt][hit-datei][seed][replic]
#*
#*  Author: Statistik Gruppe PG476
#*  =======
#*
#*  Datum: MAI 2006
#*  ======
#*
#*  Parameter:
#*  ===========
#*
#*  rechner.txt  --->   Datei die die sim-Rechner
#*                      koordiniert.
#*  hit-Datei    --->   Die eigentliche HIT-Datei
#*                      die übergeben wird.
#*  seedwert     --->   Bestimmt den Startseed-Wert
#*  replic       --->   Anzahl der Replikationen.
#*
#*********************************************************

#*** Parameter ***************************************
#
# Hier werden die benötigten Parameter deklariert.
#

set rechner      = "$1"
set hitdatei     = "$2"
set seedwert     = "$3"
set replicanzahl = "$4"
```

```
set pfad         = `echo $PWD`
set rechnerliste = `more rechner.txt`

#*** Vorrausgesetzten Parameter**************************
#
# In diesem Part wird aufgefordert, die Steuerdatei, eine
# HIT-Datei, einen Startseed-Wert und  die Anzahl der
# Replikationen einzugeben
#

if ("$1" == "") then
   echo " "
   echo "Warnung: Bitte eine Steuerdatei angeben! "
   exit 1
endif
if ("$2" == "") then
   echo " "
   echo "Warnung: Bitte eine HIT-Datei angeben! "
   exit 1
endif

if ( "$3" == "" ) then
     echo " "
     echo "Bitte geben Sie den Startseed-Wert ein! "
     echo "(Die Angabe von Primzahlen ist zu empfehlen)"
     echo -n ">> "
     set seedwert = $<
       else
         set seedwert = $3
endif

if ( "$4" == "" ) then
    echo " "
    echo "Bitte geben sie die Anzahl Ihrer
          Replikationen ein! "
    echo -n ">> "
    set replicanzahl = $<
      else
        set replicanzahl = $4
endif


#*** Löschen und erneutes Erstellen der benötigten *************
#*** Dateien und Ordner.                            *************
#
# In diesem Block werden die Hilfsordner erstellt, die für die
# Verifikation gebraucht werden, wobei in dem seeddir-Ordner
```

```
# die erzeugten Seed-Werte gespeichert sind.
# Alle durch den HI-SLANG-Compiler generierten Dateien werden
# im hitdir-Ordner abgelegt.
# Die für die einzelnen Simulationsrechner erstellten Teilskripte
# werden in den subscriptdir-Ordner geschrieben. Die hier
# erzeugten Ergebnisse werden im ergebnisdir-Ordner gespeichert.
#

if ( ! -e seeddir ) then
    mkdir seeddir
else
    rm -f seeddir/*
endif

if ( ! -e hitdir ) then
    mkdir hitdir
else
    rm -f hitdir/*
endif

if ( ! -e ergebnisdir ) then
    mkdir ergebnisdir
else
    rm -f ergebnisdir/*
endif

if ( ! -e subscriptdir ) then
    mkdir subscriptdir
else
    rm -f subscriptdir/*
endif


#*** Seed-Werte werden erstellt ************************
#
# Der Code zur Erstellung der Seed-Werte wird
# in die Datei startseed.hit eingetragen.
#

echo "%COMMON                                          "
                          >  $pfad/seeddir/startseed.hit
echo "%PARM = INDENT = 0                               "
                          >> $pfad/seeddir/startseed.hit
echo '%BIND "hit_aus" TO '"$pfad"'/newstartseed.txt  '
                          >> $pfad/seeddir/startseed.hit
echo "%END                                             "
                          >> $pfad/seeddir/startseed.hit
```

```
echo "EXPERIMENT seed METHOD SIMULATIVE;              "
                          >> $pfad/seeddir/startseed.hit
echo "VARIABLE n,z:INTEGER;                          "
                          >> $pfad/seeddir/startseed.hit
echo "        f:OUTFILE;                             "
                          >> $pfad/seeddir/startseed.hit
echo "      run:INTEGER;                             "
                          >> $pfad/seeddir/startseed.hit
echo "    replic:INTEGER;                            "
                          >> $pfad/seeddir/startseed.hit
echo "    startseed:INTEGER;                         "
                          >> $pfad/seeddir/startseed.hit
echo "    myseed:INTEGER;                            "
                          >> $pfad/seeddir/startseed.hit
echo "BEGIN                                          "
                          >> $pfad/seeddir/startseed.hit
echo ' OPEN f, "hit_aus" LENGTH 80;                  '
                          >> $pfad/seeddir/startseed.hit
echo " startseed, myseed:=$seedwert;                 "
                          >> $pfad/seeddir/startseed.hit
echo " replic := $replicanzahl;                      "
                          >> $pfad/seeddir/startseed.hit
echo " n:=2**30//replic+1;                           "
                          >> $pfad/seeddir/startseed.hit
echo " FOR run := 1 STEP 1 UNTIL replic              "
                          >> $pfad/seeddir/startseed.hit
echo " LOOP                                          "
                          >> $pfad/seeddir/startseed.hit
echo "      myseed:=get_seed(startseed,n);           "
                          >> $pfad/seeddir/startseed.hit
echo "startseed:=myseed;                             "
                          >> $pfad/seeddir/startseed.hit
echo "      WRITE FILE f, myseed;                    "
                          >> $pfad/seeddir/startseed.hit
echo "  WRITELN FILE f;                              "
                          >> $pfad/seeddir/startseed.hit
echo " END LOOP;                                     "
                          >> $pfad/seeddir/startseed.hit
echo "CLOSE f;                                       "
                          >> $pfad/seeddir/startseed.hit
echo "END EXPERIMENT seed ;                          "
                          >> $pfad/seeddir/startseed.hit
```

```
#*** Aufruf von HIT ***********************************
#
# Es findet ein Aufruf von HIT statt, um die Seed-Werte zu
# erzeugen und in die Datei newstartseed.txt zu speichern.
#

 /home/pg476/share/app_solaris/bin/hit
                              $pfad/seeddir/startseed.hit
rm t.*


#*** Aufruf des HI-SLANG-Compilers *************************

env option=norun /home/pg476/share/app_solaris/bin/hit $hitdatei
cp t.hitcode ~/t.hitcode


#*** Erzeugung einer Pipeline *************************
#
# Mit dem Befehl "mkfifo" wird eine Pipeline erzeugt.
# Die wird benötigt, um die Kommunikation zwischen dem
# parsimscript und allen parascripte zu gewährleisten.
#

if ( -e myfifo )then
    rm myfifo
    mkfifo myfifo
else
    mkfifo myfifo
endif


#*** Bestimmung der Rechneranzahl **********************

set zaehler = `expr ${#rechnerliste}`


#*** Aufspaltung der Seed-Werte ************************
#
# Die Seed-Werte aus der Datei newstartseed.txt werden,
# je nach Anzahl der Simulationsrechner, in einzelne
# kleinere Dateien aufgeteilt.
#

set teiler = `expr $replicanzahl / $zaehler`
set j = 1
set temp = $teiler
```

```
while ( $zaehler != 0 )

     gsed -n ''"$j"',"$teiler"'p' "$pfad"/newstartseed.txt
            > "$pfad"/teil"$rechnerliste[$zaehler]".txt
     @ zaehler = $zaehler - 1
     @ j = 1 + $teiler
     @ teiler = $teiler + $temp
end

@ teiler = $teiler - $temp
set rest = 1
while ( "$teiler" < "$replicanzahl" )
     @ teiler = $teiler + 1
     gsed -n ''"$teiler"'p' "$pfad"/newstartseed.txt
           >> "$pfad"/teil"$rechnerliste[$rest]".txt
     @ rest = $rest + 1
end


#*** Erzeugung der einzelnen Rechnerskripte mittels ***
#*** parascript                                     ***
#
# Es werden, je nach Simulationsrechner, Skripte erzeugt.
# Diese Skripte werden durch eine while-Schleife im
# Hintergrund gestartet.
#

set i = 1
while ( $i <= ${#rechnerliste} )
  gsed -e '1a\
   set Ausgabedatei = "ergebnis_'$rechnerliste[$i].txt'"'
   -e '1a\set Eingabedatei = "teil'$rechnerliste[$i].txt'"'
   -e '1a\set Rechner = "'$rechnerliste[$i]'"' parascript
            > parascript_$rechnerliste[$i]
  chmod 700 parascript_$rechnerliste[$i]
  @ i = $i + 1
end

set i = 1
while ( $i <= ${#rechnerliste} )

    ./parascript_$rechnerliste[$i] &
    @ i = $i + 1
end
```

```
#*** Ende des parascript's  ***************************
#
# Wenn das parascript zum Ende gekommen ist, sendet es
# an die Pipeline ein Endsignal ("End"). Das Hauptskript
# wartet, bis es von allen parascript's das Endsignal
# bekommt. Erst dann arbeitet es weiter.
#

set i = `expr ${#rechnerliste}`
while ( $i != 0 )
    set server = `cat < myfifo`
    set j = ${#server}
    while ( $j != 0 )
        if ( $server[$j] == "End" ) then
            @ i = $i - 1
        endif
        @ j = $j - 1
    end
end


#*** Ergebnis wird erstellt ***************************
#
# Das gesamt Ergebnis wird aus den einzelnen
# Teil- Ergebnissen erstellt.
#

set i = 1
if ( -e ergebnis_$rechnerliste[$i].txt ) then
    more ergebnis_$rechnerliste[$i].txt > Ergebnis.txt
    mv ergebnis_$rechnerliste[$i].txt ergebnisdir/
    @ i = $i + 1
endif
while ( $i <= ${#rechnerliste} )
    more ergebnis_$rechnerliste[$i].txt >> Ergebnis.txt
    mv ergebnis_$rechnerliste[$i].txt ergebnisdir/
    @ i = $i + 1
end


#*** temporaere Dateien werden geloescht *************
#
# Die nicht benötigten Dateien verursachen Overhead.
# Außerdem ist ihre Existenz nicht von Bedeutung und
# werden gelöscht.
#
```

```
rm myfifo
mv t.* hitdir/
mv ~/t.* hitdir/
mv parascript_* subscriptdir/
rm hitdir/*
rm subscriptdir/*
rm seeddir/*
rm ergebnisdir/*
rmdir hitdir
rmdir seeddir
rmdir ergebnisdir
rmdir subscriptdir
echo Hauptprogramm zum Ende
```

A.2. parascript

```
#!/bin/csh -f
#*** Die Ausgabedatei, falls sie existiert, wird *******
#*** gelöscht.                                   *******

if ( -e $Ausgabedatei ) then
    rm $Ausgabedatei
endif


set pfad = `echo $PWD`


#*** SED - Skript ************************************
#
# Im folgenden findet ein sed-Aufruf statt, dass
# zusätzliche Zeilen in die Datei "t.hitcode_$Rechner.sim"
# erstellt.
#

gsed -e '1i\%COMMON' -e '1i\%PARM = WARN'
 -e '1i\%BIND "hit_aus" TO extend '$PWD'/'$Ausgabedatei''
 -e '1i\%BIND "seedDateiName" TO '$PWD'/'$Eingabedatei''
 -e '1i\%END' t.hitcode.sim
            > t.hitcode_$Rechner.sim


#*** Remote SHELL ************************************
#
# Das Kernstück der Simulation. Hier werden die Anweisungen
# an die Simulationsrechner übergeben.
#

rsh $Rechner cp ~/t.hitcode /tmp/$user/t.hitcode
rsh $Rechner cp $pfad/t.hitcode_$Rechner.sim
                    /tmp/$user/t.hitcode_$Rechner.sim

while ( ! -z $Eingabedatei )
    rsh $Rechner " cd /tmp/$user ;
     /home/pg476/share/app_solaris/bin/hit
           /tmp/$user/t.hitcode_$Rechner.sim exp"
     sed -e 1d $Eingabedatei > temp$Rechner
     cp temp$Rechner $Eingabedatei
end
```

```
#*** Die temporären Dateien werden gelöscht. ************
#
# An die Pipeline wird das Endsignal gesendet.
#

rm temp$Rechner
rm $Eingabedatei
rsh $Rechner rm /tmp/$user/t.*
echo "End" > myfifo&
```

B. Validierung der Bestcasemodelle

Dieser Anhang beinhaltet die Fortsetzung der im Zwischenbericht [ZB 06] angefangenen Modellvalidierung. Es wurde jeweils ein Modell mit dem Tool HIT und dem Tool ProC/B erstellt. Es wird angenommen, dass beide Modelle das Geschehen an der Universität Dortmund hinsichtlich des Fachbereichs Informatik repräsentieren. Das Ziel ist es zu überprüfen, ob beide Modelle mit gleichen Eingaben Ergebnisse liefern, die mit hoher Wahrscheinlichkeit übereinstimmen.
Bei der Ergebnisvalidierung wird entsprechend dem Kochrezept aus dem Zwischenbericht vorgegangen. Dazu müssen alle Simulationsergebnisse, die validiert werden sollen, nach zehn Simulationsausführungen, in Tabellenform gespeichert, im R-Tool gescannt und mit dem KS-Test gegenübergestellt werden. Der KS-Test wird deswegen zur Überprüfung der Ergebnisse aus dem HIT- und ProC/B-Modell benutzt, weil die Eingabedaten für die beiden Modelle diskret sind und im Kochrezept der angewendete t-Test nur für kontinuierlichen Werte benutzt werden kann und in diesem diskreten Fall nicht anwendbar ist. Die Vorgehensweise bei der Validierung ähnelt der Validierung, die im Zwischenbericht beschrieben wird. Es wird lediglich das *Shell*-Skript für den HI-SLANG-Code des ProC/B-Modells modifiziert, um überflüssige Messpunkte zu verwerfen, Observermesspunkte zu reduzieren und die Zeilen *OPEN f2, „PROC_AUS1“ LENGTH 80;* und *CLOSE f2* an anderen Stelle zu platzieren, weil sie im *Shell*-Skript des Zwischenberichtes an falscher Stelle positioniert wurden. Das *Shell*-Skript wird mit dem Befehl

```
./sed.script
```

aufgerufen. Im Weiteren wird das *Shell*-Skript gezeigt und die geänderten Skriptzeilen mit ihren Funktionen dargestellt.
Für die bessere Übersicht wurden alle Zeilen untereinander geschrieben und durchnummeriert [1]. In den Zeilen 14 und 15 wird der Modellname ermittelt und für die weitere Verwendung im *Shell*-Skript angepasst. In den Zeilen 16 bis 24 werden die Zeilennummern ermittelt, in denen die Messpunkte *POPULATION* und *TURNAROUNDTIME* eingefügt bzw. ab welcher Zeile überflüssige Messpunkte gelöscht werden sollen. Mit der Zeile 36 werden die beiden *OUTFILES f1* und *f2* deklariert. Die Anzahl der *Observerpunkte* wird in Zeile 37 auf zehn gesetzt. Da das *OUTFILE f2* im Experimentblock geöffnet werden soll, wird es mit der Zeile 39 an entsprechender Stelle eingefügt. In Zeile 45 wird *LET SEED := myseed* gesetzt. Die Zeilen 46 und 47 bleiben mit einer Ausnahme gegenüber dem Zwischenbericht unverändert. Die Stoppbedingung *GLOBALSTOP WIDTH 5* wird gelöscht, da für die Validierung die Anzahl der Professoren und HiWis interessant ist. Deswegen sollte das Modell erst nach der Modellzeit gestoppt werden. *CLOSE f2* wird durch Zeile 55 im Experimentblock eingefügt und damit die Ausgabedatei geschlossen. Die Zeilen 69 bis 72 sind für das Löschen der unnötigen Messpunkte verantwortlich. Mit Zeile 73 wird die Variable *f2* für das gesamte Modell sichtbar gemacht.

[1] 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 = "BestCaseAlles.E1.hit"
3.set output_file = "Ausgabe1.E1.hit"
4.set findlineModel = `sed -e "/^TYPE/=" -n $input_file`
5.set addLineNumberModel = `expr $findlineModel[2]`
6.set findlineMeasureT = `sed -e "/MEASURE TURNAROUNDTIME/="
      -n $input_file`
7.set addLineNumberMeasureT = `expr $findlineMeasureT[1]`
8.set findlineMeasureP = `sed -e "/MEASURE POPULATION/="
      -n $input_file`
9.set addLineNumberMeasureP = `expr $findlineMeasureP[1]`
10.set lastNumberVar = `sed -e "/DEFAULT ESTIMATOR/="
      -n $input_file`
11.set sublastNumberVar = `expr $lastNumberVar - 1`
12.set lastNumberMeasure = `sed -e "/CONTROL AT messgeraet
      STOP MODELTIME 5200;/=" -n $input_file`
13.set sublastNumberMeasure = `expr $lastNumberMeasure - 1`
14.set Name = `sed -e "/_Typ MODEL;/p" -n $input_file`
15.set l = ` echo $Name[2] | sed "s/_Typ//g" `
16.set ModellNameLine = `sed -e "/messgeraet VIA $l;/="
       -n  $input_file`
17.set n = `expr $ModellNameLine + 1`
18.set m = `expr $ModellNameLine + 2`
19.set k = `expr $ModellNameLine + 3`
20.set ModellName = `sed -e "$n p" -n  $input_file`
21.set ModellName1 = `sed -e "$m p" -n  $input_file`
22.set ModellName2 = `sed -e "$k p" -n  $input_file`
23.set ModelNameSuper = `sed -e "$addLineNumberModel p"
       -n $input_file`
24.set b = ` echo $ModellName1[1] | sed "s/,//g" `

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

25.sed -e '/^%PARM = WARN/a\%BIND "PROC_AUS" to EXTEND
      '"$x/procb.data"''
26.-e '/^%PARM = WARN/a\%BIND "PROC_AUS1" to EXTEND
      '"$x/sem.data"''
27.-e '/^%PARM = WARN/c\%PARM = WARN, minmax'
28.-e '/^EXPERIMENT versuch METHOD SIMULATIVE/a\ VARIABLE'
```

```
29.-e '/^EXPERIMENT versuch METHOD SIMULATIVE/a\
      replic    : INTEGER;'
30.-e '/^EXPERIMENT versuch METHOD SIMULATIVE/a\
      startseed : REAL;'
31.-e '/^EXPERIMENT versuch METHOD SIMULATIVE/a\
      myseed    : REAL;'
32.-e '/^EXPERIMENT versuch METHOD SIMULATIVE/a\
      pop       : REAL;'
33.-e '/^EXPERIMENT versuch METHOD SIMULATIVE/a\
      n         : INTEGER;'
34.-e '/^EXPERIMENT versuch METHOD SIMULATIVE/a\
      run       : INTEGER;'
35.-e '/^EXPERIMENT versuch METHOD SIMULATIVE/a\
      turn      : REAL;'
36.-e '/^EXPERIMENT versuch METHOD SIMULATIVE/a\
      f1,f2     : OUTFILE;'
37.-e '/^COMPONENT modellobserver : observer/c\
      COMPONENT modellobserver : observer(5200/10);'
38.-e '/^EVALUATE MODEL/i\OPEN f1, "PROC_AUS"  LENGTH 80;'
39.-e '/^EVALUATE MODEL/i\OPEN f2, "PROC_AUS1" LENGTH 80;'
40.-e '/^EVALUATE MODEL/i\ replic :=10;'
41.-e '/^EVALUATE MODEL/i\ n := 2**30// replic+1;'
42.-e '/^EVALUATE MODEL/i\ startseed, myseed :=13;'
43.-e '/^EVALUATE MODEL/i\     FOR run :=1
      STEP 1 UNTIL replic'
44.-e '/^EVALUATE MODEL/i\         LOOP'
45.-e '/^EVALUATE MODEL '"$1"' : '"$1"'_Typ(LET SEED := 13);
      /c\EVALUATE MODEL '"$1"' :
      '"$1"'_Typ(f2,LET SEED := myseed);'
46.-e "/^MEASURE THROUGHPUT AT messgeraet;/a\MEASURE
      TURNAROUNDTIME AT $ModellName[1] DUE TO ALL
      ABSCISSA run ESTIMATOR MEAN, STANDARDDEVIATION,
      CONFIDENCE LEVEL 95;"
47.-e "/^MEASURE THROUGHPUT AT messgeraet;/a\MEASURE
      POPULATION AT $ModellName[1] DUE TO ALL
      ABSCISSA run ESTIMATOR MEAN, STANDARDDEVIATION,
      CONFIDENCE LEVEL 95;"
48.-e '/^END EVALUATE/a myseed :=get_seed (startseed, n);'
49.-e '/^END EVALUATE/a\startseed := myseed;'
50.-e '/^END EVALUATE/a\pop  := get_result("MEAN",
      "POPULATION","'$ModellName[1]'","all","dump",run);'
51.-e '/^END EVALUATE/a\turn := get_result("MEAN",
      "TURNAROUNDTIME","'$ModellName[1]'","all","dump",run);'
52.-e '/^END EVALUATE/a\ WRITELN FILE f1,pop,turn;'
53.-e '/^END EVALUATE/a\END LOOP;'
54.-e '/^END EVALUATE/a\CLOSE f1;'
55.-e '/^END EVALUATE/a\CLOSE f2;'
```

```
56.-e '/^MaxProfLS2:=0;;/i\WRITELN FILE f2,  "Lehrstuhl   ",
      "MaxProf    ", "MaxHiWi ";'
57.-e '/^MaxProfLS2:=0;;/i\WRITELN FILE f2,  "Lehrstuhl2",
      MaxProfLS2, MaxHiWiLS2;'
58.-e '/^MaxProfLS3:=0;;/i\WRITELN FILE f2,  "Lehrstuhl3",
      MaxProfLS3, MaxHiWiLS3;'
59.-e '/^MaxProfLS5:=0;;/i\WRITELN FILE f2,  "Lehrstuhl5",
      MaxProfLS5, MaxHiWiLS5;'
60.-e '/^MaxProfLS6:=0;;/i\WRITELN FILE f2,  "Lehrstuhl6",
      MaxProfLS6, MaxHiWiLS6;'
61.-e '/^MaxProfLS10:=0;;/i\WRITELN FILE f2, "Lehrstuhl10",
      MaxProfLS10, MaxHiWiLS10;'
62.-e '/^MaxProfLS12:=0;;/i\WRITELN FILE f2, "Lehrstuhl12",
      MaxProfLS12, MaxHiWiLS12;'
63.-e '/^MaxProfLSA:=0;;/i\WRITELN FILE f2,  "LehrstuhlLSA",
      MaxProfLSA, MaxHiWiLSA;'
64.-e '/^MaxProfLSB:=0;;/i\WRITELN FILE f2,  "LehrstuhlLSB",
      MaxProfLSB, MaxHiWiLSB;'
65.-e '/^MaxProfLSS:=0;;/i\WRITELN FILE f2,  "LehrstuhlLSS",
      MaxProfLSS, MaxHiWiLSS;'
66.-e '/^MaxProfLSPG:=0;;/i\WRITELN FILE f2, "LehrstuhlLPG",
      MaxProfLSPG, MaxHiWiLSPG;'
67.-e '/^MaxProfLSLNW:=0;;/i\WRITELN FILE f2,"LehrstuhlLNW",
      MaxProfLSLNW, MaxHiWiLSLNW;'
68.-e '/^MaxProfLSLNW:=0;;/i\WRITELN FILE f2, "          ";'
69.-e ''"$addLineNumberMeasureT,$sublastNumberMeasure"' c\  '
70.-e ''"$addLineNumberMeasureP,$sublastNumberMeasure"' c\  '
71.-e ''"$k,$sublastNumberVar"'c\  '
72.-e ''"$m"' c\'"$b"''
73.-e '/^TYPE '"$l"'_Typ MODEL;/c\TYPE '"$l"'_Typ MODEL
      (f2:OUTFILE); '
74.$input_file > $output_file
```

Nachdem das Skript auf den HI-SLANG-Code des ProC/B-Modells angewendet und dann das HI-SLANG-Programm ausgeführt wurde, stehen in den Dateien *procb.data* und *sem.data* die Ergebnisse des ProC/B-Modells zur Verfügung. Die Ausgaben in *procb.data* sehen wie folgt aus:

procb.data:

```
 1.835384E+003 2.335000E+002
 1.748914E+003 2.335000E+002
 1.824825E+003 2.335000E+002
 1.775550E+003 2.335000E+002
 1.798930E+003 2.335000E+002
 1.741931E+003 2.335000E+002
 1.723647E+003 2.335000E+002
 1.710777E+003 2.335000E+002
 1.707556E+003 2.335000E+002
```

```
 1.836688E+003 2.335000E+002
```

Hierbei entspricht die erste Spalte der *Population* und die zweite Spalte der *Turnaroundtime*. Bei *sem.data* werden drei Spalten ausgegeben. Die erste Spalte zeigt die angeforderten Lehrstühle. In der zweiten und dritten Spalte wird die maximale Anzahl der angeforderten Professoren bzw. HiWis in SWS angezeigt. Da die *sem.data* mehr als 20000 Zeilen enthält, werden hier nur das erste Semester des ersten Laufs und das letzte Semester des letzten Laufs dargestellt.

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             8         88
Lehrstuhl3             2         27
Lehrstuhl5             2         27
Lehrstuhl6             4         44
Lehrstuhl10            0        172
Lehrstuhl12            0          0
LehrstuhlLSA           4         54
LehrstuhlLSB           4         54
LehrstuhlLSS           4         54
LehrstuhlLPG         176        176
LehrstuhlLNW           3         27
```

Die erhaltenen Ergebnisse machen die eigentliche Validierung der beiden Modelle möglich. Dazu werden im R-Tool die Tabellen *studenten.data* vom HIT-Modell und die *procb.data* vom ProC/B-Modell ausgelesen und in Arrays *x* bzw. *y* gespeichert. Die dafür benötigten Befehle sehen wie folgt aus:

```
> x<-read.table("studenten.data",header=TRUE)
> y<-read.table("procb.data",header=TRUE)
```

und liefern die folgenden Ausgaben:

```
> x
   Population Turnaroundtime
1    1684.595            234
2    1774.525            234
3    1769.940            234
4    1813.210            234
5    1793.870            234
6    1858.005            234
7    1734.660            234
8    1768.925            234
9    1830.780            234
10   1791.010            234

> y
   Population Turnaroundtime
1    1835.384          233.5
2    1748.914          233.5
3    1824.825          233.5
4    1775.550          233.5
5    1798.930          233.5
6    1741.931          233.5
7    1723.647          233.5
8    1710.777          233.5
9    1707.556          233.5
10   1836.688          233.5
```

Da die Werte für die *Turnaroundtime* als konstante Werte ausgegeben werden, kann man sie nicht mit dem KS-Test vergleichen. Die von beiden Modellen ausgegebene Zeit entspricht neun Semestern (234 Wochen = 9 Semester je 52 Wochen). Die Ursache für die halbwöchige Abweichung bei dem ProC/B-Modell liegt an der Verzögerung, die für die Synchronisation im Modell eingebaut wurde. Also wenn diese Verzögerung zu der *Turnaroundtime* vom ProC/B-Modell addiert wird, wird diese Zeit auch neun Semestern entsprechen (233.5 + 0.5 = 234 Wochen = 9 Semester je 52 Wochen). Die Mittelwerte und die Standardabweichungen der *Population* und *Turnaroundtime* der beiden Modelle sind im Folgenden dargestellt.

HIT-Modell

```
> mean(x)
    Population Turnaroundtime
      1781.952        234.000
> sd(x)
    Population Turnaroundtime
      48.82332        0.00000
```

ProC/B-Modell

```
> mean(y)
    Population Turnaroundtime
      1770.420        233.500
> sd(y)
    Population Turnaroundtime
      50.97039        0.00000
```

Die Mittelwerte der *Population* unterscheiden sich ungefähr um 10 Studenten, was durchaus akzeptabel ist. Die Standardabweichungen zeigen, dass die Schwankungen um den Mittelwert nicht so gewaltig sind, so dass die Unterschiede während der gesamten Simulationslaufzeit in dem gleichen Bereich liegen. Der darauf folgende KS-Test für die *KI-Population* ist folgendermaßen auszuwerten:

```
> ks.test(x$Population,y$Population)
        Two-sample Kolmogorov-Smirnov test
        data:  x$Population and y1
        D = 0.3, p-value = 0.787
        alternative hypothesis: two.sided
```

Der *p-value*-Wert des KS-Tests zeigt die Wahrscheinlichkeit, ob die Verteilungen der beiden Ausgaben für die *Population* gleich sind. Bei dem durchgeführten Test ist der *p-value*-Wert 78,8%. Das sagt aus, dass die beiden Verteilungen mit der Wahrscheinlichkeit von 78,8% übereinstimmen.
Im Folgenden werden die Ausgaben von Professoren und HiWis verglichen. Entgegen dem ProC/B-Modell liefert das HIT-Modell am Ende den Verbrauch von Ressourcen *Professor-SWS* bzw. *HiWi-SWS* aus dem ganzen Fachbereich Informatik und nicht getrennt nach jedem Lehrstuhl.

Ausgabe vom HIT-Modell:

```
 Professor   HiWi
   55        161
   56        161
   55        405
   56        422
   55        583
   56        528
   211       808
   212       797
   147       779
   148       726
   219       833
    .         .
    .         .
    .         .
```

Die Ausgabe des ProC/B-Modells wurde in der Datei *sem.data* gezeigt. Um auf beide Ausgaben den KS-Test anwenden zu können, müssen noch folgende Änderungen vorgenommen werden. Auf die Datei *sem.data* wird der Sed-Befehl

```
sed -e '/^$/d' -e '1d' -e '$a\MaxProf    MaxHiWi'
                                        sem.data > sem1.data
```

angewendet. Mit diesem Befehl wird eine Kopie von der Datei *sem.data* erzeugt, die *sem1.data* heißt. In dieser Datei werden die Leerzeilen gelöscht und die erste Zeile an das Ende verschoben. Dies wurde deswegen vorgenommen, um weitere notwendige Berechnungen zu vereinfachen.
Danach wird wieder ein Shell-Skript ausgeführt. Dieses addiert die Anzahl der Professoren- bzw. HiWis-SWS in einem Semester zusammen. Das wird gebraucht, um den Vergleich mit den Daten aus dem HIT-Modell durchzuführen.

```
#!/bin/csh -f
#set echo
1.set professor=0
2.set hiwi=0
3.set i=1
4.echo Professor "      " HiWi>ergebnis
5.set zahl=`sed -e "$i p" -n sem1.data`
6.while ( "$zahl" != "" )
7.     if ( $zahl[2] != "MaxProf" ) then
8.         set professor=`expr $professor + $zahl[2]`
9.         set hiwi=`expr $hiwi + $zahl[3]`
10.    else
11.         echo $professor "              " $hiwi>>ergebnis
12.         set professor=0
13.         set hiwi=0
14.    endif
15.    set i=`expr $i + 1`
16.    set zahl=`sed -e "$i p" -n sem1.data`
17.end
```

Mit der ersten und der zweiten Zeile werden zwei Variablen für die Anzahl der Professoren und HiWis des ganzen Fachbereichs deklariert. Eine Ergebnisdatei *ergebnis* wird in Zeile vier initialisiert. In Zeile fünf wird der Variable *zahl* die erste Zeile von sem1.data zugewiesen. In der *while*-Schleife wird die Datei *sem1.data* Zeile für Zeile gelesen. Der *if*-Block addiert die Anzahl von Professoren bzw. HiWis innerhalb eines Semesters. Anschließend wird die Summe der Datei *ergebnis* hinzugefügt. Nach dem Ausführen des Skripts sieht die Datei *ergebnis* wie folgt aus:

```
Professor        HiWi
  7              112
  6               84
 11              204
 12              410
 21              328
 24              476
214              604
215              787
150              524
151              732
```

```
.                .
.                .
.                .
182              575
183              758
210              614
211              757
190              595
191              784
170              553
171              761
206              609
207              723
```

Jetzt können die Ergebnisse der beiden Modelle mit dem *R*-Tool verglichen werden. Zuerst werden diese in zwei Arrays, *hitmodell* und *procbmodell*, gespeichert und dann mit dem KS-Test verglichen.

```
> hitmodell<-read.table("bestcase.data",header=TRUE)
> procbmodell<-read.table("ergebnis",header=TRUE)

> ks.test(hitmodell$Professor,procbmodell1$Professor)
        Two-sample Kolmogorov-Smirnov test
data:  hitmodell$Professor and procbmodell$Professor
D = 0.076, p-value = 1.923e-05
alternative hypothesis: two.sided
Warning message:
cannot compute correct p-values with ties in:
   ks.test(hitmodell$Professor, procbmodell$Professor)

> ks.test(hitmodell$HiWi,procbmodell$HiWi)
        Two-sample Kolmogorov-Smirnov test
data:  hitmodell$HiWi and procbmodell$HiWi
D = 0.393, p-value < 2.2e-16
alternative hypothesis: two.sided
Warning message:
cannot compute correct p-values with ties in:
              ks.test(hitmodell$HiWi, procbmodell$HiWi)
```

Die oben aufgeführten Ergebnisse können wegen der ausgegebenen *Warnung* nicht akzeptiert bzw. nicht ausgewertet werden, so dass folgende graphische Hilfsmittel zur Hilfe genommen wurden. Als erstes wird ein Box-Plot benutzt, um die Grenzwerte (Extremwerte) und die Medianwerte von beiden Datenmengen zu vergleichen. Der Box-Plot für die Professor-SWS-Anzahl wird in der Abbildung B.1 dargestellt. Der Plot für die HiWi-SWS-Anzahl wird in der Abbildung B.2 gezeigt. Bei den *Professor-SWS* sind die Extremwerte für HIT 135 SWS und 244 SWS und bei ProC/B 130 SWS und 251 SWS. Der Medianwert ist bei beiden Modellen fast identisch und liegt bei HIT auf 188 SWS und bei ProC/B auf 190 SWS. Die Quartile der beiden Modelle sind auch fast identisch und liegen bei 163 SWS und 218 SWS. Bei den *HiWi-SWS* sind die Extremwerte für HIT 549 SWS und 1035 SWS und bei ProC/B

437 SWS und 1007 SWS. Die Medianwerte sind bei beiden Modellen unterschiedlich und liegen für HIT bei 782 SWS und für ProC/B bei 659,5 SWS. Die Quartile der beiden Modelle sind auch unterschiedlich und liegen beim HIT-Modell auf 714 SWS und 846 SWS und beim ProC/B-Modell auf 583 SWS und 759 SWS.

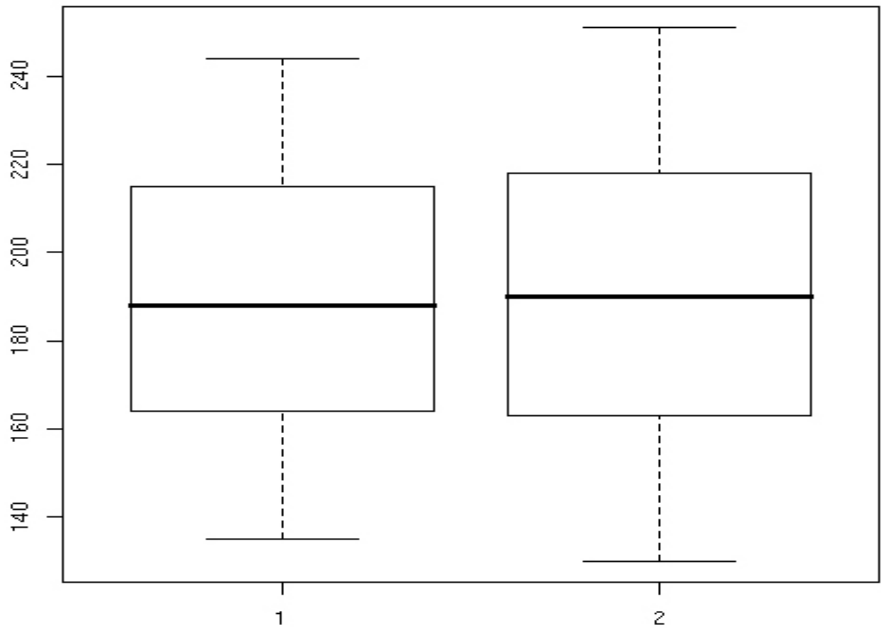


Abbildung B.1.: Boxplots Professor-SWS (1: HIT-Modell, 2: ProC/B-Modell)

Die Unterschiede der HiWi-SWS-Anzahl der beiden Modelle kommen dadurch zustande, dass bei dem HIT-Modell die HiWis auch für die Übungen der Schwerpunktgebiete mitberücksichtigt wurden, was entsprechend den Annahmen nicht modelliert werden sollte und beim ProC/B-Modell auch nicht getan wurde.
Somit können die Modelle bezüglich der HiWi-SWS-Anzahl nicht miteinander verglichen werden!
Die Medianwerte für die Anzahl von Professor-SWS bzw. HiWi-SWS wurden auch durch die Berechnungen im R-Tool bestätigt.

```
> median(hitmodell)
  Professor       HiWi
  188             782

> median(procbmodell)
  Professor       HiWi
  190             659.5
```

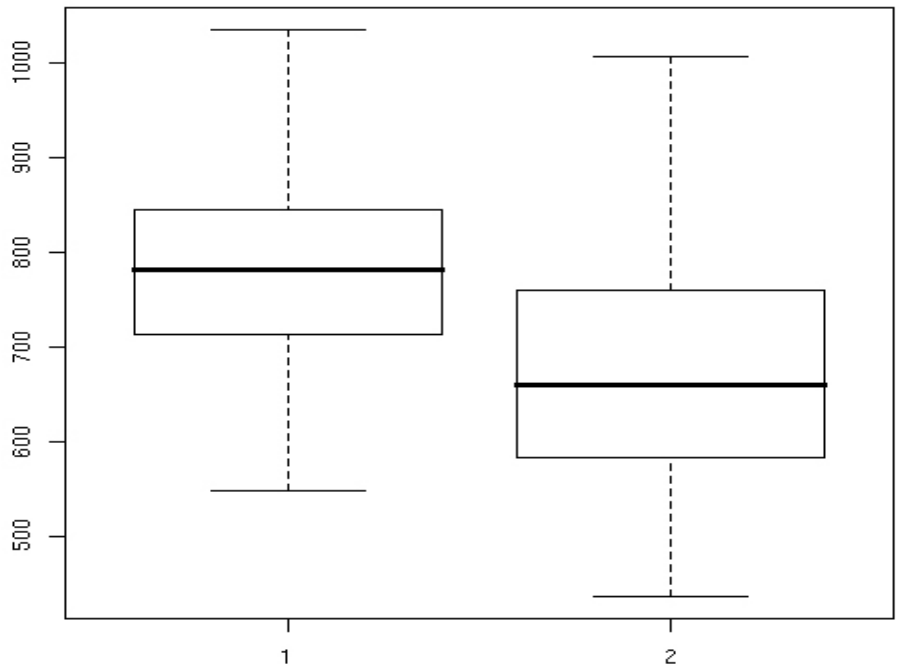


Abbildung B.2.: Boxplots HiWi-SWS (1: HIT-Modell, 2: ProC/B-Modell)

In den Abbildungen B.3 und B.4 werden die Histogramme für die Professor-SWS der beiden Modelle dargestellt. Beide Histogramme zeigen in etwa eine Gleichverteilung. Dies wird dadurch deutlich, dass die Balken eine fast gleiche Höhe aufweisen. Das wird auch in deren Verteilungsfunktion (B.5 und B.6) verdeutlicht.

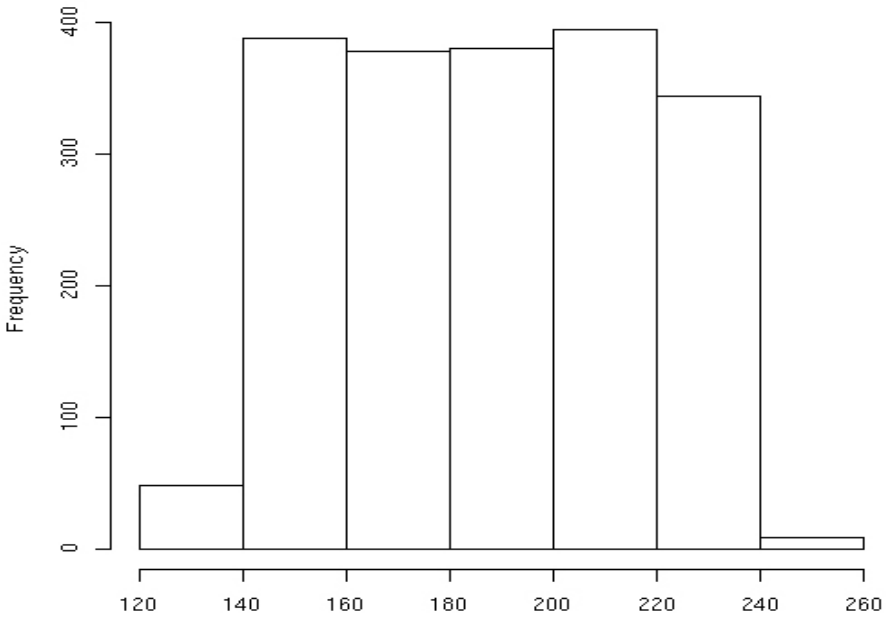


Abbildung B.3.: Histogramm für die Professor-SWS aus dem HIT-Modell

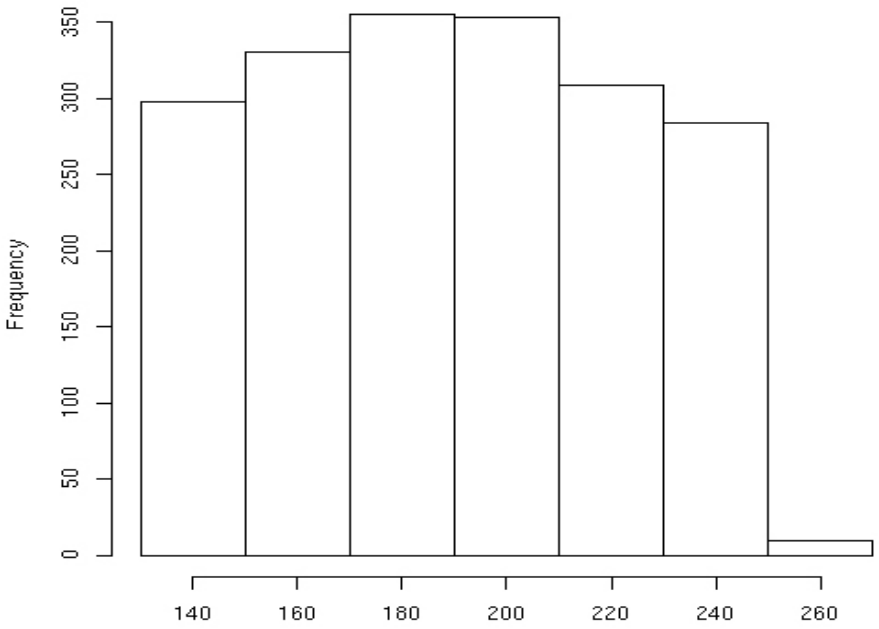


Abbildung B.4.: Histogramm für die Professor-SWS aus dem ProC/B-Modell

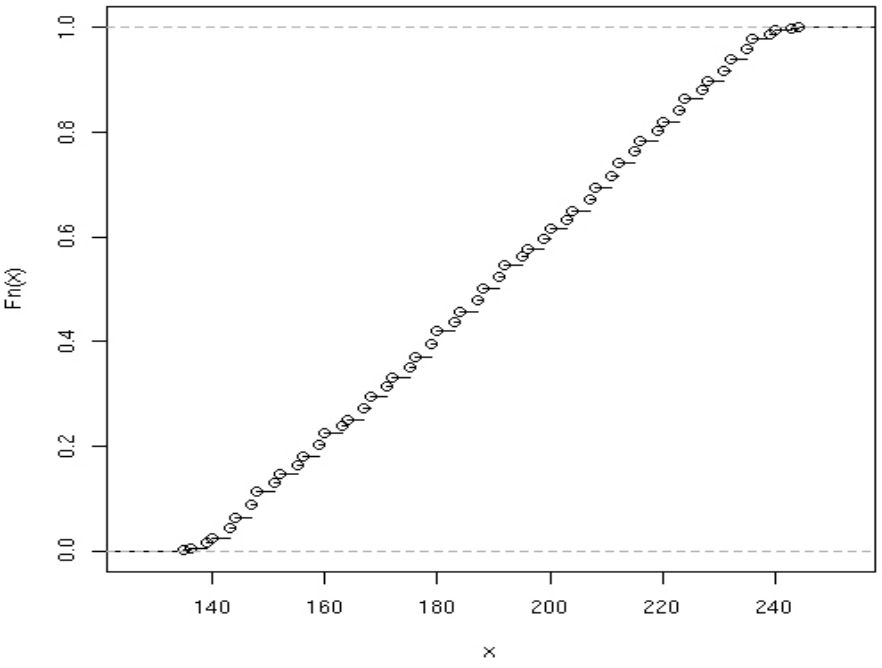


Abbildung B.5.: Verteilungsfunktion für die Professor-SWS aus dem HIT-Modell

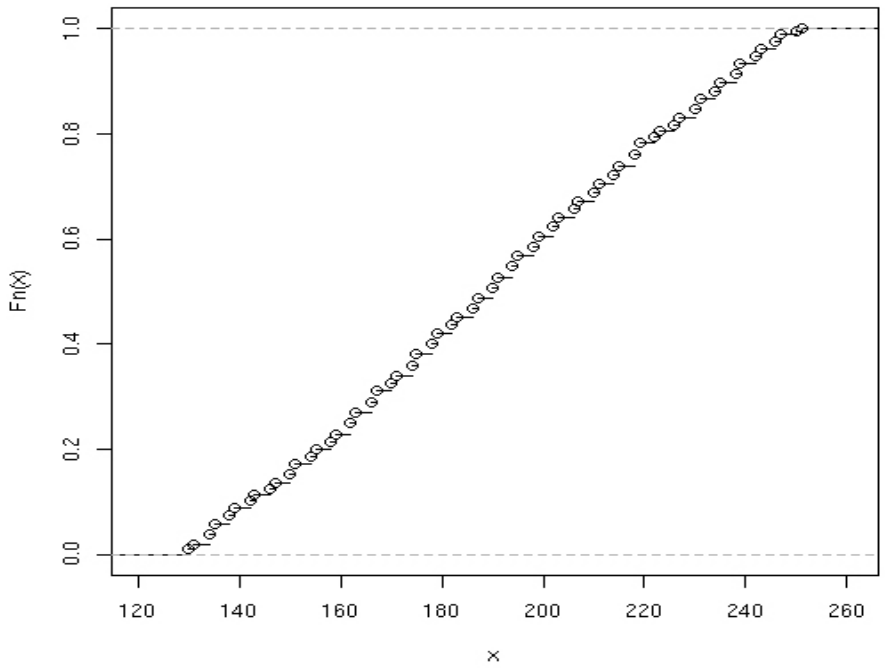


Abbildung B.6.: Verteilungsfunktion für die Professor-SWS aus dem ProC/B-Modell

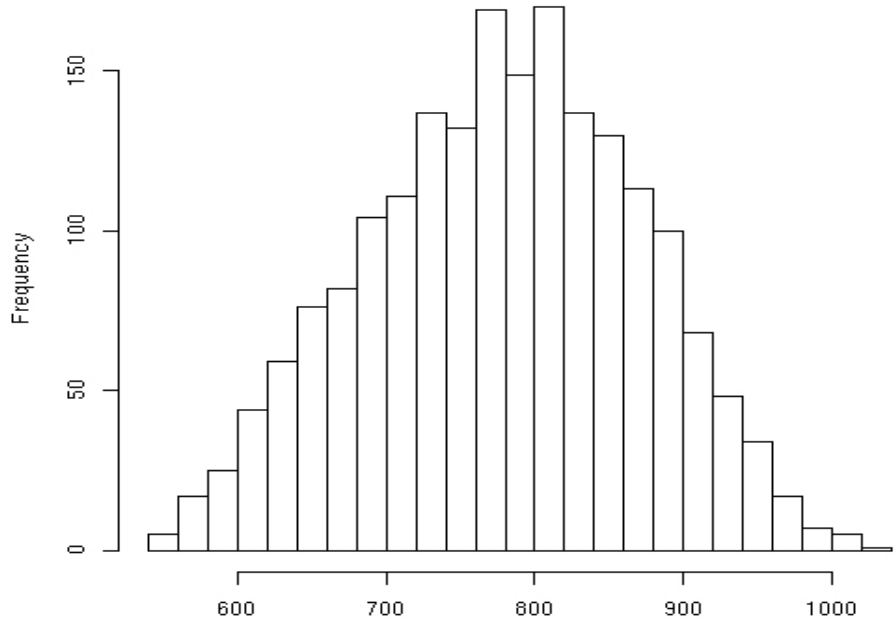


Abbildung B.7.: Histogramm für die HiWi-SWS aus dem HIT-Modell

In den Abbildungen B.7 und B.8 werden die Histogramme für die HiWi-SWS der beiden Modelle dargestellt. Dabei sieht man, dass sich beim HIT-Modell die Anzahl der HiWi-SWS um ungefähr 110 HiWi-SWS verschoben haben. Die Begründung dafür ist der Unterschied bei der HiWi-SWS Berechnung, nähmlich, dass bei dem HIT-Modell, im Vergleich zu dem ProC/B-Modell, die HiWis auch für

die Übungen der Schwerpunktgebiete miteinbezogen wurden, was laut den Annahmen nicht gemacht werden sollte. Die dazu gehörenden Verteilungsfunktionen sind in der Abbildung B.9 dargestellt. Auch sie sind ungefähr um die gleiche Anzahl an HiWi-SWS verschoben.

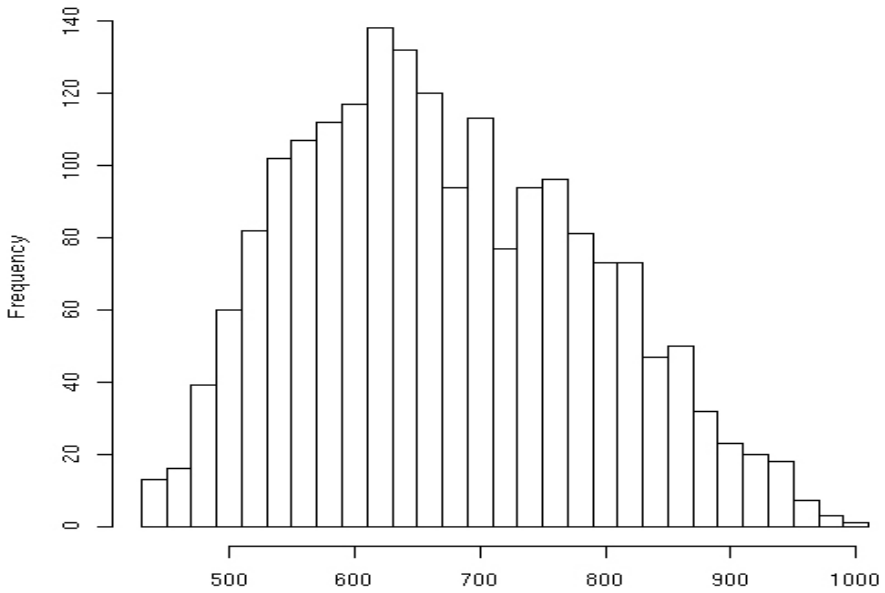


Abbildung B.8.: Histogramm für die HiWi-SWS aus dem ProC/B-Modell

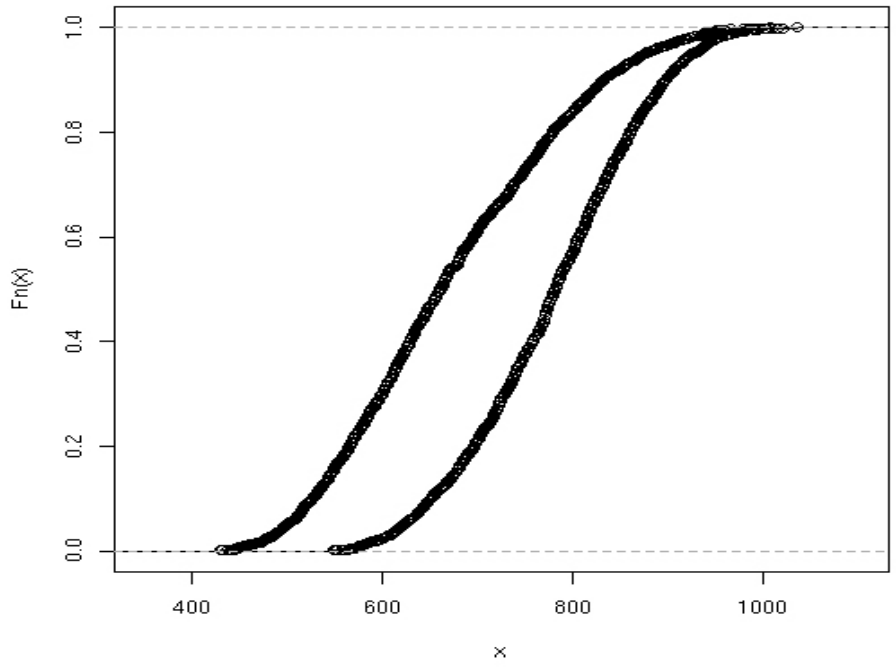


Abbildung B.9.: Verteilungsfunktionen für die HiWi-SWS aus dem HIT-Modell(rechts) und ProC/B-Modell(links)

In Rahmen der durchgeführten Validierung der beiden Modelle wurde festgestellt, dass beide Modelle an allen Punkten, bis auf einen, übereinstimmen. Dieser eine Punkt, dass bei dem HIT-Modell die Hi-Wis auch für die Übungen der Schwerpunktgebiete mitberücksichtigt wurden, was entsprechend den Annahmen nicht modelliert werden sollte, wurde im HIT-Modell entdeckt und wurde aus Zeitgründen nicht mehr behoben.

C. Vorschlag zu Abschlussberichten

In [HDG 79] hat eine hochschuldidaktische Gesprächsrunde folgende Vorschläge für Abschlussberichte von Projektgruppen erarbeitet:

4.2. Bemerkungen zum Abschlussbericht von Projektgruppen

Am Ende jeder Projektgruppe soll als abschließendes Dokument ein Abschlußbericht erstellt werden. Dabei sollte der Abschlußbericht im wesentlichen aus zwei Teilen bestehen, nämlich einem Sachbericht, der den sachlichen Inhalt des Projektes (Dokumentation) beschreibt und einem Erfahrungsbericht, der den Verlauf des Projektes aus didaktischer Sicht beschreibt und Anregungen zur Verbesserung der PG-Arbeit gibt.
Während der Sachbericht von den PG-Mitgliedern anzufertigen ist, sollte der Erfahrungsbericht von der PG-Leitung in Zusammenarbeit mit den PG-Mitgliedern entstehen.

Im folgenden sind einige Punkte angegeben, die in einem Erfahrungsbericht angesprochen werden sollten:

a) Vorbereitung der Projektgruppe
   - Wer hat sie initiiert?
   - Warum wurden Thema, Titel, Aufgaben so gewählt?
   - Welche fachlichen/außerfachlichen Lernziele wurden angestrebt?
   - Stimmen Ziel und Erwartungen überein? Welche?
     -- Wurden Ziele modifiziert?
        --- welche, von wem, warum?
   - Beschaffung von Literatur
     -- von Projektleitung alles vorgegeben und ausgesucht
     -- von Projektmitgliedern selbständig beschafft.

b) Gruppenmitglieder
   - Vorkenntnisse der Bewerber
   - Anzahl der Bewerber
   - Auswahlverfahren
   - Schwund (mit Begründung)
   - Wieviel haben durchgehalten?
   - Konflikte in der Gruppe

c) Umgebung, Sachmittel
- Projektgruppenraum
  -- Mobiliar
- Material (Kreide, Overhead-Projektor etc.)
- Spez. beantragte Geräte
  -- Kosten (Finanzierung)
  -- termingerechte Lieferung
- Schwierigkeiten mit technischer Ausstattung
  -- Rechnerbenutzung, Rechnerbetrieb

d) Verlauf der Projektgruppe
- Allgemeine Organisationsform
  -- hierarchisch, ständige Untergruppen
  -- Selbstverwaltung
  -- Plenum
  -- Protokolle
  -- externe Sitzungen
  -- Exkursionen
- Rolle des Projektgruppenleiters, Einfluß selbststeuernde Gruppe
- Betreuung durch Hochschuldidaktisches Zentrum
- Kontakt nach außen
- Zeitplan eingehalten? Warum nicht?
- Ferienarbeit
- Zwischenbericht, Abschlußbericht (rechtzeitig fertig?)
- Arbeitsaufwand der Mitglieder
- Mittelverbrauch (CPU-Zeit ...)

e) Auswertung
- Praxisrelevanz des Themas und des Ergebnisses
- Einfluß nach außen
- Inhaltliche neue Probleme
- Werden einzelne Teilnehmer in Form von Diplomarbeiten an dem Thema weiterarbeiten?
  -- oder an einem ähnlichen Thema?
- Vorschläge für ähnliche bzw. andere Projektgruppen
- Vorschläge für Nachfolge-PGs

Literaturverzeichnis

[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

[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

[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, April 2006

[Heid 86] HEIDELBERGER, P.: *Statistical analysis of parallel simulations*. http://delivery.acm.org/10.1145/320000/318448/p290-heidelberger.pdf?key1=318448&key2=7426331511&coll=GUIDE&dl=portal,ACM&CFID=11111111&CFTOKEN=2222222, Mai 2006

[HDG 79] Hochschuldidaktische Gesprächsrunde: DECKER, H.; MÄTER, J.; AN DE MEULEN, H.; MÜLLER, B.; SIEGMUND, N.; WANKMÜLLER, F.: *Das Projektstudium an der Abteilung Informatik - Materialien.* Forschungsbericht Nr. 84, Abteilung Informatik, Universität Dortmund, 1979

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

[Schu 02] SCHULZ, R.: *Parallele und Verteilte Simulation bei der Steuerung komplexer Produktionssysteme*. http://www.bibliothek.tuilmenau.de/elektr_medien/dissertationen/2003/Schulz_Roland/v47-wirqp542s389.pdf, Mai 2006

[Tepp 03] TEPPER, C.: *Anwendung simulativer Aggregierung bei der Analyse eines Verkehrszentrums*. SFB 559, Technical Report 03018, Universität Dortmund, Informatik IV, 2003

[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

[Ward 96] WARDA, A.: *Konzepte zur parallelen und verteilten Simulation von HIT-Modellen*. http://armin-warda.de/diplomarbeit.pdf, Mai 2006

[ZB 06] *Bottleneckanalyse des Informatikstudiums der Universität Dortmund*. PG476_Zwischenbericht, Universität Dortmund, Informatik, 2006