Projektgruppe 438

Entwicklung eines Performance Moduls

zur leistungsorientierten Software Entwicklung

Endbericht

SS 2004

Nadine Bramsiepe, Chang Gao, Esmeray Gencaslan, Jiuxin Huang, Thomas Kopinski, Oliver Lazar
Jan Papenfuß, Thomas Pauli, Marek Schekel, Sebastian Schürmann, Michael Stange, Reimar Twelker


HUM

Inhaltsverzeichnis

Änderungen der letzten Revisionen:

../implementierung/HUMLDatastructure/HUMLWorkload .
../implementierung/HUMLDatastructure/HUMLWorkload/CVS .
../implementierung/HUMLDatastructure/HUMLWorkload/CVS/Root .
../implementierung/HUMLDatastructure/HUMLWorkload/CVS/Repository .
../implementierung/HUMLDatastructure/HUMLWorkload/CVS/Entries .
../implementierung/HUMLDumpfileTranslator .
../implementierung/HUMLDumpfileTranslator/CVS .
../implementierung/HUMLDumpfileTranslator/CVS/Root .
../implementierung/HUMLDumpfileTranslator/CVS/Repository .
../implementierung/HUMLDumpfileTranslator/CVS/Entries .
../implementierung/HUMLGUI .
../implementierung/HUMLGUI/CVS .
../implementierung/HUMLGUI/CVS/Root .
../implementierung/HUMLGUI/CVS/Repository .
../implementierung/HUMLGUI/CVS/Entries .
../implementierung/HUMLHISLANG .
../implementierung/HUMLHISLANG/CVS .
../implementierung/HUMLHISLANG/CVS/Root .
../implementierung/HUMLHISLANG/CVS/Repository .
../implementierung/HUMLHISLANG/CVS/Entries .
../implementierung/HUMLXMITranslator .
../implementierung/HUMLXMITranslator/CVS .
../implementierung/HUMLXMITranslator/CVS/Root .
../implementierung/HUMLXMITranslator/CVS/Repository .
../implementierung/HUMLXMITranslator/CVS/Entries .
../implementierung/NanoXML .
../implementierung/NanoXML/CVS .
../implementierung/NanoXML/CVS/Root .
../implementierung/NanoXML/CVS/Repository .
../implementierung/NanoXML/CVS/Entries .
../implementierung/antlr .
../implementierung/antlr/CVS .
../implementierung/antlr/CVS/Root .
../implementierung/antlr/CVS/Repository .
../implementierung/antlr/CVS/Entries .
../klassendiagramm .
../klassendiagramm/CVS .
../klassendiagramm/CVS/Root .
../klassendiagramm/CVS/Repository .

0. Einführung

Das erste Kapitel dient der Vorstellung der Teilnehmer der Projektgruppe 438 (PG438) sowie des Themas. Es wird kurz auf das Ziel der Gruppenarbeit, die Aufgabenteilung innerhalb der Gruppe und auf einen Zeitplan, der die Aufgabenplanung der Gruppe im zweiten Semester darstellt, eingegangen.

0.1. Thema der Projektgruppe

Das Thema ist aus dem Bereich des *Software Performance Engineering* und verlangt konkret die Entwicklung eines Performanz-Moduls zur leistungsorientierten Software-Entwicklung. Das zugrunde liegende Problem ist der Umstand, dass Software-Entwicklung in der Vergangenheit zwar zunehmend unter Zuhilfenahme der Modellierungssprache UML erfolgt ist, der wichtige Performanzaspekt in diesem Verfahren allerdings ignoriert wurde. Leistungsbewertung erfolgte also bisher erst nach Abschluss der Software-Entwicklungsarbeit, d.h. leistungsbezogene Unzulänglichkeiten der entwickelten Software wurden erst im Einsatz erkannt. Eine Korrektur dieser Missstände ist zu diesem späten Zeitpunkt aber bereits sehr kostenintersiv, da eine Optimierung des Entwurfs erforderlich ist, der allerdings bereits implementiert worden ist.

Ein Grund für die Ignoranz des Performanzaspekts während der Softwareentwicklung ist sicherlich die mangelnde Unterstützung durch Entwicklungswerkzeuge und der Zeitdruck bei der Entwicklung. Die Aufgabe der PG438 besteht darin, die Modellierung in UML und die Performanzanalyse *während* der Software-Entwicklung durch ein geeignetes Werkzeug in Verbindung zu bringen. Die Gruppe setzte es sich zum Ziel, UML-Modelle um Performanzaspekte zu erweitern und ein Werkzeug zu entwickeln, das diese erweiterten UML-Modelle in eine Eingabe für das Analysewerkzeug HIT [Büttner u. a.] übersetzt. Letzteres wurde vom Lehrstuhl 4 des Fachbereichs Informatik der Universität Dortmund entwickelt.

Im ersten Semester der Gruppenarbeit wurde allgemeines Grundwissen zum Thema geschaffen, UML-Modelle auf ihre Eignung für die Performanzerweiterung geprüft sowie eine Anforderungsdefinition und ein Pflichtenheft entwickelt. Abschließend wurde eine erste Fassung des Klassendiagramms ausgearbeitet.

Zu Beginn des zweiten Semesters der Projektarbeit waren zwei Aufgabenschwerpunkte zu bearbeiten. Die Gruppenaufteilung bzgl. dieser Schwerpunkte wurde aus dem ersten Semester übernommen, aber auf zwei Gruppen reduziert:

- Die *Paradigmagruppe* hatte die Aufgabe, eine Vorgehensweise zu entwickeln, welche bei der Verarbeitung bzw. Erweiterung der für die Leistungsbewertung von der PG ausgewählten UML-Diagrammtypen (Aktivitätsdiagramm und Verteilungsdiagramm) sowie bei der Übersetzung dieser Diagramme in die HIT-Eingabesprache HI-SLANG verwendet wurde und diese in einem Bericht zu beschreiben (Kapitel 1).

- Die *Compilergruppe* hatte die Aufgabe, Compilergeneratoren und XML-Parser zu sichten, eine Auswahl zu treffen, sich in die jeweiligen Werkzeuge einzuarbeiten und ihre Verwendung in Bezug auf das Projekt zu planen.

Beide Gruppen hatten die Aufgabe, die für ihren Aufgabenbereich benötigten Klassendiagramme zu erstellen und zu dokumentieren. Nach Abschluss des Entwurfs nahmen alle Gruppenteilnehmer an der Implementierung der einzelnen Pakete des Werkzeugs HUML teil.

0.2. Teilnehmer und Betreuer

Die PG438 wurde durch den Lehrstuhl 4 der Universität Dortmund betreut und setzt sich aus folgenden Mitgliedern zusammen:

- Jürgen Mäter (PG-Betreuer)
- Nadine Bramsiepe
- Chang Gao
- Esmeray Gencaslan
- Jiuxin Huang
- Thomas Kopinski
- Oliver Lazar
- Jan Papenfuß
- Thomas Pauli
- Marek Schekel
- Sebastian Schürmann
- Michael Stange
- Reimar Twelker

0.3. Arbeitseinteilung

Dieser Abschnitt beschreibt die Gruppenaufteilung während der Projektarbeit.

- Die *Paradigmagruppe* entwickelte das HUML-Paradigma, das in Kapitel 1 beschrieben wird. Die Aufgabe bestand darin, das UML-RT [OMG (2003)] bezüglich bereits in Entwicklung befindlicher Performanzerweiterungen der Sprache UML zu analysieren, ggf. darüberhinaus erforderliche Erweiterungen zu entwickeln und die Verbindung zwischen UML-Modellen und HIT-Eingabe in HI-SLANG herzustellen. Es war zu bestimmen, in welcher Weise die ausgezeichneten UML-Diagramme um durch das UML-RT vorgeschlagene Performanz-Werte zu erweitern sind. Weiterhin war es Aufgabe festzustellen, welche weiteren Informationen (durch eigenentwickelte UML-Erweiterungen) bereitzustellen sind, um eine vollständige und gültige Eingabe für das Werkzeug HIT erzeugen zu können. Ein weiterer Aufgabenteil bestand darin, die konkrete Übersetzung der einzelnen UML-Diagramm-Bestandteile detailliert zu planen bzw. vorzuschreiben. Die Ergebnisse der *Paradigmagruppe* sind in Kapitel 1 zu finden.

  Die *Paradigmagruppe* setzte sich aus folgenden Mitgliedern zusammen:

  - Chang Gao
  - Esmeray Gencaslan
  - Jiuxin Huang
  - Oliver Lazar
  - Jan Papenfuß
  - Marek Schekel
  - Sebastian Schürmann
  - Michael Stange

- Die *Compilergruppe* sichtete zunächst das Angebot an XML-Parsern und Compilergeneratoren. Anschließend erfolgte die Auswahl eines XML-Parsers (NanoXML) und eines Compilergenerators (ANTLR), eine Einarbeitung in die Werkzeuge und der Einsatz des Parsers und des Compilergenerators. Die Aufgabe bestand darin, eine XML-Quelle (exportiertes UML-Modell) unter Zuhilfenahme des gewählten XML-Parsers in eine HUML-interne Datenstruktur zu übersetzen. Die Entwicklung dieser Datenstruktur erfolgte in Zusammenarbeit mit der *Paradigmagruppe*. Die interne Repräsentation der XML-Quelle war unter Einsatz des gewählten Compilergenerators in eine HI-SLANG-Eingabe für HIT zu übersetzen. Darüberhinaus war ein Weg zu finden, die Messergebnisse von HIT in das analysierte UML-Modell einzufügen. Auch in diesem Fall kam der Compilergenerator zum Einsatz.

  Die Compilergruppe setzte sich aus folgenden Mitgliedern zusammen:

  - Nadine Bramsiepe
  - Thomas Kopinski
  - Thomas Pauli
  - Reimar Twelker

Neben diesen Tätigkeiten wurde das im ersten Semester angefertigte Klassendiagramm umfangreich überarbeitet, erweitert und dokumentiert.

Gegen Ende des zweiten PG-Semesters widmeten sich alle Gruppenteilnehmer der Implementierung der Pakete des Werkzeugs HUML. Dabei unterteilten sich die Aufgaben in folgende Schwerpunktgebiete:

- Die grafische Benutzungsoberfläche
- Die Steuerung des Programmablaufs durch *Control*-Klassen
- Die Übersetzung des exportierten UML-Modells in eine interne Darstellung
- Die Übersetzung der internen Repräsentation in eine HI-SLANG Eingabe
- Die Rücktransformation der Messergebnisse von HIT in das UML-Modell

Weitere Tätigkeiten gegen Ende der PG-Arbeit waren die Erarbeitung eines Handbuchs sowie der Produkttest.

0.4. Zeitplan

Im Folgenden wird der Zeitplan der PG438 vorgestellt.

0.4.1. Wintersemester 2003/2004

Der Verlauf des ersten PG-Semesters ist detailliert im Zwischenbericht der PG438 beschrieben [PG438 (2004)].

0.4.2. Sommersemester 2004

Der grobe Zeitplan für das Sommersemester 2004, der am Ende des Wintersemesters erstellt wurde, ist in Tabelle 0.1 zu sehen.

Nr.	**Vorgangsname**	**Dauer**	**Anfang**	**Ende**
1	Entwurf/ Softwaredesign	25 Tage	Mo 19.04.04	Fr 21.05.04
1a	HUML-Paradigma	14 Tage	Mo 19.04.04	Do 06.05.04
1b	Klassendiagramm	12 Tage	Do 06.05.04	Fr 21.05.04
2	Implementierung	25 Tage	Fr 21.05.04	Do 24.06.04
3	Handbuch	11 Tage	Do 10.06.04	Do 24.06.04
4	Produkttest	11 Tage	Do 24.06.04	Do 15.06.05
5	Endbericht	64 Tage	Mo 19.04.04	Do 15.06.04
6	Fehlerkorrektur	12 Tage	Do.15.06.04	Fr 10.07.04

Tabelle 0.1.: Der Zeitplan für das Sommersemester 2004

0.5. Danksagung

Die PG438 möchte sich an dieser Stelle bei der Firma Gentleware bedanken, die jedem Gruppenteilnehmer freundlicherweise eine Lizenz für 'Poseidon for UML' für die Dauer der Gruppenarbeit zur Verfügung gestellt hat. Darüberhinaus hat sich Gentleware sehr für Anmerkungen und Anregungen unsererseits interessiert gezeigt. Das Werkzeug 'Poseidon for UML' erfüllt die im Rahmen der Projektgruppe an ein UML-Modellierungswerkzeug gestellten Anforderungen im Gegensatz zu vergleichbaren Konkurrenzprodukten, so dass wir 'Poseidon for UML' für die Arbeit mit HUML und HIT sehr empfehlen.

1. Das HUML-Paradigma

1.1. Motivation und Einleitung

Die Modellierung von Software mittels UML [OMG (2001)] ist ein wichtiger Teil des Softwareentwicklungsprozesses. Der Softwareentwickler verwendet dazu UML-Modellierungs-Werkzeuge wie *Poseidon for UML* [Gentleware] oder *Together* [Borland]. Ziel der Modellierung ist es, ein umfassendes Verständnis der benötigten Architektur der Software für die spätere Implementierung zu erlangen. Die UML-Diagramme liefern dabei graphische Darstellungen, die das objektorientierte Zusammenspiel der Klassen, Objekte und Methoden verdeutlichen. Ein wesentlicher Faktor ist bei der UML-Modellierung bisher unberücksichtigt: die Performanz der zu entwickelnden Software.

Der Entwickler soll im Stadium der UML-Modellierung die Möglichkeit bekommen, schon an dieser Stelle sein späteres Softwareprodukt auf Leistung hin optimieren bzw. mögliche Engpässe erkennen zu können. Im ersten Schritt müssen dazu geeignete UML-Diagramme gewählt werden. In einem zweiten Schritt werden diese Diagrammtypen um Performanz-Annotationen erweitert. Die Grundlage dafür stellt das *UML Profile for Schedulability, Performance and Time* (UML-RT) [OMG (2003)] der *Object Management Group* (OMG) dar, das im Anhang B zusammenfassend vorgestellt wird. Als dritter Schritt kann danach die eigentliche Leistungsbewertung mittels HIT [Büttner u. a.] durchgeführt werden.

Für die Leistungsbewertung von Rechensystemen spielt das Last-Maschine-Paradigma eine große Rolle. Wir bezeichnen die Aufgaben, die das Rechensystem bearbeitet allgemein als Last. Beim *Software Performance Engineering* (SPE) können wir Software als Last verstehen und die Hardware wird als Maschine interpretiert. Auf diesem Paradigma basiert auch das eingesetzte Performanz-Analyse-Werkzeug HIT.

HIT ermöglicht eine Leistungsanalyse von Modellen, die in der Sprache HI-SLANG beschrieben sind. Der Benutzer spezifiziert dabei die Experimente und wählt eine Analysetechnik aus. Die Analyse selbst wird von HIT durchgeführt. In der Modellwelt von HIT werden Systeme hierarchisch betrachtet. Auf jeder Ebene der Hierarchie, außer der obersten, werden Dienste (*services*) angeboten, die von darüber liegenden Schichten genutzt werden können. Wir unterscheiden in jeder Schicht Last und Maschine. Das System wird mit Anforderungen, die von angebotenen Diensten der Maschine erfüllt werden, belastet. Die Last wird in HIT durch in Anspruch genommene *services* erzeugt. Dabei werden die Maschinen, die diese *services* bereitstellen, belastet. Soll eine Leistungsbewertung mit HIT durchgeführt werden, muss ausgewählt werden, mit welchem Lösungsverfahren die Bewertung durchgeführt werden soll. Dabei unterscheidet man zwischen drei Lösungsverfahren: den analytisch-algebraischen, den analytisch-numerischen und den simulativen.

Zur einfachen Benutzung von HIT ist die grafische Oberfläche HITGRAPHIC [Szczittnick u. a.] vorhanden. Hierbei können die Dienste, Maschinen und ihre Beziehungen grafisch in Verbindung gesetzt werden. Um eine Verbindung von Dienst und Maschine in HITGRAPHIC herzustellen, müssen der angebotene *provided service* und der *used service* mittels eines Knotens verbunden werden.

Das Kapitel HUML-Paradigma gliedert sich in die folgenden Unterkapitel.

In 1.2 wird Leistungsbewertung im Kontext von Software eingeführt.

In 1.3 wird der Vorgang der Verarbeitung herkömmlicher UML-Diagramme durch Erweiterung und Übersetzung mit dem Ziel der Erzeugung einer Eingabe für das Leistungsbewertungswerkzeug HIT abstrakt beschrieben.

Im Kapitel 1.4 wird ein kurzer Einblick in die Modellierungssprache UML gegeben, der sich auf Anwendungsfalldiagramme (*Use Case Diagram*), Aktivitätsdiagramme (*Activity Diagram*) und Verteilungsdiagramme (*Deployment Diagram*) beschränkt.

Kapitel 1.5 befasst sich mit der Erweiterung dieser Teilmenge von Diagrammtypen um Performanzparameter unter Verwendung des UML-RT und der HUML-Performanz-Erweiterungen (siehe auch 1.3).

Im Kapitel 1.6 werden die Übersetzungen der erweiterten UML-Diagramme in eine Eingabe für das Performanz-Analyse-Werkzeug HIT beschrieben. In den Unterkapiteln 1.6.1 bis 1.6.4 wird erläutert, wie die im Rahmen des HUML-Paradigmas verwendeten Teile des UML-RT und die Erweiterungen der PG438 durch die Sprache HI-SLANG dargestellt werden können. Unterkapitel 1.6.5 gibt eine globale Sichtweise der Übersetzung von Topologien aus UML-Diagrammen nach HI-SLANG. Dafür wird die Übersetzung einfacher UML-Diagramme mit zunächst möglichst wenigen Erweiterungen betrachtet. Kapitel 1.7 fasst die von HUML verarbeiteten performanzbezogenen Eigenschaftswerte aus dem UML-RT sowie die von HUML neu hinzugefügten Eigenschaftswerte tabellarisch zusammen.

Um die abstrakte Sicht der im Rahmen des HUML-Paradigmas angewandten Vorgehensweise besser verdeutlichen zu können, wird in Kapitel 1.8 ein Anwendungsbeispiel beschrieben.

1.2. Leistungsbewertung von Software

> *To measure is to know.* (Clerk Maxwell)
>
> *You cannot control what you cannot measure* (Tom DeMarco)

In diesem Kapitel wird eine kurze Übersicht über die Softwareleistungsbewertung im Allgemeinen gegeben und die Softwareleistungsbewertung mit HUML im Besonderen erläutert.

1.2.1. Ziele der Softwareleistungsbewertung

Die Anforderungen an ein Softwareprodukt können mannigfaltig sein, aus ganz unterschiedlichen Gebieten der menschlichen Interessen stammen und somit wissenschaftlicher, technischer, ökonomischer, sozialer, politischer, medizinischer, didaktischer etc. Natur sein, m.a.W.: es sind in allen Bereichen, in denen Software zum Einsatz kommt, Anforderungen an sie gestellt.

Zu den wichtigsten Anforderungen an Software gehören im Allgemeinen Korrektheit in der Ausführung, Effizienz, Bedienbarkeit, Wartbarkeit, Dokumentation und Erlernbarkeit, Kompatibilität, Erweiterbarkeit, Interoperabilität, Sicherheit, Preis-Leistungs-Verhältnis. Die Liste ließe sich sicherlich noch beträchtlich erweitern. Allen Anforderungen gemeinsam ist jedoch das Wesen der Forderung selbst: eine Anforderung an ein Produkt drückt den Wunsch aus, dass es das *leistet*, was man von ihm erwartet.

Die Erfahrung zeigt: Reale Software entspricht mitunter den an sie gestellten Anforderungen nicht. Manchmal erfüllt sie sie ganz, oft dagegen nur teilweise oder sogar gar nicht. Folglich

gibt es Gradabstufungen von Leistung. Diese können qualitativer oder quantitativer Natur sein.

Softwareleistungsbewertung ist eine relativ neue Disziplin, die ihren Ursprung in dem Wunsch hat, mit einem auf Software bezogenen Begriff der Leistung wissenschaftlich umzugehen. Ihr Ziel ist die Entwicklung allgemeiner Verfahren zur Verbesserung von Software im Sinne der Steigerung ihrer Leistung, d.h. dass sie mehr Ansprüchen bzw. höheren Anforderungen genügt. Die Methode dieser Disziplin ist, verschiedene Begriffe von Softwareleistung zuerst einmal überhaupt zu definieren und darauf aufbauend Verfahren zu entwickeln, die es ermöglichen, je nach Art des zugrundegelegten Leistungsbegriffs qualitative oder quantitative Aussagen über konkrete Softwareprodukte zu machen. Diese Aussagen ermöglichen dann absolute Bewertungen oder können auch als Vergleichsgrößen dienen. Im Rahmen des HUML-Projekts spielen quantitative Leistungsmaße die zentrale Rolle.

Ein erster Ansatzpunkt bei der Softwareleistungsbewertung ist das Verständnis des Gegenstandsbereichs der Betrachtung, d.h. der Software selbst. Die Eigenarten von Softwareprozessen und andere Produktcharakteristika sind auf eine Weise in Modellvorstellungen zu fassen, die es erlaubt, Leistungsmaße zu definieren. Die Kapitel 1.4 und 1.5 sind diesem Thema ausführlich gewidmet.

1.2.2. Softwaremessstrategien

Für die Softwareleistungsbewertung im Allgemeinen existiert schon eine ausgearbeitete und durch das IEEE standardisierte Strategie. Sie ist ein Metaverfahren, das in einer Vorgehensweise zum Entwurf eines wissenschaftlichen Messverfahrens besteht. Abbildung 1.1 zeigt das Schema dieses Standards (Quelle: ieeesoftware (1993)):

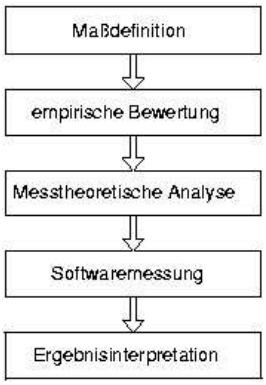

Abbildung 1.1.: Vorgehensweise der Softwareleistungsbewertung

Zu den gängigen konkreten Softwaremess- und -bewertungsstrategien im Sinne des Metaverfahrens zählen auch Verfahren, die keine eigentliche Messung bzw. messtheoretische Analyse beinhalten, aber dennoch zum Gebiet der Softwareleistungsbewertung gehören. Dazu gehören die Evaluierung, die Merkmalsabschätzung, die modellbezogene Softwareleistungsbewertung und die direkte Softwareleistungsbewertung. Diese vier Softwaremessstrategien werden im Folgenden kurz erläutert.

Die Evaluierung Die Evaluierung basiert auf stichprobenhaften Erhebungen von subjektiven Beurteilungen seitens geeignet ausgewählter Personengruppen, z.B. unter den Anwendern des untersuchten Softwareprodukts. Üblich ist die Verwendung eines Fragebogens, in dem softwareproduktbezogene Fragen gestellt sind. Diese werden von der Zielgruppe mit *Ja* oder *Nein* oder mit Vergabe von Punkten oder Noten ausgefüllt. Die Auswertung erfolgt mittels statistischer Methoden.

Die durch Evaluierung ermittelten Leistungswerte sind meistens qualitativer Natur. Sie werden jedoch durch die statistischen Methoden quantifiziert und können so zur Grundlage gradueller Aussagen über Produkteigenschaften werden.

Die Merkmalsabschätzung Die Merkmalsabschätzung erfolgt durch Bewertung in Form einer vorgegebenen Formel bzw. einer allgemeinen Berechnungsvorschrift für ein spezielles Merkmal.

Ein Beispiel für eine Berechnungsvorschrift für ein spezielles Merkmal ist die folgende Formel für die Berechnung der Produktivität:

$$Produktivitaet = \frac{Produktumfang}{Aufwand \cdot Zeit}$$

Merkmalsabschätzung ist häufig nur *a posteriori* einsetzbar, da Informationen wie z.B. die Produktionsdauer oder die Anzahl der Quellcodezeilen zur Auswertung einer Formel erforderlich sind.

Die modellbezogene Softwareleistungsbewertung Die modellbezogene Softwareleistungsbewertung erfolgt indirekt über ein Modell oder über eine abgeleitete Zwischensprache. Als Modell können u.a. Hierarchiebäume, Datenflussdiagramme oder Petrinetze dienen.

Der entscheidende Vorteil dieser Art von Softwareleistungsbewertung ist, dass sie auch *a priori*, d.h. zur Entwicklungszeit durchgeführt werden kann, wenn das Softwareprodukt selbst noch gar nicht vorliegt. Weil Fehlentwicklungen hier im voraus vermieden werden können, kommt dieser Art eine besondere Bedeutung zu. In der Tat ist sie die Grundlage des HUML-Projekts.

Die direkte Softwareleistungsbewertung Bei der direkten Softwareleistungsbewertung werden Softwareeigenschaften direkt gemessen, wie z.B. die Codelänge, die Abarbeitungszeit oder der verwendete Speicherumfang. Sie kann ausschließlich am fertigen Produkt, allenfalls an fertigen Produktteilen, durchgeführt werden und hat ihre Bedeutung in der nachträglichen Fehlerkorrektur und im Erwerb von praktischer Erfahrung zur künftigen Fehlervermeidung bzw. Erarbeitung leistungsstarker Softwareelemente. Der Nutzen ist meistens eher praktischer als wissenschaftlicher Natur.

1.2.3. Softwareleistungsbewertung mit HUML und HIT

Das Werkzeug HUML ermöglicht zusammen mit dem Leistungsbewertungswerkzeug HIT die modellbezogene Leistungsbewertung von Software-Hardware-Systemen. Diese werden durch UML-Diagramme beschrieben. Die Hardwareseite dieser Modelle besteht in den Angaben bestimmter Charakteristika der Hardware, auf denen die Software läuft. Die Modelle der Software sind prozess-, kontrollfluss- und zustandsorientiert. Die Software-Hardware-Modelle, die vom

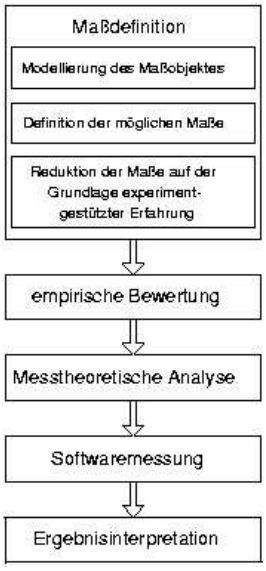

Abbildung 1.2.: Vorgehensweise bei der modellbezogenen Softwareleistungsbewertung

HUML-Werkzeug verarbeitet werden, sind keine reinen Softwaremodelle, die von jeder Hardware abstrahieren, weil die Kosten, die ein Prozess in einem Softwarezustand erzeugt, in der HUML-Modellwelt über Zeitverbrauch an Hardwareressourcen definiert ist.

Das zur Softwareleistungsbewertung durch HIT an dieses weitergegebene Modell ist eine HI-SLANG-Modellhierarchie, die vom HUML-Werkzeug aus dem UML-Modell erzeugt wird.

Die Leistungsgrößen, mit denen im Rahmen von HUML operiert wird, orientieren sich grundsätzlich an denen, die in HIT verwendet werden. Es sind hierbei Eingabe- und Ausgabegrößen zu unterscheiden. Bei den ersteren handelt es sich um eine Menge von Vorgaben, durch die Leistungsangaben für die Komponenten des Modells eines laufenden Software-Hardware-Systems gemacht werden, z.B. die Größe des Prozessaufkommens, die relative Arbeitsgeschwindigkeit von Hardwareservern oder der Zeitbedarf von bestimmten elementaren Softwareaktionen. Die Ausgabegrößen, d.h. die durch das HUML-Werkzeug unter Zuhilfenahme von HIT ermittelten Leistungsmaße, sind eine Teilmenge derjenigen, welche in der HIT-Modellwelt im Allgemeinen zur Verfügung stehen (vgl. Büttner u. a. Kapitel 5.1.2). Es handelt sich hierbei um die Population (*population*), den Durchsatz (*throughput*), die Antwortzeit (*turnaround time*), die Belegung (*occupation*) und die Auslastung (*utilization*). Welche Bedeutungen diese Maße im Einzelnen haben, ist im Kapitel 1.5.5 genau erläutert.

Die einzelnen Schritte der modellbezogenen Softwareleistungsbewertung (siehe Abbildung 1.2 und dazu ieeesoftware (1993)) lassen sich folgendermaßen auf die Softwareleistungsbewertung mit HUML übertragen:

1. **Maßdefinition:** Die Modellierung des Messobjekts erfolgt durch den Benutzer über die Erstellung eines UML-Modells des zu bewertenden Software-Hardware-Systems. HUML transformiert dieses Modell in eine HI-SLANG-Modellhierarchie, die durch HIT analysiert wird.

   Die Darstellung der Menge von Software- und Hardwareleistungsmaßen, durch die die

Charakteristika des Software-Hardware-Modells beschrieben werden, die für die Leistungsbewertung eine Rolle spielen, erfolgt im Kapitel 1.5.

Die Behandlung des Themas der Maßreduktion fand im Rahmen des HUML-Projekts nicht statt.

2. **Empirische Bewertung:** Die Leistungsbewertung im Rahmen des HUML-Projekts ist ausschließlich modellbezogen. Die Leistungswerte des Modells werden ausschließlich mittels mathematischer Methoden aus dem Bereich der Warteschlangenanalysis oder mit Hilfe von Simulation errechnet. Eine empirische Messung an realer Software findet nicht statt.

3. **Messtheoretische Analyse:** Eine messtheoretische Analyse stellt eine Kritik des Messverfahrens dar und dient normalerweise der Bestimmung von Mess(un-)genauigkeiten und besteht ferner in der Unterscheidung zwischen echten Abweichungen von Messwerten von theoretischen Vorhersagen und sogenannten systematischen Messfehlern. Da solche Analysen sich auf empirische Messungen beziehen, entfallen sie im Rahmen des HUML-Projekts.

   Eine vergleichbare Analyse in Bezug auf HUML-Modelle könnte in einer Abschätzung von Diskrepanzen zwischen Modellen und der Wirklichkeit überhaupt bestehen. Derartige modelltheoretische Untersuchungen setzen jedoch Grundlagenforschung voraus, die den Rahmen des HUML-Projekts gesprengt hätte.

   Ein Äquivalent zur Fehlerrechnung bei empirischen Verfahren besteht in der Modellanalyse durch HIT unter Anwendung des simulativen Lösungsverfahrens in der Berechnung von Standardabweichungen und in der Betrachtung von Konfidenzgraden und Konfidenzintervallen.

4. **Softwaremessung:** Die eigentliche Leistungsbewertung wird durch die Software HIT durchgeführt. HIT arbeitet mit HI-SLANG-Modellen als Eingabemenge, die vom HUML-Werkzeug aus UML-Modellen von Software-Hardware-Systemen erzeugt werden. Wie schon unter Punkt 2. gesagt, handelt es sich nicht eigentlich um eine Messung, sondern um eine Berechnung.

5. **Ergebnisinterpretation:** Die Interpretation der Ergebnisse der Leistungsmessung von HIT hat durch den Benutzer zu erfolgen. Es obliegt ihm, in den durch HIT errechneten Leistungsgrößen, die oben aufgezählt und in Kapitel 1.5.5 genauer erläutert werden, bereits als eigentliche Ergebnisse zu betrachten, oder vielmehr als Indikatoren dafür, ob ein System unter der Voraussetzung der Leistungsvorgaben, welche das System im vorhinein beschreiben, "wie gewünscht funktioniert" oder nicht. Eine sinnvolle Anwendung des HUML-Werkzeugs kann darin bestehen, Experimentserien unter ständigen Veränderungen der angegebenen Eingabewerte, d.h. der Systemvoraussetzungen, so lange durchzuführen, bis das System "funktioniert". In diesem Fall sind die zuletzt vorausgesetzten Größen das Ergebnis! Auf diese Weise wird eine Anforderung an ein Softwareprodukt, z.B. als Ganzes ein bestimmtes Prozessaufkommen "vertragen" zu können, auf eine Menge Leistungsanforderungen an kleinere Produkteinheiten (die Softwarezustände und die ihnen entsprechenden Programmstücke) zurückgeführt.

1.3. Vorgehensweise

Das HUML-Paradigma setzt auf UML-Aktivitätsdiagrammen und -Verteilungsdiagrammen zur Softwaresystem-Modellierung auf. UML-Diagramme sind von Grund auf deterministische Abfolgen von bestimmten Diagrammbestandteilen, die wiederum z.B. Softwarezustände oder Vorgänge modellieren. Im Falle von Verzweigungen, also bedingt ausgeführten Pfaden, müßten sämtliche Pfade analysiert werden. Wenn im deterministischen Sinne quantitativ präzise analysiert werden soll, müßte das Verhalten aller konkreten Anforderungen der Benutzer der zu bewertenden Software bekannt sein. Eine geschickte Möglichkeit über die Gesamtheit der Anforderungen doch verwertbare Aussagen zu treffen, bietet eine Beschreibung der Anforderungen als stochastischer Prozess, also mittels statistischer Kenngrößen wie Verteilungsfunktionen, Erwartungswerte und Varianzen. Dieses bildet die Basis zur Transformation der deterministischen Abläufe von UML nach UML-RT.

Der erste Schritt bei der Modellierung eines Systems in Hinblick auf eine spätere Leistungsbewertung durch HUML besteht darin, das Anwendungsfalldiagramm, das vom Entwickler erstellt wurde, durch ein Aktivitätsdiagramm zu ersetzen. Anwendungsfalldiagramme ignorieren die strukturellen Zusammenhänge zwischen Anwendungsfällen, die für die Leistungsbewertung allerdings von Relevanz sind. Deshalb wird bei HUML anstelle des Anwendungsfalldiagramms ein Aktivitätsdiagramm verwendet, das diese strukturellen Beziehungen ausdrückt. Dieses Aktivitätsdiagramm hat Wurzelfunktion bezüglich der Menge der übrigen Aktivitätsdiagramme, die innere Knoten und Blätter eines Aktivitätsdiagrammbaums sind. Das heißt, dass die Diagramme, die die Anwendungsfälle beschreiben, "Kinder" des neuen Wurzeldiagramms sind. Darüberhinaus können einzelne Aktionszustände durch weitere Aktivitätsdiagramme verfeinert werden. Die Beziehung von Aktionszustand und verfeinerndem Diagramm ist eine Eltern-Kind-Beziehung im Aktivitätsdiagrammbaum. Die Blätter dieses Baums sind Aktivitätsdiagramme, in denen kein Aktionszustand durch ein weiteres Aktivitätsdiagramm verfeinert wird.

Um eine Leistungsbewertung auf dieser Basis durchführen zu können, müssen die UML-Diagramme um Leistungsparameter erweitert werden. Das Thema der Erweiterung von UML um Leistungsparameter wurde bereits durch das UML-RT [OMG (2003)] behandelt. Hier wurde die Überlegung angestellt, wie spezifische Leistungsaspekte in die verschiedenen UML-Diagrammtypen eingebracht werden können.

Die PG438 hat darüberhinaus einige Leistungsparameter entwickelt, die für eine Leistungsbewertung im Rahmen von HUML und HIT benötigt werden, im UML-RT allerdings nicht vorgesehen sind bzw. vom UML-RT nicht angeboten werden.

Um ein UML-Diagramm als Eingabe für HUML zu verwenden, muss es vom Benutzer zunächst mit UML-RT erweitert werden, siehe Abbildung 1.3. Das "UML-Diagramm" wird durch Performanzerweiterungen des "UML-RT" ergänzt. Dieser Vorgang resultiert in einem "erweiterten UML-Modell".

Das mit UML-RT erweiterte Diagramm muss dann vom Benutzer mit den HUML-Erweiterungen versehen werden (siehe Abbildung 1.4). Das "erweiterte UML-Diagramm" wird durch "HUML-Erweiterungen" ergänzt. Durch diese Ergänzung entsteht das "HUML-konforme UML-Diagramm", das eine gültige Eingabe für HUML darstellt.

Unter Berücksichtigung der beiden Erweiterungen (UML-RT- und HUML-Erweiterungen) bei der Modellierung mit UML-Diagrammen, lassen sich gültige Eingaben für HUML erzeugen, die in die HIT-Eingabesprache HI-SLANG übersetzt und anschließend von HIT analysiert

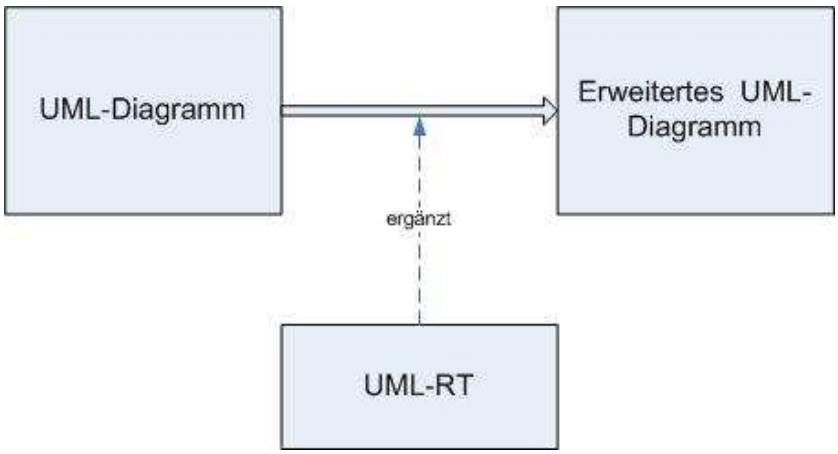

Abbildung 1.3.: Erweitern eines UML-Diagramms mit UML-RT

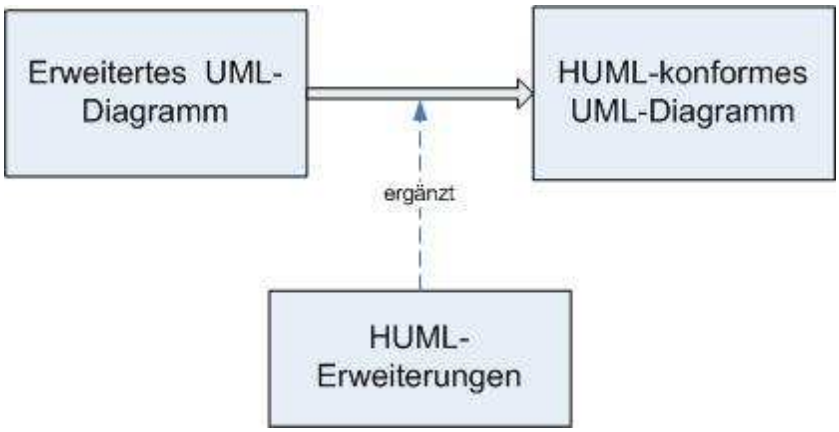

Abbildung 1.4.: Erweitern eines UML-Diagramms mit HUML-Erweiterungen

werden können (siehe Abbildung 1.5).

Abbildung 1.5.: HUML übersetzt das UML-Diagramm in eine HIT-Eingabe

1.4. UML-Diagramme

UML ist die Abkürzung für *Unified Modeling Language*, eine von der OMG entwickelte und seit 1998 standardisierte Entwurfssprache in Form von graphischer Notation, um Strukturen und Abläufe objektorientierter Programmsysteme darzustellen. In UML werden Spezifikationen, Visualisierungen, Konstruktionen und Dokumentationen von Modellen für Softwaresysteme realisiert. Sie bietet den Entwicklern die Möglichkeit, den Entwurf und die Entwicklung von

Softwaremodellen auf einheitlicher Basis zu diskutieren. Eine UML-Spezifikation besteht aus einer Sammlung sich ergänzender und teilweise überlappender Modelle, deren Summe ein Modell ist, das den strukturellen Aufbau eines Systems spezifiziert. Es beschreibt bei Anforderungsmodellen die Gegenstände der Realität, mit denen das System umgehen muss, und bei Entwurfs- und Implementierungsmodellen die Klassen, die das Problem lösen. Bei der Modellierung von Anforderungen kommt als zentrales Element das Anwendungsfallmodell hinzu, das die Benutzer-System-Interaktion aus Benutzersicht modelliert. Nach Bedarf beschreiben weitere Modelle zusätzliche Systemsichten, insbesondere dynamisches Verhalten, Funktionalität und Zusammenarbeit. UML unterstützt dabei insgesamt sieben verschiedene Sichten:

- die statische Sicht: Klassen und Objekte, strukturelle Beziehungen
- die Benutzersicht: Anwendungsfälle
- die Verhaltenssicht: Zustandsautomaten
- die Aktivitätssicht: Ablauf von Aktionszuständen
- die Interaktionssicht: Interaktion ausgewählter Objekte
- die physische Sicht: physische Systemstruktur (nur für Entwürfe)
- die Gliederungssicht: Aufgliederung der Modelle in Pakete und Subsysteme

Bei der Erweiterung von UML-Modellen um Performanzwerte ist es sinnvoll, sich vorerst auf bestimmte Arten von UML-Diagrammen zu beschränken, da ein Teil der obigen Sichten nicht für die Performanzbetrachtung geeignet sind. Die Menge der ausgewählten Diagramme wird im Folgenden vorgestellt.

1.4.1. Anwendungsfalldiagramm (*use case diagram*)

Ein Anwendungsfall beschreibt die Art und Weise wie ein Akteur (z.B eine Person) auf das System und andere Akteure (z.B. andere Programme) wirkt. Er ist eine Beschreibung für das äußerlich sichtbare Systemverhalten. Anwendungsfälle helfen, die Kommunikation zwischen den zukünftigen Systemnutzern und den Systementwicklern zu unterstützen bzw. zu verbessern. Das Anwendungsfalldiagramm modelliert demnach Interaktionen zwischen Anwendern und einem System. In einem Anwendungsfalldiagramm werden Akteure und Anwendungsfälle durch einen Pfeil assoziiert. Die Anwendungsfälle müssen durch Aktivitätsdiagramme verfeinert werden, sie gehören allerdings selbst nicht zur Eingabe für das HUML-Tool.

1.4.2. Aktivitätsdiagramm (*activity diagram*)

Die zusammenhängenden Funktionsabläufe innerhalb eines Systems, die aus einem Anwendungsfalldiagramm nicht entnommen werden können, werden durch Aktivitätsdiagramme spezifiziert. Dazu verwendet man Aktionszustände, Steuerfluss und Objektzustände als Hauptelemente. Außerdem ist es möglich, beim Steuerfluss Fallunterscheidungen und Parallelität zu modellieren sowie das Versenden und Empfangen von Signalen. Ferner ist die Zuordnung von Verantwortlichkeiten für die einzelnen Aktionszustände möglich. Aktivitätsdiagramme

unterscheiden sich dabei konzeptionell nur unwesentlich von den traditionellen Programmablaufplänen (*flowcharts*). Damit können Anwendungsfälle durch Aktivitätsdiagramme präzisiert werden. In der Regel besitzt ein Aktivitätsdiagramm einen wohl definierten Start- und Endpunkt. Dazwischen werden Aktionszustände, Entscheidungen und/oder Transitionen modelliert. Ein Aktionszustand enthält die Beschreibung einer im System ausgeführten Aktion (z.B. " Speichern einer Datei "). Von einem Aktionszustand gehen die Transitionen aus, die in UML als gerichtete Pfeile zwischen Aktionszuständen dargestellt werden und den Abschluss der Aktion und den Übergang zum nächsten Aktionszustand darstellen. Gehen mehrere Transitionen von einem Aktionszustand aus, so müssen diese mittels Bedingungen voneinander zu unterscheiden sein.

1.4.3. Verteilungsdiagramm (*deployment diagram*)

Das Verteilungsdiagramm (*deployment diagram*) wird in UML verwendet, um zu zeigen, wie Komponenten auf unterschiedliche Computer verteilt werden, über welche Protokolle diese Komponenten zusammenarbeiten und/oder welche Anforderungen an die einzelnen Einheiten gestellt werden. Verteilungsdiagramme zeigen die Topologie der Hardware. In Verteilungsdiagrammen werden Knoten benutzt, die z.B. Speicher, CPUs oder ganze Rechner repräsentieren. Um die zur Verfügung stehende Hardware eines Systems in UML abbilden zu können, kann also ein Verteilungsdiagramm verwendet werden. Leistungsbezogen gesehen eignen sich Verteilungsdiagramme im Sinne des "Last-Maschine-Paradigmas" also zur Darstellung von Maschinenkomponenten. Die zur Leistungsbewertung benötigten Daten der Maschine, z.B. die Geschwindigkeit der Maschine, werden dabei jeder Maschinenkomponente zugewiesen.

1.5. UML-Diagramme erweitert um Performanzwerte

Um Leistungsbewertungen vornehmen zu können, müssen die deterministischen UML-Diagramme um Leistungsparameter erweitert werden. Dazu ist die Kenntnis des UML-RT erforderlich. Es enthält Konzepte wie die UML um Performanzaspekte erweitert werden kann. Von der PG438 werden allerdings nur Teile des UML-RT genutzt. Darüberhinaus werden von der PG438 entwickelte Annotationen für die Leistungsbewertung vorgestellt, die nicht im UML-RT enthalten sind. Im Folgenden erfolgt zunächst ein Überblick über die zu bearbeitenden Bereiche.

1.5.1. Überblick

Die Erweiterung um Leistungsparameter muss nicht unbedingt der Entwickler der Diagramme selbst durchführen, die Leistungsparameter können aber bereits bei der Modellierung ergänzt werden. Es gibt vier große Bereiche, bei denen Ergänzungen gemacht werden müssen: Arbeitslast, Aktionszustände, Ressourcen und Messdaten.

Die Ergänzungen erfolgen größtenteils über Annotationen an stereotypisierten Aktionszuständen. Diese Annotationen sind folgendermaßen aufgebaut:

```
<<Stereotyp>>
{
...Auflistung der Tags, durch Kommata getrennt...
}
```

Beispiel:

```
<<PAstep>>
{
HUMLrepGeo = 3,
PAdelay = ('assm','mean','exponential', 3.0, 's'),
HUMLsubAct = 'diagramm'
}
```

Die genauen Ergänzungen sind in den Kapiteln 1.5.2 bis 1.5.5 nachzulesen. Im Folgenden soll zunächst einmal ein Überblick über die Erweiterungen gegeben werden.

Arbeitslast Die Arbeitslastbeschreibung eines Systems erfolgt durch die Stereotypisierung an einem Aktionszustand im modellierten System mit dem Stereotyp «PAclosedLoad» für ein geschlossenes oder mit dem Stereotyp «PAopenLoad» für ein offenes System. In Abbildung 1.6 ist eine Aktivität zu sehen, die u.a. mit «PAclosedLoad» stereotypisiert ist und an der eine Annotation befestigt ist. Der markierte Bereich der Annotation enthält die Arbeitslastbeschreibung des modellierten Systems. Für ein geschlossenes System sind die Population und ggf. die verlangte Antwortzeit zu spezifizieren. Die Population wird durch den Eigenschaftswert PApopulation angegeben, der im Beispiel 5 beträgt. Das heißt, dass stets 5 Prozesse im System zirkulieren und Last erzeugen. Die Antwortzeit des Systems wird unter Verwendung des Eigenschaftswerts PArespTime spezifiziert. Im Beispiel soll die mittlere (Zusatz "mean" in PArespTime) Antwortzeit, also die mittlere Zeit zwischen Eingabe an das System und Ausgabe durch das System, maximal 8 ms betragen. Der Parameter "req" in PArespTime charakterisiert den Eigenschaftswert als "erforderlich" (*required*, benötigt). Der Vergleich dieser Art von Vorgaben mit den ermittelten Leistungswerten nach der Leistungsbewertung wird nicht von HUML durchgeführt und ist somit dem Benutzer überlassen. Näheres zur Arbeitslast offener und geschlossener Systeme und den möglichen Eigenschaftswerten ist in Kapitel 1.5.2 zu finden.

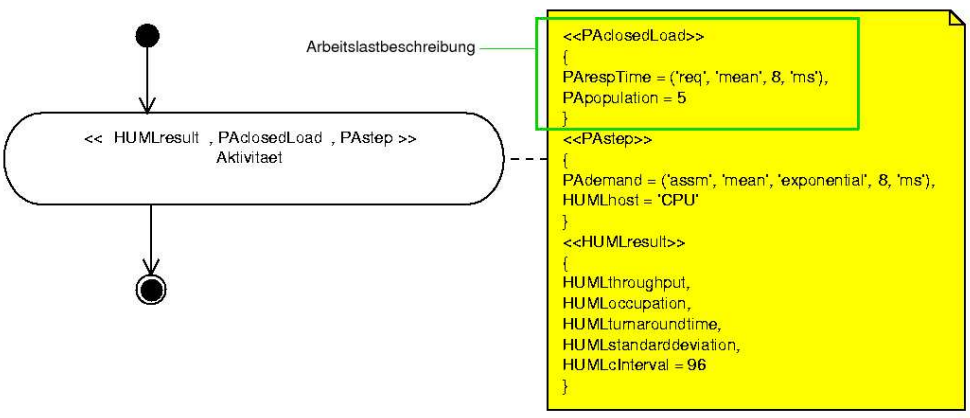

Abbildung 1.6.: Annotation der Arbeitslast an einem Aktionszustand

Aktionszustände Ein Aktionszustand eines Aktivitätsdiagramms beschreibt einen Prozess mit endlicher Ausführungszeit. Jeder Aktionszustand, der in die Leistungsbewertung einbezogen werden soll, ist mit dem Stereotyp «PAstep» zu versehen. Abbildung 1.7 zeigt einen mit «PAstep» stereotypisierten Aktionszustand, an dem eine Annotation befestigt ist. Der markierte Teil der Annotation enthält die Prozessbeschreibung. Die Menge der Eigenschaftswerte sowie weitere Informationen zum Stereotyp «PAstep» werden in Kapitel 1.5.3 beschrieben. Im Beispiel enthält die Prozessbeschreibung zwei Eigenschaftswerte. Die Ausführungszeit des annotierten Prozesses auf einer Hardware mit Referenzgeschwindigkeit 1 wird durch PAdemand beschrieben. Der Wert dieses Eigeschaftswerts beträgt in Abbildung 1.7 8 Millisekunden. Die Parameter "assm" und "mean" charakterisieren diesen Wert als "vermutet" (*assumed*, angenommen) und als Mittelwert einer exponentialverteilten Zufallsvariable. Das heißt, dass der Wert von 8 Millisekunden der vermutete Mittelwert der Ausführungszeit des Prozesses auf Referenzhardware ist. Der Eigenschaftswert HUMLhost bringt den Prozess mit der Hardware in Verbindung, auf der er ausgeführt wird. Die Beziehung basiert auf Namensgleichheit zwischen dem Wert des Eigenschaftswerts HUMLhost und dem Namen der Maschineninstanz im Verteilungsdiagramm, siehe Abbildung 1.8. Also wird der Prozess in Abbildung 1.7 auf einer Hardware ausgeführt, die den Bezeichner "CPU" trägt.

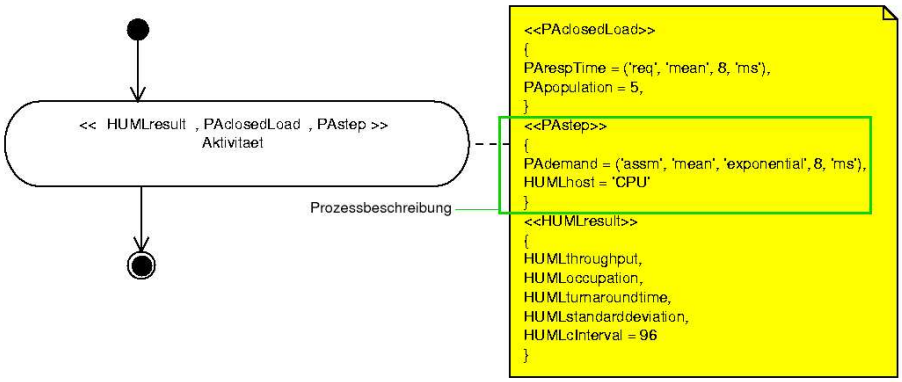

Abbildung 1.7.: Die Verbindung eines Aktionszustands mit einer Instanz einer Maschine "CPU"

Ressourcen Die Topologie der Hardwareumgebung eines Systems wird in UML-Verteilungsdiagrammen modelliert. HUML beschränkt sich in Bezug auf Verteilungsdiagramme ausschließlich auf Maschineninstanzen, siehe Abbildung 1.8. Eine Maschineninstanz beschreibt entweder eine aktive oder eine passive Ressource. Diese Eigenschaft wird durch die Stereotypisierung einer Maschine mit entweder «PAhost» für eine aktive Ressource oder «PAresource» für eine passive Ressource beschrieben. In Abbildung 1.8 ist an der Maschine eine Annotation angebracht, die die Eigenschaften der Maschine festlegt. Kapitel 1.5.4 enthält Informationen zu den beiden Stereotypen und den zur Ressourcenbeschreibung eingesetzten Eigenschaftswerten. Abbildung 1.8 zeigt eine aktive Ressource "CPU", die folglich als «PAhost» stereotypisiert ist. Die Ressource wird durch zwei Eigenschaftswerte charakterisiert. Für aktive Ressourcen ist zum Einen die Geschwindigkeit relativ zu einer Referenzhardware der Geschwindigkeit 1 anzugeben. Dies erfolgt durch die Verwendung des Eigenschaftswerts PArate. Die Maschine im Beispiel hat also die doppelte Geschwindigkeit der Referenzhardware. Zum Anderen legt

der Eigenschaftswert PAschdPolicy die Prozessverarbeitungsrichtlinie der Maschine fest. Für eine Liste der möglichen Richtlinien siehe Kapitel 1.5.4. Im Beispiel werden Prozesse nach der Richtlinie FIFO verarbeitet.

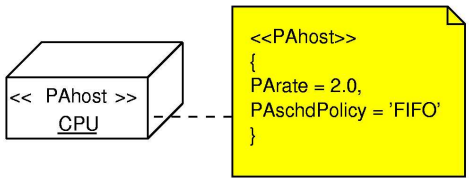

Abbildung 1.8.: Verteilungsdiagramm mit einer annotierten Maschinen-Instanz

Messdaten Sollen an einem Aktionszustand oder einer Ressource Leistungsdaten gemessen werden, so erfolgt dies durch die Stereotypisierung der jeweiligen Entität mit «HUMLresult». Die zu messenden Werte werden in der zur Entität gehörigen Annotation in einem ebenfalls mit «HUMLresult» gekennzeichneten Block eingetragen (s. Abb. 1.9). Dabei wird der bloße Name des zu bestimmenden Werts aufgeführt:

- HUMLthroughput
- HUMLoccupation
- HUMLturnaroundtime
- HUMLutilization
- HUMLpopulation

Darüberhinaus kann für ein simulatives Lösungsverfahren das Konfidenzintervall (HUMLc-Interval in Abb. 1.9) angegeben werden, sowie ob die Standardabweichung (HUMLstandard-deviation in Abb. 1.9) berechnet werden soll.

Der Stereotyp «HUMLresult» und die zugehörigen Eigenschaftswerte werden in Kapitel 1.5.5 näher beleuchtet.

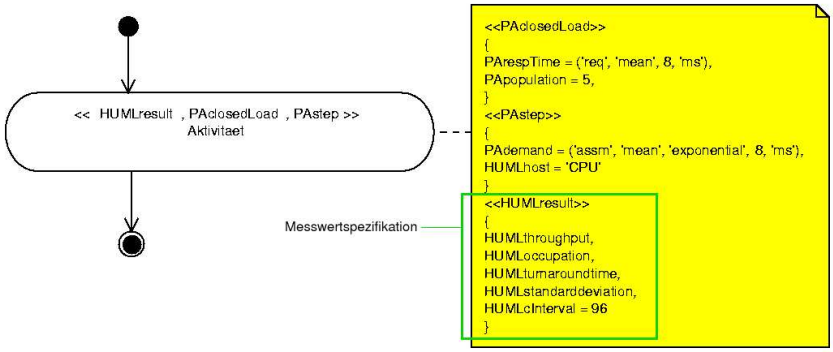

Abbildung 1.9.: Annotation der zu ermittelnden Werte mit dem Stereotyp «HUMLresult»

1.5.2. Arbeitslast in HUML

Die Arbeitslast ist am ersten Aktionszustand eines Szenarios bzw. Aktivitätsdiagrammes zu spezifizieren, d.h. der erste mit «PAstep» stereotypisierte Aktionszustand wird zusätzlich mit «PAopenLoad» oder «PAclosedLoad» stereotypisiert.

Im Folgenden werden die Stereotype «PAopenLoad» und «PAclosedLoad» für eine Arbeitslast in einem offenen bzw. geschlossenen System vorgestellt. Für die Eigenschaftswerte wird die Grammatik in BNF angegeben. BNF-Literale stehen in doppelten Anführungszeichen, Nichtterminalsymbole in einfachen spitzen Klammern. Optionale Parameter stehen in eckigen Klammern. Ein Stern steht für den endlichen Abschluss. Alternativen werden durch vertikale Balken getrennt. Innerhalb der BNF-Literale können TVL (*Tag Value Language*)-Literale stehen, welche zwischen einfachen Anführungszeichen stehen. Weitere Informationen zur TVL finden sich im Anhang A.

«PAopenLoad» Dieses Stereotyp wird zur Definition einer Arbeitslast in einem offenen System verwendet. Die dazugehörigen Eigenschaftswerte sind:

- **PAoccurence**
  Der Eigenschaftswert PAoccurence vom Typ *RTarrivalPattern* definiert die Ankunftsrate der Prozesse im System.

- **PArespTime**
  Der Eigenschaftswert PArespTime vom Typ *PAperfvalue* spezifiziert die erwartete Zeit zwischen dem Starten und Beenden eines Szenarios und wird bei der Leistungsbewertung durch HUML nicht berücksichtigt.

- **PApriority**
  Der Eigenschaftswert PApriority spezifiziert die Priorität der Arbeitslast. Dieser Eigenschaftswert wird von HUML nicht berücksichtigt.

Die Eigenschaftswerte von PAopenLoad werden in Tabelle 1.1 zusammengefasst. Der Typ

Tag	**Type**
PAoccurence	RTarrivalPattern
PArespTime	PAperfValue
PApriority	Integer

Tabelle 1.1.: Die Eigenschaftswerte von PAopenLoad

PAperfValue wird folgendermaßen aufgebaut:

```
PAperfValue :: =  "("<source-modifier>","<type-modifier>","<time-value>")
```

- `<source-modifier> :: = 'req' | 'assm' | 'pred' | 'msr'`

  beschreibt, ob der Leistungsparameter gemessen ('msr' für measured), angenommen ('assm' für assumed), benötigt ('req' für required) oder vorhergesagt ('pred' für predicted) wurde.

- `<type-modifier>::='mean'|'sigma'|'kth-mom',<Integer>|'max'|'percentile' ,<real>|'dist'`

gibt die statistische Bedeutung des Wertes an: Mittelwert (mean), Standardabweichung (sigma), k-tes Moment (kth-mom, <k>), Maximum (max), prozentualer Anteil (percentile, <real>) oder Verteilung (dist). HUML unterstützt nur den Mittelwert (mean) und die Verteilung (dist).

- `<Time-value>` vom Typ *RTtimeValue* schließlich gibt den Wert selbst an. *RTtimeValue* ist ein intuitiv verständlicher, aber umfangreicher Typ, der im *UML-RT* [OMG (2003)] nachgeschlagen werden kann. In HUML wird folgende Syntax für RTtimeValue verwendet:
  `<Time-value>::= "("('exponential' "," )?  Real "," <timeStr>")"`
  `<timeStr>::= 'ms'|'s'|'min'|'h'`

Beispiel:

1. `PArespTime = ('assm', 'mean', (3.0, 's'))` beschreibt eine gleichverteilte erwartete Antwortzeit.
2. `PArespTime = ('assm', 'dist', ('exponential', 10.0, 'ms'))` beschreibt eine exponentialverteilte erwartete Antwortzeit.

Der Typ *RTarrivalPattern* wird nicht in seinem kompletten Umfang von HUML unterstützt. Im Folgenden wird die Syntax beschrieben, wie sie von HUML verwendet wird.
`RTarrivalPattern :: =  "(" ('periodic' | 'unbounded') "," <Time-value> ")"`
Beispiel:

1. `PAoccurence = ('periodic', (2.0, 's'))` beschreibt ein periodisches Ankunftsmuster.
2. `PAoccurence = ('unbounded', (5.0, 's'))` beschreibt ein Ankunftsmuster, das durch eine Wahrscheinlichkeits-Verteilungsfunktion spezifiziert ist.

«PAclosedLoad» Eine Arbeitslast in einem geschlossenen System ist durch eine feste Anzahl von Benutzern oder Aufträgen charakterisiert. Die dazugehörigen Eigenschaftswerte sind:

- **PApopulation**
  Der Eigenschaftswert PApopulation spezifiziert die Anzahl der Prozesse im System, die eine Last erzeugen.
- **PAextDelay**
  Der Eigenschaftswert PAextDelay spezifiziert die Zeitverzögerung zwischen dem Ende einer Antwort und dem Start einer neuen Anfrage vom System. Die externe Zeitverzögerung wird auch als "Denk-Zeit" bezeichnet, da sie häufig zur Modellierung systemexterner Verzögerungen benutzt wird.
- **PArespTime** und **PApriority**
  Die Eigenschaftswerte PArespTime und PApriority haben für «PAclosedLoad» dieselbe Bedeutung wie für «PAopenLoad».

Die Eigenschaftswerte von «PAclosedLoad» sind in Tabelle 1.2 zusammengefasst.

Tag	Type
PApopulation	Integer
PAextDelay	PAperfValue
PArespTime	PAperfValue
PApriority	Integer

Tabelle 1.2.: Die Eigenschaftswerte von PAclosedLoad

1.5.3. Aktionszustände in HUML

Wird ein Aktionszustand in einem Aktivitätsdiagramm als «PAstep» stereotypisiert, so können Eigenschaftswerte *(tagged values)* angegeben werden, die die Eigenschaften des Aktionszustands als Prozess festlegen. «PAstep» wird durch folgende Eigenschaftswerte näher beschrieben: PAdemand, PArep, HUMLrepGeo, PAdelay, PAinterval, HUMLsubAct, HUMlhost und PAextOp. Die mit "PA" beginnenden *tagged values* entstammen dem *UML-RT* [OMG (2003)]. Bei den mit dem Präfix "HUML" versehenen *tagged values* handelt es sich um durch die PG438 eingeführten Eigenschaftswerte.

- **PAdemand**
  Der Eigenschaftswert PAdemand beschreibt die benötigte Zeit zur vollständigen Ausführung des durch den annotierten Aktionszustand beschriebenen Prozesses auf einer Ressource mit Referenzgeschwindigkeit 1. Das heißt, dass die tatsächliche Ausführungszeit des Prozesses auf einer Ressource auch stark von deren Geschwindigkeit abhängt.
  Beispiel:

  1. `PAdemand = ('assm', 'mean', 'exponential', 3.0, 's')`
     Die angenommene ('assm', *assumed*) Verarbeitungszeit eines Prozesses auf einer Hardware mit Referenzgeschwindigkeit 1 ist der Mittelwert ('mean') einer exponentialverteilten Zufallsvariablen ('exponential') und beträgt 3 Sekunden ('s', *second*).

- **PArep**
  PArep gibt eine deterministische Anzahl an Schleifendurchläufen an. Das heißt, dass der durch den annotierten Aktionszustand repräsentierte Prozess wiederholt durchgeführt wird und die Zahl der Ausführungen deterministisch ist.
  Beispiel:

  1. `PArep = 3`
     Die deterministische Anzahl der Widerholungen des Prozesses beträg 3.

- **HUMLrepGeo**
  HUMLrepGeo spezifiziert eine nichtdeterministische Anzahl an Schleifendurchläufen. Das heißt, dass der durch den annotierten Aktionszustand repräsentierte Prozess wiederholt ausgeführt wird und die Zahl der Ausführungen der Wert einer geometrisch verteilten Zufallsvariablen ist.
  Beispiel:

  1. `HUMLrepGeo = 3`
     Die nichtdeterministische Anzahl (HUMLrepGeo im Gegensatz zu PArep) der Wiederholungen des Prozesses beträgt 3. Der Wert ist Mittelwert einer Zufallsvariablen.

- **PAdelay**
  Der Eigenschaftswert PAdelay beschreibt eine Zeitverzögerung, die *vor* der Ausführung des durch den annotierten Aktionszustand repräsentierten Prozesses auf der assoziierten Maschine auftritt. Das heisst, dass diese Form der Verzögerung im Falle wiederholter Ausführung (PArep, HUMLrepGeo) nur einmal auftritt.
  Beispiel:

  1. `PAdelay = ('assm', 'mean', 'exponential', 5.0, 's')`
     Die Verzögerung, die vor der Ausführung des Prozesses auftritt, ist der Mittelwert ('mean') einer exponentialverteilten ('exponential') Zufallsvariablen und beträgt 5 Sekunden.

- **PAinterval**
  PAinterval gibt eine Zeitverzögerung an, die *während* der Ausführung des durch den annotierten Aktionszustand repräsentierten Prozesses auf der assoziierten Maschine auftritt. Das heißt, dass diese Form der Verzögerung im Falle wiederholter Ausführung (PArep, HUMLrepGeo) für jede Schleifeniteration auftritt.
  Beispiel:

  1. `PAinterval = ('assm', 'mean', 'exponential', 2.0, 's')`
     Die angenommene ('assm', assumed) Verzögerung während der Ausführung (PAinterval im Gegensatz zu PAdelay) des Prozesses beträgt 2 Sekunden. Der Wert ist der Mittelwert ('mean') einer exponentialverteilten ('exponential') Zufallsvariablen.

- **HUMLsubAct**
  HUMLsubAct verweist auf ein Aktivitätsdiagramm, das den durch den annotierten Aktionszustand beschriebenen Prozess verfeinert. Die Beziehung basiert auf Namensgleichheit zwischen dem Wert des Eigenschaftswerts und dem Namen des Aktivitätsdiagramms.
  Beispiel:

  1. `HUMLsubAct = 'Bericht zusammenstellen'`
     Der Prozess wird in einem Aktivitätsdiagramm namens "Bericht zusammenstellen" verfeinert.

- **HUMLhost**
  HUMLhost verweist auf eine aktive Ressource, auf der der durch den Aktionszustand modellierte Prozess ausgeführt wird. Die Beziehung basiert auf Namensgleichheit zwischen dem Wert des Eigenschaftswerts und dem Namen der Ressource im Verteilungsdiagramm.
  Beispiel:

  1. `HUMLhost = 'Netzwerkverbindung'`
     Der Prozess wird von einer aktiven Ressource (HUMLhost im Gegensatz zu PAextOp) namens "Netzwerkverbindung" verarbeitet.

- **PAextOp**
  PAextOp verweist auf eine oder mehrere passive Ressourcen, die der Prozess benutzt. Die Beziehung basiert auf Namensgleichheit zwischen dem bzw. den Namen der Ressourcen im Eigenschaftswert und den Namen der Ressourcen im Verteilungsdiagramm.

Neben der Prozess-Maschine-Beziehung kann PAextOp benutzt werden, um die Belastung der Maschine auf zweierlei Art auszudrücken. Einerseits besteht die Möglichkeit, die Gesamtbelastung in Form eines PAperfValue anzugeben (Beispiel 1), andererseits ist es möglich, die Anzahl der Zugriffe in Form eines Integer-Werts zu spezifizieren (Beispiel 2).
Beispiele:

1. `PAextOp = ('Festplatte', 'assm', 'mean', 'exponential', 4.0, 's')`
   Der Prozess beansprucht eine Ressource namens "Festplatte" einmalig für 4 Sekunden. Dieser Wert ist der angenommene ('assm', *assumed*) Mittelwert ('mean') einer exponentialverteilten ('exponential') Zufallsvariablen.
2. `PAextOp = ('Festplatte', 61250)`
   Der Prozess beansprucht eine Ressource namens "Festplatte" 61250 mal pro Ausführung. Der Zeitverbrauch entsteht dabei ressourcenseitig durch die Zugriffszeit pro Inanspruchnahme.

Tabelle 1.3 ordnet jedem der genannten Eigenschaftswerte einen Datentyp zu.

Tag	**Type**
PAdemand	PAperfValue
PArep	Integer
PAinterval	PAperfValue
PAdelay	PAperfValue
HUMLrepGeo	Integer
HUMLsubAct	String
HUMLhost	String
PAextOp	String, Integer | <time-value>

Tabelle 1.3.: Eigenschaftswerte des Stereotyps «PAstep»

Die Definition von `<time-value>` kann im *UML-RT* nachgelesen werden.

1.5.4. Ressourcen in HUML

Das UML-Profil zur Erweiterung um Performanzangaben (*UML-RT* [OMG (2003)]) unterscheidet aktive und passive Ressourcen.

Unter einer aktiven Ressource (*processing resource*) versteht man eine Hardwarekomponente wie einen Prozessor oder ein sonstiges Gerät ohne echte Parallelverarbeitung, an der zu jedem Zeitpunkt höchstens ein Prozess in Bedienung sein kann. Auch wenn die aktive Ressource von mehreren Prozessen zugleich belegt wird, kann sie diese nur nacheinander oder in gewissen Zeitabständen abwechselnd (*round robin*-Verfahren) bedienen.

In *UML-RT* ist eine passive Ressource (*passive resource*) als eine Hardwarekomponente definiert, die in einer Beziehung zu Softwareprozessen steht, aber nicht eigentlich an ihrer Bedienung im engeren Sinne, d.h. ausführend, beteiligt ist. Mit dieser Art von Ressourcen sind vor allem Speichermedien wie Festplattenspeicher oder RAM gemeint, aber allgemeiner auch jede Art von physischen Geräten. Eine wichtige Grundeigenschaft von passiven Ressourcen ist, dass sie sich gegenüber Softwareprozessen rein reaktiv verhalten.

Im Folgenden werden aktive und passive Ressourcen beschrieben.

Aktive Ressourcen Aktive Ressourcen werden durch Maschinenknoteninstanzen in Verteilungsdiagrammen dargestellt, die durch «PAhost» stereotypisiert sind (Abb. 1.10).

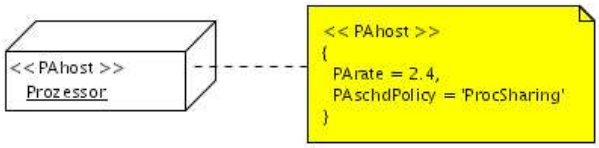

Abbildung 1.10.: Beispiel einer aktiven Ressource

Die Bedienung eines Prozesses durch eine aktive Ressource wird in einem Aktivitätsdiagramm am jeweiligen Aktionszustand durch den für das HUML-Projekt neu eingeführten Eigenschaftswert HUMLhost des Stereotyps «PAstep» ausgedrückt. Die mittlere Verarbeitungszeit des Prozesses durch eine Maschine mit Referenzgeschwindigkeit 1 wird durch den Eigenschaftswert PAdemand angegeben (vgl. Kapitel 1.5.3 und Abbildung 1.11).

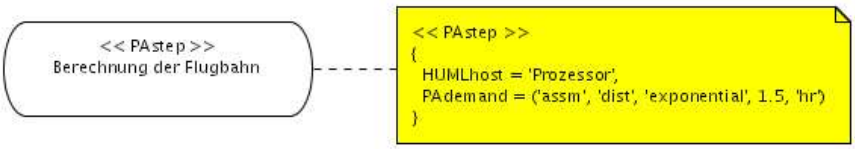

Abbildung 1.11.: Bedienung eines Prozesses durch eine aktive Ressource

Die Eigenschaftswerte (*tagged values*) des «PAhost»-Stereotyps dienen der näheren Beschreibung der aktiven Ressource:

- **PArate**
  Der Eigenschaftswert PArate gibt einen relativen Geschwindigkeitsfaktor an, der sich auf Prozessbedienungszeiten als Divisor bzw. auf Prozessbedienungsraten als Multiplikator auswirkt.

- **PAschdPolicy**
  Der Eigenschaftswert PAschdPolicy dient der Festlegung der Prozessplanungsrichtlinie (*scheduling policy*) für die Bedienung konkurrierender Prozesse. Es handelt sich um einen Enumerationstyp (siehe Tabelle 1.4). Für manche Enumerationswerte ist es notwendig, eine gesonderte Angabe hinzuzufügen, die festlegt, ob eine Prozesswarteschlange präemptiv ist oder nicht. Der zugehörige Eigenschaftswert heißt PApreemptable, wird aber im HUML-Werkzeug nicht verwendet.

- **PAthroughput** und **PAutilization**
  Die in *UML-RT* vorgesehenen Eigenschaftswerte PAthroughput und PAutilization finden im HUML-Werkzeug keine Berücksichtigung, weil die durch sie angegebenen Werte zur Ergebnismenge der Leistungsbewertung seitens HIT gehören und gemäß der Anforderungsdefinition für das HUML-Werkzeug bei den Eigenschaftswerten des «HUMLresult»-Stereotyps eingetragen werden.

- **PActxtSwT**
  Der Eigenschaftswert PActxtSwT wird ebenfalls ignoriert, weil Zeitverzögerungen bei

Kontextwechseln im Rahmen von HIT nicht dargestellt werden können und in den meisten Fällen sowieso vernachlässigbar sind. Ein weiterer Grund, auf den Eigenschaftswert PActxtSwT zu verzichten, ist, dass in *UML-RT* die Möglichkeit fehlt, im Fall einer Ressource mit *time sharing*-Richtlinie die Zeitscheibe für Kontextwechsel anzugeben.

- **PAprioRange**
  Der Eigenschaftswert PAprioRange spielt für das HUML-Werkzeug auch keine Rolle, da die Angabe von Prozessprioritäten sich nach den Möglichkeiten von HIT zu richten hat, wo sie sich im Bereich von 0 bis 32767 bewegen. (Im übrigen ist die Angabe von Prozessprioritäten im HUML-Werkzeug zunächst nicht implementiert; vgl. dazu Kapitel 1.6.3)

Es ergibt sich die Tabelle 1.4 der vom HUML-Werkzeug interpretierten Eigenschaftswerte des Stereotyps «PAhost».

Tag	**Type**	
PArate	Real	optional (Standard = 1.0)
PAschdPolicy	Enumeration: 'ProcSharing', 'FIFO', 'LIFO', 'HeadOfLine', 'PreemtResume', 'PrioProcSharing'	obligatorisch
PApreemptable	Boolean	nur bei 'LIFO' und 'PrioProcSharing'

Tabelle 1.4.: Eigenschaftswerte des Stereotyps «PAhost»

Passive Ressourcen Der Hauptaspekt einer passiven Ressource ist, dass sie im weitesten Sinne "Raum" bereitstellt, der Prozessen durch einen Zugriffssteuerungsmechanismus zur Verfügung gestellt wird. Eine passive Ressource kann z.B. eine Menge von in Blöcken zuteilbarem RAM-Speicher sein, aber auch ein einzelnes I/O-Gerät oder auch ein redundant ausgelegtes System von I/O-Geräten, wie beispielsweise eine Druckerstation mit mehreren parallel arbeitenden Druckern, die in der Lage sind, Prozesse echt parallel zu bedienen, indem Druckanfragen auf jeweils freie Geräte verteilt werden.

Eine aktive Ressource wie ein Prozessor kann während des ganzen Lebenszyklus eines Prozesses von diesem belegt sein, muss jedoch u.U. mit anderen Prozessen geteilt werden, während passive Ressourcen angefordert, zugeteilt, benutzt und wieder freigegeben werden. Ist eine passive Ressource (oder eine ihrer Untereinheiten) einem Prozess zugeteilt, hat er diese i.d.R. exklusiv, bis er sie wieder freigibt, es sei denn, sie ist präemptiv und kann ihm "gewaltsam" weggenommen werden. Das ist oft bei RAM der Fall, das durch eine betriebssystemgesteuerte dynamische Speicherverwaltung, die einen *swap*-Mechanismus beinhaltet, einem Prozess entzogen und einem anderen zugeteilt werden werden kann.

Passive Ressourcen werden durch Maschinenknoteninstanzen in Verteilungsdiagrammen dargestellt, die durch «PAresource» stereotypisiert sind (Abb. 1.12). Das Stereotyp ähnelt zwar demjenigen für aktive Ressourcen, passive Ressourcen werden aber durch z.T. andere Eigenschaftswerte beschrieben:

- **PAthroughput** und **PAutilization**
  PAthroughput und PAutilization sind ihnen gemeinsam, werden vom HUML-Werkzeug

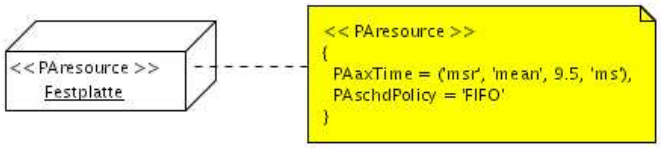

Abbildung 1.12.: Beispiel einer passiven Ressource

in beiden Fällen aber nicht berücksichtigt.

- **PArespTime** und **PAwaitTime**
  «PAresource» kennt ferner die Eigenschaftswerte PArespTime und PAwaitTime, die ebenfalls zu den nichtberücksichtigten Eigenschaftswerten gehören.

- **PAaxTime**
  PAaxTime beschreibt die Zugriffszeit der Ressource. Die gesamte Bedienungszeit eines Prozesses an einer passiven Ressource ist nun offenbar das Produkt aus der Zugriffszeit und der Anzahl der Zugriffe zuzüglich einer eigentlichen Bedienungszeit (z.B. ist die Gesamtbedienungszeit für einen Schreibvorgang auf einer Festplatte das Produkt aus der Zeit für eine Plattenkopfpositionierung und der Anzahl der Spurwechsel zuzüglich der eigentlichen Zeit für das Schreiben der Daten). Sowohl die eigentliche Bedienungszeit als auch die Zahl der dabei stattfindenden Zugriffe kann durch den Eigenschaftswert PAextOp des Stereotyps «PAstep» angegeben werden (Abb. 1.13). Die angegebenen Zeiten werden kumulativ interpretiert, d.h. addiert.

  Weil Bedienungszeiten auch so modelliert werden können, dass sie gänzlich als Zeitangaben im Eigenschaftswert PAextOp stehen, ohne dass eine Zugriffsanzahl eine Rolle spielt, ist der Eigenschaftswert PAaxTime optional.

Abbildung 1.13.: Beispiel eines kumulativen Gebrauchs des Eigenschaftswerts PAextOp

- PAschdPolicy ist ein Enumerationstyp, dessen mögliche Werte sich von denen des gleichnamigen Eigenschaftswerts von aktiven Ressourcen unterscheiden.

  Von den Enumerationswerten des Eigenschaftswerts PAschdPolicy des Stereotyps «PAresource» kann nur 'FIFO' eindeutig einem Standardservertyp zugeordnet werden, der in HIT zur Verfügung steht. Es handelt sich dabei entweder um einen einfachen Standardserver ohne Parallelverarbeitung, der eine Warteschlange mit FIFO-Richtlinie besitzt, oder einen mit echter Parallelverarbeitung, bei dem jeder bediente Prozess die volle Bedienungsleistung erhält, d.h. jeder Prozess wird mit derselben Bedienungsrate bedient, als ob er der einzige Prozess wäre, der das Bedienungszentrum belegt. Ein realistisches Beispiel dafür ist ein redundant ausgelegtes System, z.B. eine Druckerstation mit mehreren Druckern. Die Aufnahmefähigkeit eines solchen Systems ist durch den Grad der

Redundanz begrenzt. Belegen mehr Prozesse ein Bedienungszentrum als dieses parallel zu verarbeiten vermag, bildet sich eine Warteschlange mit FIFO-Richtlinie.

Die übrigen Enumerationswerte des Eigenschaftswerts PAschdPolicy haben Bedeutungen, die aus dem *UML-RT* nicht ganz ersichtlich sind. Da ihre Entsprechungen zu von HIT bereitgestellten Standardkomponententypen unklar sind, wird ihre Verwendung in der Eingabemenge des HUML-Werkzeugs ausgeschlossen.

- **PAschdParam**
  Der Eigenschaftswert PAschdParam kommt nur als Ergänzung zum Einsatz, wenn als Wert für PAschdPolicy 'PriorityCeiling' gewählt wurde. Auch er spielt also für das HUML-Werkzeug keine Rolle.

- **PAcapacity**
  PAcapacity ist ein Eigenschaftswert, der die maximale Zahl parallel verarbeiteter Prozesse am Bedienungszentrum angibt. Er entscheidet also über den Grad der Redundanz.

Es ergibt sich die Tabelle 1.5 der vom HUML-Werkzeug interpretierten Eigenschaftswerte des Stereotyps «PAresource». Dabei ist zu beachten, dass mindestens der Eigenschaftswert PAschdPolicy angegeben werden muss, da zu einem späteren Zeitpunkt eventuell auch weitere *scheduling policies* von HUML unterstützt werden könnten.

Tag	**Type**	
PAaxTime	PAperfValue	optional
PAschdPolicy	Enumeration: nur 'FIFO' möglich	obligatorisch
PAcapacity	Integer	optional (Standard = 1)

Tabelle 1.5.: Eigenschaftswerte des Stereotyps «PAresource»

Der Umgang mit passiven Ressourcen in einem UML-Modell wirft bei der Übertragung in ein HIT-Modell das Problem der Einbeziehung eines Ressourcenmanagements auf. Die Zugriffssteuerung auf passive Ressourcen wird von modernen Betriebssystemen durch Semaphore bewerkstelligt. Diese stehen auch als Standardkomponententypen in HIT zur Verfügung. Eine wesentliche Schwierigkeit besteht aber darin, dass in *UML-RT* keine Modellierungsmittel vorgesehen sind, die eine ununterbrochene Belegung einer passiven Ressource auch über mehrere Aktionen eines Aktivitätsdiagramms hinweg und eine anschließende Freigabe ausdrücken.

Diese Lücke wird im Rahmen des HUML-Projekts nicht geschlossen. Die Angabe einer passiven Ressource an einem Aktionszustand bedeutet, dass diese beim Eintritt in den Aktionszustand angefordert wird, der Prozess oder Prozessstrang bis zur Zuteilung schläft, die Aktion ausgeführt, d.h. der Prozess bedient wird, und die Ressource beim Verlassen des Aktionszustands wieder freigegeben wird. Der UML-Modellierer muss zunächst damit leben, dass bei jeder Transition von einem Aktionszustand zum Nächsten die Zuteilung einer passiven Ressource verlorengeht, auch wenn er durch die Angabe derselben passiven Ressource an beiden Aktionszuständen auszudrücken versucht, daß sie kontinuierlich belegt wird.

Auf eine Behebung dieses Mangels wurde im Rahmen der Projektgruppenarbeit verzichtet, weil sie auf die Erweiterung von UML um einen kompletten Mechanismus des Ressourcenmanagements hinausliefe, was den Rahmen gesprengt hätte.

Das Zusammenspiel von aktiven und passiven Ressourcen kann durch ein realistisches Szenario deutlich gemacht werden: Bei Softwareprozessen kommt es häufig vor, dass sie Vorgänge

beinhalten, bei denen es sich um zeitliche Folgen von abwechselnden *CPU bursts* und *I/O bursts* handelt. In einem UML-Modell ordnet man daher einem Softwareaktionszustand, z.B. einem Dateizugriff, sowohl eine aktive Ressource, z.B. einen Prozessor, als auch eine passive Ressource, z.B. eine Festplatte, zu (Abb. 1.14). Die Wechsel zwischen den Inanspruchnahmen beider werden durch eine gegenseitige Synchronisierung bewirkt. Beide Ressourcen bleiben auch außerhalb der unmittelbaren Inanspruchnahme für den Prozess reserviert, also sowohl der Prozessor während eines *I/O bursts* als auch das I/O-Medium oder -gerät während eines *CPU bursts*.

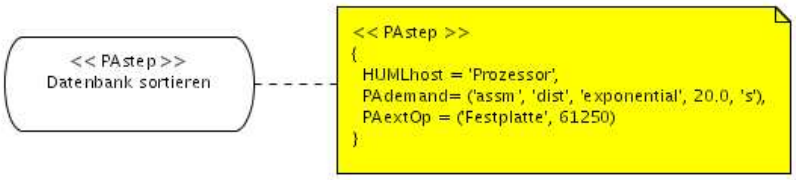

Abbildung 1.14.: Belegung sowohl einer aktiven als auch einer passiven Ressource durch einen Prozess

Reservierungen von Ressourcen werden durch einen Zugriffsmechanismus des Betriebssystems, der meistens durch Semaphore implementiert ist, gesteuert.

1.5.5. Messdaten in HUML

Das Stereotyp «HUMLresult» ist nicht Teil des *UML-RT* [OMG (2003)]. Es ist eine Erweiterung zur übersichtlichen Ein- und Ausgabe von gewünschten Leistungsmaßen und Messergebnissen. Die Ergebnisse der Leistungsbewertung durch HIT werden im UML-Modell in die HUMLresult-Abschnitte der Annotationen eingetragen.

Der Vorteil der Benutzung dieses neuen Stereotyps besteht darin, dass für den Anwender die gewünschten Leistungsbewertungsparameter und Ergebnisse blockweise dargestellt werden. Diese Lösung erhöht die Benutzerfreundlichkeit und die Übersichtlichkeit. Andererseits hat dies aber den Nachteil, dass keine Lastpfade (siehe Büttner u. a.) gebildet und gezielt ausgewertet werden können.

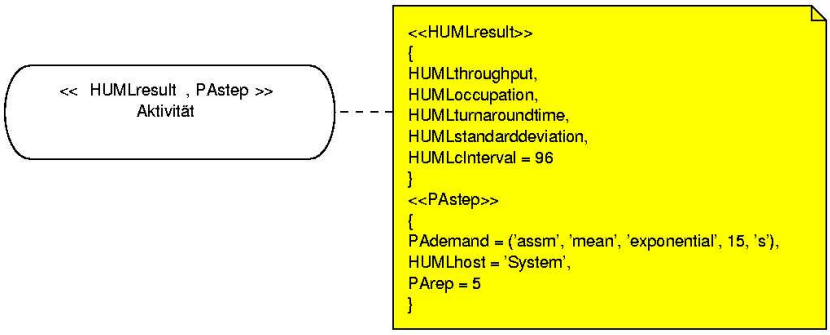

Abbildung 1.15.: «HUMLresult»-Block

Um Leistungsdaten an einem Aktionszustand oder einer Ressource zu ermitteln, muss dieser bzw. diese mit «HUMLresult» stereotypisiert werden. In einer angehängten Annotation

werden die Leistungsdaten, die berechnet werden sollen, innerhalb eines mit «HUMLresult» überschriebenen Blocks spezifiziert, siehe Abb. 1.15. Nicht erwünschte Werte werden nicht aufgeführt. Die zugehörigen *tagged values*, die für «HUMLresult» angegeben werden können und die in Abbildung 1.15 in der Annotation zu sehen sind, werden im Folgenden näher erläutert.

- **HUMLpopulation**
  Die *Population* ist die durchschnittliche Warteschlangenlänge. In Bezug auf einen Softwarezustand oder eine Hardwareressource ist die Warteschlangenlänge als die Zahl von Softwareprozessen zu verstehen, die sich gleichzeitig in dem Zustand bzw. an der Hardwareressource befinden.

- **HUMLthroughput**
  Der *Durchsatz* beschreibt für einen Softwarezustand oder eine Hardwaressource die Zahl der Prozesse, die über einen bestimmten Zeitraum (eine Zeiteinheit) hinweg diesen Zustand bzw. diese Hardwareressource verlassen haben.

- **HUMLturnaroundtime**
  Die *Antwortzeit* bezeichnet für einen Softwarezustand bzw. eine Hardwareressource die durchschnittliche Zeit, die sich ein Prozess in diesem Zustand oder an dieser Ressource befindet.

- **HUMLoccupation** (nicht bei analytischen Lösungsverfahren verwendbar)
  Die *Belegung* ist die Wahrscheinlichkeit, einen Softwarezustand bzw. eine Hardwareressource belegt anzutreffen. Dies ist immer dann der Fall, wenn sich in dem Softwarezustand mindestens ein Prozess befindet bzw. die Hardwareressource mindestens einen Prozess bedient. Wird der Eigenschaftswert HUMLoccupation angegeben, ist nur das simulative Lösungsverfahren anwendbar.

- **HUMLutilization** (nur bei Harwareressourcen verwendbar)
  Die *Auslastung* ist eine Größe, die in der HUML-Modellwelt nur an Hardwareressourcen eine Rolle spielt. Sie ist das zeitliche Mittel des Produkts der Anzahl der Prozesse, die am Hardwareserver tatsächlich in Bearbeitung (nicht "schlafend" in der Warteschlange) sind, und dem Quotienten aus anteiliger relativer Bedienungsgeschwindigkeit, die je einem dieser Prozesse zukommt, und der relativen Standardgeschwindigkeit. Die anteilige relative Bedienungsgeschwindigkeit ist bei HIT-Standardservern mit FIFO-Warteschlangen und jeweils nur einem bedienten Prozess gleich der Standardgeschwindigkeit selbst. Bei einem HIT-Standardserver ohne Warteschlange und mit *shared*-Bedienungsrichtlinie ist sie gleich dem Quotienten aus relativer Standardgeschwindigkeit und der Anzahl der bedienten Prozesse. Bei HIT-Standardservern mit *sdequal*-Richtlinie ist sie gleich dem Produkt aus Standardgeschwindigkeit und zustands-, d.h. von der Anzahl der bedienten Prozesse, abhängiger Normgeschwindigkeit. Zu diesen nicht ganz trivialen Begriffen vgl. auch Kapitel 1.6.3 sowie Büttner u. a. Kapitel 5.1.2 und Anhang F.3.4.

  Die recht abstrakte Definition der Auslastung hat für die Hardwareressourcentypen der HUML-Modellwelt je nachdem, auf was für HIT-Servertypen sie abgebildet werden, die folgenden Konsequenzen: Bei einer aktiven Ressource ist die Auslastung gleich der Belegung. Bei einer passiven Ressource wird die Auslastung durch HUML über eine besondere Manipulation von Normgeschwindigkeiten innerhalb des HI-SLANG-Modells (vgl. dazu

Kapitel 1.6.3) so skaliert, dass sie effektiv gleich dem Quotienten aus Population und Kapazität der Ressource ist.

Der Vorteil der Auslastung ist, dass sie bei aktiven Ressourcen anstelle der Belegung verwendet werden kann und dann auch bei der Anwendung analytischer Lösungsverfahren zur Verfügung steht.

- **HUMLstandarddeviation** (nur bei Ressourcen sinnvoll)
  Der Eigenschaftswert HUMLstandarddeviation beschreibt die Standardabweichung der Messergebnisse vom geschätzten Mittelwert.

- **HUMLcInterval** (nur bei Ressourcen sinnvoll)
  Der Eigenschaftswert HUMLcInterval dient der Angabe einer Wahrscheinlichkeit (Integer zwischen 90 und 99), mit der der reale Mittelwert in einem bestimmten Intervall um den berechneten Mittelwert liegt. Dabei wird die Standardabweichung automatisch mitberechnet.

1.5.6. Aktivitätsdiagramme in HUML

Aktivitätsdiagramme beschreiben strukturierte, deterministische Abfolgen von Aktionszuständen. In HUML dienen Aktivitätsdiagramme außerdem dazu, eine baumförmige Gesamtstruktur des Systems zu modellieren. Dazu gibt es genau ein Aktivitätsdiagramm, das als Wurzeldiagramm des Systems fungiert. An dessen erstem Aktionszustand ist die Arbeitslast zu spezifizieren (siehe Kapitel 1.5.2).

Die Menge der übrigen Aktivitätsdiagramme bildet innere Knoten und Blätter eines Aktivitätsdiagrammbaums. Die Verknüpfung von Aktivitätsdiagrammen erfolgt in HUML durch den Eigenschaftswert HUMLsubAct in den Prozessbeschreibungen an Aktionszuständen (siehe Kapitel 1.5.3). Die Verknüpfung eines Aktionszustandes mit einem Aktivitätsdiagramm bedeutet, dass der Aktionszustand durch das Diagramm verfeinert wird.

Bei der Modellierung von Wiederholungen bzw. Schleifen muss beachtet werden, dass diese nicht durch Rückwärtskanten in dem Diagramm sondern durch die Eigenschaftswerte PArep und HUMLrepGeo (siehe Kapitel 1.5.3) modelliert werden. Soll eine Schleife modelliert werden, die mehrere Aktionszustände umfasst, muss dies über ein Subaktivitätsdiagramm erfolgen. Abbildung 1.16 zeigt zwei Aktivitätsdiagramme. Das rechte Diagramm verfeinert den Aktionszustand "Aktionszustand_1". Der Eigenschaftswert PArep an diesem Aktionszustand bedeutet, dass die Aktionszustände des Aktivitätsdiagramms "SubActDiagramm" dreimal wiederholt werden.

Desweiteren muss beachtet werden, dass die Übergangswahrscheinlichkeiten bei der Benutzung von Verzweigungen (*branch*) nicht an den Aktionszuständen annotiert werden, sondern als Kantenbeschriftung mit dem Eigenschaftswert HUMLtransProb angegeben werden müssen.

1.6. Übersetzung der UML-Stereotype nach HI-SLANG

Dieses Kapitel soll einen Überblick geben, wie die Annotationen und Eigenschaften in UML-Diagrammen nach HI-SLANG übersetzt werden können. Zu Beginn wird die Arbeitslast, die mit «PAopenload» oder «PAclosedload» modelliert wird, näher betrachtet. Desweiteren werden hier auch die Besonderheiten beschrieben, die bei diesen Stereotypen beachtet werden müssen. Im Abschnitt über die Aktionszusände wird die Übersetzung aller *tag-*

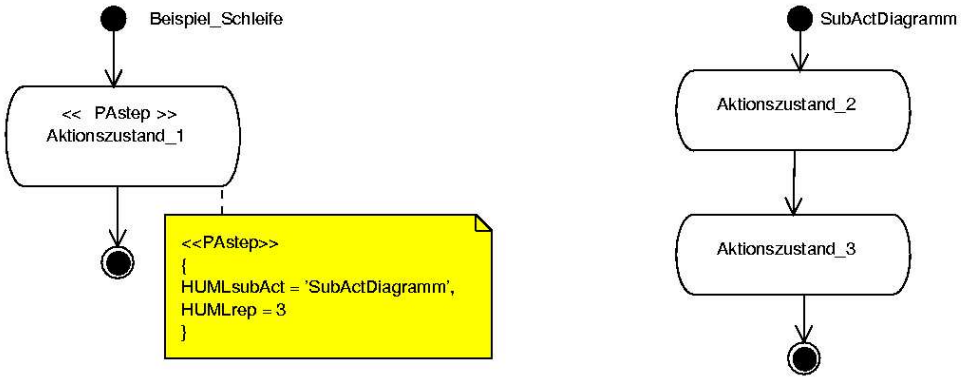

Abbildung 1.16.: Beispiel einer Modellierung einer Schleife, die mehrere Aktionszustände beinhaltet

ged valus von dem Stereotyp «PAstep» nach HIT erklärt. Gefolgt wird dieser Abschnitt von den Erläuterungen zur Maschinenbeschreibung, die anhand der Stereotype «PAhost» und «PAresource» vorgenommen wird. Dieser Teil bezieht sich speziell auf die Hardware-Modellierung. Im nachfolgenden Abschnitt wird die Messwertspezifikation erläutert, die durch das Stereotyp «HUMLresult» angegeben wird. Das Hauptaugenmerk liegt auf den zu ermittelnden Parametern und wie diese nach HI-SLANG übersetzt werden können. Desweiteren gehört zu diesem Abschnitt auch die Betrachtung der EXPERIMENTS, wie sie in HIT genannt werden. Hier wird angegeben unter welchen Bedingungen, die Ergebnisse ermittelt werden sollen. Im letzten Kapitel wird gezeigt, wie spezielle Strukturen in UML-Diagrammen, z.B. parallele Abläufe oder Schleifen, in die entsprechenden HITGRAPHIC-Modelle umgesetzt werden können. Dabei wird bewusst zur Veranschaulichung nicht der konkrete HI-SLANG-Text zur Übersetzung gezeigt, sondern die entsprechenden HITGRAPHIC-Modelle vorgestellt.

1.6.1. UML-Stereotype zur Arbeitslastbeschreibung

Für jedes UML-Projekt ist genau eine Arbeitslast zu spezifizieren. An dem obersten Aktivitätsdiagramm, das das Szenario auf allgemeinster Ebene beschreibt, befindet sich an der ersten Aktivität eine Notiz mit dem Stereotyp «PAopenLoad» bzw. «PAclosedLoad». Jedes Szenario wird in ein eigenständiges HIT-Modell übersetzt. Pro HIT-Modell existiert also genau eine Arbeitslast. Diese wird im ACTIVITY-Block der obersten HIT-Modell-Ebene beschrieben. Die ACTIVITY-Blocks der anderen Ebenen sind leer.

Handelt es sich bei der Arbeitslast um einen «PAopenload», so muss die Annotation ein PAoccurence aufweisen, das die Ankunftsrate der Prozesse spezifiziert. Die Ausprägungen der bei PAoccurence angegebenen Werte werden folgendermaßen in das CREATE-Statement von HI-SLANG eingetragen:

- *unbounded-string* und *periodic-string* werden mit EVERY v übersetzt, wobei v meist eine Verteilung ist, im Falle des periodic-string aber auch ein Integer sein kann.

- ein *irregular-string* gibt einen oder mehrere Zeitpunkte an, was in HI-SLANG mit AT notiert werden kann. Bei mehreren Zeitpunkten entstehen mehrere CREATE-Statements.

- Intervalle, in denen evtl. Prozesse gestartet werden, was mit *bounded-string* oder *bursty-string* ausgedrückt werden kann, sind für eine Leistungsanalyse zu ungenau und werden daher nicht unterstützt.

Bei einem «PAclosedLoad» muss statt PAoccurence PApopulation mit einem Integer x angegeben werden. Dies kann einfach mit CREATE x PROCESS nach HI-SLANG übersetzt werden. Ein geschlossenes System wird in HIT mit LOOP im Dienst der obersten Ebene realisiert. Ist ein PAextDelay d angegeben, wird dieser mit hold(d) ans Ende der Schleife gestellt.

Bei beiden Typen von System können die Eigenschaftswerte PArespTime und PApriority angegeben werden. Eine Angabe einer Priorität wird bei HUML nicht berücksichtigt, da nur eine Arbeitslast pro Modell angegeben werden soll. PArespTime wird von HUML auch nicht überprüft. Der Benutzer kann diesen Wert aber für seine eigenen Kontrollen angeben. Nach der erfolgreichen Leistungsbewertung ist es möglich aus den entsprechenden Resultaten der HUMLturnaroundtimes zu errechnen, ob die geforderte Antwortzeit des Benutzers eingehalten werden kann.

1.6.2. UML-Stereotyp zur Aktionszustandbeschreibung

Aktionszustände (*action states*) in Aktivitätsdiagrammen, die in die Leistungsbewertung einfließen sollen, werden mit «PAstep» stereotypisiert und durch die zugehörigen Eigenschaftswerte PAdemand, HUMLhost, PAextOp, HUMLsubAct, PAdelay, PArep, HUMLrepGeo und PAinterval näher beschrieben.

In einem HI-SLANG-Programm werden Aktionszustände durch Instanzen von selbstdefinierten Komponententypen repräsentiert. Damit die Aktionszustände klar voneinander unterschieden sind und einzeln bewertet werden können, entspricht jedem Aktionszustand genau ein Komponententyp mit einer einzigen Instanz.

Grundsätzlich muss zwischen zwei verschiedenen Arten von Aktionszuständen unterschieden werden:

- Ein einfacher Aktionszustand ist die kleinste Softwaremodellierungseinheit in einem UML-Modell. Er wird nicht durch ein weiteres Aktivitätsdiagramm erweitert. Ein Prozess in einem solchen Aktionszustand beansprucht mindestens eine Hardware-Ressource, von ihnen darf jedoch höchstens eine eine aktive Ressource sein. Wenn der Prozess durch eine aktive Ressource ausgeführt wird, besitzt das Stereotyp «PAstep» des Aktionszustandes die Eigenschaftswerte PAdemand und HUMLhost (siehe Beispiel Abb. 1.17), wenn er eine oder mehrere passive Ressourcen belegt, den Eigenschaftswert PAextOp. Weitere zulässige Eigenschaftswerte sind PAdelay, PArep, HUMLrepGeo und PAinterval.

- Ein Prozess in einem erweiterten Aktionszustand beansprucht nicht unmittelbar Hardware-Ressourcen, sondern durchläuft statt dessen innere Aktionszustände, die in einem Aktivitätsdiagramm dargestellt sind, durch das der Aktionszustand erweitert wird (Abb. 1.18 und 1.19). Das Stereotyp «PAstep» eines solchen erweiterten Aktionszustandes besitzt den Eigenschaftswert HUMLsubAct. Weitere zulässige Eigenschaftswerte sind ebenfalls PAdelay, PArep, HUMLrepGeo und PAinterval.

Es muss mindestens einer der Eigenschaftswerte PAdemand, PAextOp oder HUMLsubAct definiert werden. PAdemand und PAextOp können zusammen auftreten, andere Kombinatio-

nen der drei Eigenschaftswerte sind dagegen nicht erlaubt. Wenn PAdemand definiert wird, dann muss auch HUMLhost definiert werden.

Die optionalen Eigenschaftswerte PArep und HUMLrepGeo schließen sich gegenseitig aus. Der optionale Eigenschaftswert PAinterval darf nur in Verbindung mit PArep oder HUMLrepGeo verwendet werden.

Der optionale Eigenschaftswert PAdelay kann immer angegeben werden.

Die Übersetzung eines Aktionszustandes in HI-SLANG-Code besteht im wesentlichen aus der Definition eines Komponententyps.

Ein HI-SLANG-Programm besteht aus einem Modelldefinitionsteil und einem Experimentteil. Im Modelldefinitionsteil könnte die Hierarchie der Modellkomponenten von der Definition der obersten Modellebene bis herab zu den untersten Komponenten durch eine Schachtelung von Typdefinitionen dargestellt werden. Das ist aber nur möglich, wenn die Hierarchie, soweit sie die selbstdefinierten Komponententypen unter Ausschluss der Standardkomponentypen betrifft, baumförmig ist. Das ist aber bei Modellen, bei denen verschiedene Aktionszustände durch ein identisches Aktivitätsdiagramm erweitert werden, nicht gegeben. Statt dessen müssen bei dem Ansatz der PG 438 sämtliche Komponenten des Modells als gemeinsam genutzte Komponenten (*shared components*) instanziiert werden, die Definitionen ihrer Komponententypen sind dann nicht geschachtelt, sondern eine einfache Sequenz.

Jede Definition eines Komponententyps enthält einen Anweisungsblock ("Rumpf"). Bei einem Komponententyp, der einen einfachen Aktionszustand repräsentiert, besteht der Rumpf aus einem oder mehreren Aufrufen von benutzten Diensten, die mit den angebotenen Diensten von Instanzen von Standardkomponentypen verknüpft sind. Diese repräsentieren Hardware-Ressourcen (vgl. 1.5.4). Wiederholungen (definiert durch PArep oder HUMLrepGeo) werden in Schleifen übersetzt. Zusätzliche Verzögerungen (definiert durch PAdelay und/oder PAinterval) werden durch Aufrufe der HI-SLANG-Standardprozedur `hold()` realisiert.

Bei einem Komponententyp, der einen erweiterten Aktionszustand repräsentiert, spiegelt der Rumpf die Topologie des erweiternden Aktivitätsdiagramms wider. Es ist in der UML gestattet, zwischen Aktionszuständen desselben Aktionsstrangs beliebige Transitionen zu definieren. Dies eröffnet die Möglichkeit, dass ein Modellierer Softwareprogrammabläufe entwirft, für deren Darstellung sich die in modernen Programmiersprachen, und damit auch in HI-SLANG, üblichen Kontrollstrukturen (Sequenz, Verzweigung, Schleifenbildung) nur bedingt eignen. HI-SLANG bietet jedoch zwei besondere Befehle an, die die Übersetzung von Aktivitätsdiagrammen mit beliebiger Topologie erleichtern: `OPEN_CHAIN` und `CLOSED_CHAIN`.

Die Topologie eines Aktivitätsdiagramms kann normalerweise in einen `OPEN_CHAIN`-Block übersetzt werden. Nur wenn die Arbeitslast eine geschlossene ist, wird die Topologie des Wurzelaktivitätsdiagramms in einen `CLOSED_CHAIN`-Block übersetzt.

Eine einschränkende Besonderheit der HI-SLANG-Syntax macht es bei der Anwendung dieser Befehle jedoch notwendig, eine Unterscheidung zwischen zwei Arten von erweiterten Aktionszuständen zu machen. In `CHAIN`-Blöcken ist es ausschließlich zulässig, einfache Aufrufe benutzter Dienste unterzubringen. Schleifen und andere Kontrollstrukturen, selbst bloße Sequenzen von Anweisungen, sind verboten.

Wenn ein erweiterter Aktionszustand keinen anderen Eigenschaftswert als HUMLsubAct besitzt, ist dieser Umstand unproblematisch. Die Eigenschaftswerte PArep und HUMLrepGeo werden jedoch in Schleifen übersetzt. Es ist nicht möglich, eine Schleife den `OPEN_CHAIN`-Block umschließen zu lassen. Um die Schleife zu realisieren, muss der `OPEN_CHAIN`-Block in einen eigens definierten Komponententyp einer Unterkomponente gekapselt werden. Der un-

tergeordnete Komponententyp bietet dann einen Dienst an, der im Rumpf des übergeordneten Komponententyps innerhalb der Schleife aufgerufen wird.

Es folgen Schablonen für die Übersetzung der verschiedenen Arten von Aktionszuständen in HI-SLANG-Komponententypdefinition:

Übersetzung eines einfachen Aktionszustandes ohne HUMLsubAct:

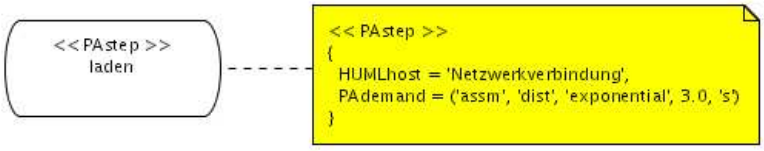

Abbildung 1.17.: Beispiel eines einfachen Aktionszustandes

```
TYPE <Aktionszustand>_class COMPONENT;

PROVIDE
  SERVICE <Aktionszustand>_service;
END PROVIDE;

TYPE <Aktionszustand>_service SERVICE;

  USE
    <Benutzte Dienste>
  END USE;

  [<Definition lokaler Variablen>]

  BEGIN
    <Rumpf>

END TYPE <Aktionszustand>_service;

<Deklaration der Hardwarekomponenten>

REFER
  <Aktionszustand>_service
TO
  <Hardwarekomponenten>
EQUATING
  <Dienstverknüpfungen>
END REFER;

END TYPE <Aktionszustand>_class;
```

<Aktionszustand>
Bezeichner für den Aktionszustand

<Benutzte Dienste>
Eine oder mehrere Deklarationen der benutzten Dienste von aktiven und/oder passiven Ressourcen:

- *Aktive Ressource:*
  `SERVICE` <Hardwarekomponente>`_used_service(`<Formale Parameter>`)`
- *Passive Ressourcen:*
  `SERVICE` <Hardwarekomponente>`_used_service(`<Formale Parameter>`)`
  `SERVICE` <Hardwarekomponente>`_semaphore_p;`
  `SERVICE` <Hardwarekomponente>`_semaphore_v;`

<Formale Parameter>

- *Beim Dienst einer Hardware-Ressource mit einer Warteschlangenrichtlinie ohne Berücksichtigung von Prioritäten:*
  `amount : REAL`
- *Beim Dienst einer Hardwareressource mit Prioritätswarteschlange:*
  `amount : REAL, prio : INTEGER`

<Deklaration der Hardwarekomponenten>
Eine oder mehrere Deklarationen von Komponenten, durch die aktive und/oder passive Ressourcen repräsentiert sind:

- *Aktive Ressource:*
  `ENCLOSE` <Hardwarekomponente> `:` <Hardwarekomponententyp>`;`
- *Passive Ressourcen:*
  `ENCLOSE` <Hardwarekomponente> `: server;`
  `ENCLOSE` <Hardwarekomponente>`_semaphore : semaphor;`

<Hardwarekomponente>
Bezeichner für eine Hardwarekomponente, mit der der Aktionszustand assoziiert ist

<Hardwarekomponententyp>
`server` *oder* `prioserver`

<Hardwarekomponenten>
Einer oder mehrere Komponentenbezeichner, durch Kommata getrennt:

- *Aktive Ressource:*
  <Hardwarekomponente>
- *Passive Ressourcen:*
  <Hardwarekomponente>`,` <Hardwarekomponente>`_semaphore`

<Dienstverknüpfungen>
Eine oder mehrere Verknüpfungen von benutzten mit angebotenen Diensten in der Form:

- *Aktive Ressource:*
  <Aktionszustand>.<Hardwarekomponente>`_used_service`
  `WITH` <Hardwarekomponente>`.request;`

- *Passive Ressourcen:*
  <Aktionszustand>.<Hardwarekomponente>`_used_service`
  `WITH` <Hardwarekomponente>`.request;`
  <Aktionszustand>.<Hardwarekomponente>`_semaphore_p`
  `WITH` <Hardwarekomponente>`_semaphore.p;`
  <Aktionszustand>.<Hardwarekomponente>`_semaphore_v`
  `WITH` <Hardwarekomponente>`_semaphore.v;`

<Definition lokaler Variablen>

- *Wenn das Stereotyp* «PAstep» *den Eigenschaftswert* PArep *enthält:*
  `VARIABLE i : INTEGER;`
- *Wenn der Aktionszustand mit mindestens einer passiven Ressourcen assoziiert ist, in deren Bedienungszeit die Anzahl der Zugriffe und die Zugriffszeit eingehen:*
  `VARIABLE j : INTEGER;`
- *Wenn beides der Fall ist:*
  `VARIABLE i : INTEGER;`
  `VARIABLE j : INTEGER;`

<Rumpf>
[<Anforderungen passiver Ressourcen>]
[`hold(`<PAdelay-Wert>`);`]
<Innerer Rumpf>
[<Freigaben passiver Ressourcen>]

<Anforderungen passiver Ressourcen>
Jede passive Ressource wird beim Betreten eines Aktionszustandes angefordert und durch ein Semaphor zugeteilt:
<Hardwareressource>`_semaphore_p;`
<Hardwareressource>`_semaphore_p;`
...

<Freigaben passiver Ressourcen>
Jede passive Ressource wird beim Verlassen eines Aktionszustandes wieder freigegeben:
<Hardwareressource>`_semaphore_v;`
<Hardwareressource>`_semaphore_v;`
...

<PAdelay-Wert>
Verzögerungszeit zwischen dem Betreten eines Aktionszustandes und dem Beginn der eigentlichen Aktion

<Innerer Rumpf>

- *Wenn das Stereotyp* «PAstep» *weder den Eigenschaftswert* HUMLrepGeo *noch den Eigenschaftswert* PArep *enthält:*
  <Hardwaredienstaufrufe>

- *Wenn das Stereotyp* «PAstep» *den Eigenschaftswert* HUMLrepGeo *enthält:*
  `AVERAGE` <HUMLrepGeo-Wert> `TIMES LOOP`
  [`hold(`<PAinterval-Wert>`);`]
  <Hardwaredienstaufrufe>
  `END LOOP;`

- *Wenn das Stereotyp* «PAstep» *den Eigenschaftswert* PArep *enthält:*
  `FOR i := 0 STEP 1 UNTIL` <PArep-Wert> `LOOP`
  [`hold(`<PAinterval-Wert>`);`]
  <Hardwaredienstaufrufe>
  `END LOOP;`

<HUMLrepGeo-Wert>
Mittlere Anzahl (genauer: Erwartungswert) von Wiederholungen der Hardwaredienstaufrufe in einer Schleife. Die Zahl der Schleifendurchläufe ist eine diskrete Zufallsvariable mit geometrischer Verteilung.

<PArep>-Wert
Genaue Anzahl von Wiederholungen der Hardwaredienstaufrufe in einer Schleife. Die Zahl der Schleifendurchläufe ist keine Zufallsvariable, es handelt sich um eine deterministische Schleife.

<PAinterval-Wert>
Verzögerungszeit innerhalb einer Wiederholungsschleife

<Hardwaredienstaufrufe>
Eine oder mehrere Dienstaufrufe in der Form:

- Aktive Ressource:
  <Hardwarekomponente>`_used_service(`<Bedienungszeit>[`,`<Priorität>]`);`

- *Passive Ressourcen, deren Normalbedienungszeit durch den Eigenschaftswert* PAextOp *explizit angegeben ist:*
  <Hardwarekomponente>`_used_service(`<Bedienungszeit>`);`

- *Passive Ressourcen, deren Bedienungszeit als Produkt von Zugriffsanzahl und Zugriffszeit ausgedrückt ist:*
  `FOR j := 0 STEP 1 UNTIL` <Zugriffsanzahl> `LOOP`
  <Hardwarekomponente>`_used_service(`<Zugriffszeit>`);`
  `END LOOP;`

<Bedienungszeit>
Prozessbedienungszeit, sofern der aufrufende Prozess der einzige ist, der die Ressource belegt. Bei einer aktiven Ressource wird der Wert aus dem Eigenschaftswert PAdemand *gewonnen, bei einer passiven Ressource aus der zweiten Stelle des* PAextOp-*Wertepaares, sofern diese ein* PAperfValue-*Wert ist (vgl. Kap. 1.3.2).*

<Zugriffsanzahl>
Anzahl der Zugriffe auf eine passive Ressource. Sie wird als Integer-*Wert an der zweiten Stelle des* PAextOp-*Wertepaares angegeben.*

<Zugriffszeit>
Zugriffszeit einer passiven Ressource, wie sie durch den Eigenschaftswert PAaxTime *des Stereotyps* «PAresource» *definiert wird*

<Priorität>
Integer-*Wert, der die Prioritätsstufe des Aktionszustandes angibt. Er kommt nur bei Hardwarekomponenten mit Prioritätswarteschlangen vor. Eine Möglichkeit, die Priorität als Eigenschaftswert anzugeben, ist z.Z. noch nicht vorgesehen, daher wird als fester Wert* 32767 *gewählt.*

Übersetzung eines erweiterten Aktionszustandes mit dem Eigenschaftswert HUMLsubAct und ohne einen weiteren Eigenschaftswert:

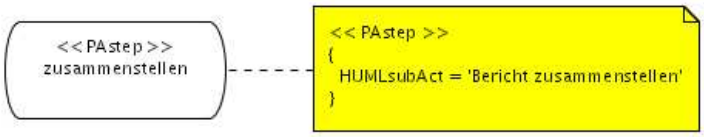

Abbildung 1.18.: Beispiel eines erweiterten Aktionszustandes ohne weitere Eigenschaftswerte

```
TYPE <Aktionszustand>_class COMPONENT;

  PROVIDE
    SERVICE <Aktionszustand>_service;
  END PROVIDE;

  TYPE <Aktionszustand>_service SERVICE;

    USE
      <Benutzte Dienste>
    END USE;

    BEGIN
      OPEN_CHAIN
        <OPEN_CHAIN-Block>
      END OPEN_CHAIN;

  END TYPE <Aktionszustand>_service;

  <Deklaration der Unteraktionskomponenten>

  REFER
    <Aktionszustand>_service
  TO
    <Benutzte Komponenten>
  EQUATING
    <Dienstverknüpfungen>
  END REFER;
```

```
END TYPE <Aktionszustand>_class;
```

<Aktionszustand>
Bezeichner für den Aktionszustand

<Unteraktionszustand>
Bezeichner für einen Unteraktionszustand, d.h. einen Aktionszustand im erweiternden Aktivitätsdiagramm

<Benutzte Dienste>
Liste der Form:

```
SERVICE <Unteraktionszustand>_used_service;
SERVICE <Unteraktionszustand>_used_service;
  ...
```

<OPEN_CHAIN-Block>
Menge von QNODE- *und* PROB-*Anweisungen, die die Topologie desjenigen Aktivitätsdigramms widerspiegelt, auf das der Eigenschaftswert* HUMLsubAct *referiert*

<Deklaration der Unteraktionskomponenten>
Eine oder mehrere Deklarationen in der Form:

```
ENCLOSE <Unteraktionszustand> : <Unteraktionszustand>_class;
```

<Benutzte Komponenten>
Einer oder mehrere Bezeichner von Unteraktionszuständen, durch Kommata getrennt

<Dienstverknüpfungen>
Eine oder mehrere Verknüpfungen von benutzten mit angebotenen Diensten in der Form:

```
<Aktionszustand>.<Unteraktionszustand>_used_service
    WITH <Unteraktionszustand>.<Unteraktionszustand>_service;
```

Übersetzung eines erweiterten Aktionszustandes mit dem Eigenschaftswert HUMLsubAct und mindestens einem anderen Eigenschaftswert:

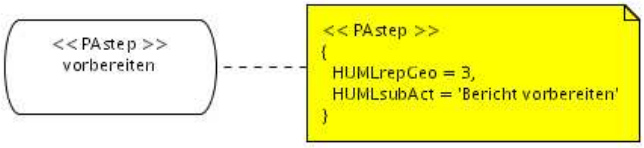

Abbildung 1.19.: Beispiel eines erweiterten Aktionszustandes mit weiteren Eigenschaftswerten

```
TYPE <Aktionszustand>_class COMPONENT;
  PROVIDE
    SERVICE <Aktionszustand>_service;
  END PROVIDE;
  TYPE <Aktionszustand>_sub_class COMPONENT;
```

```
    PROVIDE
      SERVICE <Aktionszustand>_sub_service;
    END PROVIDE;

    TYPE <Aktionszustand>_sub_service SERVICE;

      USE
        <Benutzte Dienste>
      END USE;

      BEGIN
        OPEN_CHAIN
          <OPEN_CHAIN-Block>
        END OPEN_CHAIN;

    END TYPE <Aktionszustand>_sub_service;

    <Deklaration der Unteraktionskomponenten>

    REFER
      <Aktionszustand>_sub_service
    TO
      <benutzte Komponenten>
    EQUATING
      <Dienstverknüpfungen>
    END REFER;

  END TYPE <Aktionszustand>_sub_class;

  TYPE <Aktionszustand>_service SERVICE;

    USE
      SERVICE <Aktionszustand>_sub_used_service;
    END USE;

    BEGIN
      <Rumpf>

  END TYPE <Aktionszustand>_service;

  COMPONENT <Aktionszustand>_sub : <Aktionszustand>_sub_class;

  REFER
    <Aktionszustand>_service
  TO
    <Aktionszustand>_sub
  EQUATING
    <Aktionszustand>_service.<Aktionszustand>_sub_used_service;
      WITH <Aktionszustand>_sub.<Aktionszustand>_sub_service;
  END REFER;

END TYPE <Aktionszustand>_class;
```

<Aktionszustand>
Bezeichner für den Aktionszustand

<Unteraktionszustand>
Bezeichner für einen Unteraktionszustand, d.h. einen Aktionszustand im erweiternden Aktivitätsdiagramm

<Benutzte Dienste>
Liste der Form:

```
SERVICE <Unteraktionszustand>_used_service;
SERVICE <Unteraktionszustand>_used_service;
  ...
```

<`OPEN_CHAIN`-Block>
Menge von `QNODE`*- und* `PROB`*-Anweisungen, die die Topologie desjenigen Aktivitätsdigramms widerspiegelt, auf das der Eigenschaftswert* HUMLsubAct *referiert*

<Deklaration der Unteraktionskomponenten>
Eine oder mehrere Deklarationen in der Form:

```
ENCLOSE <Unteraktionszustand> : <Unteraktionszustand>_class;
```

<Benutzte Komponenten>
Einer oder mehrere Bezeichner von Unteraktionszuständen, durch Kommata getrennt

<Dienstverknüpfungen>
Liste der Verknüpfungen von benutzten mit angebotenen Diensten in der Form:

```
<Aktionszustand>_sub.<Unteraktionszustand>_used_service
    WITH <Unteraktionszustand>.<Unteraktionszustand>_service;
```

<Rumpf>

- *Wenn das Stereotyp* «PAstep» *weder den Eigenschaftswert* PArep *noch den Eigenschaftswert* HUMLrepDet *enthält:*

```
hold(<PAdelay-Wert>);
<Aktionszustand>_sub_used_service;
```

- *Wenn das Stereotyp* «PAstep» *den Eigenschaftswert* HUMLrepGeo *enthält:*

```
[hold(<PAdelay-Wert>);]
AVERAGE <HUMLrepGeo-Wert> TIMES LOOP
  [hold(<PAinterval-Wert>);]
  <Aktionszustand>_sub_used_service;
END LOOP;
```

- *Wenn das Stereotyp* «PAstep» *den Eigenschaftswert* PArep *enthält:*

```
VARIABLE i : INTEGER;

BEGIN
[hold(<PAdelay-Wert>);]
FOR i := 0 STEP 1 UNTIL <PArep-Wert> LOOP
  [hold(<PAinterval-Wert>);]
  <Aktionszustand>_sub_used_service;
END LOOP;
```

<PAdelay-Wert>
Verzögerungszeit zwischen dem Betreten eines Aktionszustandes und dem Beginn der eigentlichen Aktion

<HUMLrepGeo-Wert>
Mittlere Anzahl (genauer: Erwartungswert) von Wiederholungen der Aufrufe von Diensten der Unteraktionskomponenten in einer Schleife. Die Zahl der Schleifendurchläufe ist eine diskrete Zufallsvariable mit geometrischer Verteilung.

<PArep-Wert>
Genaue Anzahl von Wiederholungen der Aufrufe von Diensten der Unteraktionskomponenten in einer Schleife. Die Zahl der Schleifendurchläufe ist keine Zufallsvariable, es handelt sich um eine deterministische Schleife.

<PAinterval-Wert>
Verzögerungszeit innerhalb einer Wiederholungsschleife

1.6.3. UML-Stereotyp zur Maschinenbeschreibung

Hardwareressourcen, die für die Leistungsbewertung eine Rolle spielen, müssen in UML-Modellen als Maschinenknoteninstanzen in Verteilungsdiagrammen dargestellt werden, die entweder durch «PAhost» (aktive Ressourcen) oder «PAresource» (passive Ressourcen) stereotypisiert sind und durch die jeweiligen Eigenschaftswerte beschrieben werden (vgl. Kap 1.5.4).

In einem HIT-Modell, welches dem Ansatz der PG 438 folgt, werden alle Hardwareressourcen als "gemeinsam genutzte Komponenten" (*shared components*) dargestellt. Eine gemeinsam genutzte Komponente wird in einem HI-SLANG-Programm auf der obersten Syntaxebene des '`TYPE` <Modellname> `MODEL;` ... `END TYPE` <Modellname>`;`' Blocks definiert.

Jede Hardwarekomponente wird zunächst durch genau eine Instanz eines Standardkomponententyps dargestellt. Im HUML-Projekt sind das, abhängig davon, ob Prozessprioritäten berücksichtigt werden oder nicht, `prioserver` oder `server`. Zusätzlich müssen bei passiven Ressourcen Standardkomponententypen instanziiert werden, die der Zugriffssteuerung dienen. Zur Zeit werden hierzu ausschließlich Semaphore verwendet. Der Grund dafür ist, dass bei einer Warteschlangenbildung von Prozessen an einem Bedienungszentrum die Warteschlangenrichtlinie eines Semaphors per Definition FIFO ist, wie es für die einzige Art von passiven Ressourcen gefordert ist, deren Umsetzung in eine HIT-Modellkomponente durch das HUML-Werkzeug zur Zeit angeboten wird (vgl. Kap. 1.5.4).

Von der Verwendung des Standardkomponententyps `counter` zum Zweck der Zugriffssteuerung wird abgesehen, weil in seinem Fall eine Warteschlange nach dem Zufallsprinzip abgearbeitet wird. Eine zukünftige Verwendung des Komponententyps `counter` in Verbindung mit anderen Arten von passiven Ressourcen ist jedoch angeraten, weil die Verwendung von randomisierten Warteschlangen die Leistungsbewertung durch numerische Lösungsverfahren zulässt, die FIFO-Warteschlangen der Semaphore dagegen nur die Simulation.

Aktive Ressourcen

Die Definition einer aktiven Ressource hat im HI-SLANG-Programm die folgende Form:

```
COMPONENT <Servername> : <Servertyp> (
  LET schedule := <Prozesswarteschlangenrichtlinie>,
```

```
  LET dispatch := <Prozessverteilungsrichtinie> (
    LET speed := <relativer Geschwindigkeitsfaktor>
  )
);
```

Der Wert des Parameters `speed` ist genau der Real-Wert des Eigenschaftswerts PArate. Die Eigenschaftswerte PAschdPolicy und PApreemptable werden wie in Tabelle 1.6 beschrieben umgesetzt:

PAschdPolicy	**PApreemptable**	**Servertyp**	`schedule`	`dispatch`
'ProcSharing'	-	server	immediate	shared
'FIFO'	-	server	fcfs	equal
'LIFO'	$false	server	lcfs	equal
'LIFO'	$true	server	lcfspr	equal
'HeadOfLine'	-	prioserver	prionp	equal
'PreemptResume'	-	prioserver	priopres	equal
'PrioProcSharing'	$false	prioserver	prionp	shared
'PrioProcSharing'	$true	prioserver	priopres	shared

Tabelle 1.6.: Die Eigenschaftswerte PAschdPolicy und PApreemptable

Passive Ressourcen

Die Definition einer passiven Ressource inklusive des dazugehörigen Semaphors sieht im HI-SLANG-Programm folgendermaßen aus:

```
COMPONENT <Servername>_semaphore : semaphor (
  LET sem_init := <Kapazität>
);

COMPONENT <Servername> : server (
  LET schedule := immediate,
  LET dispatch := sdequal (
    LET sdspeeds := [[1], [1/<Kapazität>]],
    LET speed := <Kapazität>
  )
);
```

Der Wert des Parameters `sem_init` ist genau der Integer-Wert des Eigenschaftswerts PAcapacity. Die Werte `immediate` für die Warteschlangenrichtlinie und `sdequal` für die Prozessverteilungsrichtlinie bedeuten, dass jedem bedienten Prozess ohne Verzögerung die volle Leistung des Bedienungszentrums zur Verfügung steht, sobald er die Zuweisung der Hardwareressource durch den Semaphor erhalten hat. An diesem bildet sich ggf. eine FIFO-Warteschlange, so dass an der eigentlichen Komponente keine weitere Warteschlange modelliert werden muss, was durch den Wert `immediate` dargestellt wird. Durch den Wert `sdequal` ist ausgedrückt, dass bei einer größeren Kapazität als 1 das Bedienungszentrum eine echte Parallelverarbeitung ohne Einbußen leisten kann, denn jeder Prozess wird mit derselben Geschwindigkeit bedient.

Der Wert `sdequal` bezeichnet im Gegensatz zu `equal` eine zustandsabhängige (*state dependent*) Prozessverteilungsrichtlinie, wobei der Zustand durch die Zahl der bedienten Prozesse

gegeben und die abhängige Größe die Bedienungsgeschwindigkeit ist. Dabei wird im Unterschied zu `equal` bei `sdequal` insbesondere zwischen Normgeschwindigkeit(en), einer Standardgeschwindigkeit und Effektivgeschwindigkeit(en) der Komponente unterschieden. Das Feld `sdspeeds` enthält eine zweispaltige Tabelle, durch die die Normgeschwindigkeiten in Abhängigkeit vom Komponentenzustand festlegt sind. Die Standardgeschwindigkeit ist durch den Parameter `speed` gegeben. Die zustandsabhängige effektive Geschwindigkeit ist das jeweilige Produkt beider (vgl. *HI-SLANG Reference Manual for the Hierarchical Evaluation Tool HIT* Büttner u. a. Anhang F.3.4).

Dadurch, dass in dem obigen Fall das Feld `sdspeeds` mit nur einem Wertepaar für die Zustandsmenge "1 Prozess oder mehr" belegt ist, ist die Normgeschwindigkeit zustandsunabhängig. Die Belegung der Standardgeschwindigkeit mit dem Wert der Kapazität der passiven Ressource und der Normgeschwindigkeit mit dem reziproken Wert führt zu einer konstanten Effektivgeschwindigkeit von 1. Die Komponente verhält sich also nicht anders, als ob man einfach `equal` als Prozessverteilungsrichtlinie angegeben hätte.

Der Sinn besteht jedoch darin, dass bei der Auswertung des Leistungsmodells durch HIT ein etwaiger UTILIZATION-Ergebniswert (vgl. Kap. 1.5.5) mittels einer Division durch den Wert der Standardgeschwindigkeit, der in diesem Fall der Kapazität entspricht, in bezug auf die Größe eben dieser Kapazität skaliert wird (vgl. dazu besonders Büttner u. a. Kap. 5.1.2). Diese Normierung ist notwendig, damit der UTILIZATION-Wert in ein sinnvolles Verhältnis zur Beschaffenheit der passiven Ressource gebracht wird. Ohne sie würden durch HIT Werte errechnet werden, die die Fähigkeit der Komponente zu einer echten Parallelverarbeitung einer Anzahl von Prozessen (genau das ist die Kapazität) unberücksichtigt ließen und darum um den entsprechenden Faktor zu groß wären. Dabei könnte es sogar zu verwirrenden Werten über 1 kommen, welche gemäß der Definition des "Ausnutzungsanteils" eigentlich gar nicht möglich sind.

1.6.4. UML-Stereotyp zur Messwertspezifikation

Das für die Leistungsanalyse zu definierende Experiment wird durch anpassbare Einstellungen in der HUML-Software verwirklicht. Hier kann dann zwischen simulativen und analytischen Verfahren mit entsprechenden Parameterangaben gewählt werden. Ohne entsprechende Abbruchbedingungen, würde das Experiment allerdings nicht stoppen. Das bedeutet, dass bei SIMULATIVE als Abbruchbedingung zusätzlich die CPUTIME und die MODELTIME angegeben werden muss (Standard jeweils: 10000). Bei LIN2 und MARKOV ist eine ACCURACY anzugeben (Standard bei LIN2: 1.0 und bei MARKOV: 0.1), bei MARKOV zusätzlich noch die CPUTIME (Standard: 5000). Für weitere Informationen, wie die einzelnen Bedingungen zu benutzen sind, sei hier auf [Büttner u. a.] verwiesen. Aus diesen zusätzlichen Parametern wird dann ein Experimentblock erzeugt.

Das Experiment wird in HI-SLANG folgendermaßen deklariert beschrieben:

```
EXPERIMENT <name>_exp METHOD <methode>
BEGIN
-----------------------------------------------
{evaluation statements}
-----------------------------------------------
END EXPERIMENT <name>_exp;
```

<name>
Name des Experimentes; der Name wird von HUML intern vergeben.

<method>
das zu wählende Lösungsverfahren; folgende Kombinationen sind möglich:
`ANALYTICAL "DOQ4"; | ANALYTICAL "LIN2"; | ANALYTICAL "MARKOV"; | SIMULATIVE;`

Um nun einzelne Komponenten der Software zu bewerten, müssen diese mit «HUMLresult» stereotypisiert sein. Da jeder Aktionszustand bei HUML zu einer selbstdefinierten Komponente in HIT wird, ist es möglich jede dieser Komponenten zu bewerten. Dazu wird jeder zu bewertenden Komponente eine Evaluation angehängt. Die Befehlszeile für eine Evaluation sieht folgendermaßen aus:

```
EVALUATE
    MODEL <model_name>:<model_type_name>;

    EVALUATIONOBJECT
       <ev_obj_name> VIA <Komponentenhierarchie> [,...]
       DEFAULT
       OUTPUT TABLE "TABLE" | DUMPFILE "DUMPFILE" ;
BEGIN
     MEASURE <stream> [,...] AT <ev_obj_name>
     DUE TO ALL
     ESTIMATOR
         MEAN
       [STANDARDDEVIATION],
       {bei Simulation}
       [CONFIDENCE LEVEL <HUMLcInterval>]
       ;
CONTROL
       {bei Simulation}
       AT <ev_obj_name>
         STOP MODELTIME <modeltime_wert>
         AT <ev_obj_name>
         STOP CPUTIME <cputime_wert>

       {bei MARKOV}
         STOP ACCURACY <accuracy_wert> OR CPUTIME <cputime_wert>

       {bei LIN2}
         STOP ACCURACY <accuracy_wert>
      ;
END EVALUATE;
```

<model_name>
Name der Evaluation, der intern vergeben wird.

<model_type_name>
Name des Modells, der intern vergeben wird.

<ev_obj_name>
Name des Evaluationsobjektes, der intern vergeben wird.

<Komponentenhierarchie>
Konkatenation von Komponenten, die sich auf dem Pfad von der Quelle der Last bis hin zur Komponente mit dem Evaluationsobjekt befinden. Der Konkatenationoperator ist ein ".".

<stream>
Hierbei handelt es sich konkret um THROUGHPUT, TURNAROUNDTIME, POPULATION, UTILIZATION und OCCUPATION. Je nachdem, ob im «HUMLresult»-Block die einzelnen Parameter ausgewählt werden, muss der entsprechende Wert hier eingetragen werden. Bei Simulation können keine Freiheitsgrade (DEGREE) angegeben zu werden, es wird automatisch DEGREE 10 von HIT verwendet.

<HUMLcInterval>
Der im «HUMLresult»-Block angegebene Prozentsatz für das Vertrauensintervall.

<modeltime_wert>
Der vom Benutzer angegebene MODELTIME-Wert.

<accuracy_wert>
Der vom Benutzer angegebene ACCURACY-Wert.

<cputime_wert>
Der vom Benutzer angegebene CPUTIME-Wert.

Werden mehrere Evaluationsobjekte definiert, so werden diese direkt hintereinander in HI-SLANG durch Kommata getrennt aufgeführt. Als Ausgabe wird ein DUMPFILE erzeugt. Das HI-SLANG Fragment DUE TO ALL bezieht sich auf die HIERARCHIES, die aufgrund der ausschließlichen Bewertung von UML-Aktivitäts- und Verteilungsdiagrammen nicht im einzelnen unterstützt werden können. Somit kann eine Leistungsanalyse nur mit der Einstellung DUE TO ALL durchgeführt werden.

Der Mittelwert MEAN ist standardmäßig ausgewählt. STANDARDDEVIATION und CONFIDENCE LEVEL sind optional bzw. nur bei Simulation zu verwenden.
Der CONTROL-Block wird nur bei den drei im HI-SLANG-Code-Beispiel angegebenen Verfahren erzeugt, bei DOQ4 gibt es ihn nicht.

1.6.5. Übersetzung von UML-Topologien nach HI-SLANG

Dieser Abschnitt beschreibt eine globale Sichtweise der Übersetzung von UML nach HI-SLANG. Dafür wird die Übersetzung von Topologien einfacher UML-Diagramme mit möglichst wenigen Annotationen gezeigt. In späteren Abschnitten wird die Übersetzung aller erlaubten Performanz-Annotationen aufgeführt. Da in den HITGRAPHIC-Modellen die Namen maximal 15 Zeichen lang sein dürfen, sind die eigentlichen Namen nach dieser Länge in den Modellen abgeschnitten.

Zunächst wird mit einem einfachen Aktivitätsdiagramm begonnen, das aus einem Startpunkt, einem Endpunkt und einer Aktivität besteht. Dieses Diagramm ist in Abbildung 1.20 zu sehen. Für den Anfang wird die Annotation außer Acht gelassen. Diese wird später miteinbezogen.

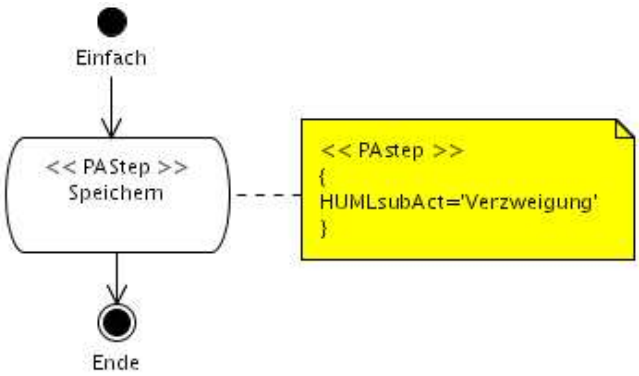

Abbildung 1.20.: einfaches Beispiel eines Aktivitätsdiagrammes

Das Aktivitätsdiagramm aus Abbildung 1.20 wird folgenderweise in ein HITGRAPHIC-Modell übersetzt. Aus der einen vorhandenen Aktivität wird in HIT eine selbstdefinierte Komponente. Im Beispiel wird der übergeordnete *service* "Einfach_service" mit dem angebotenen *service* "Speichern_service" der selbstdefinierten Komponente verbunden. Dieses HITGRAPHIC-Modell ist in Abbildung 1.21 zu sehen.

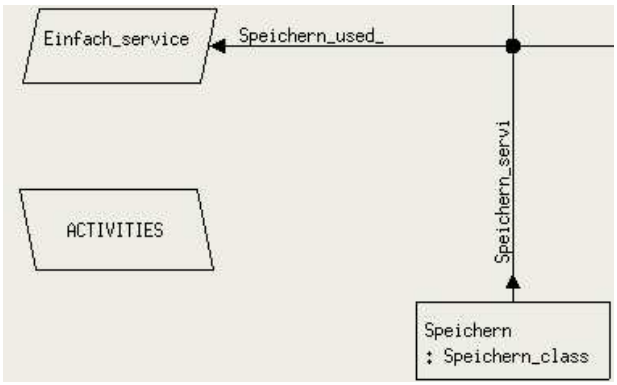

Abbildung 1.21.: HITGRAPHIC-Modell, das aus dem Aktivitätsdiagramm in Abbildung 1.20 gewonnen wird

Der Aufruf des angebotenen *services* einer selbstdefinierten Komponente erfolgt in dem *body* der übergeordneten Komponente, die diesen *service* benutzt. Eine Topologie von aufeinanderfolgenden Aktionen wird in HI-SLANG in einem OPEN_CHAIN-Block oder einem CLOSED_CHAIN-Block angegeben. Näheres zu der jeweiligen Verwendung dieser beiden Anweisungen ist auch im Kapitel 1.6.2 nachzulesen. Für das weitere Vorgehen sei hier nur die OPEN_CHAIN-Anweisung genutzt. Die Aktionen werden als QNODEs definiert. Der Aufbau eines OPEN_CHAIN-Blocks folgt folgender Struktur:

```
open_chain_statement ::=
         OPEN_CHAIN [ arrival-prob_part ]
      qnode [ ...]
         END OPEN_CHAIN;

qnode ::=
```

```
        QNODE qnode-identifier [prob_part]

prob_part ::=
        {       PROB simple_real_expression     : qnode-identifier;} [ ..]
        [       ELSE                            : qnode-identifier;]
```

Im Beispiel aus Abb. 1.20 gibt es nur eine einzige Aktion ("Speichern"). Im *body* der Komponente "Einfach_service" steht deshalb nur ein QNODE ("Speichern_used_service"), der mit Wahrscheinlichkeit 1 aufgerufen wird. Wenn der OPEN_CHAIN-Block nicht mit einer expliziten PROB-Anweisung beginnt, wird automatisch der erste QNODE abgearbeitet. Der *body* des services "Einfach_serivce" ist im Folgenden dargestellt.

```
OPEN_CHAIN
 QNODE Speichern_used_service
END OPEN_CHAIN;
```

Jetzt wird zusätzlich die Annotation aus Abbildung 1.20 miteinbezogen. Sie gibt an, dass die Aktivität "Speichern" durch ein weiteres Sub-Aktivitätsdiagramm verfeinert wird. Dieses Sub-Aktivitätsdiagramm ist in Abbildung 1.22 zu sehen.

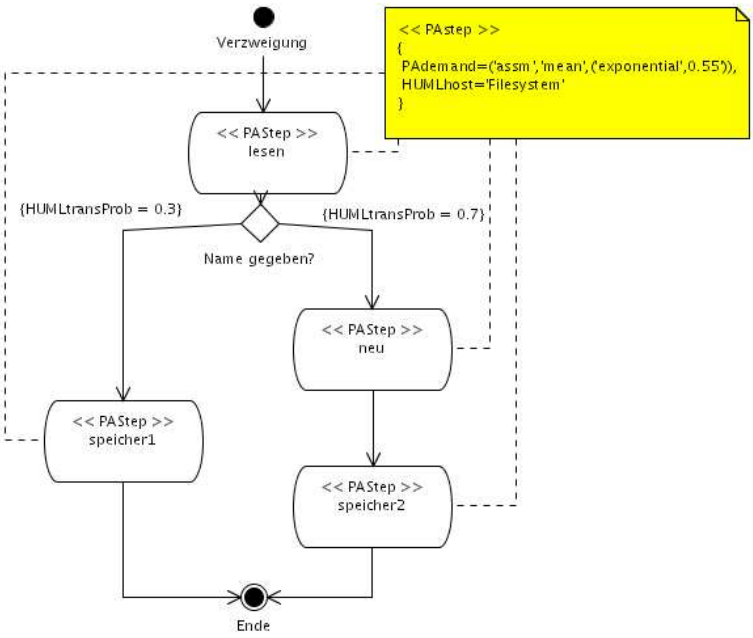

Abbildung 1.22.: Das Sub-Aktivitätsdiagramm zu Abbildung 1.20

Das Sub-Aktivitätsdiagramm wird nun in das HIT-Modell aus Abbildung 1.21 eingefügt. Dabei dient es zur Definition der selbstdefinierten Komponente "Speichern_class". In Abbildung 1.22 ist weiterhin zu sehen, dass eine Verzweigung vorhanden ist, und nicht mehr nur einfache sequentiell ablaufende Aktivitäten betrachtet werden.

Jede der Aktivitäten in Abbildung 1.22 wird wieder zu einer selbstdefinierten Komponente in HIT. Damit ergeben sich vier weitere Komponenten. Sie beinhalten jeweils nur einen angebotenen Dienst und greifen alle auf dieselbe Maschine zu. Dies ist im Aktivitätsdiagramm

durch die Angabe des *tagged values* HUMLhost an den Aktivitäten zu erkennen. Diese einfache Zuordnung von Dienst und Maschine in HIT ist in Abbildung 1.23 zu sehen.

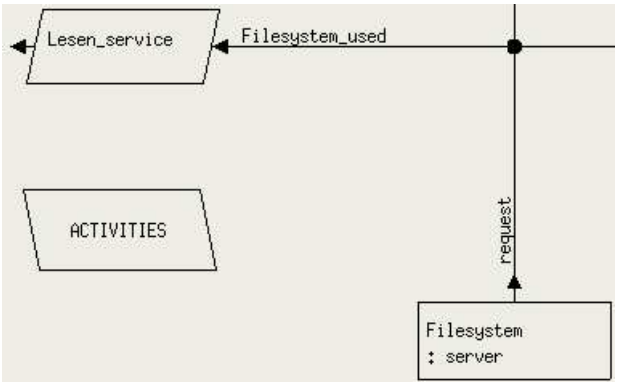

Abbildung 1.23.: Die Komponente "Lesen_class" aus Abbildung 1.24

Die selbstdefinierte Komponente "Speichern_class" besteht also aus weiteren vier selbstdefinierten Komponenten. Diese vier Komponenten werden mit einem angebotenem *service* "Speichern_service" verknüpft. Dieses Modell ist in Abbildung 1.24 zu sehen.

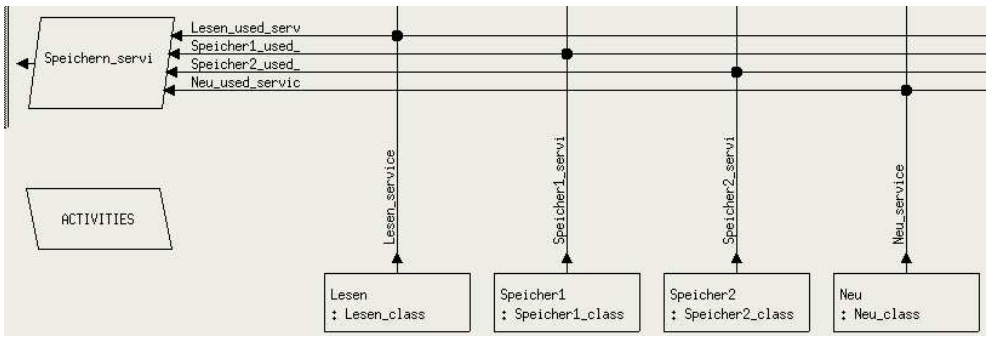

Abbildung 1.24.: HITGRAPHIC-Modell zum Sub-Aktivitätsdiagramm aus Abbildung 1.22

Zusätzlich muss allerdings noch der Verzweigungsknoten aus dem Aktivitätsdiagramm modelliert werden. Dies geschieht im *body* des *services* "Speichern_service". Auch hier wird die OPEN_CHAIN-Anweisung aus HI-SLANG genutzt. Zunächst wird der erste QNODE "Lesen_used_service" ausgeführt. Nachfolgend wird mit einer Wahrscheinlichkeit von 30% "Speicher1_used_service" aufgerufen. Zu 70% wird erst der *used service* "Neu_used_service" und dann "Speicher2_used_service" ausgeführt. Wie im folgenden Quelltext des *body* des *services* "Speichern_service" zu sehen, werden dabei wieder QNODE-Aufrufe verwendet. Es ist zu beachten, dass die Wahrscheinlichkeiten der Transitionen im Entscheidungsfall direkt in dem Namen der Kanten durch z.B. {HUMLtransProb=0.3} angegeben werden müssen und nicht an den folgenden Aktivitäten.

```
OPEN_CHAIN
 QNODE Lesen_used_service
   PROB 0.3: Speicher1_used_service;
   ELSE Neu_used_service
```

```
 QNODE Speicher1_used_service
 QNODE Neu_used_service
   PROB 1: Speicher2_used_service;
END OPEN_CHAIN;
```

Damit sind die beiden voneinander abhängigen Aktivitätsdiagramme in ein HITGRAPHIC-Modell übersetzt. Hierbei wurden schon teilweise spezielle HUML-Annotationen verwendet, die so nicht im *UML-RT* [OMG (2003)] definiert sind, für dieses Beispiel aber unabdingbar sind. Um wirklich eine Leistungbewertung durchführen zu können, fehlen noch die Parameter der Last. Die Modellierung der Last in UML-Modellen und ihre Übersetzung in HI-SLANG wird in den Kapiteln 1.5.3 und 1.6.2 beschrieben.

Nach dieser ersten Übersetzung von UML nach HI-SLANG bzw. HITGRAPHIC soll nun ein weiteres Konstrukt aus UML betrachtet werden. Im Folgenden wird gezeigt, wie parallel ablaufende Aktivitäten in HIT modelliert werden können. Dazu ist zunächst in Abbildung 1.25 ein entsprechendes Aktivitätsdiagramm dargestellt. Hier werden die Aktivitäten "lesen1" und "schreiben" parallel zu "eingabe" durchgeführt.

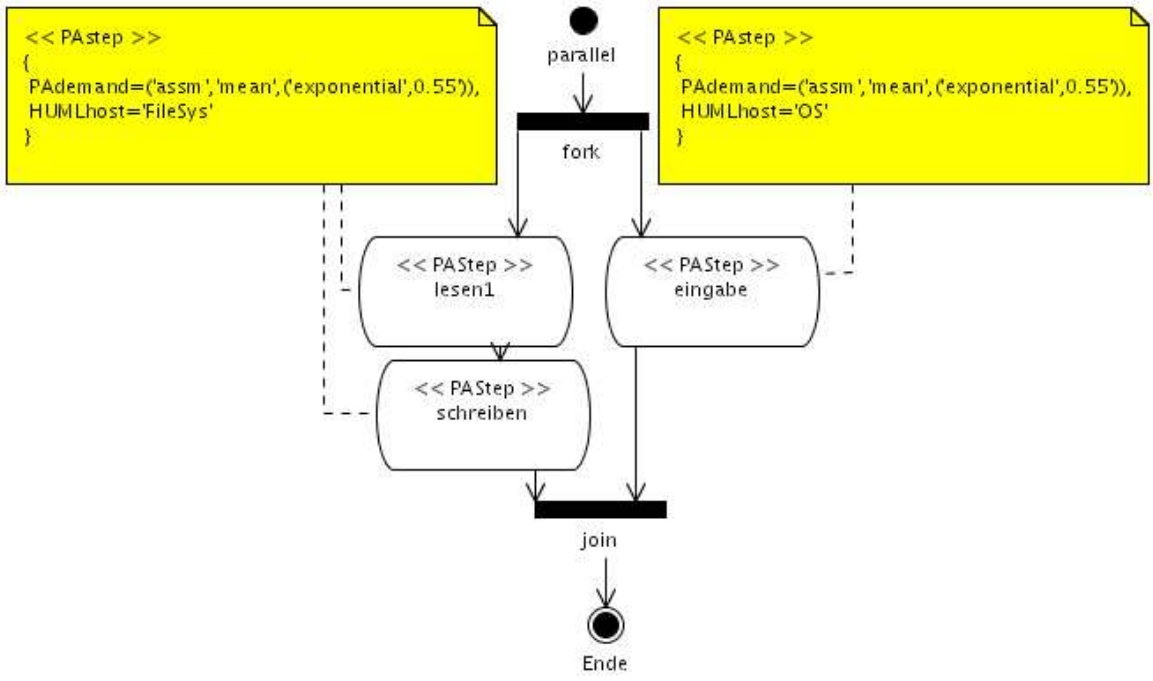

Abbildung 1.25.: Ein Aktivitätsdiagramm mit parallel ablaufenden Aktivitäten

In HIT werden für die zwei parallelen Handlungsstränge zwei Pfade als selbstdefinierte Komponenten angelegt. Die *bodies* der angebotenen Dienste der selbstdefinierten Komponenten bestehen wieder aus OPEN_CHAIN-Anweisungen, die die Struktur des jeweiligen Handlungsstranges modellieren. Auf der untersten Ebene besteht die Zuordnung eines Dienstes zu einer Maschine, ähnlich wie bereits in Abbildung 1.23 gezeigt wurde. Das entsprechende HIT-Modell ist in Abbildung 1.26 zu sehen.

Der eigentliche parallele Ablauf wird im *body* des *services* "Parallel_service" realisiert. Hier wird das HI-SLANG-Konstrukt CONCURRENT eingesetzt. Dadurch können in HI-SLANG definierte Anweisungen parallel ausgeführt werden. So ist es möglich, die beiden *used services*

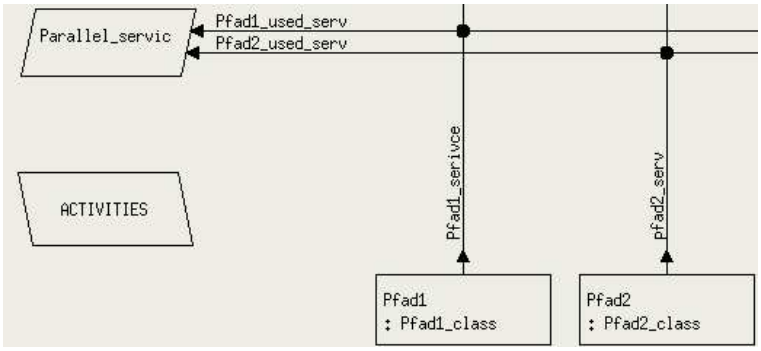

Abbildung 1.26.: übersetzes HITGRAPHIC-Modell aus Abbildung 1.25

"Pfad1_used_service" und "Pfad2_used_service" parallel ablaufen zu lassen. Dabei wird zur Übersetzung keine OPEN_CHAIN-Anweisung verwendet.

In diesem Beispiel werden die beiden selbstdefinierten Komponenten "Pfad1" und "Pfad2" also gleichzeitig aufgerufen und ausgeführt. Der *body* des *services* "Parallel_service" ist im folgenden HI-SLANG-Fragment zu sehen.

```
CONCURRENT
  Pfad1_used_service;
 TO
  Pfad2_used_service;
END CONCURRENT;
```

Die selbstdefinierte Komponente "Pfad1_class" enthält die Struktur des ersten Handlungsstranges der parallel ablaufenden Prozesse. In unserem Beispiel werden also die Aktivitäten "lesen1" und "schreiben" ausgeführt. Das HITGRAPHIC-Modell dazu ist in Abbildung 1.27 zu sehen.

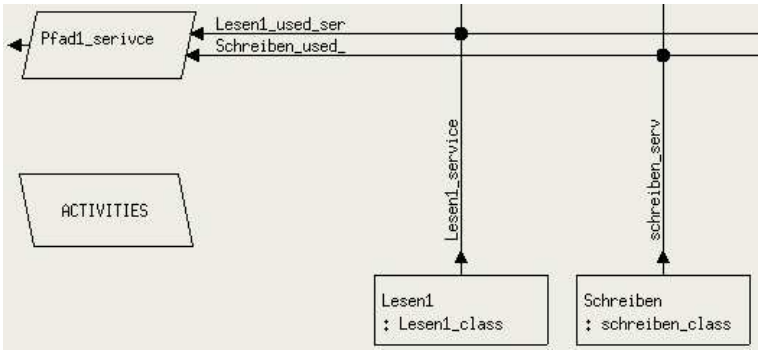

Abbildung 1.27.: HITGRAPHIC-Modell des ersten Handlungstranges aus Abbildung 1.25

Im *body* des *services* "Pfad1_service" steht die OPEN_CHAIN-Anweisung, um die Struktur des Ablaufes des ersten Stranges zu modellieren. Zuerst wird der *used service* "Lesen1_used_service" und danach der *used service* "Schreiben_used_service" ausgeführt. Dies ist im folgenden Quelltext zu sehen.

```
OPEN_CHAIN
```

```
 QNODE Lesen1_used_service
   PROB 1: Schreiben_used_service;
 QNODE Schreiben_used_service
END OPEN_CHAIN;
```

Die Übersetzung des 2. Pfades folgt daraus analog.

Im Folgenden wird die Übersetzung von Schleifen in UML-Aktivitätsdiagrammen nach HI-SLANG betrachtet. Dabei muss zwischen zwei Typen unterschieden werden. Zum einen gibt es Schleifen, die deterministisch durchlaufen werden. Dies sind Schleifen, bei denen die Anzahl der Durchläufe festgelegt ist. Zum Anderen gibt es Schleifen, deren Anzahl an Durchläufen probabilistisch ist.

Zunächst wird die erste Variante von Schleifen betrachtet. Um die beiden Arten von Schleifen unterscheiden zu können, wird bei HUML ein zusätzliches Sprachmittel eingeführt. Wird zur Schleifenmodellierung "PArep" genutzt, wird die Schleife in HIT deterministisch modelliert. Soll eine probabilistische Schleife in HIT eingesetzt werden, muss stattdessen "HUMLrepGeo" verwendet werden. Die Anzahl der Schleifendurchläufe wird hinter dem jeweiligen Schlüsselwort angegeben. Bei beiden Schleifenarten werden die Werte als Integer angegeben. Falls die Schleife aus mehreren Aktivitäten besteht, müssen diese in einem Sub-Aktivitätsdiagramm dargestellt werden, wie in dem ersten Beispiel gesehen. Die Struktur des Schleifenablaufes muss also in diesem Sub-Aktivitätsdiagramm explizit modelliert werden. In welchem Sub-Aktivitätsdiagramm die Verfeinerung stattfindet, wird durch die HUMLsubAct-Anweisung spezifiziert.

Ein Beispiel für eine deterministische Schleife, bei der die Aktivität "zaehle1" dreimal wiederholt wird, ist in Abbildung 1.28 zu sehen. Da nur eine einzelne Aktivität wiederholt wird, wird die Schleife nicht in einem Sub-Aktivitätsdiagramm verfeinert.

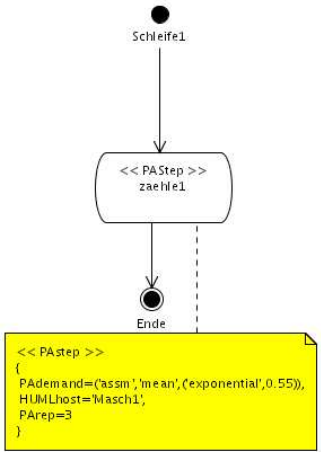

Abbildung 1.28.: Ein Aktivitätsdiagramm mit einer deterministischen Schleife

In dem Ansatz der PG 438 werden in HITGRAPHIC dazu mehrere Ebenen von Modellen genutzt. Auf der obersten Ebene steht ein HITGRAPHIC-Modell mit einer OPEN_CHAIN-Anweisung, die eine selbstdefinierte Komponente aufruft. Dieses erste Modell ist in Abbildung 1.29 und der zugehörige *body* des *services* "Schleife1_service" ist im Anschluss zu sehen.

```
OPEN_CHAIN
```

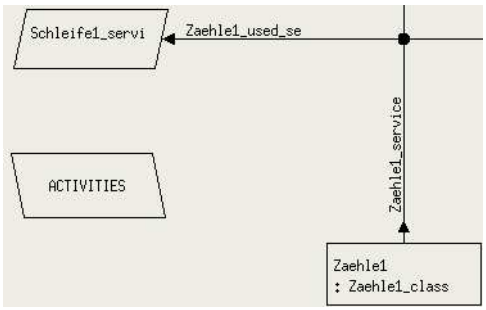

Abbildung 1.29.: HITGRAPHIC-Modell zum Aktivitätsdiagramm in Abbildung 1.28

```
 QNODE Zaehle1_used_service
END OPEN_CHAIN;
```

Die selbstdefinierte Komponente "Zaehle1_class" bildet die zweite Ebene. Auf dieser Ebene wird die eigentliche Schleifendurchführung modelliert. Hier wird der *service* "Zaehle1_service" direkt mit einer Maschine verbunden. Würde die Schleife mehrere Aktivitäten beinhalten, müsste eine weitere selbstdefinierte Komponente eingefügt werden. Das Modell ist in Abbildung 1.30 zu sehen.

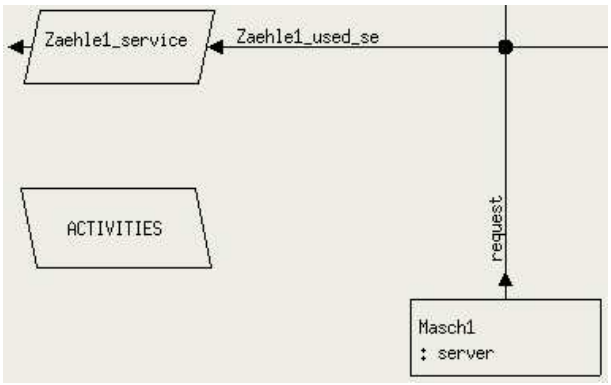

Abbildung 1.30.: HITGRAPHIC-Modell der selbstdefinierten Komponente "Zaehle1_class" aus Abbildung 1.29

Interessant an dieser Ebene ist der *body* des *services* "Zaehle1_service". Dort wird der eigentliche Schleifenaufruf durchgeführt. Für eine deterministische Schleife wird die FOR-Anweisung von HI-SLANG benutzt. Es wird dreimal der angebotene Dienst der Maschine aufgerufen. Dies sieht in HI-SLANG wie folgt aus.

```
FOR i := 0 STEP +1 UNTIL 3
LOOP
  Zaehle1_used_service;
END LOOP;
```

Abschließend werden die probabilistischen Schleifen behandelt. Bei diesen Schleifen ist die Anzahl der Durchläufe probabilistisch. Dies wird durch den Eigenschaftswert HUMLrepGeo angegeben. Da die Schleife mehrere Aktivitäten umfasst, wird das Schlüsselwort HUMLsubAct

mit dem Namen des Sub-Aktivitätsdiagramms angegeben. An dieser Stelle soll das Sub-Aktivitätsdiagramm aber nicht weiter beachtet werden. Es tritt lediglich in der selbstdefinierten Komponente "Zaehle2_sub_class", anstatt der Maschine im Beispiel der deterministischen Schleife, im HITGRAPHIC-Modell auf. Das Aktivitätsdiagramm, das die eigentliche Schleife modelliert, ist in Abbildung 1.31 zu sehen.

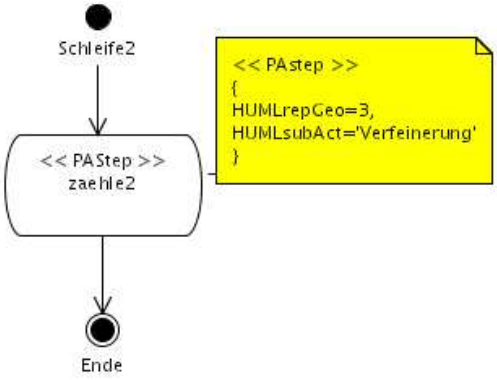

Abbildung 1.31.: Ein Aktivitätsdiagramm mit einer probabilistischen Schleife

In HITGRAPHIC sieht das entsprechende Modell genauso aus wie das der deterministischen Schleifenart. Allerdings wird anstelle des FOR-Konstruktes eine AVERAGE-Anweisung benutzt, die den Mittelwert der geometrisch verteilten Anzahl der Schleifendurchläufe angibt. Der *body* eines solchen *services* ist nachfolgend gezeigt:

```
AVERAGE 3 TIMES
LOOP
  Zaehle2_sub_used_service;
END LOOP;
```

1.7. Zusammenfassung: Stereotype und Eigenschaftswerte

Im Folgenden werden alle im Rahmen des HUML-Projektes verwendeten Stereotype und Eigenschaftswerte zusammenfassend in der Tabelle 1.7 aufgelistet.

Stereotyp	**Tag**	**Typ**
«PAopenLoad»	PAoccurence	RTarrivalPattern
«PAclosedLoad»	PApopulation	Integer
	PAextDelay	PAperfValue
«PAstep»	PAdemand	PAperfValue
	PArep	Integer
	PAinterval	PAperfValue
	PAdelay	PAperfValue
	HUMLrepGeo	Integer
	HUMLsubAct	String
	HUMLhost	String
	PAextOp	String, (Integer | <time-value>)
«PAhost»	PArate	Real
	PAschdPolicy	Enumeration
	PApreemptable	Boolean
«PAresource»	PAaxTime	PAperfValue
	PAschdPolicy	Enumeration
	PAcapacity	Integer
«HUMLresult»	HUMLthroughput	–
	HUMLutilization	–
	HUMLturnaroundtime	–
	HUMLpopulation	–
	HUMLoccupation	–
	HUMLstandarddeviation	–
	HUMLcInterval	Integer

Tabelle 1.7.: Zusammenfasssung der Stereotype und Eigenschaftswerte

1.8. Anwendungsbeispiel: Bürosystem

Dieser Abschnitt soll zeigen, wie es möglich ist, von einer Problemstellung über UML-Diagramme zu einer korrekten Eingabe für HUML zu gelangen. Zur Darstellung dieser Problemstellung wurde ein Anwendungbeispiel für die Leistungsmodellierung aus dem Modellierungspraktikum, das im WS 03/04 stattfand, entnommen. Im Beispiel wird die Übersetzung, die HUML vornimmt, durch HITGRAPHIC-Modelle veranschaulicht. Im Folgenden wird zunächst die Aufgabenstellung beschrieben und anschließend werden die UML-Diagramme, die eine Lösung der Aufgabenstellung modellieren. Im Rahmen des HUML-Projekts werden als UML-Diagramme ausschließlich Aktivitätsdiagramme und Verteilungsdiagramme benutzt. Danach werden diese UML-Diagramme um Performanz-Werte ergänzt, und es wird gezeigt, wie diese erweiterten UML-Diagramme in korrespondierende HITGRAPHIC-Darstellungen übersetzt werden können. Da zum Zeitpunkt der Erstellung der HITGRAPHIC-Modelle die Abbildungskonventionen zur Übersetzung der Namen aus UML-Diagrammen in HITGRAPHIC-Namen nicht feststand, können Namensdifferenzen auftreten.

Aufgabenstellung Bürosystem Es existiere eine Firma, die aus 20 Abteilungen besteht. Jede Abteilung liefert pro Tag einen Bericht ab. Für die Erstellung der Berichte ist je ein Sekretär zuständig. Ein Sekretär erarbeitet seinen Bericht, indem er einzelne Dokumente vorbereitet und sie danach zusammenstellt. Ein Bericht besteht aus drei separaten Dokumenten, daher besteht der Arbeitsablauf der Berichterstellung aus drei Vorbereitungen und einer Zusammenstellung. Technisch gesehen bestehen alle Arbeitsgänge aus derselben Tätigkeit, nämlich dem rechnergestützten Editieren von Texten. Zunächst lädt der Sekretär einen der zu verarbeitenden Texte auf seinen lokalen Rechner, der über die Netzwerkverbindung "Netzwerkverbindung" mit dem Firmennetzwerk verbunden ist. Danach bearbeitet er den Text und speichert ihn ab. Die Bearbeitung des Texts erfolgt dreimal hintereinander zur Vorbereitung der Einzeldokumente und einmal abschließend bei der Zusammenstellung des Berichts; das Speichern erfolgt einmal. Für die Überprüfung und für das Abschließen der Berichte ist eine eigene Dienststelle zuständig, in der ein Sachbearbeiter die Berichte sichtet und eine Auswahl von ihnen archiviert. Für diese Arbeiten verwendet er einen lokalen Rechner. Auch dieser ist mit dem Firmennetzwerk über "Netzwerkverbindung" verbunden.

In einem Anwendungsfalldiagramm wird die Aufgabenverteilung des Firmenpersonals festgehalten. Die Anwendungsfälle werden anschließend durch Aktivitätsdiagramme verfeinert.

Anwendungsfalldiagramm für das "Bürosystem" Das Diagramm in Abbildung 1.32 veranschaulicht die Akteure Sachbearbeiter und Sekretär, die jeweils mit den Aktionen "Bericht überprüfen", "Bericht beenden", "Bericht vorbereiten" und "Bericht zusammmenstellen" verbunden sind.

Da die Erweiterung eines Anwendungsfalldiagramms um Leistungsinformationen aus Sicht der PG438 für die *Software Performance Engineering* nicht geeignet erscheint, wird dieses Diagramm im Weiteren nicht mehr betrachtet und durch ein spezielles Aktivitätsdiagramm ersetzt.

Das neue Diagramm enthält alle Anwendungsfälle des Anwendungsfalldiagramms in Form von Aktionszuständen, siehe Abbildung 1.33. Dieses Diagramm ist damit so etwas wie die Wurzel des Gesamtsystems. Es ersetzt das Anwendungsfalldiagramm und modelliert zudem die für die Leistungsbewertung wichtige Struktur der Anwendungsfälle. Die übrigen Aktivitätsdiagramme verfeinern je einen Aktionszustand des neuen Aktivitätsdiagramms. In HUML

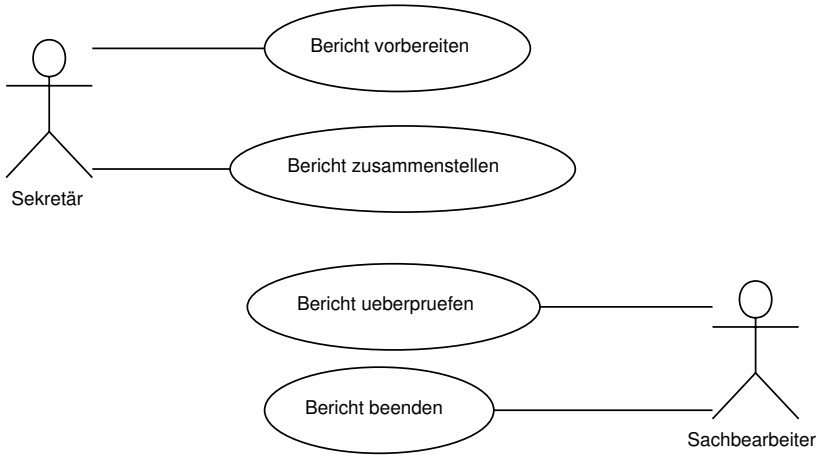

Abbildung 1.32.: Das Anwendungsfalldiagramm für das Bürosystem

werden also für die Leistungserweiterung ausschließlich Aktivitätsdiagramme und Verteilungsdiagramme benutzt.

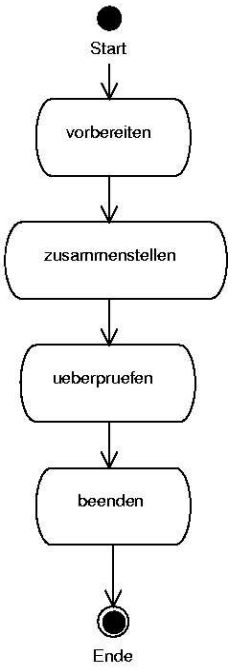

Abbildung 1.33.: Das Aktivitätsdiagramm für das Bürosystem

Jeder ursprüngliche Anwendungsfall muss nun durch ein Aktivitätsdiagramm verfeinert werden.

Aktivitätsdiagramme für das "Bürosystem" Um den "Bericht zusammenstellen" zu können, nutzt der Sekretär dieselben technischen Mittel, wie beim Vorbereiten des Berichts. Da das Vorbereiten und das Zusammenstellen im Sinne von HIT auf Maschinenebene identische Prozesse sind, erfolgen damit auch dieselben Aktionszustände. Abbildungen 1.34 und 1.35 stellen die genauen Abläufe der Anwendungsfälle "Bericht vorbereiten" und "Bericht zusammenstellen" dar, die jeder Sekretär zu erledigen hat. Diese Aktivitätsdiagramme verdeutlichen die zuvor im Anwendungsfalldiagramm nicht erkennbaren Prozesse "laden", "ändern" und "speichern", die beim Vorbereiten und Zusammenstellen der Berichte sequentiell ablaufen.

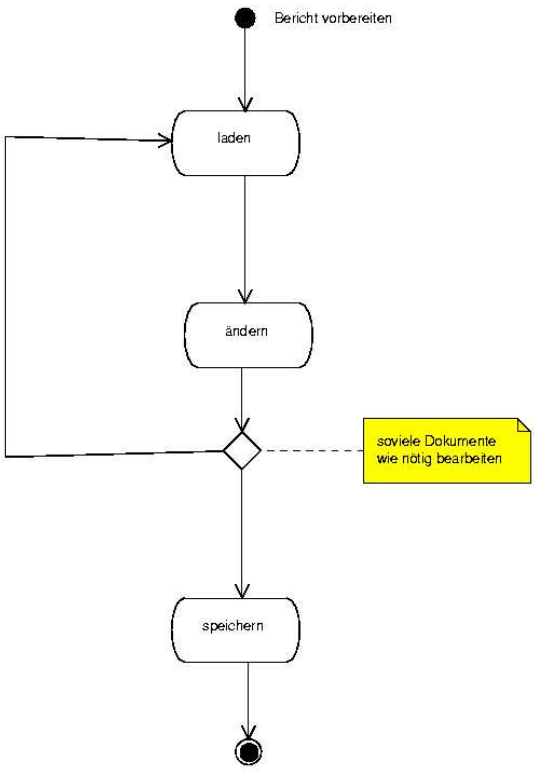

Abbildung 1.34.: Das Aktivitätsdiagramm für das Vorbereiten eines Berichts

Der Sachbearbeiter muss indes die Überprüfung der Dokumente vornehmen. Der genaue Prozessablauf, bestehend aus den Aufträgen "laden" und "auswählen", wird in Abbildung 1.36 veranschaulicht.

Schließlich muss zum Archivieren des endgültigen Berichts das Speichern vorgenommen werden. Der genaue Ablauf des Prozesses "Bericht beenden" wird in Abbildung 1.37 wiedergegeben.

Folgerungen Durch Aktivitätsdiagramme konnten die Funktionen der einzelnen Personalmitglieder veranschaulicht werden. Trotz der Verfeinerung der Anwendungsfälle mittels Aktivitätsdiagrammen fehlen Leistungsinformationen, deren Ergänzung durch Erweiterungen der Modellierungssprache UML erfolgt.

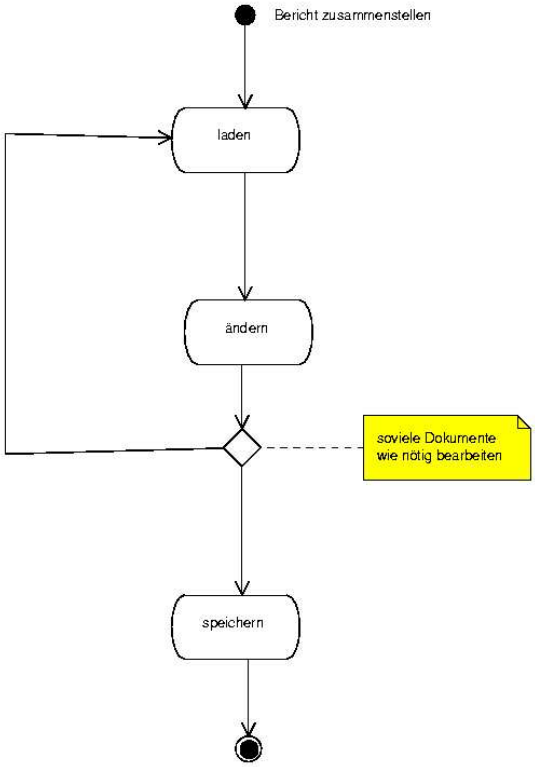

Abbildung 1.35.: Das Aktivitätsdiagramm für das Zusammenstellen eines Berichts

Im folgenden Abschnitt werden die UML-Diagramme des Bürosystems durch Leistungsparameter erweitert. Desweiteren müssen Maschinenknoten, die bei der Durchführung der einzelnen Funktionen benutzt werden, dargestellt werden. Durch den Einsatz eines Verteilungsdiagramms können die benutzten Hardwareknoten abgebildet werden.

Anwendungsbeispiel "Bürosystem" mit Erweiterung um Leistungsdaten Dieser Abschnitt beschreibt das um Performanzeigenschaften erweiterte Anwendungsbeispiel "Bürosystem". Wichtige bisher noch nicht vorhandene Informationen, z.B. die Bearbeitungszeit eines Prozesses, können durch den Einsatz leistungscharakteristischer Werte verdeutlicht werden.

Abbildung 1.38 zeigt das Wurzelaktivitätsdiagramm nach der Erweiterung um Leistungsparameter, Messdaten und Strukturinformationen. Die Aktionszustände wurden mit mehreren Stereotypen versehen. «PAstep» kennzeichnet einen Aktionszustand/Prozess als relevant für die Leistungsbewertung. Wird dieses Stereotyp nicht benutzt, so spielt der Aktionszustand im Verlauf der Leistungsbewertung keine Rolle. «PAstep» bietet einige Eigenschaftswerte, die das Verhalten des Prozesses charakterisieren und darüberhinaus Strukturinformation: Die Tatsache, dass die einzelnen Aktionszustände des Diagramms von den obigen Aktivitätsdiagrammen der Teilprozesse erweitert werden, wird durch den Eigenschaftswert HUMLsubAct in dem mit «PAstep» überschriebenen Teil der Annotation eines jeden Aktionszustands konkretisiert. Der Wert von HUMLsubAct ist der Name des verfeinernden Aktivitätsdiagramms.

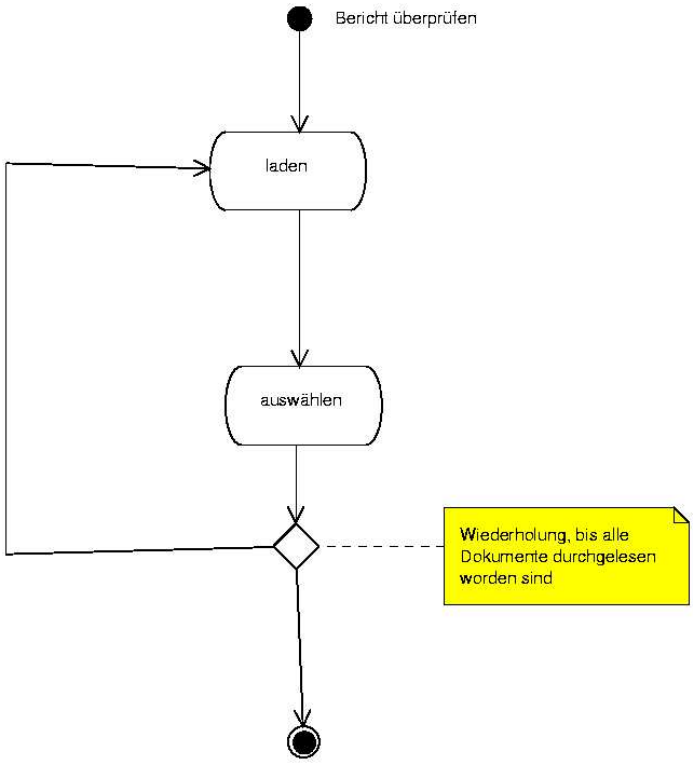

Abbildung 1.36.: Das Aktivitätsdiagramm für die Prüfung eines Berichts

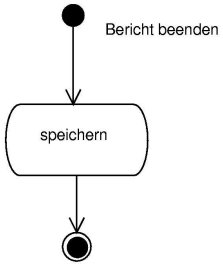

Abbildung 1.37.: Das Aktivitätsdiagramm für das Abschließen eines Berichts

Bei dem modellierten System handelt es sich um ein geschlossenes System mit einer konstanten Population. Das Stereotyp «PAclosedLoad» am ersten Aktionszustand in Abbildung 1.38 kennzeichnet diese Eigenschaft. Die Population des Systems besteht aus den 20 Abteilungen der Firma. Diese Eigenschaft wird durch den Wert PApopulation=20 in dem mit «PAclosedLoad» überschriebenen Teil der Annotation spezifiziert.

Die Eigenschaft, dass das Vorbereiten eines Berichts aus 3 gleichartigen Schritten besteht, wird durch HUMLrepGeo=3 am Aktionszustand “vorbereiten” in 1.38 angegeben. Dabei ist

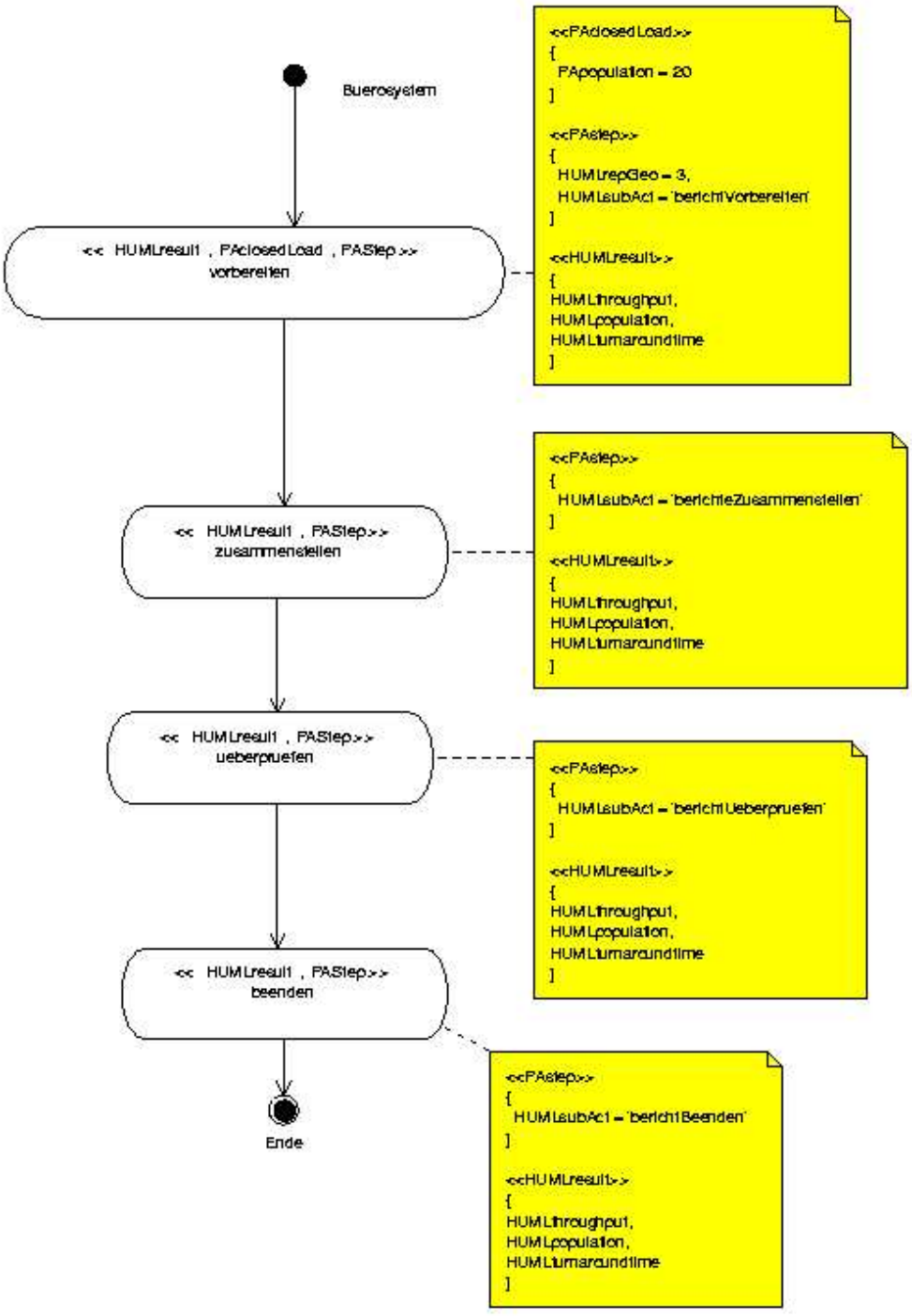

Abbildung 1.38.: Das Aktivitätsdiagramm für das Bürosystem mit Annotationen

zu beachten, dass dieser Wert aussagt, dass die Anzahl der Schleifendurchläufe der Wert einer geometrisch verteilten Zufallsvariablen mit Mittelwert 3 ist.

Jeder Aktionszustand in Abbildung 1.38 ist darüberhinaus mit «HUMLresult» stereotypisiert. Das Stereotyp erlaubt es dem Benutzer, die gewünschten Messwerte für einen Prozess in dem mit «HUMLresult» überschriebenen Teil der zugehörigen Annotation zu spezifizieren. Abbildung 1.38 zeigt, dass z.B. für den Prozess "vorbereiten" der Durchsatz (HUMLthroughput), die Population (HUMLpopulation) und die Antwortzeit (HUMLturnaroundtime) ermittelt werden sollen. Nach der Leistungsbewertung werden an diesen Stellen die Messergebnisse eingetragen.

Im nächsten Schritt der performanzorientierten Erweiterung der UML-Diagramme ist das Aktivitätsdiagramm, das das Vorbereiten eines Berichts modelliert, um Performanzinformationen zu ergänzen, siehe Abbildung 1.34. Das erweiterte Diagramm ist in Abbildung 1.39 zu sehen. Der erste Aktionszustand des Diagramms modelliert das Laden und Ändern eines Teilberichts. Dieser Prozess findet pro Vorbereitung im Mittel 10 Mal statt, so dass der Prozess in einer Schleife 10 mal durchgeführt wird. Da das Laden und Ändern aus mehreren Einzelprozessen besteht, muss auch dieser Aktionszustand durch ein weiteres Diagramm verfeinert werden. Dies wird wie üblich durch die Verwendung von HUMLsubAct in der Annotation zu "ladenUndAendern" in Abbildung 1.39 kenntlich gemacht. Die Anzahl der Schleifendurchläufe wird durch den Eigenschaftswert HUMLrepGeo mit dem Wert 10 spezifiziert.

Der zweite Teilprozess der Berichtsvorbereitung ist das Speichern des editierten Berichts. Dieser Prozess belastet die Netzwerkverbindung, die als aktive Ressource modelliert wird. Der Eigenschaftswert PAdemand in der Annotation zu "speichern" in Abbildung 1.39 sagt aus, dass der vermutete ("assm", *assumed*) Zeitverbrauch exponential verteilt ist und auf einer Standardmaschine im Mittel ("mean") 3 Sekunden beträgt. Der Prozess und die ihn bearbeitende Maschine werden durch den Eigenschaftswert HUMLhost über Namensgleichheit in Verbindung gebracht. Das heißt, dass in einem Verteilungsdiagramm ein Maschinenknoten namens "Netzwerkverbindung" zu modellieren ist.

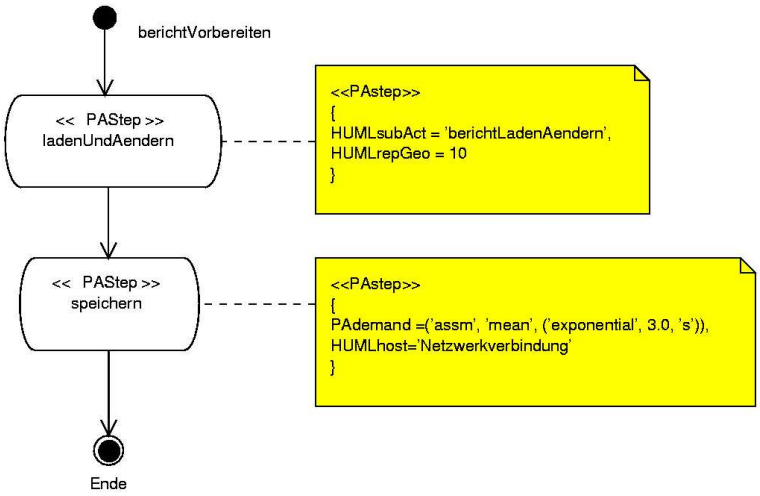

Abbildung 1.39.: Das Aktivitätsdiagramm "berichtVorbereiten" mit Leistungsdaten

Abbildung 1.40 zeigt das erweiterte Aktivitätsdiagramm, das das Laden und Editieren einer Berichtsseite modelliert. Das Laden einer Seite beansprucht die aktive Ressource "Netzwerkverbindung". Die Beziehung zwischen Prozess und Aktivität wird durch den Eigenschaftswert HUMLhost hergestellt. Die vermutete ("assm", *assumed*) Verarbeitungszeit auf Standardhardware ist exponential verteilt und beträgt im Mittel ("mean") 3 Sekunden.

Das Ändern einer Berichtsseite wird auf der passiven Ressource "SAP_1" (einem Rechner) durchgeführt. Zur Spezifikation dieses Sachverhalts wird der Eigenschaftswert PAextOp verwendet. Die Bearbeitung des Prozesses "aendern" ist exponential verteilt und beansprucht auf der Ressource "SAP_1" eine angenommene ("assm", *assumed*) mittlere ("mean") Zeit von 4.0 Sekunden.

Das Ändern einer Berichtsseite geschieht in (im Mittel) 5 Wiederholungen eines Prozesses. Diese Eigenschaft wird durch den Wert HUMLrepGeo=5 beschrieben. Während jedes Arbeitsschrittes tritt eine exponential verteilte Verzögerung von im Mittel ("mean") 20 Sekunden seitens des Sachbearbeiters auf. Diese Verzögerung wird durch den Eigenschaftswert PAinterval modelliert. In Abbildung 1.40 sagt dieser aus, dass während jeder Schleifeniteration eine vermutete ("assm", *assumed*) mittlere ("mean") Verzögerung von 20 Sekunden auftritt.

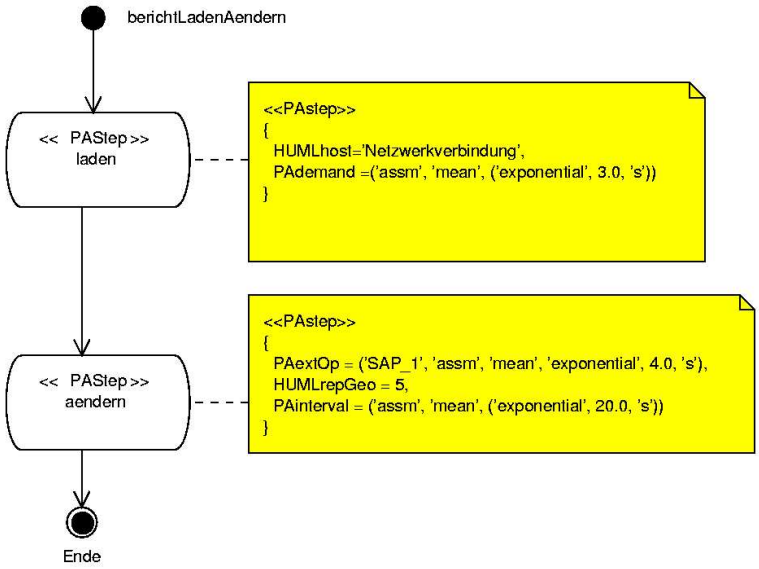

Abbildung 1.40.: Das Subaktivitätsdiagramm "berichtLadenAendern" mit Leistungsdaten

Abbildung 1.41 zeigt das um Performanzeigenschaften erweiterte Aktivitätsdiagramm für das Zusammenstellen eines Berichts. Da die Topologie der Diagramme identisch ist, sei hier auf die Besprechung von Abbildung 1.39 verwiesen.

Abbildung 1.42 zeigt das Aktivitätsdiagramm, das das Überprüfen des Berichts modelliert. Das Diagramm besteht aus einem einzelnen Aktionszustand, der das Laden und Auswählen von Teilberichten repräsentiert. Dieser Schritt wird im Mittel 10 Mal durchgeführt (HUMLrepGeo) und im Aktivitätsdiagramm "berichtLadenAuswaehlen" verfeinert (HUMLsubAct).

Das Aktivitätsdiagramm "ladenUndAuswaehlen" ist in Abbildung 1.43 zu sehen. Es besteht aus den Schritten "laden" (einer Berichtsseite) und "auswaehlen" (einer Berichtsseite). Das La-

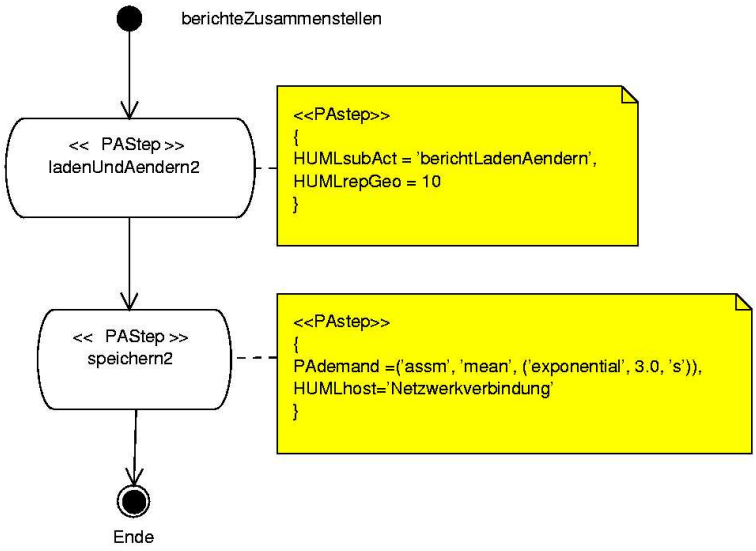

Abbildung 1.41.: Das Aktivitätsdiagramm "berichteZusammenstellen" mit Leistungsdaten

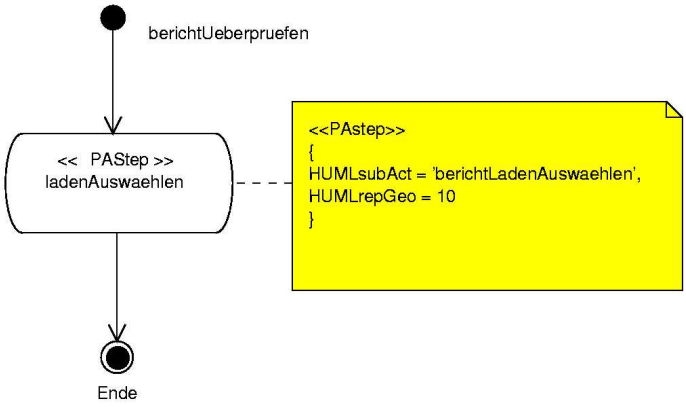

Abbildung 1.42.: Das Aktivitätsdiagramm "berichtUeberpruefen" mit Leistungsdaten

den belastet die aktive Ressource "Netzwerkverbindung" (HUMLhost) und nimmt im Mittel ("mean" in PAdemand) eine vermutete ("assm", *assumed* in PAdemand) Zeit von 3 Sekunden in Anspruch, bei der es sich um den Wert einer exponential verteilten Zufallsvariablen handelt. Das Auswählen einer Berichtsseite belastet die passive Ressource "SAP_2" (einem Rechner). Der Eigenschaftswert PAextOp sagt aus, dass diese Ressource vermutlich ("assm", *assumed*) im Mittel ("mean") 2 Sekunden lang zur Prozessbearbeitung benutzt wird. Der Vorgang des Auswählens umfasst die dreimalige Wiederholung eines Arbeitsschritts. Diese Eigenschaft wird wie üblich durch Verwendung von HUMLrepGeo spezifiziert. Die Zahl der Schleifendurchläufe ist der Wert einer geometrisch verteilten Zufallsvariablen und beträgt im Mittel 3. Darüber-

hinaus fällt pro Schleifeniteration eine Verzögerung seitens des Sachbearbeiters an, die per PAinterval angegeben wird. Sie ist der Wert einer exponential verteilten Zufallsvariablen und beträgt im Mittel 60 Sekunden.

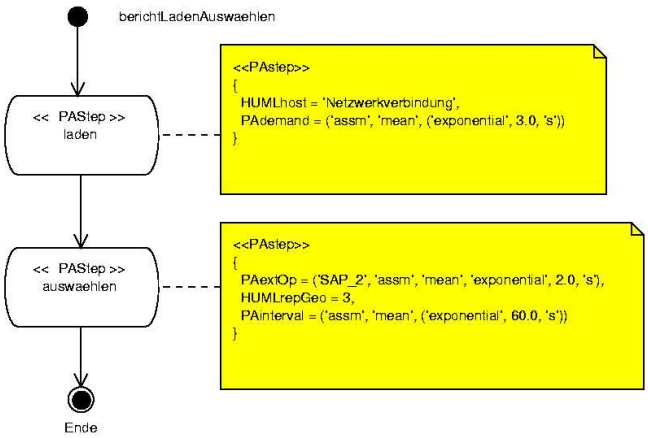

Abbildung 1.43.: Das Subaktivitätsdiagramm "berichtLadenAuswaehlen" mit Leistungsdaten

Nach dem Auswählen der zu archivierenden Berichtsteile werden diese gespeichert. Das Aktivitätsdiagramm, das diesen Vorgang modelliert, ist in Abbildung 1.44 zu sehen. Der Vorgang des Speicherns belastet die aktive Ressource "Netzwerkverbindung" (HUMLhost). Die zeitliche Belastung bzgl. einer Standardhardware beträgt eine vermutete ("assm"), mittlere ("mean") und exponential verteilte Zahl von 3.0 Sekunden. Das Speichern erfolgt in 20 Wiederholungen eines Arbeitsschritts. Diese Eigenschaft wird durch die Benutzung von HUMLrepGeo=20 spezifiziert. Die Zahl der Schleifendurchläufe ist der Wert einer geometrisch verteilten Zufallsvariablen und beträgt im Mittel ("mean") 20. Schließlich fällt pro Schleifeniteration eine Verzögerung von vermuteten ("assm") mittleren ("mean") 30.0 Sekunden an (PAinterval). Diese Zahl ist der Wert einer exponential verteilten Zufallsvariablen.

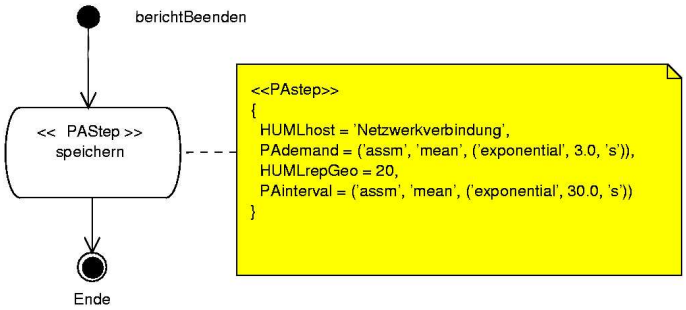

Abbildung 1.44.: Das Aktivitätsdiagramm "berichtBeenden" mit Leistungsdaten

Neben den Aktivitätsdiagrammen ist ein Verteilungsdiagramm erforderlich, das die aktive Ressource "Netzwerkverbindung" und die passiven Ressourcen "SAP_1" und "SAP_2" defi-

niert, siehe Abbildung 1.45. Die aktive "Netzwerkverbindung" ist mit «PAhost» und «HUMLresult» stereotypisiert. «PAhost» klassifiziert die Ressource als aktiv. In dem zugehörigen Annotationsblock werden Geschwindigkeit und Prozessbearbeitungsrichtlinie dieser Ressource spezifiziert. PArate gibt die Geschwindigkeit in Relation zu einer Standardmaschine an, im Beispiel beträgt PArate 1, d.h. dass die Geschwindigkeit der Maschine der Geschwindigkeit der Standardhardware entspricht. Die Prozessbearbeitungsrichtlinie wird durch Verwendung des Eigenschaftswerts PAschdPolicy angegeben. Die Bearbeitungsrichtlinie der aktiven Ressource ist *processor sharing* (PAschdPolicy='ProcSharing').

Die beiden Ressourcen "SAP_1" und "SAP_2" sind durch «PAresource» als passive Ressourcen angegeben. Da "SAP_1" und "SAP_2" identisch sind, gelten folgende Beschreibungen für beide. Durch den Eigenschaftswert PAcapacity wird die Kapazität der Maschine angegeben, d.h. die Zahl der maximal gleichzeitig bedienbaren Prozesse. Für beide passiven Ressourcen beträgt die Kapazität 20. Der Wert PAschdPolicy ist für beide passiven Ressourcen *first in first out* (PAschdPolicy='FIFO').

Alle Maschinen sind neben «PAhost» mit «HUML=result» stereotypisiert. Damit wird wie schon bei Aktionszuständen angezeigt, dass an der Maschine Leistungsmessungen vorgenommen werden sollen. An der "Netzwerkverbindung" sollen z.B. Durchsatz (HUMLthroughput), Population (HUMLpopulation), Antwortzeit (HUMLturnaroundtime) und Auslastung (HUMLutilization) gemessen werden.

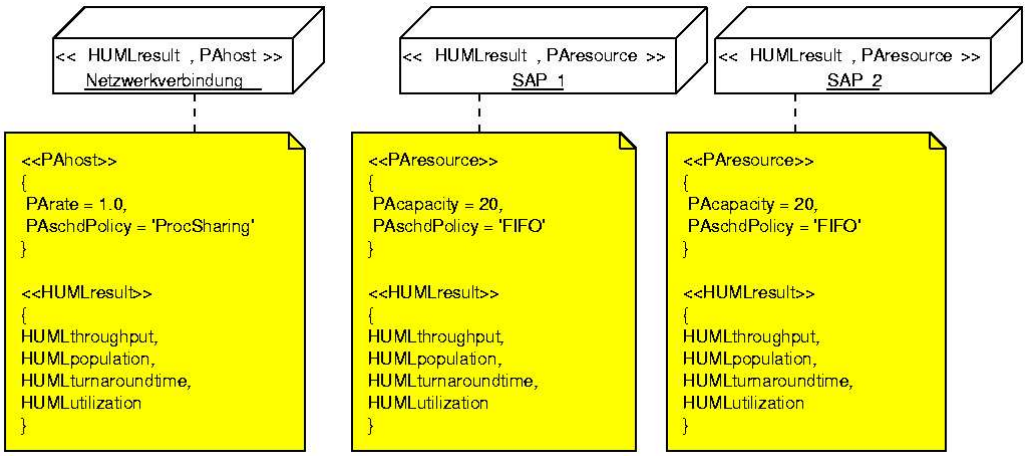

Abbildung 1.45.: Das Verteilungsdiagramm mit Leistungsdaten

Fazit der UML-Modellierungsphase Mit der Aktivitätsdiagrammhierarchie 1.46 und dem Verteilungsdiagramm wurde das Bürosystem modelliert. Das Anwendungsfalldiagramm wird in ein Aktivitätsdiagramm transformiert welches wiederum durch weitere um Performanzwerte erweiterte Aktivitätsdiagramme verfeinert werden kann.

Durch die Erweiterungen um Stereotype und Eigenschaftswerte wurden Informationen hinzugefügt, deren Spezifikation in UML-Diagrammen von UML nicht vorgesehen ist, die für die Leistungsbewertung aber unverzichtbar sind. Der nächste Schritt in der Leistungsbewertung ist die Übersetzung der Diagramme in ein HIT-Modell.

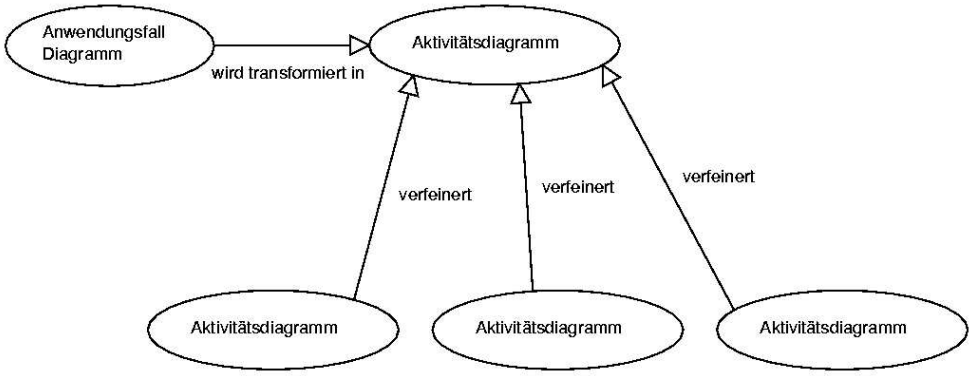

Abbildung 1.46.: Die Aktivitätsdiagrammhierarchie

Übersetzung der UML-Diagramme in ein HIT-Modell Nun sind die um Performanzeigenschaften erweiterten UML-Diagramme in HI-SLANG darzustellen. Dabei wird darauf hingewiesen, dass die Anzahl der Zeichen für die Namensgebung der Komponenten und Dienste mit einem Bildbearbeitungsprogramm erhöht wurde, da die Namensgebung in HITGRAPHIC beschränkt ist.

In Abbildung 1.55 wird das gesamte Bürosystem mitsamt der Dienste und Maschinenkomponenten in allen Ebenen vorgestellt. Der Übersichtlichkeit wegen, wurde es am Ende des Kapitels eingefügt.

Dabei wird nur ein Pfad von der obersten Modellebene bis zu einer Maschinenebene exemplarisch verfolgt. Und zwar verfolge man die Komponente "vorbereiten" des Komponententyps "vorbereiten_class", die von der Komponente "vorbereiten_sub" des Komponententyps "vorbereiten_sub_class" angebotenen Dienste in Anspruch nimmt. Desweiteren verfolgt man die Komponente "ladenundaendern1" des Komponententyps "ladenundaendern1_class" und die Komponente "ladenundaendern1_sub" des Komponententyps "ladenundaendern1_sub_class". Die darunter liegende Schicht bildet die Komponente "laden1" des Komponententyps "laden1_class" und "aendern1" des Komponententyps "aendern1_class". Letztendlich sind diese Komponenten auf Maschinen-Ebene mit den Hardware-Ressourcen "SAP_1" und "Netzwerkverbindung" verbunden, da diese die vom Sekretär benutzten technischen Mittel sind.

Das in Abbildung 1.47, 1.48 und 1.49 aufgeführte HITGRAPHIC-Modell stellt die oberste Modell-Ebene des Bürosystems dar. Aus Platzgründen wurde das Modell auf drei Abbildungen aufgeteilt.

Auf der obersten HIT-Modellebene werden die Aktionszustände aus Abbildung 1.38 zu Maschinenkomponenten der obersten Modellebene, die mit den entsprechenden Diensten verbunden sind. Hinzu kommen die Komponenten "SAP_1", "SAP_2" und "Netzwerkverbindung" (siehe Abbildungen 1.47, 1.48 und 1.49), die in dieser HITGRAPHIC-Darstellung als unverbundene Komponenten stehen, da dies Komponenten der untersten Ebene darstellen. Diese sind auf der untersten Modell-Ebene genutzte Hardwareressourcen. Während die Komponente "SAP_1" einen vom Sekretär benutzten Rechner darstellt, wird "SAP_2" vom Sachbearbeiter verwendet. Die "Netzwerkverbindung" ist eine gemeinsam genutzte Komponente.

Der Rumpf des Dienstes "buerosystem", der der Struktur des Kontrollflusses des Aktivitätsdiagramms aus Abbildung 1.38 entspricht, lautet:

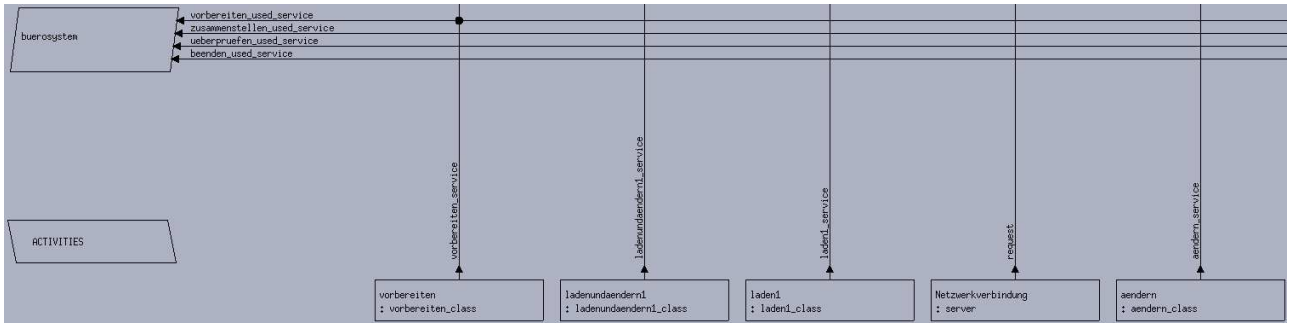

Abbildung 1.47.: Bürosystem: Die oberste Modellebene (1. Teil)

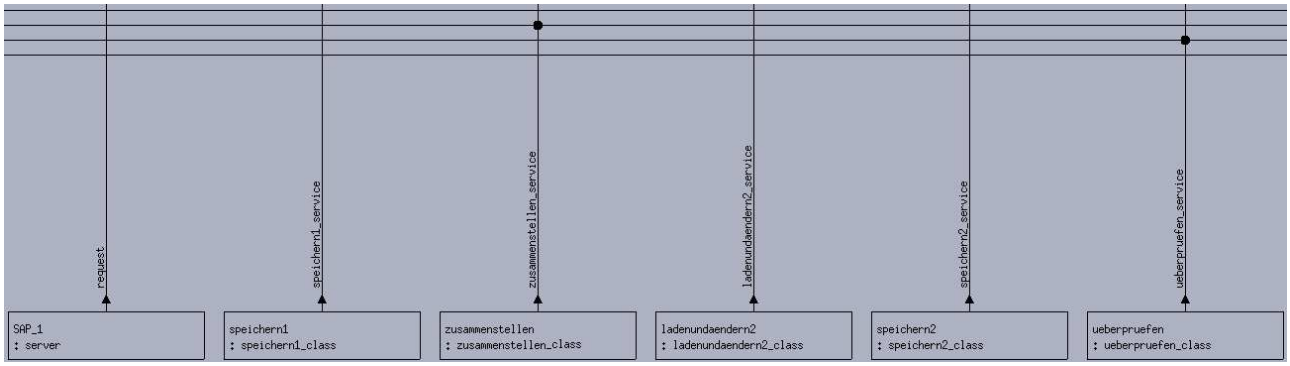

Abbildung 1.48.: Bürosystem: Die oberste Modellebene (2. Teil)

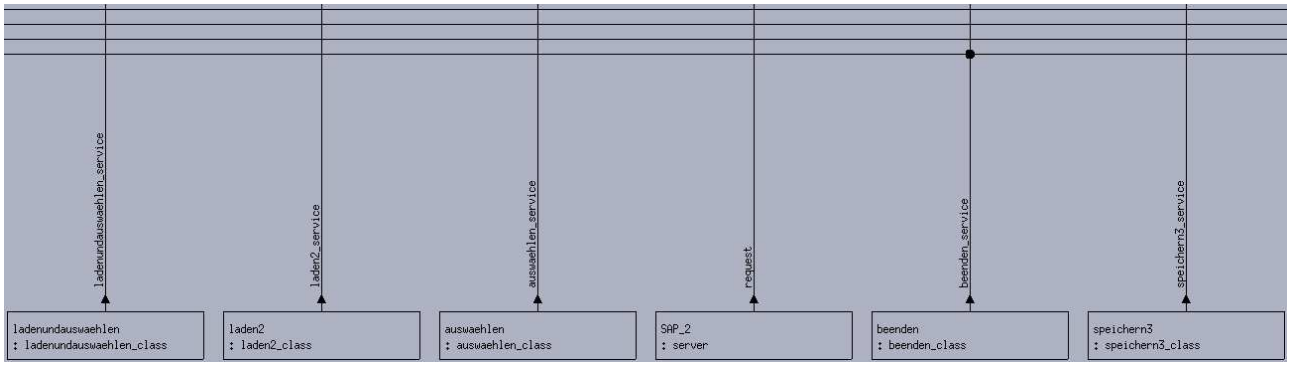

Abbildung 1.49.: Bürosystem: Die oberste Modellebene (3. Teil)

```
CLOSED_CHAIN
    QNODE vorbereiten_used_service
        PROB 1 : zusammenstellen_used_service;
    QNODE zusammenstellen_used_service
        PROB 1 : ueberpruefen_used_service;
    QNODE ueberpruefen_used_service
```

```
        PROB 1 : beenden_used_service;
    QNODE beenden_used_service
        PROB 1 : vorbereiten_used_service;
END CLOSED_CHAIN
```

Anschließend wird der *Activities*-Abschnitt vorgestellt, in dem die Anzahl der Prozesse angegeben werden, die an der ersten Aktivität aus Abbildung 1.38 als Annotation vermerkt wurden:

```
CREATE 20 PROCESS buerosystem
```

Da hier nur exemplarisch ein Pfad gezeigt werden soll, betrachtet man ausschließlich die Komponente "vorbereiten" vom selbstdefinierten Komponententyp "vorbereiten_class".

Dieser Komponententyp wird in einer tieferen Ebene betrachtet. Der Komponententyp "vorbereiten_class" entspricht einem erweiterten Aktivitätsdiagramm (Abbildung 1.39), auf das in Abbildung 1.38 als "HUMLsubAct = ' berichtVorbereiten"' referenziert wurde.

Die Abbildung 1.50 stellt eine solche Modell-Ebene dar. Hier wird der benutzte Dienst "vorbereiten_sub_used_service", der im Dienst "vorbereiten_service" enthalten ist, mit dem angebotenen Dienst "vorbereiten_sub_service", der von der Maschine "vorbereiten_sub" angeboten wird, verknüpft.

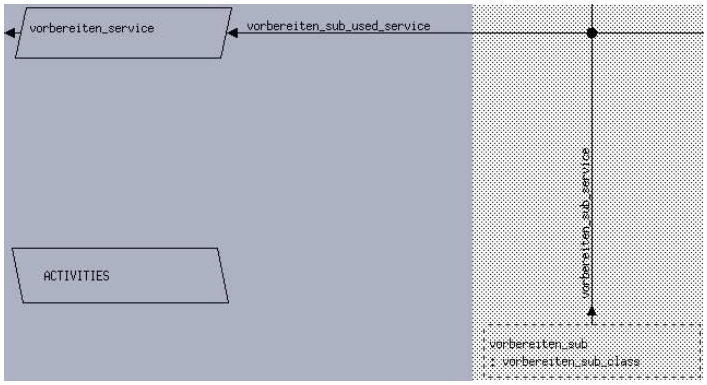

Abbildung 1.50.: Bürosystem: Der Komponententyp "vorbereiten _class"

Der Rumpf des angebotenen Dienstes "vorbereiten_service" des Komponententyps "vorbereiten_class" besitzt folgende Zeilen:

```
AVERAGE 3 TIMES LOOP
    vorbereiten_sub_used_service;
END LOOP
```

Der Inhalt des Rumpfes "vorbereiten_service" entspricht der Annotation (aus Abbildung 1.38), die an der Aktion "vorbereiten" angehängt wurde. In dieser Notiz sind mittels Eigenschaftswerten die Schleifendurchläufe angegeben worden.

Nun betrachtet man die hierarchisch tiefer liegende selbstdefinierte Komponente "vorbereiten_sub_class".

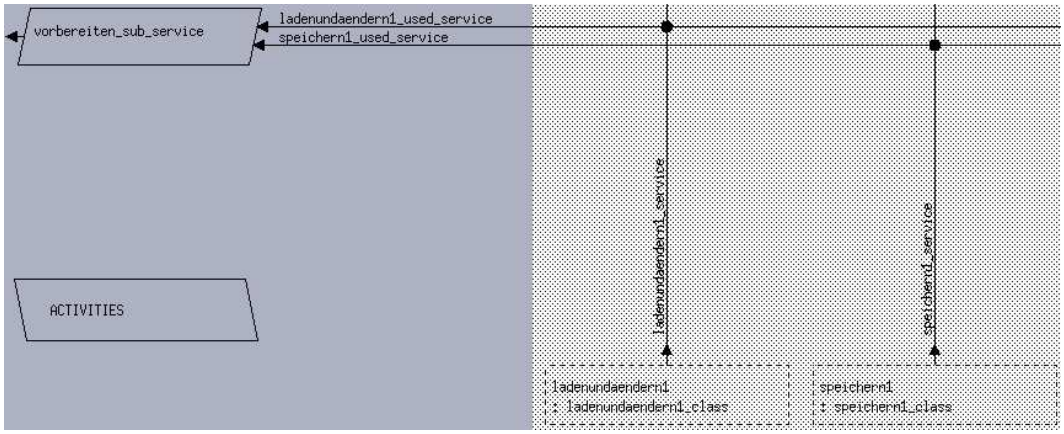

Abbildung 1.51.: Bürosystem: Der Komponententyp "vorbereiten _sub_class"

Dieses in Abbildung 1.51 veranschaulichte HITGRAPHIC-Modell ist eine Gegenüberstellung zum Aktivitätsdiagramm (aus Abbildung 1.39).

Der Dienst "vorbereiten_sub_service" des Komponententyps "vorbereiten_sub_class", der der höheren Ebene zur Verfügung gestellt wird, besitzt im Rumpf folgende Zeilen:

```
OPEN_CHAIN
        PROB 1 : ladenundaendern1_used_service;
    QNODE ladenundaendern1_used_service
        PROB 1 : speichern1_used_service;
    QNODE speichern1_used_service
END OPEN_CHAIN
```

Im Folgenden ist die selbstdefinierte Komponente "ladenundaendern1_service" vom Typ "ladenundaendern1_class" zu betrachten.

Das in Abbildung 1.52 vorgestellte Modell zeigt eine tiefere Modell-Ebene in HITGRAPHIC. Der benutzte Dienst "ladenundaendern1_sub_used_service" ist mit dem angebotenen Dienst "ladenundaendern1_sub_service", der von der Komponente "ladenundaendern_sub_service" des Typs "ladenundaendern1_sub_class" angeboten wird, verbunden.

Die Anzahl der Schleifendurchläufe des Prozesses "ladenundaendern1_sub_used_service" wird im Rumpf des Dienstes "ladenundaendern1_service" des Komponententyps "ladenundaendern1_class" festgehalten:

```
AVERAGE 10 TIMES LOOP
    ladenundaendern1_sub_used_service;
END LOOP
```

Nun verfolge man die Ebene, bei der der Dienst "ladenundaendern1_service" bereitgestellt wird. In Abbildung 1.53 wird diese tiefere Modell-Ebene, die dem Subaktivitätsdiagramm für "Bericht vorbereiten"(aus Abbildung 1.40) gegenübergestellt werden kann, vorgestellt. Der Dienst "ladenundaendern1_sub_service" enthält die zu benutzenden Dienste

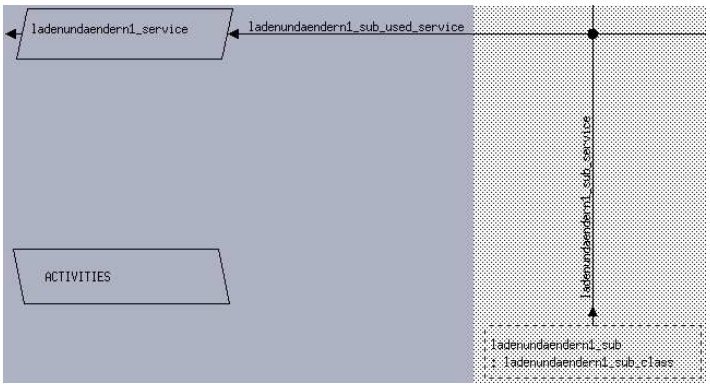

Abbildung 1.52.: Bürosystem: Der Komponententyp "ladenundaendern1_class"

"laden1_used_service" und "aendern1_used_service", die mit den jeweiligen angebotenen Diensten "laden1_service" und "aendern1_service" verknüpft sind. Diese Dienste werden jeweils von den Komponenten "laden1" vom Typ "laden1_class" und "aendern1" vom Typ "aendern1_class" zur Verfügung gestellt.

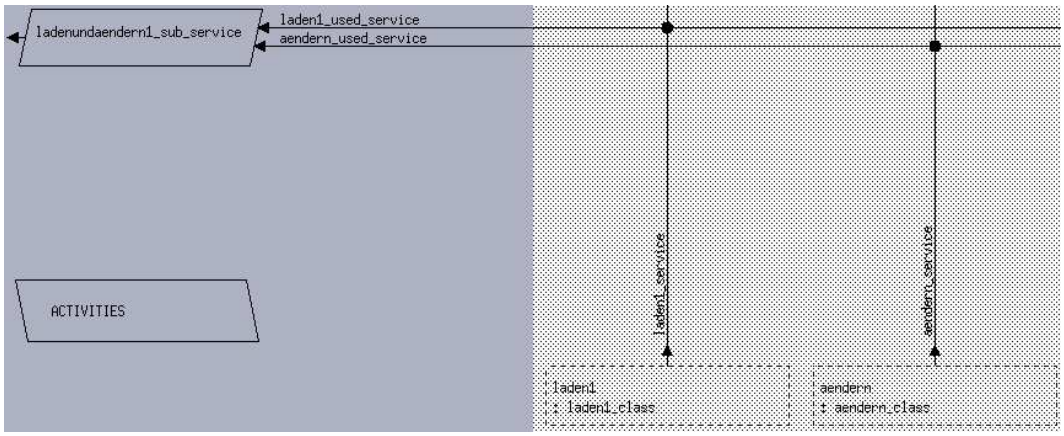

Abbildung 1.53.: Bürosystem: Der Komponententyp "ladenundaendern1_sub_class"

Der Rumpf des angebotenen Dienstes "ladenundaendern1_sub_service" des Komponententyps "ladenundaendern1_sub_class" sieht folgendermaßen aus:

```
OPEN_CHAIN
        PROB 1 : laden1_used_service;
    QNODE laden1_used_service
        PROB 1 : aendern1_used_service
END OPEN_CHAIN
```

Man verfolge im Anschluss die tiefere Modell-Ebene, in der der Dienst "aendern1_service" bereitgestellt wird. Die in Abbildung 1.54 vorgestellte Maschinen-Ebene zeigt in der horizon-

talen Ebene die Maschinenkomponente "SAP_1" vom Typ "server". Dies ist die vom Sekretär genutzte Komponente, die zur technischen Unterstützung bereit steht.

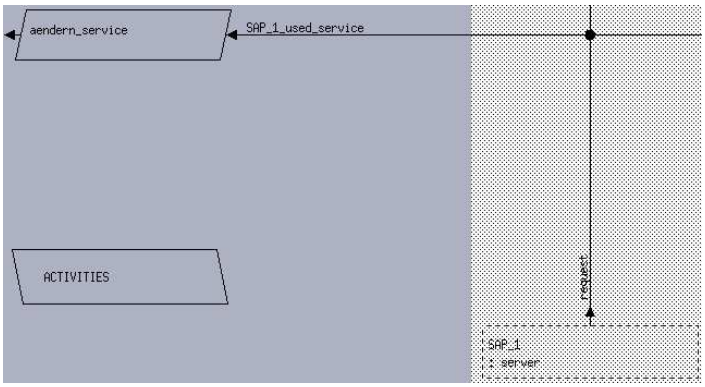

Abbildung 1.54.: Bürosystem: Der Komponententyp "aendern1_class"

Folgerungen Die hier aufgeführten HITGRAPHIC-Modelle bilden nur eine Teilmenge der HI-SLANG-Modelle welche aus um Performanzwerte ergänzten UML-Modellen herleitbar sind.

Die im Rumpf der Dienste erstellten HI-SLANG-Befehle können aus den in Aktivitätsdiagrammen enthaltenen Annotationen, in der die Leistungsdaten durch Eigenschaftswerte angegeben sind, abgeleitet werden. Dabei stellte jede Einzelaktion aus den Aktivitätsdiagrammen eine selbstdefinierte Komponente dar, die mit den entsprechenden benutzten Diensten (*used service*) verbunden wurden. Die Information, welche Komponente von welcher Aktion beansprucht wird, wurde aus Annotationen zu den Einzelaktionen entnommen.

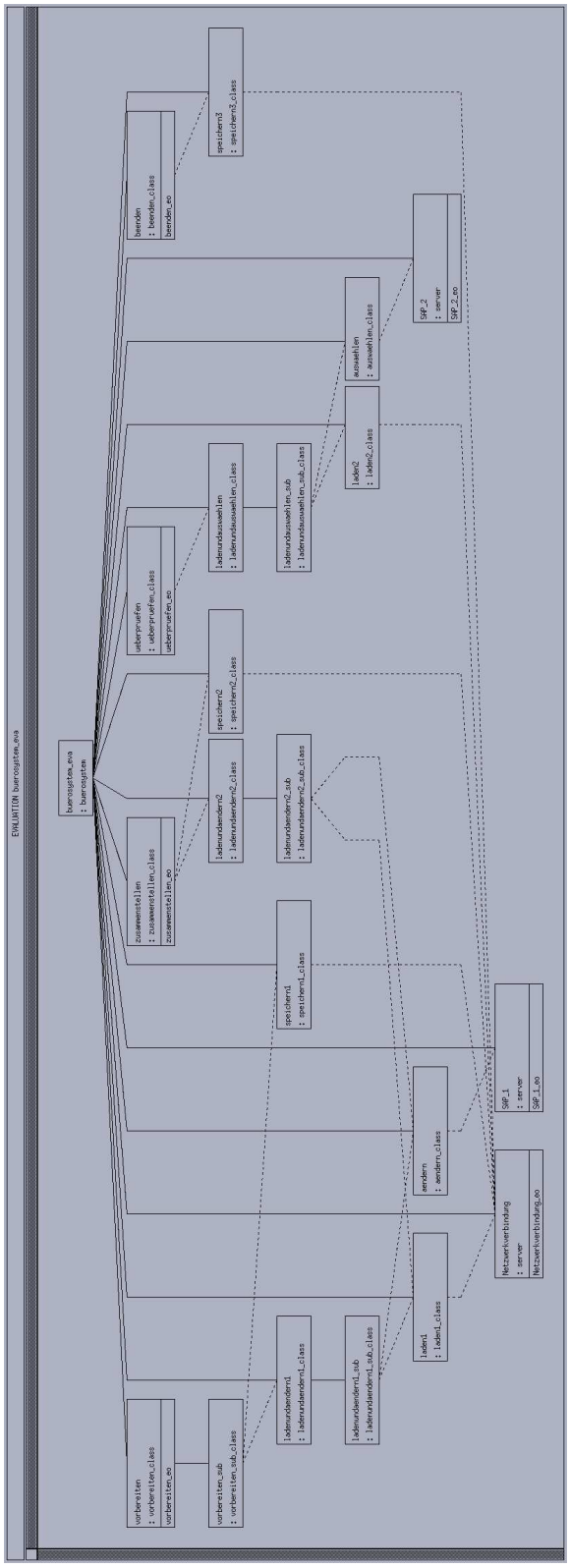

Abbildung 1.55.: Bürosystem: Die gesamte Modell-Hierarchie

2. Implementierung

Im Folgenden werden die Pakete des Werkzeugs HUML beschrieben. Es gibt ein GUI-Paket (*HUMLGUI*), das die grafische Benutzeroberfläche beschreibt. Das Paket *HUMLControl* enthält die Menge der Kontrollklassen, die den Ablauf von HUML steuern. Das Paket *HUMLDatastructure* enthält die Klassen der HUML-internen Datenstruktur zur Repräsentation der XMI-Eingabe. Die in *HUMLDatastructure* enthaltenen Pakete *HUMLWorkload*, *HUMLEvaluation*, *HUMLActionStateDiagram* und *HUMLResource* werden von HUML zur Erzeugung einer HI-SLANG-Eingabe für HIT verwendet. In *HUMLXMITranslator* befinden sich Klassen zur Übersetzung der XMI-Eingabe in die HUML-Datenstruktur. Im Paket *HUML* befinden sich das Paket *HUMLHISLANG* sowie weitere Verwaltungsklassen und Interfaces. Die HUML-Datenstruktur wird von den Klassen des Pakets *HUMLHISLANG* in eine HISLANG-Eingabe für das Werkzeug HIT übersetzt. Das Paket *HUMLDumpfileTranslator* bietet Funktionen zur Ergänzung der XMI-Eingabe um die Messergebnisse der HIT-Leistungsanalyse. Es wurden während der Entwicklung von HUML zwei frei verfügbare Werkzeuge eingesetzt. Das Paket *ANTLR* enthält die Klassen des Parsergenerators ANTLR ANTLR, und das Paket *NanoXML* enthält die Klassen des XML-Parsers NanoXML. Die Paketstruktur ist in Abbildung 2.1 dargestellt.

2.1. Paket HUMLGUI

Das Paket *HUMLGUI* besteht aus den Klassen *HUMLMain, HUMLConfiguration, HUMLNewProject, HUMLSaveDialog, HUMLLogFenster, HUMLExperiment* und *Version*, die zur Erstellung der graphischen Oberfläche von HUML notwendig sind.

HUMLMain Die Klasse erbt von der Klasse *JFrame* aus dem Paket *javax.swing* und stellt das Hauptfenster von HUML bereit, das zur zentralen Kontrolle sämtlicher Fenster und Dialoge dient. Die Klasse *HUMLMain* steht deshalb mit allen anderen GUI-Klassen sowie der Klasse *HUMLn* vom Paket *HUMLControl* in Verbindung. *HUMLMain* enthält sämtliche "ActionPerformed"-Methoden. Diese ermöglichen es, Fenster und Dialoge anzuzeigen oder Methoden aus dem Paket *HUMLControl* aufzurufen.

HUMLConfiguration Die Klasse erbt von der Klasse *JFrame* aus dem Paket *javax.swing* und verwaltet das Fenster der HUML-Konfiguration, in dem der Benutzer Grundeinstellungen für HUML vornehmen kann.

HUMLNewProject Die Klasse erbt von der Klasse *JFrame* aus dem Paket *javax.swing* und zeigt das Fenster zum Erzeugen eines neuen Projekts an.

HUMLSaveDialog Die Klasse *HUMLSaveDialog* wird zum Erzeugen des Fensters zum Speichern eines Projekts verwendet. Sie erbt von der Klasse *JFrame* aus dem Paket *javax.swing*

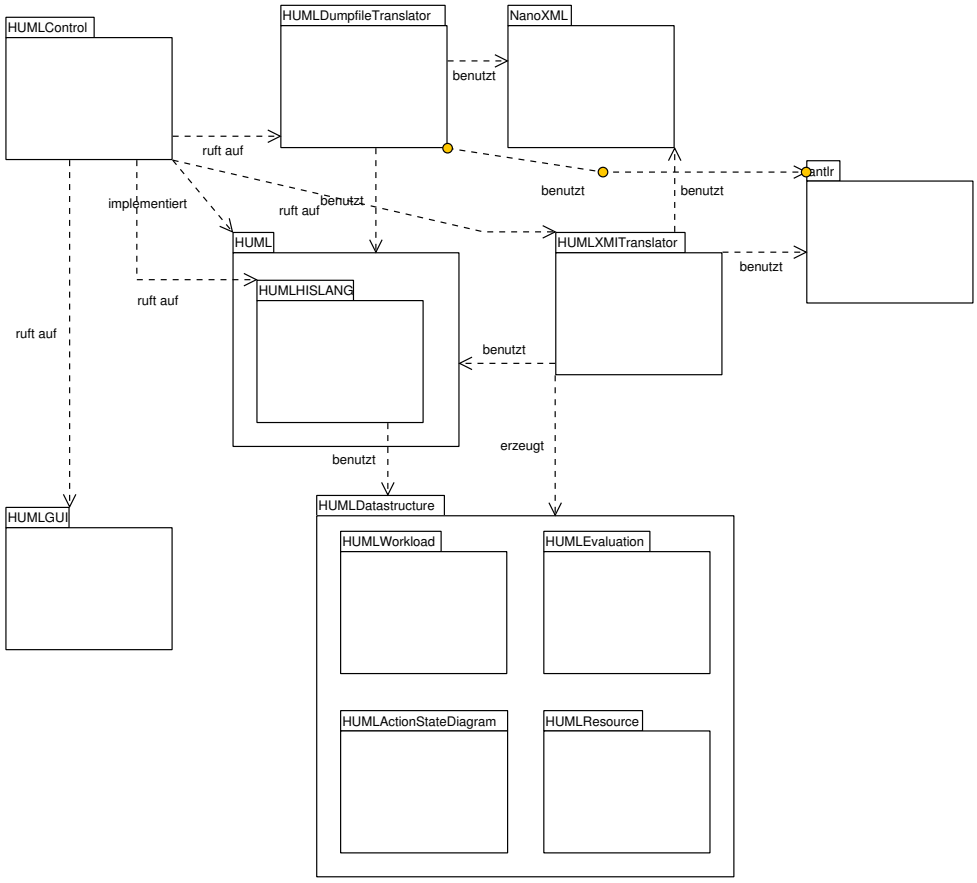

Abbildung 2.1.: Paketstruktur von HUML

und ermöglicht dem Benutzer das aktuelle Projekt unter anderem Namen im Projektverzeichnis zu speichern.

HUMLLogFenster Die Klasse erbt von der Klasse *JDialog* aus dem Paket *javax.swing* und stellt das Fenster bereit, das zum Anzeigen der Compiler-Logdatei und HIT-Logdatei verwendet wird. Die dazugehörigen Objekte werden beim Aufrufen der Methoden *showCompLog* und *showHitLog* der Klasse *HUMLMain* erzeugt.

HUMLExperiment *HUMLExperiment* erbt von der Klasse *JFrame* aus dem Paket *javax.swing*. Beim Erzeugen eines Objektes dieser Klasse wird das Fenster "Experiment" angezeigt, das aus mehreren Textblöcken besteht. Der Benutzer trägt in diesem Fenster die für die Durchführung der Leistungsbewertung benötigten Parameter ein.

Version Die Klasse *Version* wird verwendet, um Informationen über HUML wie z.B Version, Entwickler, Kontakt etc. anzuzeigen. Sie erbt von der Klasse *JFrame* aus dem Paket *javax.swing*.

2.2. Paket HUMLControl

Das Package *HUMLControl* enthält Kontrollklassen zur Steuerung des Programmablaufs. Diese Klassen werden im Folgenden einzeln vorgestellt.

HUMLn Die Klasse *HUMLn* enthält die main-Methode zum Starten des Programms. Beim Programmstart wird im HUML-Installationsverzeichnis die Datei *huml.ini* ausgelesen, sofern diese existiert. Sollte die Datei noch nicht vorhanden sein, so wird sie angelegt. *huml.ini* enthält Pfadangaben zur HIT-Installation und zum gewünschten Projektpfad. Sind diese Angaben nicht vorhanden, wird automatisch das Konfigurationsfenster zum Einstellen der genannten Pfade aufgerufen. Die Klasse stellt Methoden zum Setzen und Lesen dieser beiden Pfade zur Verfügung.

Beim Beenden des Programms wird die Datei *humln.ini* gesichert und ein möglicherweise geöffnetes Projekt kontrolliert beendet. Neben den Klassen *ProjectControl* und *HITControl* aus demselben Package wird auch das GUI-Hauptfenster *HUMLMain* aus dem Package *HUMLGUI* instanziiert. Außerdem wird der *MainThread* gestartet.

ProjectControl Die Klasse *ProjectControl* steuert sämtliche Operationen zur Verwaltung von Projekten. Dazu gehört das Anlegen, Öffnen, Speichern und das Löschen eines Projekts. Beim Erzeugen eines neuen Projekts wird ein neuer Ordner mit dem Namen des Projekts am Projektpfad erstellt. Danach wird die Datei *project.ini* generiert, die projektspezifische Daten (z.B. das Lösungsverfahren) enthält. Diese Werte werden mit Standardwerten belegt und können dann später im Experimentenfenster der Klasse HUMLExperiment aus dem Package HUMLGUI wunschgemäß geändert werden. Beim Öffnen eines Projektes ist die Datei *project.ini* bereits vorhanden und muss nicht neu erzeugt werden. In jedem Fall werden dann die Kontrollklassen *ProjectPropertyControl* und *Translator* instanziiert. Beim benutzerinitiierten Speichern des Projekts unter neuem Namen wird ein neuer Ordner am Projektpfad erstellt und alle Dateien aus dem aktuellen Projektordner werden dorthin kopiert. Das Löschen des Projektes geschieht durch Löschen aller Dateien aus dem Projektordner und des Projektordners selbst.

Für ein kontrolliertes Schließen eines geöffneten Projekts (z.B. beim Beenden) werden die Projekteinstellungen, die als Attribute in der Klasse *ProjectPropertyControl* vorliegen, in der Datei *project.ini* im Projektordner gespeichert.

ProjectPropertyControl Die Klasse *ProjectPropertyControl* verwaltet projektspezifische Attribute, die über das Experimentfenster eingegeben werden können. Hier werden auch je nach Status des Projektes die Lampenfarben für die GUI eingestellt.

HITControl Die Klasse *HITControl* ist eine Schnittstellenklasse zur Steuerung von HIT. Vor dem Start von HIT wird ein Shellskript dynamisch erzeugt. Dieses Skript enthält zwei Anweisungen zum Exportieren der Variablen für die im Experimentfenster angegebenen Werte für *HI-SLANG-Compiler Memory* und *HIT-Experiment Memory*. Die dritte Zeile dieses Skripts ruft dann das Programm HIT auf. Nachdem das Skript erzeugt wurde, werden die Rechte davon auf *ausführbar* gesetzt und anschließend ausgeführt. Dabei werden die von HIT erzeugten Konsolenmeldungen in die Datei *hit.log* umgeleitet.

Translator Die Klasse *Translator* steuert alle Übersetzungsvorgänge durch HUML. Dazu ruft sie die Methoden zum Importieren der XMI-Datei, zur Übersetzung in die interne Datenstruktur, zum Erzeugen von HISLANG-Code aus der internen Datenstruktur, zum Import der HIT-Messergebnisse in HUML und zum Zurückschreiben der Ergebnisse in die XMI-Datei auf. Es werden die Klassen *NameService* und *XMITranslator* instanziiert und eine Methode zum Schreiben in die Datei *compiler.log* angeboten.

MainThread Damit die GUI während des Programmablaufs aktualisiert wird, werden die Berechnungen in eigenen Threads durchgeführt. Damit die GUI-Ereignisse trotzdem in der Reihenfolge ihres Auftretens bearbeitet werden, müssen sie in den *Event Dispatcher Thread* geschrieben werden. Dies ist der Thread in dem die *Java Virtual Machine* alle Ereignisse, die aus der GUI stammen, sammelt. Der *MainThread* wird beim Start des Programms gestartet und läuft bis zum Beenden des Programms. Wenn der Benutzer ein Projekt erstellt, öffnet oder speichert oder wenn einer der Buttons für einen Übersetzungsschritt gedrückt wird, startet der *MainThread* einen *GUIThread*.

GUIThread Dieser vom *MainThread* gestartete Thread schreibt GUI-Ereignisse in den *Event Dispatcher Thread*. Dabei wird ein *StatusControl-Objekt* übergeben. Übersetzungsschritte werden an die Klasse *Translator* weitergeleitet, Projektverwaltung an *ProjectControl*. Beim Erstellen eines *GUIThread* wird bestimmt, welcher Programmcode ausgeführt werden soll, d.h. auf welches GUI-Ereignis reagiert werden soll.

StatusControl Ein *StatusControl-Objekt* enthält den Programmcode für das Initialisieren bzw. Setzen der Lampen und der Statuszeile. Die Klasse *StatusControl* sorgt dafür, dass die Lampen und die Statuszeile rechtzeitig und in der richtigen Reihenfolge gesetzt werden.

OrdnerException Diese Ausnahme behandelt Probleme, die beim Arbeiten mit HUML-Projekten auftreten.

2.3. Paket HUMLDatastructure

Das Paket *HUMLDatastructure* beinhaltet die verschiedenen Klassen und Pakete für die Realisierung der Datenstruktur: Das Paket *HUMLActionStateDiagram*, *HUMLResource*, *HUML-Workload* und *HUMLEvaluation*. Des Weiteren sind noch die Klassen *PseudoState* und *DistributionValue* vorhanden.

PseudoState Diese Klasse erbt von der abstrakten Klasse *Node* und modelliert die verschiedenen UML-Pseudostates innerhalb eines Diagramms: Start- und Endknoten, Verzweigungsknoten, Forks und Joins (Parallelläufigkeit), Maschinenknoten und Aktionszustände, die nicht mit «PAstep» stereotypisiert sind.

DistributionValue Die Klasse *DistributionValue* steht mit *Workload* und mit *ActionState* in Verbindung. Sie enthält Attribute, die angeben, ob ein Wert Mittelwert einer gleich- oder exponentialverteilten Zufallsvariable ist.

2.3.1. Paket HUMLActionStateDiagram

Das Package *HUMLActionStateDiagram* beschreibt UML-Aktivitätsdiagramme.

Model Die Klasse *Model* repräsentiert den Kopf eines UML-Modells. Sie enthält je ein Objekt der Klasse *Episode,* das das erste Aktivitätdiagramm repräsentiert und *Workload* , das die Arbeitslastinformation für das UML-Modell repräsentiert.

Node Die abstrakte Klasse *Node* modelliert einen Knoten in der Repräsentation eines Aktivitätsdiagramms. Sie ist Oberklasse von *ActionState*, *PseudoState*, *Episode* und *ConcurrentEpisodes*. Die Klasse verweist auf ein Objekt der Klasse Transition, das die Menge aller ausgehenden und eingehenden Kanten des Knotens enthält.

Episode Die Klasse *Episode* beschreibt einen Strang von *Node*-Objekten, erbt von der Klasse *Node* und steht in Beziehung zu der Klasse *Model*.

ActionState Die abstrakte Klasse *ActionState* beschreibt einen UML-Aktionszustand. Sie ist die Oberklasse der Klassen *AtomicActionState* und *ExtendedActionState* und erbt von der Klasse *Node*.

AtomicActionState Die Klasse *AtomicActionState* modelliert einen Aktionszustand, der in einem UML-Diagramm mit einem Maschinenobjekt (Objekt der Klasse *HardwareResource*) verbunden ist und nicht weiter verfeinert wird.

ExtendedActionState Die Klasse *ExtendedActionState* beschreibt einen UML-Aktionszustand, der durch ein weiteres Aktivitätsdiagramm, also ein Objekt der Klasse *Episode*, verfeinert wird.

ConcurrentEpisodes Die Klasse *ConcurrentEpisodes* modelliert Nebenläufigkeit in UML-Diagrammen. Sie besteht aus 2 oder mehr Objekten vom Typ *Episode*, die die parallellaufenden Aktionsstränge darstellen.

Transition Die Klasse *Transition* beschreibt eine UML-Transition und die Wahrscheinlichkeit, mit der der Zielknoten der Transition, von dem Quellknoten der Transition aus, erreicht wird.

2.3.2. Paket HUMLResource

Das Paket *HUMLResource* enthält die Klasse *HardwareResource*, *ActiveResource* und *PassiveResource*, die Maschinenkomponenten darstellen.

HardwareResource Die abstrakte Klasse *HardwareResource* repräsentiert eine Maschinenressource.

ActiveResource Die Klasse *ActiveResource* erbt von der Klasse *HardwareResource* und repräsentiert eine aktive Maschinenkomponente.

PassiveResource Die Klasse *PassiveResource* erbt von der Klasse *HardwareResource* und repräsentiert eine passive Maschinenkomponente.

2.3.3. Paket HUMLEvaluation

Das Paket *HUMLEvaluation* beschreibt die Evaluationsobjekte, die für die Leistungsbewertung benutzt werden.

Evaluation Die Klasse ist abstrakt und ist Oberklasse von den beiden Klassen *ActionEvaluation* und *HardwareEvaluation*, die der Spezifikation von Experimenten zur Leistungsbewertung dienen.

ActionEvaluation Die Klasse modelliert ein Evaluationsobjekt zur Leistungsbewertung von Aktionszuständen. Sie erbt von *Evaluation* und steht mit einer beliebigen Zahl von *ActionState*-Objekten in Verbindung.

HardwareEvaluation Die Klasse modelliert ein Evaluationsobjekt zur Leistungsbewertung von Maschinenknoten. Sie erbt von *Evaluation* und steht mit einer beliebigen Zahl von *HardwareResource*-Objekten in Verbindung.

2.3.4. Paket HUMLWorkload

Das Paket *HUMLWorkload* besteht aus einer abstrakten Klasse *Workload* und zwei Klassen *OpenLoad* und *ClosedLoad*, die von der abstrakten Klasse erben. Diese Klassen verwalten die Arbeitslastinformationen. Jede Workload wird mit einem Model assoziiert.

Workload Die abstrakte Klasse *Workload* verwaltet Informationen, die für Arbeitslasten von sowohl geschlossenen als auch offenen Systemen gelten.

OpenLoad Die Klasse *OpenLoad* modelliert eine offene Arbeitslast.

ClosedLoad Die Klasse *ClosedLoad* modelliert eine geschlossene Arbeitslast.

2.4. Paket HUMLXMITranslator

Das Paket dient der Transformation eines nach XMI exportierten UML-Diagramms in eine Repräsentation durch die HUML-interne Datenstruktur. Die Klasse *XMITranslator* steuert die Übersetzung. Es wird zunächst das Paket *NanoXML* benutzt, um die XMI-Eingabe zu analysieren und in eine Java-Baumstruktur zu überführen. Dies ist möglich, da XMI ein XML-Dialekt ist. Anschließend durchläuft *XMITranslator* den Baum und benutzt die *Builder*-Klassen, um die Bestandteile der HUML-internen Datenstruktur zu erzeugen und zusammenzusetzen. Vom Modellierer an UML-Aktionszuständen und Maschinenknoten angebrachte Kommentare werden durch die *ANTLR*-Klassen analysiert und in Form eines Objekts der Klasse *CommentInfo* verfügbar gemacht. Den Abschluss bildet die Verwendung der Klasse *GraphProcessor*, die alle Knoten der Klassenzugehörigkeit *PseudoState* (Paket *HUMLDatastructure*) aus dem bis dahin erzeugten Graphen entfernt.

XMITranslator Die Klasse übersetzt das XMI-Quelldokument in eine interne Repräsentation durch die HUML-Datenstruktur. Der erste Schritt ist der Import des XMI-Dokuments durch die NanoXML-Klassen. Das Resultat dieses Vorgangs ist ein Objekt der Klasse *XMLElement* des Pakets *NanoXML* ist, das die Wurzel einer Java-Baumrepräsentation der XMI-Quelle ist. *XMITranslator* durchläuft diesen Java-Baum und verarbeitet die einzelnen Teilbäume in folgender Reihenfolge:

- Übersetzung von Maschinenknoten, Kommentaren (UML-Annotationen), Stereotypen. Maschinen werden in *HardwareResource*- und Kommentare in *CommentInfo*-Objekte übersetzt. Die Stereotype werden als einfache Strings verwaltet.
- Übersetzung der Aktionszustände und anderer Bestandteile eines Aktivitätsdiagramms in Objekte der Klassen *PseudoState* und *ActionState* des Pakets *HUMLDatastructure-.HUMLActionStateDiagram* durch die Klassen *NodeBuilder*, *ActionStateBuilder* und *PseudoStateBuilder*.
- Erzeugung von Transitionen in Form von *Transition*-Objekten des Pakets *HUMLDatastructure.HUMLActionStateDiagram* zwischen diesen Knoten durch die Klasse *TransitionBuilder*.
- Erzeugung eines *Episode*-Objekts des Pakets *HUMLDatastructure.HUMLActionStateDiagram* für jedes Aktivitätsdiagramm durch die Klasse *EpisodeBuilder*.

Die erzeugten Bestandteile werden in Hashtables des Pakets *java.util* gespeichert und zu einer Datenstruktur zusammengefügt. Ein Objekt der Klasse *Model* des Pakets *HUMLDatastructure.HUMLActionStateDiagram* bildet die Wurzel dieser Datenstruktur. Dieses Objekt wird zu einem späteren Zeitpunkt an die Klasse *HISLANGTranslator* des Pakets *HUMLHISLANG* weitergegeben.

ANTLRCommentLexer Die Klasse wurde mit Hilfe des Parsergenerators ANTLR erstellt und zerlegt den Eingabezeichenstrom (eine UML-Annotation im UML-Quelldiagramm) in einen Tokenstrom, der von einer Instanz der Klasse *ANTLRCommentParser* weiterverarbeitet wird.

ANTLRCommentParser Die Klasse wurde mit Hilfe des Parsergenerators ANTLR erstellt und transformiert den Tokenstrom, der ihr durch eine Instanz der Klasse *ANTLRCommentLexer* zugänglich gemacht wird, in ein Objekt der Klasse *CommentInfo*.

ANTLRCommentParserTokenTypes Das Inferface wurde von ANTLR automatisch erzeugt und wird von den *ANTLRComment**-Klassen implementiert. Das Interface macht beiden Klassen die Tokendefinitionen zugänglich.

CommentInfo Die Klasse beschreibt die Menge aller für HUML relevanten Informationen, die ein Modellierer per UML-Annotation an Aktionszuständen und Maschinenknoten spezifizieren kann. Die Klasse *ANTLRCommentParser* erzeugt für jede Annotation ein Objekt der Klasse. *CommentInfo*-Objekte werden durch die Klassen *NodeBuilder* und *HardwareResourceBuilder* verwendet, um *Node*- und *HardwareResource*-Objekte des Pakets *HUMLDatastructure.HUMLActionStateDiagram* zu erzeugen.

DistributionValueInfo Die Klasse *DistributionValueInfo* wird von *ANTLRCommentParser* genutzt, um einen Verteilungswert zu repräsentieren. Diese Klasse beschreibt eine Verteilung (gleichverteilt oder exponentialverteilt, da die PG438 sich auf diese beiden Verteilungen beschränkt hat) und eine Zeitinformation.

Host Die Klasse *Host* wird vom *ANTLRCommentParser* genutzt, um alle Informationen, die in UML-Annotationen unter dem Eigenschaftswert PAextOp bezüglich einer passiven Ressource gefunden werden, zu repräsentieren, d.h. den Namen der passiven Ressource und die Anzahl der an sie gerichteten Aufrufe oder die Gesamtzugriffszeit.

EpisodeBuilder Die Klasse wird von der Klasse *XMITranslator* aufgerufen und verarbeitet einen Teilbaum der Java-XML-Repräsentation, der einem Aktivitätsdiagramm entspricht, zu einem Objekt der Klasse *Episode* des Pakets *HUMLDatastructure.HUMLActionStateDiagram*.

NodeBuilder Die Klasse bewertet einen Teilbaum der Java-XML-Repräsentation hinsichtlich seines Typs und ruft entweder eine Methode der Klasse *ActionStateBuilder* oder der Klasse *PseudoStateBuilder* auf. Darüberhinaus gibt sie das von der jeweiligen Methode erzeugte *Node*-Objekt des Pakets *HUMLDatastructure.HUMLActionStateDiagram* zurück. *NodeBuilder* wird von *XMITranslator* aufgerufen.

ActionStateBuilder Die Klasse verarbeitet einen Teilbaum der Java-XML-Repräsentation zu einem Objekt der Klasse *ActionState* des Pakets *HUMLDatastructure.HUMLActionStateDiagram*. Sie wird von der Klasse *NodeBuilder* aufgerufen und verwendet Informationen eines *CommentInfo*-Objekts.

PseudoStateBuilder Die Klasse verarbeitet einen Teilbaum der Java-XML-Repräsentation zu einem Objekt der Klasse *PseudoState* des Pakets *HUMLDatastructure*. Sie wird von der Klasse *NodeBuilder* aufgerufen.

HardwareResourceBuilder Die Klasse bewertet einen Teilbaum der Java-XML-Repräsentation hinsichtlich seines Typs und ruft entweder eine Methode der Klasse *ActiveResourceBuilder* oder der Klasse *PassiveResourceBuilder* auf. Darüberhinaus gibt sie das von der jeweiligen Methode erzeugte *HardwareResource*-Objekt des Pakets *HUMLDatastructure.HUMLActionStateDiagram* zurück. *HardwareResourceBuilder* wird von *XMITranslator* aufgerufen.

ActiveResourceBuilder Die Klasse verarbeitet einen Teilbaum der Java-XML-Repräsentation zu einem Objekt der Klasse *ActiveResource*. Sie wird von der Klasse *HardwareResourceBuilder* aufgerufen und verwendet Informationen eines *CommentInfo*-Objekts.

PassiveResourceBuilder Die Klasse verarbeitet einen Teilbaum der Java-XML-Repräsentation zu einem Objekt der Klasse *PassiveResource*. Sie wird von der Klasse *HardwareResourceBuilder* aufgerufen und verwendet Informationen aus einem *CommentInfo*-Objekt.

TransitionBuilder Die Klasse wird von *XMITranslator* aufgerufen und erzeugt zwischen zwei Objekten der Klasse *Node* des Pakets *HUMLDatastructure.HUMLActionStateDiagram* eine Transition der Klasse *Transition* des Pakets *HUMLDatastructure.HUMLActionStateDiagram*. Dazu werden die Klassen *TransitionLexer* und *TransitionParser* verwendet, die eine UML-Kantenbeschriftung auf das Auftreten einer Übergangswahrscheinlichkeit hin analysieren.

TransitionLexer Die Klasse wurde mit Hilfe des Parsergenerators ANTLR erstellt und zerlegt den Eingabezeichenstrom (eine UML-Kantenbeschriftung) in einen Tokenstrom, der von einer Instanz der Klasse *TransitionParser* weiterverarbeitet wird.

TransitionParser Die Klasse wurde mit Hilfe des Parsergenerators ANTLR erstellt und transformiert den Tokenstrom, der ihr durch eine Instanz der Klasse *TransitionLexer* zugänglich gemacht wird, in einen String, der das für eine Übergangswahrscheinlichkeit durch HUML reservierte Schlüsselwort und den dazugehörigen Wert enthält.

TransitionParserTokenTypes Das Inferface wurde von ANTLR automatisch erzeugt und wird von den Klassen *TransitionLexer* und *TransitionParser* implementiert. Das Interface macht beiden Klassen die Tokendefinitionen zugänglich.

WorkloadBuilder Die Klasse wird von der Methode *traverseModelComments* aus der Klasse *XMITranslator* aufgerufen. Sie entscheidet anhand der Einträge in dem übergebenen *CommentInfo*, ob eine offene Arbeitslast der Klasse *OpenLoad* oder eine geschlossene Arbeitslast der Klasse *ClosedLoad* erstellt werden muss. Für diese Entscheidung werden die Methoden *isWorkloadOpenLoad()* und *isWorkloadClosedLoad()* des *CommentInfo*-Objekts genutzt. Ist entweder ein neues Objekt der Klasse *OpenLoad* oder *ClosedLoad* erstellt worden, wird dieses Objekt mit den vorhandenen Daten aus dem *CommentInfo*-Objekt gefüllt. Das erstellte *Workload*-Objekt wird zurückgegeben.

EvaluationBuilder Der *EvaluationBuilder* erzeugt ein Evaluationsobjekt zu einer Maschine (*HardwareEvaluation*) oder zu einem Aktionszustand (*ActionEvaluation*).

Die Methode *buildHardwareEvaluation()* dieser Klasse wird von der Klasse *HardwareResourceBuilder* aufgerufen. Es wird ihr ein *CommentInfo*-Objekt und ein *HardwareResource*-Objekt übergeben. In der Methode wird ein neues *HardwareEvaluation*-Objekt erzeugt. Anhand des *CommentInfo*-Objekts werden die Attribute des *HardwareEvaluation*-Objekts gesetzt. Dem *HardwareEvaluation*-Objekt wird auch die übergebene *HardwareResource* zugeordnet. Das neu erstellte *HardwareEvaluation*-Objekt wird schließlich zurück gegeben.

Die Methode *buildEvaluationAction()* des *EvaluationBuilder* wird von der Klasse *ActionStateBuilder* aufgerufen. Ihr wird ein *CommentInfo*-Objekt und ein *ActionState*-Objekt übergeben. Es wird ein neues Objekt der Klasse *ActionEvaluation* erstellt. Anhand der Attribute im *CommentInfo*-Objekt werden die Attribute des *ActionEvaluation*-Objektes gefüllt. Außerdem wird das übergebene *ActionState*-Objekt explizit dem *ActionEvaluation*-Objekt zugeordnet. Das neu erzeugte und gefüllte *ActionEvaluation*-Objekt wird zurückgegeben.

GraphProcessor Die Methode *processGraph()* wird von der Methode *build()* der Klasse *XMITranslator* aufgerufen. Sie bekommt die initiale Episode des aus der XMI-Datei erstellten Graphen übergeben. Bei einem rekursiven Durchlauf durch die initiale Episode werden für

die Leistungsbewertung irrelevante Knoten aus dem Graphen entfernt oder umgewandelt, Übergangswahrscheinlichkeiten zu nachfolgenden Knoten berechnet und alle *Episode*-Objekte vervollständigt. Nach Beendigung von *processGraph()* stellt der durch die initiale Episode gekapselte Graph eine gültige Eingabe für die Übersetzung nach HISLANG dar.

2.5. Paket HUML

Das Paket HUML dient der Übersetzung der HUML-Datenstruktur in HI-SLANG. In diesem Paket ist eine Menge von Schnittstellen enthalten. Die Schnittstellen *Translator* und *IdentifierGenerator* werden von den Klassen *Translator* und *IdentifierGenerator* des Unterpakets HUML implementiert.

Translator Das Interface *Translator* deklariert Funktionen, die dafür sorgen, dass die Datenstruktur in den Code eines Programms in einer bestimmten Programmiersprache übersetzt wird. Im Rahmen des HUML-Projekts ist dies die Programmiersprache HI-SLANG, was durch die Implementierung des Interfaces *Translator* durch die Klasse *Translator* im Paket *HUML.HISLANG* realisiert wird.

IdentifierGenerator Das Interface *IdentifierGenerator* stellt für die HUML-Namensverwaltung, d.h die Klasse *NameService*, einen Generator für eindeutige symbolische Bezeichner innerhalb des Programmcodes einer bestimmten Programmiersprache zur Verfügung.

Output Das Interface *Output* bietet die Möglichkeit, in eine Datei zu schreiben.

LogOutput Das Interface *LogOutput* stellt eine Methode zur Verfügung, mit der Fehler- und Erfolgsmeldungen in eine Log-Datei geschrieben werden können.

NameService Das Interface *NameService* stellt die Möglichkeit zur Verfügung, Namen für z.B. *Evaluation*-Objekte oder *HardwareResource*-Objekte zu erzeugen.

Translatable Das Interface *Translatable* deklariert die Methode *translate()*. Jede Klasse innerhalb des Paketes *HUMLDatastructure* erbt von diesem Interface und kann so übersetzt werden.

2.5.1. Paket HISLANG

Im Paket *HUML.HISLANG* sind alle Funktionalitäten enthalten, die spezifisch für die Erzeugung von HI-SLANG-Code sind. Es enthält die Klasse *Translator*, die das Interface *Translator* implementiert, in der zahlreiche Übersetzungsmethoden für die diversen Komponenten von UML-Modellen deklariert sind, und *IdentifierGenerator*, die das Interface *IdentifierGenerator* implementiert, das Funktionen zur Erzeugung von symbolischen Bezeichnern enthält.

Translator Die Klasse *Translator* implementiert das Interface *HUML.Translator*. Die Implementierung realisiert die Methoden des Interfaces so, dass sie die Komponenten der Datenstruktur in HI-SLANG-Programmcode übersetzen. Da im zu erzeugenden HI-SLANG-Programmcode Einrückungen enthalten sein sollen, enthält die Klasse *Translator* Funktionen, durch die jede Programmzeile mit der erwünschten Einrückung versehen wird. Folglich muss nach einer Einrückung eine Zeichenkette ausgegeben werden können. Für die Erzeugung und Ausgabe der HI-SLANG-Zeichenkette werden Methoden aus der Klasse *HUML.Output* aufgerufen. Zusätzlich wird der Ausgabezeichenstrom sowie die Zwischenspeicherung von Referenzen auf diejenigen Instanzen, die selbstdefinierte Komponenten, Standard(hardware-)komponenten und Evaluationsobjekte des HIT-Modells repräsentieren, erzeugt. Weiterhin veranlasst die Klasse *Translator* den Durchlauf des HIT-Modellgraphen und die Generierung aller selbstdefinierten Komponententypen, Standardkomponententypen und Evaluationsobjekte. Die Evaluationsobjekte eines HIT-Modells repräsentieren die Stereotypisierungen von Aktionszuständen in Aktivitätsdiagrammen und Maschinenknoteninstanzen in Verteilungsdiagrammen des UML-Modells mit dem Stereotyp «HUMLresult». Damit können im Experimentblock des HI-SLANG-Programmcodes die erwünschten Evaluationsblöcke erzeugt und definiert werden. In der Klasse *Translator* werden die passiven und aktiven Ressourcen erzeugt und separat behandelt. Desweiteren stehen Funktionen für offene Systeme zur Verfügung, durch die die Topologie eines Aktivitätsdiagramms (oder bei nebenläufigen Prozessen eines einzelnen Aktionsstrangs) auf einen OPEN_CHAIN-Block innerhalb der Definition des Dienstes der obersten HIT-Modellebene bzw. eines angebotenen Dienstes eines selbstdefinierten Komponententyps abgebildet werden kann. Im Falle von geschlossenen Systemen, d.h. wenn die Arbeitslast im UML-Modell durch das Stereotyp «PAclosedLoad» definiert wird, werden entsprechende Funktionen benutzt, um die Topologie des obersten Aktivitätsdiagramms auf einen CLOSED_CHAIN-Block innerhalb der Definition des Dienstes der obersten HIT-Modellebene abzubilden.

IdentifierGenerator Die Klasse *IdentifierGenerator* implementiert das Interface *HUML.IdentifierGenerator*, so dass die zu realisierenden Funktionen eindeutige symbolische Bezeichner erzeugen, die in der Syntax von HI-SLANG gültig sind. Diese Klasse beinhaltet Methoden, die zur Erzeugung eindeutiger Bezeichner durch Weiterzählen eines numerischen Namensanteils, genutzt werden.

2.6. Paket HUMLDumpfileTranslator

Das Paket dient der Integration der Ergebnisse einer HIT-Ergebnisdatei (dumpfile) in die XMI-Eingabe des exportierten UML-Modells, um eine, um die ermittelten Messwerte erweiterte, XMI-Repräsentation des Eingabemodells zu erhalten. Es werden drei, mit dem Compilergenerator ANTLR erzeugte, Klassen verwendet: *ANTLRDumpfileLexer*, *ANTLRDumpfileParser* und *ANTLRDumpfileParserTokenTypes*. Diese werden für die Analyse der Ergebnisdatei verwendet und geben ein Objekt der Klasse *EvaluationObjectInfo* zurück.

Die Hauptklasse dieses Pakets ist die Klasse *DumpfileTranslator*, die den Übersetzungsvorgang durch die ANTLR-Klassen startet und darüber hinaus die Erweiterung der XMI-Eingabedatei um die ermittelten Messergebnisse vornimmt.

ANTLRDumpfileLexer Die Klasse zerlegt den Eingabestrom (dumpfile) in einen Tokenstrom, der von der Klasse *ANTLRDumpfileParser* weiterverarbeitet wird.

ANTLRDumpfileParser Die Klasse verarbeitet den Tokenstrom (bereitgestellt durch ein Objekt der Klasse *ANTLRDumpfileLexer*) zu einem Objekt der Klasse *EvaluationObjectInfo*.

ANTLRDumpfileParserTokenTypes Das Interface enthält alle in der dumpfile-Grammatik spezifizierten Tokens und wird von *ANTLRDumpfileLexer* und *ANTLRDumpfileParser* implementiert.

EvaluationObjectInfo Die Klasse modelliert einen Container für die durch HIT ermittelten Messergebnisse. Die Werte werden als Tripel (simulatorisches Lösungsverfahren: Messwert, Standardabweichung und Confidence-Intervall) bzw. als einzelner Wert (analytisches oder numerisches Lösungsverfahren: Messwert) in Objekten der Klasse *HUMLresultContainer* verwaltet.

DumpfileTranslator Die Klasse ist die Hauptklasse der Übersetzung der Ergebnisdatei. Sie erzeugt Instanzen von *ANTLRDumpfileLexer* und *ANTLRDumpfileParser* und startet die Übersetzung. Anschließend verarbeitet sie die vom Parser erzeugte Instanz der Klasse *EvaluationObjectInfo*, indem die einzelnen *HUMLresultContainer*-Objekte ausgelesen und ihre Bestandteile an die jeweilige Stelle in der Java-Baumrepräsentation der XMI-Eingabe geschrieben werden. Den Abschluss bildet die Verwendung der Klasse *XMLWriter* des Pakets *NanoXML*, die die Java-Baumstruktur in eine XMI-Zieldatei transformiert.

HUMLresultContainer Die Klasse verwaltet einen HIT-Messwert und für den Fall, dass ein simulatorisches Lösungsverfahren von HIT verwendet wurde, die Standardabweichung und das Konfidenz-Intervall.

2.7. Paket ANTLR

Das Paket *antlr* enthält alle Pakete und Klassen, die die von ANTLR erzeugten Parser- und Lexer-Klassen zur Ausführung benötigen. Das Packet enthält die Klasse *CharScanner* und *LLkParser*, von denen die erzeugten Parser- und Lexer-Klassen erben.

2.8. Paket NanoXML

Das Paket stellt einen XML-Parser zur Verfügung, der im Rahmen des HUML-Werkzeugs zur Analyse von UML-Modellen eingesetzt wird, die zuvor durch ein UML-Modellierungswerkzeug exportiert wurden. Die Klassen *XMLLexer* und *XMLParser* verarbeiten eine XMI-Eingabe zu einer Java-Baumrepräsentation, die ausschließlich aus Objekten der Klasse *XMLElement* besteht. Dies ist möglich, da XMI ein XML-Dialekt ist.

Die Klasse *XMLWriter* kann eine solche Repräsentation in eine XMI-Datei transformieren. Diese Eigenschaft wird durch das Werkzeug HUML verwendet, um einen um Messergebnisse erweiterten Java-Baum in eine XMI-Zieldatei zurückzuschreiben.

XMLLexer Die Klasse zerlegt den Eingabestrom der XMI-Datei in einen Tokenstrom, der von der Klasse *XMLParser* weiterverarbeitet wird.

XMLParser Die Klasse analysiert den Tokenstrom (von einem Objekt der Klasse *XMLLexer* bereitgestellt) und erzeugt eine Java-Baumrepräsentation wobei jedes *XMI-Tag* durch ein Objekt der Klasse *XMLElement* dargestellt wird.

XMLElement Die Klasse repräsentiert ein *XMI-Tag* in der Java-Baumrepräsentation. Ein Objekt trägt einen Namen (Name des *XMI-Tags*) und hat u.a. eine Menge von Attributen (Attribute des *XMI-Tags*) und eine Menge von Kindern (untergeordnete *XMI-Tags*). Weitere Bestandteile der Klasse werden vom HUML-Werkzeug nicht verwendet.

XMLWriter Die Klasse übersetzt eine Baumrepräsentation eines XMI-Dokuments in ein XMI-Dokument. Dazu wird ihr die Wurzel der Java-Baumstruktur übergeben. Die Klasse wird eingesetzt, um die um die Messwerte erweiterte Java-Struktur in eine XMI-Zieldatei zu transformieren.

3. Produkttest

Zum Testen des HUML-Werkzeugs wurden unterschiedliche xmi-Dateien erstellt, die verschiedene Aspekte der Leistungsbewertung beinhalten.

Zuerst wurde HUML mit der xmi-Datei des Bürosystems aus Kapitel 1.8 getestet (siehe Kapitel 3.1). Das Bürosystem beinhaltet aber nicht alle Modellierungstechniken, die HUML unterstützt. Es fehlt die Modellierung einer Parallelität und die Modellierung einer Verzweigung. Es wurde für jede dieser Modellierungstechniken ein seperates UML-Modell erstellt und die aus diesen Modellen resultierenden xmi-Dateien getestet (siehe Kapitel 3.2 und 3.3).

Um die korrekte Behandlung von Fehlern und Fehleingaben zu testen, wurden weitere xmi-Dateien erzeugt, die verschiedene Fehleingaben beinhalteten (siehe Kapitel 3.4).

3.1. Test des Bürosystems

Das Bürosystem beinhaltet passive und aktive Ressourcen, Subaktivitätsdiagramme und eine geschlossene Arbeitslast. Die Bilder des Verteilungsdiagramms und die des Aktivitätsdiagramms zu dem Bürosystem sind im Kapitel 1.8 zu finden und werden hier nicht nochmal aufgeführt.

Die Lösung zum Vergleich der Korrektheit zu dem Bürosystem wurden schon im Modellierungspraktikum berechnet (siehe Zwischenbericht [PG438 (2004)]). In Tabelle 3.1(zusammen mit Tabelle 3.2 auf der folgenden Seite) werden diese Ergebnisse aus dem Modellierungspraktikum nochmals aufgeführt.

Das System aus dem Modellierungspraktikum wurde in UML modelliert und mit den entsprechenden Performanz-Annotationen versehen. Die UML Diagramme wurden in einer xmi-Datei gespeichert und dienten als Eingabe für das HUML Werkzeug. Nach der Leistungsbewertung durch HUML, ergaben sich folgende Ergebnisse, die in Tabelle 3.2 zu sehen sind.

Beim Vergleich der Ergebnisse aus dem Modellierungspraktikum mit den Ausgaben von HUML fällt auf, dass Abweichungen vorhanden sind. Es besteht lediglich eine geringe prozentuale Abweichung dieser Werte voneinander. Die Abweichungen treten auf, da als Lösungsverfahren die simulative Berechnung gewählt wurde. Bei dem simulativen Lösungsverfahren weichen die Ergebnisse immer etwas voneinander ab. Also ist aus dem Test zu schließen, dass HUML die korrekten Ergebnisse berechnet, die auch im Modellierungspraktikum ermittelt wurden, bis auf eine leichte Abweichung aufgrund der simulativen Berechnung.

Name	Population	Throughput	Turnaroundtime	Utilization
vorbereiten	9.857501	0.002590	3806.462687	–
zusammenstellen	3.285834	0.002590	1268.820896	–
ueberpruefen	4.978817	0.002590	1922.564451	–
beenden	1.877848	0.002590	725.128901	–
Netzwerkverbindung	1.198960	0.191636	6.256445	0.574908
SAP_1	2.071740	0.517935	4.0	0.103587
SAP_2	0.155380	0.077690	2.0	0.007769

Tabelle 3.1.: Lösungen der Leistungsbewertung für das Bürosystem aus dem Modellierungspraktikum

Name	Population	Throughput	Turnaroundtime	Utilization
vorbereiten	9.954683	0.002534725	3918.494	–
zusammenstellen	3.274935	0.002533602	1291.921	–
ueberpruefen	4.940725	0.002532104	1949.159	–
beenden	1.829658	0.002531355	722.5931	–
Netzwerkverbindung	1.185217	0.1905039	6.221495	0.5711576
SAP_1	2.089592	0.5219025	4.003793	0.1048182
SAP_2	0.1533545	0.07695095	1.992896	0.007706418

Tabelle 3.2.: Lösungen der Leistungsbewertung für das Bürosystem durch HUML

3.2. Test mit Parallelität

Zum Testen der Modellierung von Parallelität wurde ein Aktivitätsdiagramm erzeugt, das ein *fork* und ein *join* enthält. Dieses Aktivitätsdiagramm ist in Abbildung 3.1 zu sehen. In dem vorherigen Test wurde ein System mit einer geschlossenen Arbeitslast getestet. Um die richtige Übersetzung einer Spezifikation einer offenen Arbeitslast in ein HI-SlANG-Modell zu testen, wird dieses Test-System mit einer offenen Arbeitslast modelliert.

Desweiteren wurde ein Verteilungsdiagramm (siehe Abbildung 3.2) erstellt, das die beiden Ressourcen enthält, auf die die Aktivitäten des Aktivitätsdiagramms zugreifen.

Aufgrund der Synchronisation durch *join* können die theoretischen Lösungen nur näherungsweise bererchnet werden. Da beide Maschinen dieselben Leistungsparameter haben, wird im Fogenden nur die Lösungen für 'Maschine1' berechnet. Die Lösungen der 'Maschine2' sind identisch.

Bedienrate (spezifiziert durch PArate/ PAdemand) = 1000/1 = 1000
Ankunftsrate (PAoccurence gibt an, wann Prozesse ankommen: alle 0.002 s) = 1/0.002 = 500
Ankunftsrate (spezifiziert durch PAoccurence) = 0.002

Population: = Ankunftsrate/Turnaroundtime = 0.002/0.002 = 1.0
Utilization (Auslastung): Ankunftsrate/ Bedienrate = 500 / 1000 = 0.5
Wartezeit = (Utilization/Bedienrate)/(1-Utilization) = (0.5/1000)/(1-0.5) = 0.001
Turnaroundtime (Durchlaufzeit): Wartezeit + (1/Bedienrate) = 0.001 + (1/1000) = 0.002
Throughput (Durchsatz): Utilization · Bedienrate = 0.5 · 1000 = 500

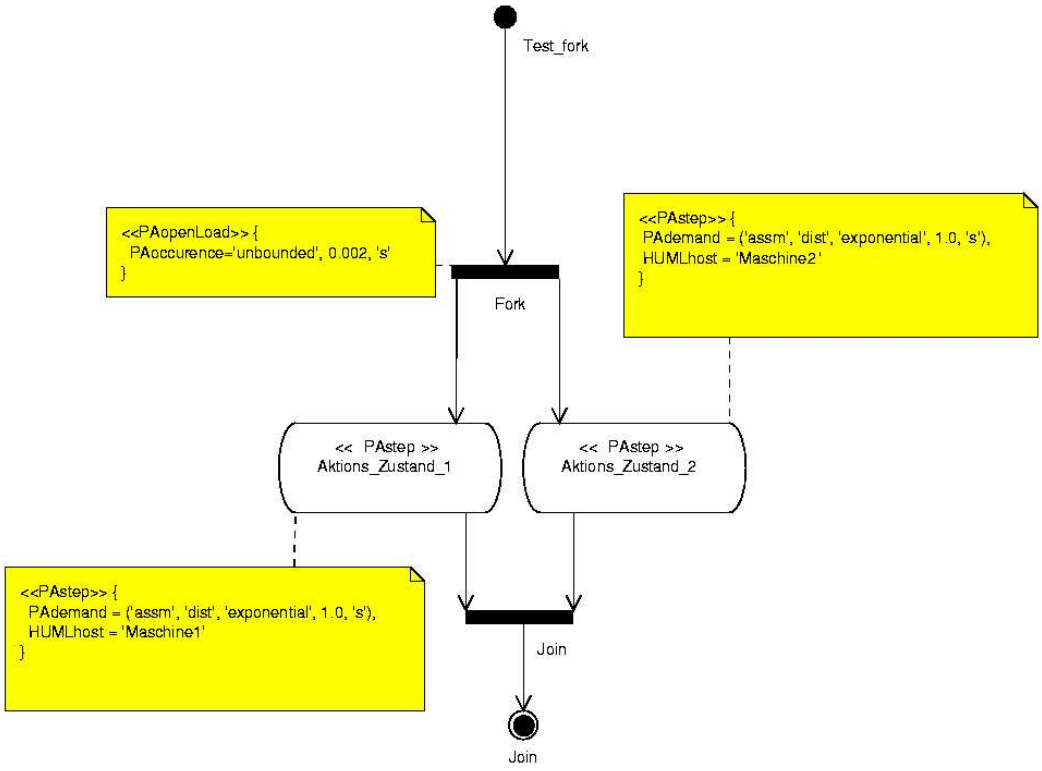

Abbildung 3.1.: Aktivitätsdiagramm des Parallelitäts-Test

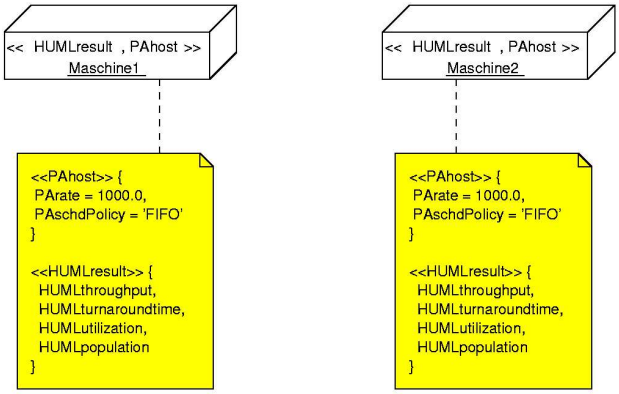

Abbildung 3.2.: Verteilungsdiagramm des Parallelitäts-Tests

Nach der Leistungsbewertung mit HUML/HIT wurden folgende, in Tabelle 3.3 aufgelisteten, Ergebnisse an den Ressourcen 'Maschine1' und 'Maschine2' ermittelt.

Leistungsparameter	Maschine1	Maschine2
Population	1.005019	1.007340
Throughput	500.003583	500.008168
Turnaroundtime	0.002008616	0.002011411
Utilization	0.5023523	0.5025203

Tabelle 3.3.: Lösungen der Leistungsbewertung für den Parallelitäts-Test

3.3. Test mit Verzweigung

Zum Testen der Modellierung von Verzweigungen wurde ein Aktivitätsdiagramm erzeugt, das ein *branch* enthält. Dieses Aktivitätsdiagramm ist in Abbildung 3.3 zu sehen.

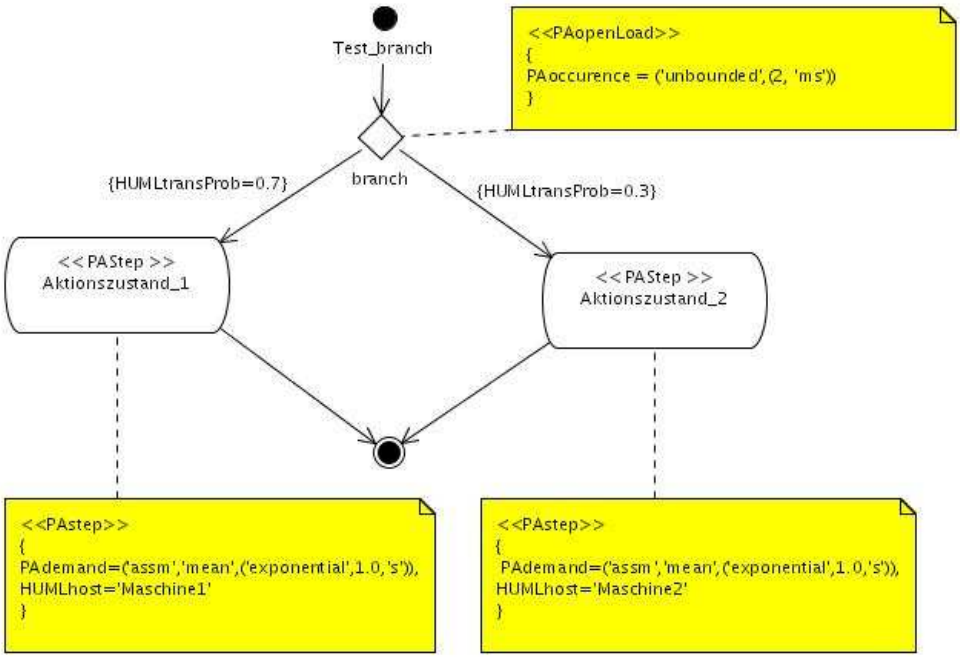

Abbildung 3.3.: Aktivitätsdiagramm des Verzweigungs-Test

Desweiteren wurde ein Verteilungsdiagramm (siehe Abbildung 3.4) erstellt, das zwei aktive Ressourcen enthält, die die Prozesse des Aktivitätsdiagramms bearbeiten.

Die Leistungsbewertung der *Maschine1* und die *Maschine2* können folgendermaßen theoretisch berechnet werden:

Bedienrate: PArate / PAdemand = 1000 / 1 = 1000 Pakete / sec
Ankunftsrate: PAoccurence = 0.002 sec = 500 Pakete / sec

Theoretische Ergebnisse für Maschine1 Utilization (Auslastung) = (Ankunftsrate / Bedienrate) · HUMLtransProb = (500 / 1000) · 0,7 = 0.35

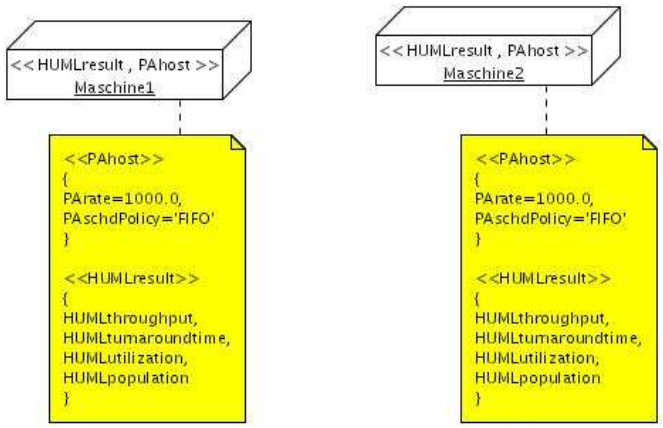

Abbildung 3.4.: Verteilungsdiagramm des Verzweigungs-Test

Wartezeit = (Utilization / Bedienrate) / (1 - Utilization) = (0.35 / 1000) / (1 - 0.35) = 0.00054
Turnaroundtime (Durchlaufzeit) = Wartezeit + (1 / Bedienrate) = 0.00054 + (1 / 1000) =0.00154
Throughput (Durchsatz) = Utilization · Bedienrate = 0.35 · 1000 = 350

Theoretische Ergebnisse für Maschine2 Utilization (Auslastung) = (Ankunftsrate / Bedienrate) · HUMLtransProb = (500 / 1000) · 0,3 = 0.15
Wartezeit = (Utilization / Bedienrate) / (1 - Utilization) = (0.15 / 1000) / (1 - 0.15) = 0.00018
Turnaroundtime (Durchlaufzeit) = Wartezeit + (1 / Bedienrate) = 0.00018 + (1 / 1000) = 0.00118
Throughput (Durchsatz) = Utilization · Bedienrate = 0.15 · 1000 = 150

Leistungsparameter	**Maschine1**	**Maschine2**
Throughput	350	150
Turnaroundtime	0.0014	0.0006
Utilization	0.35	0.15

Tabelle 3.4.: Theoretische Lösungen der Leistungsbewertung für den Verzweigungs-Test

Nach der Leistungsbewertung mit HUML und HIT wurden folgende, in Tabelle 3.4 aufgelisteten, Ergebnisse an den Ressourcen 'Maschine1' und 'Maschine2' ermittelt.

Leistungsparameter	**Maschine1**	**Maschine2**
Throughput	350.4066	149.5942
Turnaroundtime	0.001153	0.001053
Utilization	0.35138	0.14979

Tabelle 3.5.: Lösungen der Leistungsbewertung für den Verzweigungs-Test

Daraus ist zu erkennen, dass HUML die richtigen Ergebnisse berechnet, die auch auf theoretischem Wege ermittelt wurden.

3.4. Test mit Fehleingaben in der xmi-Datei

Zum Testen der Fehlermeldungen wurde HUML zunächst mit der xmi-Datei eines einfachen Aktivitätsdiagramms getestet, in dem eine Arbeitslast in einem offenen System modelliert wird (siehe Abbildung 3.5). In diesem offenen System, das durch «PAopenLoad» stereotypisiert wird, wird anstelle des korrekten Eigenschaftswerts PAoccurence der ungültige Eigenschatswert PApopulation definiert.

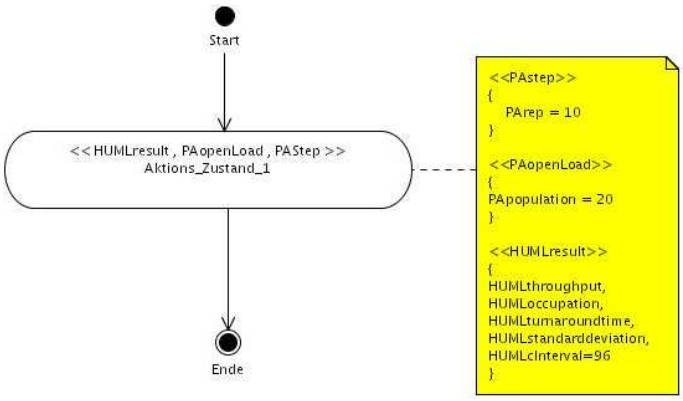

Abbildung 3.5.: Aktivitätsdiagramm mit Fehleingabe 'PApopulation'

HUML erkennt diese Fehleingabe und erzeugt eine Fehlermeldung. Nach dem Importieren der fehlerhaften xmi-Datei, wird in der erzeugten HUML-Log-Datei die Fehlermeldung angegeben, die in Abbildung 3.6 zu sehen ist. Folglich kann durch die Modellierung eines offenen Systems mit einer Spezifikation eines geschlossenen Systems, die Datenstruktur nicht erzeugt werden.

```
In dieser Log-Datei wird der ?bersetzungsvorgang kommentiert.

XMI-Datei wird importiert:
XMI-Import beendet

Datenstruktur wird erstellt:
Erfolg: Comments erfolgreich geparst
Erfolg: Stereotype erfolgreich eingetrage
Erfolg: Aktive Ressource erfolgreich erzeugt
Erfolg: Hardware Resource erzeugt.
Erfolg: NodeInstances eingetragen
Erfolg: Action Evaluation erzeugt
Erfolg: buildActionState durchgelaufen
Erfolg: buildActionState durchgelaufen
Fehler: keine Workload spezifiziert
Datenstruktur konnte nicht erstellt werden
```

Abbildung 3.6.: Die durch die Fehleingabe 'PApopulation' erzeugte HUML-Log-Datei

Anschließend wurde die xmi-Datei eines Aktivitätsdiagramms (Abbildung 3.7) überprüft, in das ein geschlossenes System zwar modelliert, jedoch mit dem Eigenschaftswert PAoccurence spezifiziert wurde.

Die nach dem Importieren der entsprechenden xmi-Datei erzeugte HUML-Log-Datei ist in Abbildung 3.8 zu sehen. HUML meldet das Fehlen einer Population für den Stereotyp

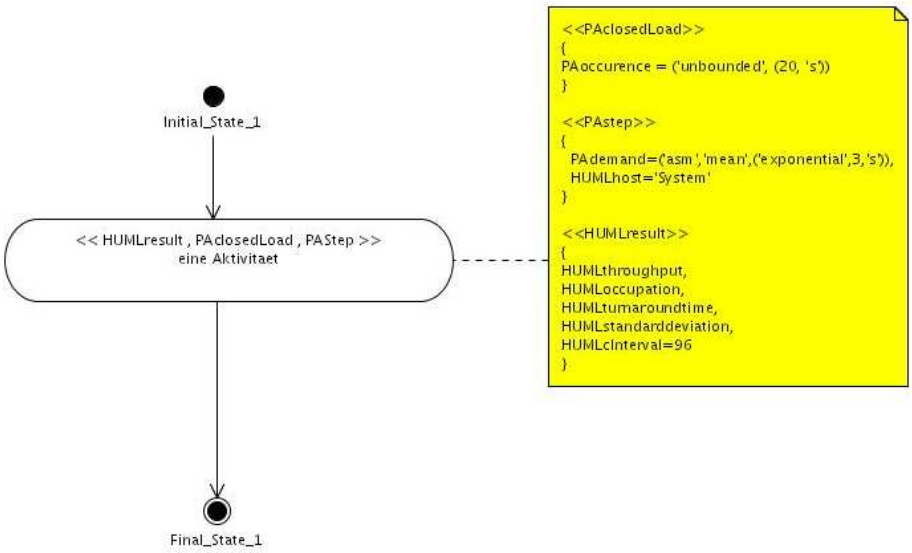

Abbildung 3.7.: Aktivitätsdiagramm mit Fehleingabe 'PAoccurence'

«PAclosedLoad».

```
In dieser Log-Datei wird der ?bersetzungsvorgang kommentiert.

XMI-Datei wird importiert:
XMI-Import beendet

Datenstruktur wird erstellt:
Fehler: keine Population bei geschlossener Arbeitslast angegeben!
Erfolg: Comments erfolgreich geparst
Erfolg: Stereotype erfolgreich eingetrage
Erfolg: Aktive Ressource erfolgreich erzeugt
Erfolg: Hardware Resource erzeugt.
Erfolg: NodeInstances eingetragen
Erfolg: Action Evaluation erzeugt
Erfolg: buildActionState durchgelaufen
Fehler: keine Workload spezifiziert
Datenstruktur konnte nicht erstellt werden
```

Abbildung 3.8.: Die durch die Fehleingabe 'PAoccurence' erzeugte HUML-Log-Datei

Als Nächstes wurde die xmi-Datei eines Aktivitätsdiagramms (siehe Abbildung 3.9) getestet, in der ein Verweis auf eine aktive Ressource mittels HUMLhost enthalten ist, jedoch eine passive Ressource mittels PAresource im Verteilungsdiagramm (siehe hierzu Abbildung 3.10) spezifiziert wird.

Da HUML die falsche Spezifikation erkennt, konnte somit auch die Datenstruktur nicht erzeugt werden. Die generierte HUML-Log-Datei, in der u.a. die Fehlermeldungen abgebildet werden, ist in Abbildung 3.11 zu sehen.

Schließlich wird die xmi-Datei eines Aktivitätsdiagramms (siehe Abbildung 3.12) überprüft, in dem mittels PAextOp auf eine passive Ressource verwiesen wird, jedoch im Verteilungsdiagramm eine aktive Ressource (Abbildung 3.13) mittels «PAhost» spezifiziert wird.

Die durch HUML erkannte Fehlermeldung wird in Abbildung 3.14 dargestellt.

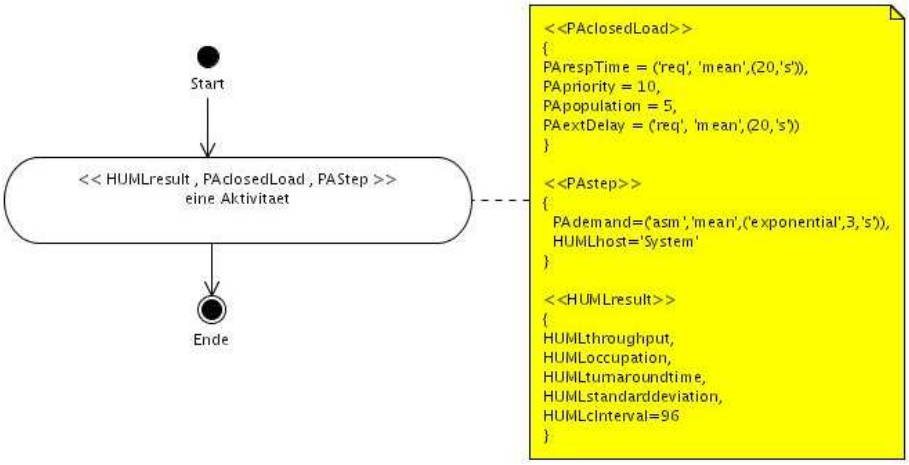

Abbildung 3.9.: Aktivitätsdiagramm mit Fehleingabe 'HUMLhost'

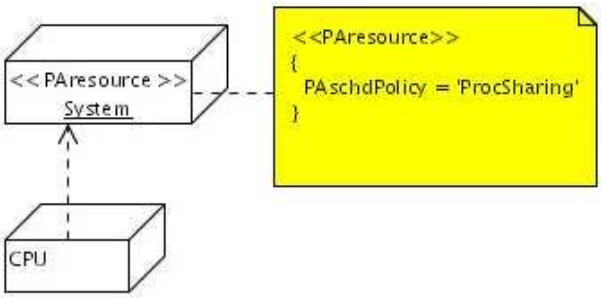

Abbildung 3.10.: Verteilungsdiagramm mit Fehleingabe 'PAresource'

```
In dieser Log-Datei wird der ?bersetzungsvorgang kommentiert.

XMI-Datei wird importiert:
XMI-Import beendet

Datenstruktur wird erstellt:
Erfolg: Comments erfolgreich geparst
Erfolg: Stereotype erfolgreich eingetrage
Erfolg: Passive Hardware Ressource erstellt.
Erfolg: Hardware Resource erzeugt.
Erfolg: NodeInstances eingetragen
Fehler: Es existiert keine aktive Ressource mit dem Namen System, nur eine passive Ressource.
Fehler: traverseAGNodes fehlerhaft
Fehler: in der Methode traverseActivityGraph
Datenstruktur konnte nicht erstellt werden
```

Abbildung 3.11.: Die durch die Fehleingabe 'HUMLhost' erzeugte HUML-Log-Datei

Als Nächstes wird die xmi-Datei eines Aktivitätsdiagramms getestet, in dem zwar ein Verweis auf ein weiteres Aktivitätsdiagramm mittels HUMLsubAct beschrieben wird, jedoch solches nicht exisitiert. Dieses Aktivitätsdiagramm, in dem nur der Verweis auf ein Aktivitätsdiagramm gegeben wird, ist in Abbidung 3.15 dargestellt.

HUML identifiziert diese Fehleingabe, die in der erzeugten HUML-Log-Datei (siehe Abbildung 3.16) zu lesen ist.

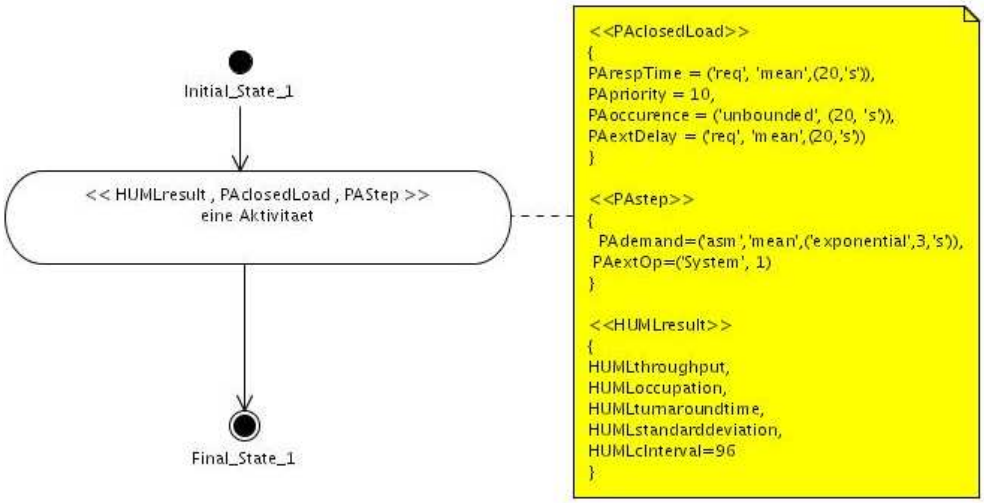

Abbildung 3.12.: Aktivitätsdiagramm mit Fehleingabe 'PAextOp'

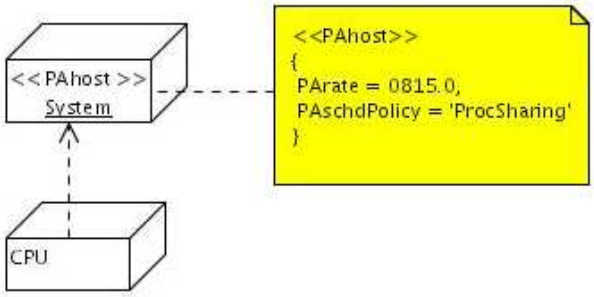

Abbildung 3.13.: Verteilungsdiagramm mit Fehleingabe 'PAhost'

```
In dieser Log-Datei wird der ?bersetzungsvorgang kommentiert.

XMI-Datei wird importiert:
XMI-Import beendet

Datenstruktur wird erstellt:
Erfolg: Comments erfolgreich geparst
Erfolg: Stereotype erfolgreich eingetrage
Erfolg: Passive Hardware Ressource erstellt.
Erfolg: Hardware Resource erzeugt.
Erfolg: NodeInstances eingetragen
Fehler: Es existiert keine aktive Ressource mit dem Namen System, nur eine passive Ressource.
Fehler: traverseAGNodes fehlerhaft
Fehler: in der Methode traverseActivityGraph
Datenstruktur konnte nicht erstellt werden
```

Abbildung 3.14.: Die durch die Fehleingabe 'PAextOp' erzeugte HUML-Log-Datei

Schließlich wird die Spezifikation multipler Arbeitslasten in einem Aktivitätsdiagramm (Abbildung 3.17) getestet. Wie in der Abbildung 3.18 zu sehen ist, erkennt HUML das Einführen mehrerer Arbeitslasten und gibt eine Fehlermeldung aus, da für HUML nur eine Arbeitslast spezifiziert werden darf.

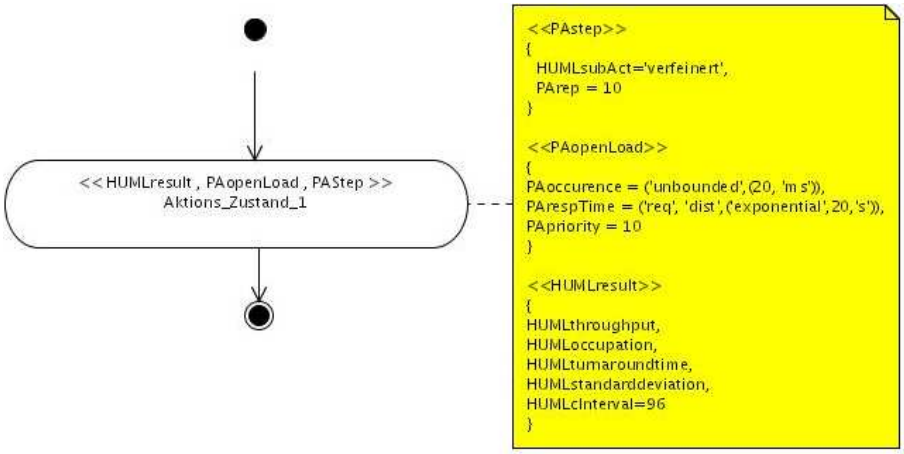

Abbildung 3.15.: Aktivitätsdiagramm mit Fehleingabe 'HUMLsubAct'

```
In dieser Log-Datei wird der ?bersetzungsvorgang kommentiert.

XMI-Datei wird importiert:
XMI-Import beendet

Datenstruktur wird erstellt:
Erfolg: Comments erfolgreich geparst
Erfolg: Stereotype erfolgreich eingetrage
Erfolg: Aktive Ressource erfolgreich erzeugt
Erfolg: Hardware Resource erzeugt.
Erfolg: NodeInstances eingetragen
Erfolg: Action Evaluation erzeugt
Erfolg: buildActionState durchgelaufen
Erfolg: buildActionState durchgelaufen
Fehler: Es existiert kein Aktivit?tsdiagramm mit dem Namen verfeinert
Datenstruktur konnte nicht erstellt werden
```

Abbildung 3.16.: Die durch die Fehleingabe 'HUMLsubAct' erzeugte HUML-Log-Datei

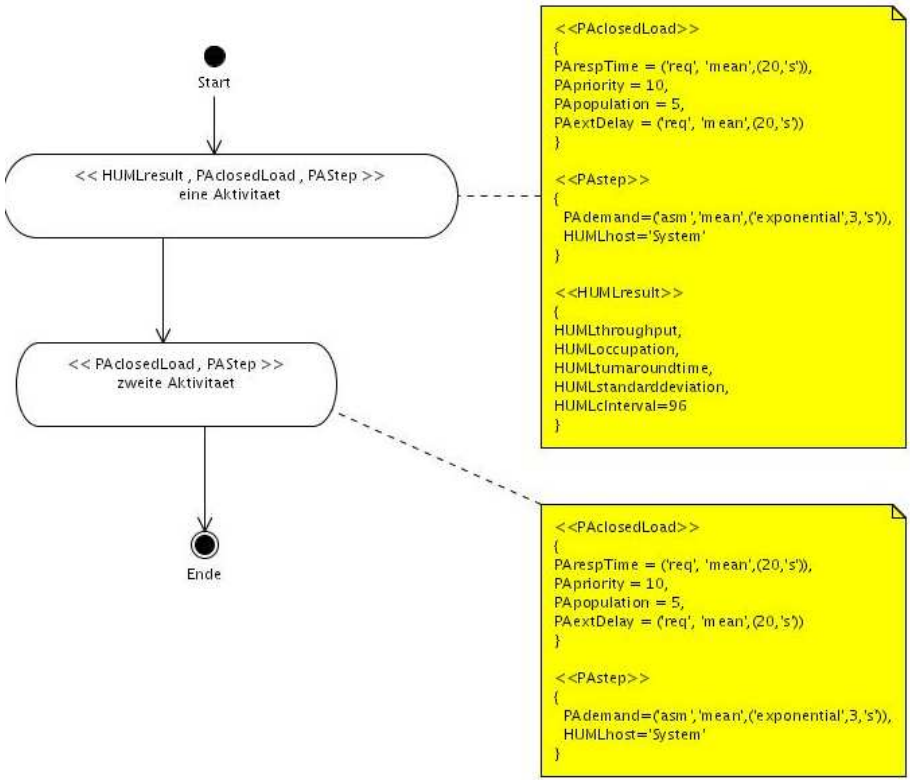

Abbildung 3.17.: Aktivitätsdiagramm mit multiplen Arbeitslasten

```
In dieser Log-Datei wird der Übersetzungsvorgang kommentiert.

XMI-Datei wird importiert:
XMI-Import beendet

Datenstruktur wird erstellt:
Fehler: mehrere Workloads wurden spezifiziert
Fehler: Comments fehlerhaft
Erfolg: Stereotype erfolgreich eingetrage
Erfolg: Aktive Ressource erfolgreich erzeugt
Erfolg: Hardware Resource erzeugt.
Erfolg: Aktive Ressource erfolgreich erzeugt
Erfolg: Hardware Resource erzeugt.
Erfolg: NodeInstances eingetragen
Fehler: in der Methode traverseModel
Datenstruktur konnte nicht erstellt werden
```

Abbildung 3.18.: Die durch multiple Arbeitslasten erzeugte HUML-Log-Datei

4. Benutzungshandbuch

4.1. Einführung

Dieses Benutzungshandbuch erklärt die Verwendung des von der PG438 entwickelten Softwarewerkzeuges HUML. Die Motivation für die Entwicklung von HUML wird im Zwischenbericht der PG438 vom WS03/04 [PG438 (2004)] besprochen.

Mit HUML steht Ihnen ein Werkzeug zur Unterstützung der Leistungsanalyse von sich noch in der Entwicklung befindlichen Softwaresystemen zur Verfügung. HUML ermöglicht es, den Performanzaspekt bei der Softwareentwicklung mittels UML zu berücksichtigen. Dazu werden UML-Modelle, die mit Leistungsparametern versehene Aktivitäts- und Verteilungsdiagramme enthalten, in Modelle des Performanzanalysewerkzeugs HIT übersetzt. Anschließend wird die Leistung des Softwaresystems, dargestellt durch das HIT-Modell, mittels des HIT-Werkzeuges berechnet und die Ergebnisse dieser Analyse in das Eingabe-UML-Modell eingetragen. HUML schlägt also eine Brücke zwischen der UML-Modellierung, bei der Performanzaspekte bisher unberücksichtigt bleiben, und einem Leistungsanalysewerkzeug. Die eigentliche Leistungsanalyse wird mit dem vom Informatik-Lehrstuhl 4 der Universität Dortmund entwickelten Leistungsbewertungs-Tool 'HIT' durchgeführt. Schließlich kann der Entwickler anhand der von HUML in das analysierte UML-Diagramm eingefügten Analyseergebnisse sein Softwareprodukt leistungsbezogen optimieren. Eine Leistungsbewertung ist damit noch vor der Implementierung des Entwurfs möglich.

Als UML-Werkzeug wird 'Poseidon for UML' [Gentleware] empfohlen, da HUML nur für dieses Werkzeug getestet wurde. **ACHTUNG: Bei der Erstellung von Diagrammen ist darauf zu achten, dass beim Entfernen von Modelelementen fehlerhafter XMI-Code erzeugt wird. Daher ist stets ein neues Modell zu erstellen. Weitere bisher unbekannte Besonderheiten, die zu einem für das Tool "HUML" unbrauchbaren XMI-Code führen, sind nicht auszuschließen.**

In dem folgenden Kapitel werden zunächst die Vorraussetzung für den Einsatz von HUML angegeben. Anschließend wird die Installation von HUML behandelt. Im Anschluss daran folgt die Erklärung der Benutzungsoberfläche. Im nächsten Kapitel wird die Vorgehensweise beschrieben, an die sich der Benutzer bei der Verwendung von HUML in Verbindung mit 'Poseidon for UML' und HIT halten sollte, um schnell Resultate zu erzielen. Im letzten Abschnitt des Kapitels wird kleinschrittig ein Anwendungsbeispiel durchgespielt, um dem Benutzer einen Eindruck von der Arbeit mit HUML zu vermitteln.

4.2. Systemanforderungen

HUML setzt eine installierte Java Virtual Machine ab Version 1.4.2 voraus. Getestet und entwickelt wurde HUML unter Verwendung des UML-Tools 'Poseidon for UML' [Gentleware], daher wird die Verwendung dieses Tools empfohlen. Für die Funktionsfähigkeit mit anderen UML-Tools wird keine Garantie übernommen.

Für die Verwendung von HUML kann jede beliebige UNIX-Umgebung (z.B. SunOS, Linux oder cygwin unter Windows) mit installiertem sed-Skriptinterpreter eingesetzt werden. Auf dem Rechner, der für die Leistungsbewertung mit HIT benutzt werden soll (siehe auch Abschnitt 4.5.2), muss darüberhinaus das Leistungsbewertungswerkzeug HIT installiert sein. Damit das Benutzungshandbuch angezeigt werden kann, muss der Acrobat Reader ab Version 3.x auf dem System installiert und über den Befehl "acroread" aufrufbar sein.

4.3. Installation von HUML

Das Programm liegt als ZIP-Archiv mit dem Namen "HUML.zip" vor. Dieses Archiv enthält das Programm selbst als JAR-Archiv, das Benutzungshandbuch und ein Skript zum Starten des Programms. Für die Installation von HUML ist dieses ZIP-Archiv in ein beliebiges Verzeichnis zu entpacken. Auf einem Linux/Unix/Solaris System wechselt man zunächst in das Verzeichnis in welches das Archiv "HUML.zip" entpackt werden soll und entpackt es anschließend mit dem Befehl "unzip HUML.zip". Danach startet man das Programm aus diesem Verzeichnis durch das Skript "startHuml". Dabei handelt es sich um ein Shell-Script, das den Befehl "java -jar huml.jar" enthält. Das Programm wird aus dem JAR-Archiv "huml.jar" heraus gestartet. Das Skript "startHuml" kann bei Bedarf modifiziert werden.

Wird HUML zum ersten Mal gestartet, öffnet sich zusätzlich zum Startfenster das Konfigurationsfenster in dem die Konfiguration des Programms vorgenommen werden muss. Ohne durchgeführte Konfiguration kann HUML nicht weiter ausgeführt werden. (siehe hierzu Abschnitt "Menüpunkt Konfiguration" in 4.4.2.2).

4.4. Benutzungsoberfläche von HUML

In diesem Abschnitt wird die grafische Benutzungsoberfläche (siehe Abbildung 4.1) des Softwarewerkzeugs HUML vorgestellt.

HUML wird über eine graphische Benutzungoberfläche bedient. Dabei wird zwischen einem Standardmodus und einem Expertenmodus sowie einem Startfenster (siehe Abbildung 4.2) unterschieden. Beim Starten des Programms erscheint das Startfenster mit einer Menüleiste und dem Hinweis, ein neues Projekt anzulegen oder ein bereits vorhandenes Projekt zu öffnen. Bei der ersten Benutzung öffnet sich zudem das Konfigurationsfenster. Nach durchgeführter Konfiguration kann ein neues Projekt angelegt oder ein bereits existierendes Projekt geöffnet werden. Die Oberfläche wechselt sowohl beim Anlegen eines neuen Projektes als auch beim Öffnen eines bereits existierenden Projektes zum Standardmodus. Im Standardmodus werden die einzelnen Arbeitsschritte von HUML automatisiert durchgeführt, während im Expertenmodus die Möglichkeit besteht, die einzelnen Schritte einer Bearbeitung manuell durchzuführen. Nähere Beschreibungen zu den Modi sind im Abschnitt 4.5 zu finden.

Zunächst werden die verschiedenen Formen der Statusmeldungen von HUML vorgestellt. Im Anschluss wird die Struktur der Menüleiste besprochen, die sich in beiden Benutzungsmodi nicht unterscheidet.

4.4.1. Statusmeldungen in HUML

Eine Leistungsbewertung mit Hilfe von HUML besteht aus mehreren Schritten, die entweder automatisiert (Standardmodus) oder manuell (Expertenmodus) durchgeführt werden. HUML

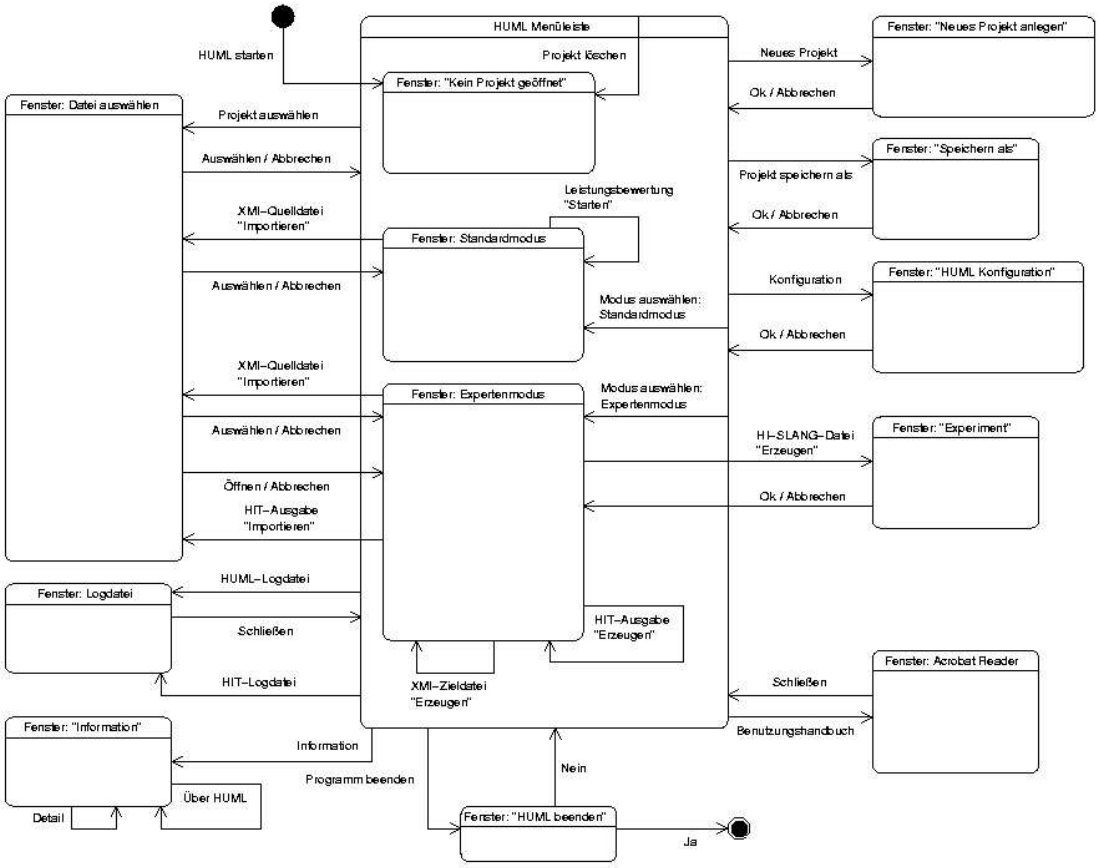

Abbildung 4.1.: Die Dialogstruktur von HUML

Abbildung 4.2.: Das Startfenster

bietet zweierlei Statusmeldungen. Im Standardmodus (siehe Kapitel 4.5.1) gibt es am unteren Rand des Hauptfensters eine Statusleiste, in der der aktuelle Arbeitsschritt dokumentiert

wird. Im Expertenmodus (siehe Kapitel 4.5.2) ist eine solche Anzeige nicht vorhanden, da die Schritte einzeln ausgeführt werden. Über Erfolg und Misserfolg der Arbeitsschritte geben in beiden Modi Lampen, die jeweils neben einem durch den Benutzer auszulösenden Arbeitsschritt (im Standardmodus zwei, im Expertenmodus fünf) angebracht sind, Auskunft. Jede Farbe dokumentiert einen Status des jeweiligen Arbeitsschritts.

- Grau: Der Vorgang wurde noch nicht durchgeführt
- Gelb: Der Vorgang ist gerade in Bearbeitung
- Grün: Der Vorgang wurde erfolgreich beendet
- Rot: Der Vorgang wurde aufgrund eines Fehlers abgebrochen

Wenn HUML vor Abschluss aller Arbeitsschritte beendet wird, wird der Projektstatus des aktuellen Projekts gespeichert, so dass dieser bei einem späteren Öffnen des Projekts wiederhergestellt werden kann. Dann werden auch die Lampen wieder in den entsprechenden Zustand gesetzt.

Zusätzlich werden zwei Logdateien angelegt, die genauere Meldungen über Ereignisse, die bei der Benutzung von HUML oder HIT aufgetreten sind, enthalten. Genauere Informationen zu den Logdateien finden Sie in den Paragraphen "Menüpunkt HUML-Logdatei" und "Menüpunkt HIT-Logdatei in 4.4.2.2.

Treten Fehler beim Parsen auf, werden diese nicht in die Logdatei geschrieben, sondern werden vom Parser direkt auf der Konsole ausgegeben. Die Fehlermeldungen beginnen mit `line`, gefolgt von einer Zeilenangabe und einer Beschreibung des Fehlers. Diese Meldungen deuten auf Rechtschreibfehler, falsche Eigenschaftswerte, falsche Typen von Eigenschaftswerten oder fehlende Klammern/Kommata in einer Annotation hin.

4.4.2. Die HUML-Menüleiste

In diesem Unterkapitel werden die einzelnen Menüpunkte der Menüleiste des HUML-Hauptfensters erklärt. In Kapitel 4.4.2.1 wird das Menü "Datei", in Kapitel 4.4.2.2 das Menü "Ansicht und in Kapitel 4.4.2.3 das Hilfe-Menü erläutert.

4.4.2.1. Menü "Datei"

Über das Menü "Datei" (siehe Abbildung 4.3) ist es dem Benutzer möglich, ein neues HUML-Projekt anzulegen, ein bereits angelegtes Projekt zu öffnen, eine Kopie des geöffneten Projekts zu erzeugen und das aktuelle Projekt zu löschen. Darüber hinaus kann HUML über dieses Menü beendet werden.

Menüpunkt "Neues Projekt" Nach Auswählen des Menüpunkts "Neues Projekt" öffnet sich eine Dialogbox, in der ein neues Projekt angelegt werden kann (siehe Abbildung 4.4). Dazu muss der Name des Projekts angegeben werden, der keine Leerzeichen enthalten darf. Mit einem Klick auf "OK" wird das Projekt angelegt, mit einem Klick auf "Abbrechen" wird der Vorgang abgebrochen. In beiden Fällen kehrt das Programm zum Hauptfenster zurück. Wurde ein neues Projekt angelegt, wird der Name des Projekts als Titel des Hauptfensters angezeigt. Bei Abbruch bleibt der alte Titel erhalten.

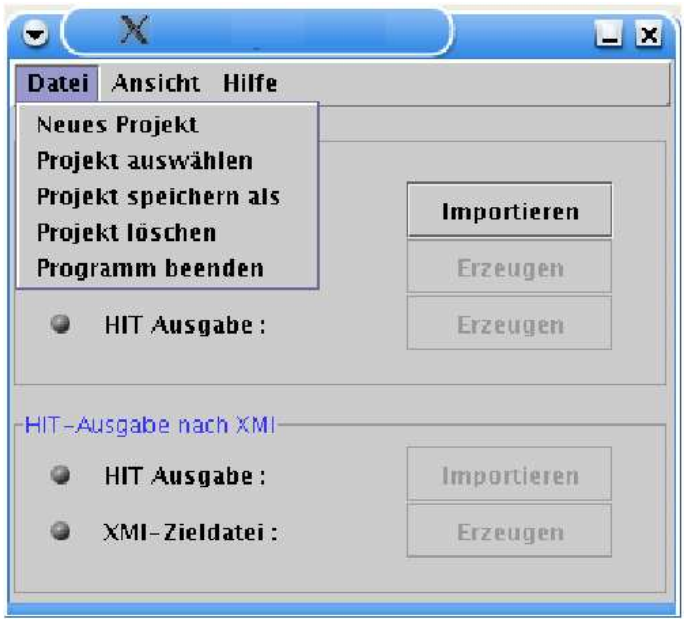

Abbildung 4.3.: Das Menü "Datei"

Abbildung 4.4.: Das Dialogfenster für das Anlegen eines neuen Projekts

Menüpunkt "Projekt öffnen" Nach dem Auswählen des Menüpunkts "Projekt öffnen" erscheint ein Auswahldialog, in dem ein bereits existierendes Projekt ausgewählt werden kann. In Abbildung 4.5 ist dieses Fenster dargestellt.

Menüpunkt "Projekt speichern als" Nach Auswahl des Menüpunkts "Projekt speichern als" ist es möglich, ein zuvor angelegtes Projekt unter einem anderen Namen abzuspeichern um so z.B. eine Kopie des Projekts, das gerade bearbeitet wird, zu erstellen. Der erscheinende Dialog (siehe Abbildung 4.6) erwartet die Angabe eines neuen Projektnamens. Unter diesem Namen wird eine Kopie des Projekts in dem Verzeichnis, das in der Konfiguration als "Projekt-Pfad" angegeben wurde, erstellt. Es werden alle aktuellen Projektdaten kopiert. Nach dem erfolgreichen Speichern wird das neu gespeicherte Projekt direkt geöffnet.

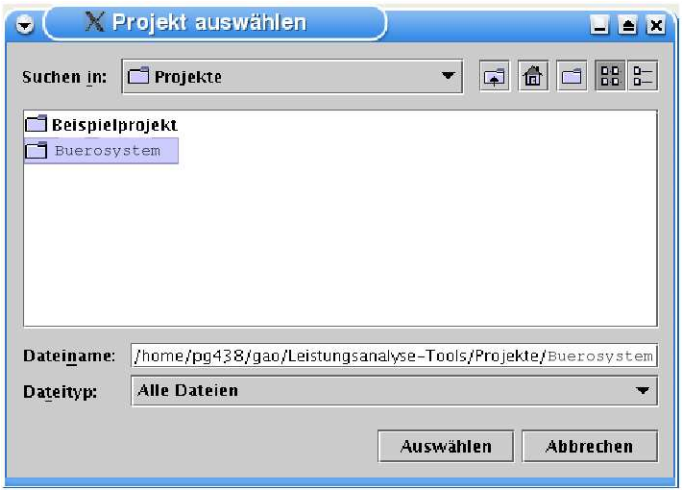

Abbildung 4.5.: Das Dialogfenster für das Auswählen eines Projekts

Abbildung 4.6.: Das Auswahlfenster für das Speichern eines Projekts

Menüpunkt "Projekt löschen" Nach Auswahl des Menüpunkts "Projekt löschen" werden alle zum geöffneten Projekt gehörenden Daten, inklusive des Projektordners, nach einer Sicherheitsabfrage an den Benutzer gelöscht.

Menüpunkt "Programm beenden" Nach Auswahl des Menüpunkts "Programm beenden" und anschließender Bestätigung der Sicherheitsabfrage, wird die Konfiguration von HUML und der Status des aktuellen Projekts gespeichert und das Programm beendet.

4.4.2.2. Menü "Ansicht"

Über das Menü "Ansicht" (siehe Abbildung 4.7) kann der Benutzer die Logdateien von HUML oder HIT einsehen (sofern diese bereits existieren), den Benutzungsmodus wechseln und die Konfiguration von HUML ändern.

Menüpunkt "Konfiguration" Nach Auswahl des Menüpunkts "Konfiguration" öffnet sich ein Dialogfenster (Abbildung 4.8), in dem Pfad-Einstellungen durchgeführt werden können. Der Projekt-Pfad gibt den Pfad zu dem Verzeichnis an, in dem die Projektordner angelegt werden. Existiert das angegebene Verzeichnis noch nicht, bietet das Programm die Möglichkeit an

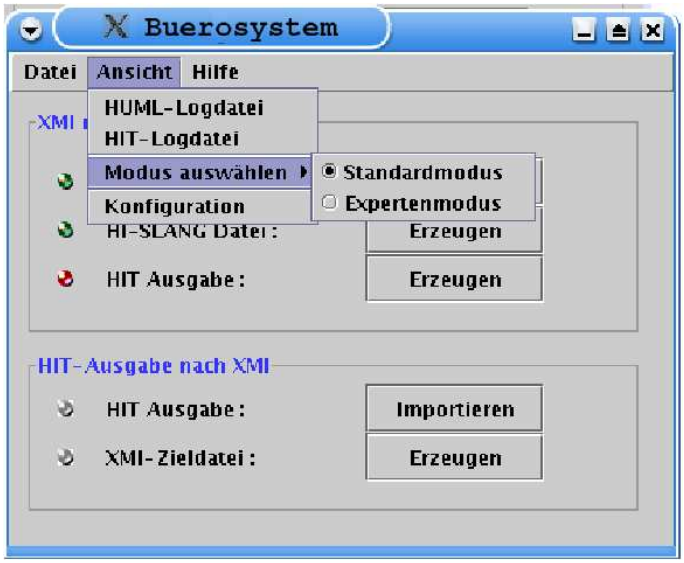

Abbildung 4.7.: Das Menü "Ansicht"

es anlegen zu lassen. Wird dies nicht gewünscht muss ein bereits existierendes Verzeichnis angegeben werden. Der HIT-Pfad gibt den Pfad zu dem Verzeichnis an, in dem HIT gestartet wird. Bei erstmaligem Start von HUML wird der Konfigurationsdialog automatisch geöffnet. Die Verwendung von HUML erfordert korrekte Pfadangaben.

Abbildung 4.8.: Das Dialogfenster für die Konfiguration

Menüpunkt "HUML-Logdatei" Nach Auswahl des Menüpunkts "HUML-Logdatei" öffnet sich ein Fenster, in dem die durch HUML erzeugte Logdatei dargestellt wird (siehe Abbildung 4.9). Die Logdatei zeichnet Fehler- und Erfolgsmeldungen der einzelnen Programmbestandteile auf.

Menüpunkt "HIT-Logdatei" Nach Auswahl des Menüpunkts "HIT-Logdatei" öffnet sich ein Fenster, in dem die durch 'HIT' erzeugte Logdatei dargestellt wird (siehe Abbildung 4.10). Die Logdatei gibt Auskunft über eventuell aufgetretene Fehler, die während der Benutzung von HIT entstanden sind.

Abbildung 4.9.: Das HUML-Logdatei-Fenster

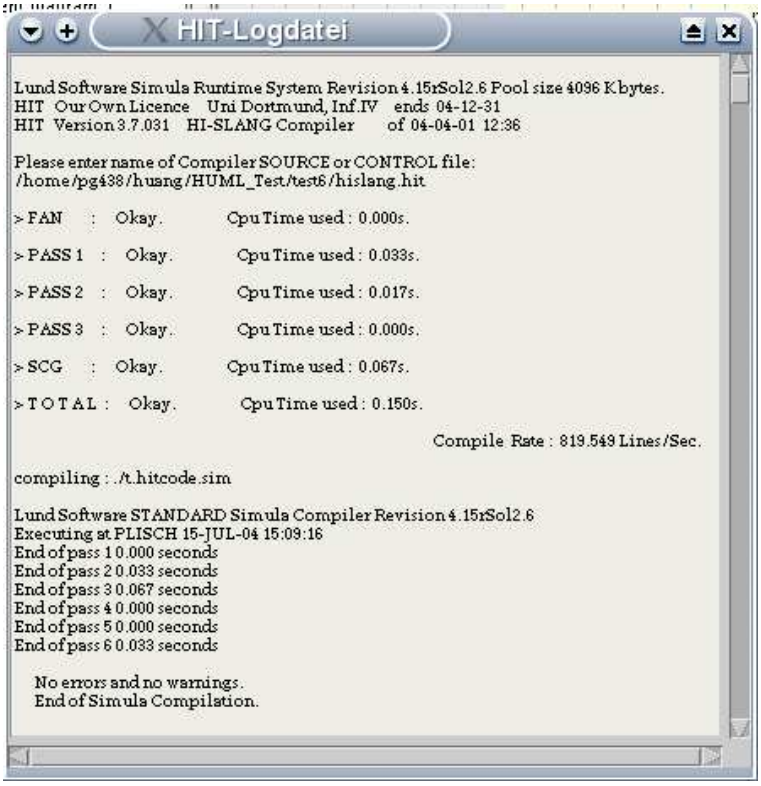

Abbildung 4.10.: Das HIT-Logdatei-Fenster

Menüpunkt "Modus auswählen" Unter dem Menüpunkt "Modus auswählen" kann zwischen dem Standardmodus und dem Expertenmodus (siehe Abschnitt 4.5) gewechselt werden.

4.4.2.3. Menü "Hilfe"

Über das Hilfemenü kann der Benutzer das Handbuch sowie einige Informationen über HUML einsehen.

Menüpunkt "Benutzungshandbuch" Nach Auswahl des Menüpunkts "Benutzungshandbuch" wird das Benutzungshandbuch im PDF-Format geöffnet. Hierzu muß Acrobat Reader instal-

liert sein. Dieser muss mit dem Befehl "acroread" aufgerufen werden können.

Menüpunkt "Information" Nach Auswahl des Menüpunkts "Information" wird ein Fenster geöffnet, in dem Informationen über die Softwareentwickler, Softwarerevisionen und Kontaktmöglichkeiten zu den Entwicklern angezeigt werden.

4.5. Benutzungsmodi

Das HUML-Werkzeug kann wahlweise im Standard- oder Expertenmodus betrieben werden. Im Standardmodus wird der Ablauf der Leistungsanalyse weitestgehend automatisiert und mit Standardeinstellungen durchgeführt. Im Expertenmodus können die einzelnen Schritte manuell ausgeführt werden und die Leistungsbewertung kann konfiguriert werden (z.B. durch Auswahl des Lösungsverfahrens). Im Folgenden werden die beiden Benutzungsmodi erläutert.

4.5.1. Standardmodus

Im Standardmodus werden die einzelnen Schritte zur Leistungsbewertung von, in eine XMI-Datei exportierten, UML-Aktivitätsdiagrammen automatisch durchgeführt. Die Leistungsbewertung wird durch Simulation, die bei einer MODELTIME von 10.000.000 Schritten oder einer CPUTIME von 300 Sekunden stoppt, durchgeführt. Abbildung 4.11 zeigt das Hauptfenster im Standardmodus.

Abbildung 4.11.: Das Hauptfenster im Standardmodus

Im Standardmodus muss vor der Durchführung einer Leistungsanalyse die mittels 'Poseidon for UML' [Gentleware] erzeugte XMI-Datei importiert werden. Dies geschieht durch einen Klick auf den Knopf (XMI Quelldatei) "Importieren". Es öffnet sich das in Abbildung 4.12 dargestellte Fenster. Die Quelldatei wird markiert und durch Bestätigung mit "Auswählen" importiert. Nach dem erfolgreichen Import der XMI-Datei (Meldung in der Statusleiste und grüne Lampe) wird durch einen Klick auf den Button 'Starten' die Leistungsanalyse gestartet. Die dabei automatisch durchgeführten Schritte der Leistungsanalyse werden über die Statusleiste dokumentiert. Der Wechsel in den Expertenmodus ist während einer ablaufenden Leistungsanalyse nicht möglich.

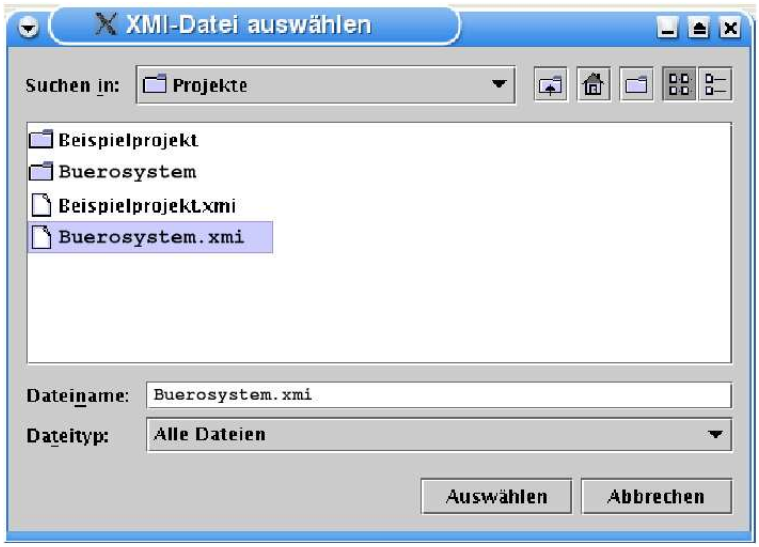

Abbildung 4.12.: Das Auswahlfenster zum Importieren einer XMI-Quelldatei

4.5.2. Expertenmodus

Eine Alternative zum Standardmodus bildet der Expertenmodus, in dem die einzelnen Schritte der Leistungsanalyse manuell durchgeführt werden müssen. Dadurch wird es möglich, die Leistungsbewertung mit HIT auf einem externen Rechner auszuführen, wenn HIT nicht auf dem verwendeten Rechner installiert ist. So kann der HI-SLANG Code erzeugt, und per externem Datenträger auf dem HIT-Rechner ausgeführt werden. Anschließend kann die HIT-Ergebnisdatei in den Projektordner kopiert werden und dann mit HUML ausgewertet werden. Abbildung 4.13 zeigt das Hauptfenster im Expertenmodus. Dieses Fenster ist in zwei Bereiche eingeteilt:

1. XMI nach HIT-Ausgabe
   Der erste Schritt ist der Import der XMI-Quelldatei. Ein Klick auf die Schaltfläche "Importieren" öffnet einen Auswahldialog, der in Abbildung 4.12 zu sehen ist. Nach Auswahl der XMI-Quelldatei wird durch einen Klick auf die Schaltfläche "Auswählen" der Importvorgang begonnen, durch "Abbrechen" wird der Importvorgang abgebrochen. War der Importvorgang erfolgreich (dargestellt durch eine grüne Lampe; siehe hierzu auch Kapitel 4.4.1) kann die HI-SLANG Datei durch einen Klick auf die Schaltfläche "Erzeugen" generiert werden. Es öffnet sich das Experimentfenster (siehe Abbildung 4.14), in dem der Benutzer das Lösungsverfahren auswählen kann, mit dem das Softwaremodell bewertet werden soll und in dem HIT eine bestimmte Menge Arbeitspeicher zur Bewertung zur Verfügung gestellt werden kann. Die erfolgreiche Generierung wird durch eine grüne Lampe dargestellt.

   HIT bietet drei verschiedene Typen von Lösungsverfahren.

   - Analytisch-algebraische Löser
     Zu den analystisch-algebraischen Lösungsverfahren zählen DOQ4 und LIN2. Diese Verfahren sind für die Leistungsbewertung separabler Netze einsetzbar. DOQ4

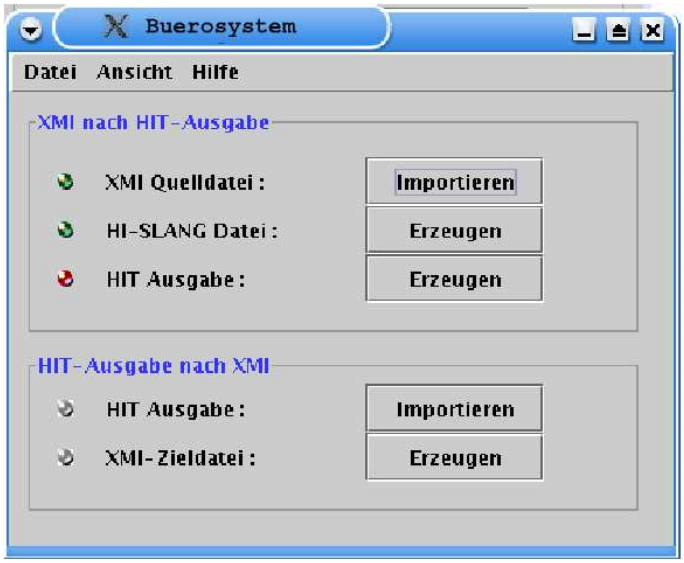

Abbildung 4.13.: Das Hauptfenster im Expertenmodus

Abbildung 4.14.: Das Experiment-Fenster

berechnet eine exakte Lösung während LIN2 die Lösung mit einer vom Benutzer spezifizierbaren Genauigkeit (ACCURACY im Experimentfenster) approximiert. Der ACCURACY-Wert muss dabei eine reelle Zahl zwischen 0 und 5 sein, die von HIT auf einen Integer gerundet wird. Je höher der Wert, desto genauer sind die

Ergebnisse. Bei großen Systemen ist das LIN2 Lösungsverfahren dem DOQ4 Lösungsverfahren aufgrund seiner Geschwindigkeit zu bevorzugen. In einem solchen Fall ist allerdings damit zu rechnen, dass HIT einen großen ACCURACY-Wert zugunsten der Berechnungszeit reduziert. Dies wird dem Benutzer in Form einer HIT-Warnung gemeldet.

- Analytisch-numerische Löser
  Zu dieser Klasse zählt das MARKOV Lösungsverfahren. Mittels der Einstellung ACCURACY im Experimentfenster lässt sich die zu erzielende Genauigkeit (maximaler Fehler in Prozent) bei der Approximation der Ergebnisse festlegen. Es wird ein Genauigkeitswert geringer als 1% empfohlen (ein ACCURACY-Wert kleiner als 0.01). Die Bedeutungen der ACCURACY-Werte für LIN2 und MARKOV weichen ausdrücklich voneinander ab, siehe Büttner u. a.. Die Einstellung CPUTIME ermöglicht die Vorgabe einer Stopbedingung für die Auswertung durch MARKOV. Die angegebene CPUTIME muss ein positiver reeller Wert sein. Sie gibt das Maximum der für die Berechnung aufzuwendenen CPU-Zeit an.
- Simulative Löser
  Da für die analytischen Lösungsverfahren sehr starke Einschränkungen bezüglich der analysierbaren Systeme gelten, bietet HIT darüberhinaus das simulative Lösungsverfahren SIMULATIVE. Die Einstellung CPUTIME ermöglicht die Vorgabe einer Stopbedingung für die Auswertung durch SIMULATIVE. Sie muss als positiver reeller Wert angegeben werden und gibt das Maximum der für die Berechnung aufzuwendenen CPU-Zeit an. Die Einstellung MODELTIME ist eine weitere Möglichkeit, eine Stopbedingung für die Simulation vorzugeben. Die Modell-Zeit beginnt für jeden Simulationsdurchlauf bei 0.0 und schreitet im Laufe eines Durchlaufs bei bestimmten Ereignissen im analysierten System voran. Der MODELTIME-Wert gibt die maximale Modell-Zeit pro Simulationsdurchlauf an.

Es empfielt sich, bezüglich der einzelnen Lösungsverfahren und der Einschränkungen, die für ihren Einsatz gelten, die entsprechenden Kapitel im HI-SLANG Reference Manual Büttner u. a. zu studieren.

Mittels "Compilation" und "Experiment" im Abschnitt "Memory" des Experimentfensters wird angegeben, wieviel Speicher HIT für die Leistungsanalyse zur Verfügung gestellt wird. Je mehr Speicher HIT zur Verfügung steht, desto schneller wird die Leistungsanalyse durchgeführt.

2. HIT-Ausgabe nach XMI
   Die Schritte dieses Abschnitts dienen der Integration der HIT-Messergebnisse in die XMI-Eingabe, um dem Benutzer einen anschaulichen, leicht verständlichen Zugang zu den Messergebnissen zu ermöglichen. Die ergänzte XMI-Datei kann dann in 'Poseidon for UML' importiert werden.
   Über den Knopf (HIT-Ausgabe) "Importieren" lässt sich ein externes HIT-dumpfile in das Projekt importieren. Dieser Schritt ermöglicht den, wie bereits angesprochen, Import von HIT-Ergebnisdateien, die auf einem externen Rechner erzeugt wurden. Sind die HIT-Ergebnisse im Arbeitsschritt (HIT Ausgabe) "Erzeugen" mit HUML erzeugt worden, steht diese Importfunktion nicht zur Verfügung.
   Nach dem Klick auf (XMI-Zieldatei) "Erzeugen" wird der um HIT-Messergebnisse er-

weiterte XMI-Code generiert. Dabei liest HUML die dump-Datei und überträgt die Ergebnisse eine die lokale Kopie der XMI-Quelldatei. Diese Datei kann dann in 'Poseidon for UML' [Gentleware] importiert und betrachtet werden.

Die erfolgreiche Durchführung eines jeden Vorgangs wird wiederum durch grüne Lampen dokumentiert (siehe hierzu auch Kapitel 4.4.1).

4.6. Einschränkungskriterien und bekannte Fehler

Folgende Einschränkungen und bekannte Fehler sind in der aktuellen Version von HUML vorhanden:

- Die Eigenschaftswerte PAthroughput und PAutilization des *UML-RT* finden im HUML-Werkzeug keine Berücksichtigung, weil sowohl Durchsatz als auch Auslastung bei der Leistungsbewertung berechnet werden können. Damit macht es keinen Sinn, diese Werte bei der Modellierung abzuschätzen. Gemäß der Anforderungsdefinition für das HUML-Werkzeug werden die entsprechenden beiden Bewertungsanforderungen als Eigenschaftswerte des «HUMLresult»-Stereotyps eingetragen.

- PArespTime und PAwaitTime gehören ebenfalls zu den nichtberücksichtigten Eigenschaftswerten.

- Desweiteren wird der Eigenschaftswert PActxtSwT ignoriert, weil Zeitverzögerungen bei Kontextwechseln im Rahmen von HIT nicht dargestellt werden können. Außerdem fehlt im *UML-RT* die Möglichkeit, einer Ressource mit *time sharing*-Richtlinie die Zeitscheibe für Kontextwechsel anzugeben.

- Der Eigenschaftswert PAprioRange spielt für das HUML-Werkzeug keine Rolle, da sich die Angabe von Prozessprioritäten nach den Möglichkeiten von HIT zu richten hat.

- Passive Ressourcen können mit den verwendeten Erweiterungen nur für die Dauer einer Aktivität belegt werden. Die Belegung einer Ressource über mehrere Schritte erfordert die Entwicklung eines neuen Konzepts für das Ressourcenmanagement von passiven Ressourcen. Die Entwicklung dieses Konzepts übersteigt jedoch den Rahmen dieser Projektgruppe.

- Ein HI-SLANG-Programm besteht aus einem Modelldefinitionsteil und einem Experimentteil. Im Modelldefinitionsteil könnte die Hierarchie der Modellkomponenten von der Definition der obersten Modellebene bis herab zu den untersten Komponenten durch eine Schachtelung von Typdefinitionen dargestellt werden. Das ist aber nur möglich, wenn die Hierarchie, soweit sie die selbstdefinierten Komponententypen unter Ausschluss der Standardkompontentypen betrifft, baumförmig ist. Das ist aber bei Modellen, bei denen verschiedene Aktionszustände durch ein identisches Aktivitätsdiagramm erweitert werden, nicht gegeben. Stattdessen müssen bei dem Ansatz der PG 438 sämtliche Komponenten des Modells als gemeinsam genutzte Komponenten (*shared components*) instanziiert werden, die Definitionen ihrer Komponententypen sind dann nicht geschachtelt, sondern eine einfache Sequenz.

- Im Falle einer Arbeitslast eines offenen Systems, muss das Stereotyp PAopenLoad ein PAoccurence aufweisen, das die Ankunftsrate spezifiziert. Die Ausprägungen der bei PAoccurence angegebenen Werte sind in dem Sinne eingeschränkt, dass Intervalle, in denen evtl. Prozesse gestartet werden, für eine Leistungsanalyse zu ungenau sind und daher nicht unterstützt werden.

- Eine weitere Einschränkung bildet das HI-SLANG Fragment DUE TO ALL, das sich auf die HIERARCHIES bezieht, die aufgrund der ausschließlichen Bewertung von UML-Aktivitäts- und Verteilungsdiagrammen nicht im einzelnen unterstützt werden können. Somit kann eine Leistungsanalyse nur mit der Einstellung DUE TO ALL durchgeführt werden.

- *Poseidon for UML* [Gentleware], das von der PG438 benutzte UML-Werkzeug, unterstützt die Verschachtelung von Diagrammen. Diese Möglichkeit wurde durch die PG438 nicht ausgeschöpft.

- Annotationen, die der XMI-Datei des Eingabemodells durch HUML hinzugefügt werden, sind nach dem Import mit *Poseidon for UML* nicht richtig formatiert. Dieser Mangel konnte nicht behoben werden, da die Ursache höchstwahrscheinlich bei *Poseidon for UML* zu suchen ist.

- Sämtliche Dateinamen, Pfade und Beschreibungen dürfen keine Umlaute enthalten.

- Beschreibungen und Kommentare, die nicht der Spezifikation von Leistungsparametern dienen bzw. nicht anderweitig Informationen für HUML enthalten, müssen in einer eigenen Annotation stehen und dürfen nicht mit "«" beginnen.

4.7. Vorgehensweise für eine HUML-Leistungsbewertung

Dieses Kapitel beschreibt eine Vorgehensweise zur Verwendung von HUML. Das Paradigma enthält das zugrundeliegende Konzept der Software HUML und Anweisungen für den korrekten Einsatz von UML-Modellen und Erweiterungen, um eine Leistungsanalyse mit dem Performanz-Analyse-Werkzeug HIT [Büttner u. a.] zu ermöglichen.

HUML verwendet Aktivitätsdiagramme (siehe Kapitel 1.4.2) und Verteilungsdiagramme (siehe Kapitel 1.4.3) zur Leistungsbewertung. Die UML-Diagramme müssen bei "Poseidon for UML" direkt auf der Modellebene, der obersten Ebene, erstellt werden. Aus dem Anwendungsfalldiagramm der zu entwickelnden Software wird das Aktivitätsdiagramm der obersten Ebene erstellt. Dazu müssen die einzelnen Anwendungsfälle in eine Ablaufreihenfolge gebracht werden.

Was beschreiben Aktivitätsdiagramme? Aktivitätsdiagramme beschreiben deterministische Folgen von Aktionszuständen. Die Folge von Aktionszuständen kann durch Verzweigungen, Nebenläufigkeiten und Schleifen strukturiert werden. In HUML spielen Aktionszustände und ihre Beziehungen eine wichtige Rolle. Die Möglichkeiten zum Senden und Empfangen von Nachrichten sowie Objektzustände in Aktivitätsdiagrammen werden von HUML ignoriert.

Was beschreiben Verteilungsdiagrammme? Ein Verteilungsdiagramm beschreibt Hardwarekomponenten und -instanzen sowie die zwischen ihnen bestehenden Beziehungen. Damit repräsentiert ein Verteilungsdiagramm die Topologie der Hardware eines modellierten Systems. Im Kontext von HUML werden nur Hardwarekomponenten betrachtet.

Was ist darüberhinaus erforderlich? Verteilungs- und Aktivitätsdiagramme beschreiben ein System nicht ausreichend. Um eine Leistungsbewertung durchzuführen fehlen noch einige Informationen:

- Maschinenbeschreibungen
  Diese werden durch Stereotypisierung der Maschinen als aktive (Stereotyp *«PAhost»*) oder passive Ressourcen (Stereotyp *«PAresource»*) beschrieben (siehe Kapitel 1.5.4).
- Arbeitslastbeschreibung
  Die Arbeitslast wird durch Stereotypisierung des ersten Aktionszustands als *«PAopenLoad»* oder *«PAclosedLoad»* beschrieben (siehe Kapitel 1.5.2).
- Informationen über Aktionszustände
  Wenn ein Aktionszustand in die Leistungsbewertung eingehen soll, muss er mit *«PAstep»* stereotypisiert werden. Die Eigenschaften eines Aktionszustandes, z.B. die Anzahl der Wiederholungen oder die Verarbeitungszeit, können durch verschiedene Eigenschaftswerte (z.B. *PArep* oder *PAdemand*) beschrieben werden. Eine genauere Beschreibung des Stereotyps *«PAstep»* und dessen Eigenschaftswerten sind in Kapitel 1.5.3 zu finden.
- Die Verbindung zwischen Maschine und Aktionszustand
  Die Verbindung wird durch die Eigenschaftswerte *PAextOp* (Verbindung zu einer oder mehreren passiven Ressourcen) oder *«HUMLhost»* (Verbindung zu einer aktiven Ressource) an einem als *«PAstep»* stereotypisierten Aktionszuständen angegeben. Erklärungen zu den beiden Eigenschaftswerten sind in Kapitel 1.5.3 zu finden.
- Informationen über gewünschte Messdaten
  Die zu messenden Leistungsinformationen werden in einer mit *«HUMLresult»* stereotypisierten Annotation angegeben. Leistungsdaten können an Aktionszuständen und Maschinen gemessen werden. Genauere Informationen zur Stereotypisierung mit *«HUMLresult»* sind in Kapitel 1.5.5 zu finden.

Darüberhinaus muss die Struktur des Aktivitätsdiagramms verschiedenen Anforderungen genügen. Diese Anforderungen werden in Kapitel 1.6.5 näher erläutert.

Ein Anwendungsbeispiel, in dem UML-Diagramme durch Stereotypisierung und Annotation von Eigenschaftswerten erweitert werden, ist in Kapitel 4.8 zu finden. Das Nachvollziehen der einzelnen gezeigten Modellierungsschritte und der verwendeten Annotationen hilft dem Anwender bei der späteren Entwicklung eigener Modelle.

4.8. Anwendungsbeispiel

Im Folgenden wird die Funktionsweise von "HUML", ausgehend von einem leicht verständlichen UML-Diagramm, erklärt. Es soll ein System analysiert werden, das eine Aktion auf einer CPU ausführt. Dieses Anwendungsbeispiel kann Schritt für Schritt mittels "Poseidon for UML" und "HUML" durchgeführt werden.

Installation In diesem Beispiel wird die Datei "HUML.zip" durch den Befehl "unzip HUML.zip" in das Verzeichnis "Leistungsanalyse-Tools" entpackt.

Erstellen der UML-Diagramme Um ein System zu bewerten, muss dieses zuerst in Form von UML-Diagrammen modelliert werden und anschließend mit Leistungsparametern versehen werden. Dazu wird zunächst "Poseidon for UML" gestartet, um die UML-Diagramme zu erzeugen.

In einem Verteilungsdiagramm werden die verwendeten Hardwarekomponenten durch einen Maschinenknoten modelliert und mit Leistungsdaten annotiert. In dem Beispiel gibt es nur eine Hardwarekomponente, und zwar eine CPU. Das zugehörige Verteilungsdiagramm sehen Sie in Abbildung 4.15. Eine CPU ist eine aktive Ressource, und hat in diesem Beispiel eine Geschwindigkeit von 2.0. Der Wert 2.0 gibt den relativen Geschwindigkeitsfaktor an, um den sich Prozessverarbeitungsgeschwindigkeit dieser CPU von der Referenzgeschwindigkeit 1 unterscheidet. In diesem Fall werden die Prozesse doppelt so schnell verarbeitet, was bewirkt, dass sich alle angegebenen Prozessverarbeitungszeiten halbieren. Die ankommenden Prozesse werden von der CPU nach der Bearbeitungsrichtlinie FIFO bearbeiten.

Die Hardware wird durch das Stereotyp «PAhost» als aktive Ressource charakterisiert. Die Eigenschaften der Hardware (Geschwindigkeit und Prozessbearbeitungsrichtlinie) werden durch Eigenschaftswerte in einer ebenfalls mit «PAhost» versehenen Annotation beschrieben. Der Eigenschaftswert PArate beschreibt den relativen Geschwindigkeitsfaktor zur Referenzgeschwindigkeit 1, der Eigenschaftswert PAschdPolicy die Prozessbearbeitungsrichtlinie FIFO.

Das beschriebene Verteilungsdiagramm sieht nach dem Einfügen der Leistungsparameter so aus, wie in Abbildung 4.15 abgebildet. In dieser Abbildung ist also ein Hardwareknoten zu sehen, dessen Geschwindigkeit und Verhalten bzgl. eintreffender Prozesse vorgegeben sind.

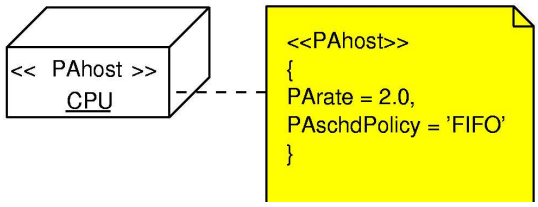

Abbildung 4.15.: Beispiel: erweitertes UML-Verteilungsdiagramm für einen Maschinenknoten

Softwareseitig soll ein System erzeugt werden, das nur aus einem einzigen Aktionszustand besteht. Es wird also zunächst ein herkömmliches UML-Aktivitätsdiagramm angelegt, das aus Startknoten, Aktionszustand, Zielknoten und Kanten besteht (Abb. 4.16).

Aufgrund der Tatsache, dass das Diagramm das Gesamtsystem beschreiben soll, ist es zunächst notwendig eine Arbeitslast zu spezifizieren. In diesem Beispiel handelt sich um ein geschlossenes Softwaresystem, in dem eine feste Anzahl von Prozessen zirkuliert. Dies wird durch die Stereotypisierung des ersten Aktionszustandes als «PAclosedLoad» angegeben. Ein offenes System würde mit dem Stereotyp «PAopenLoad» gekennzeichnet (in einem solchen Fall würde die Angabe der die Last charakterisierenden Eigenschaftswerte leicht von der folgenden Vorgehensweise abweichen).

Die Spezifikation der Last muss in einer an den ersten Aktionszustand des Gesamtsystems

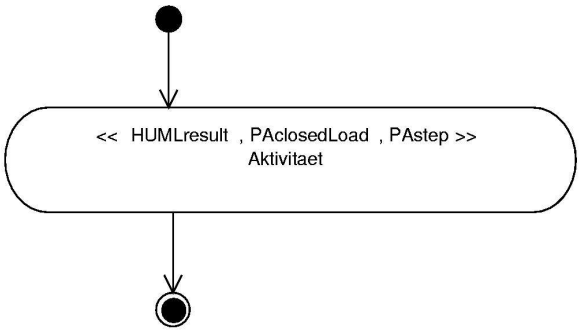

Abbildung 4.16.: einfaches UML-Aktivitätsdiagramm für einen Aktionszustand

annotierten Notiz stehen. In dem Beispiel ist das in Abbildung 4.17 zu sehen. Ist ein Aktionszustand mehrfach stereotypisiert, bietet es sich an, alle Spezifikationen in einer Notiz anzugeben. Deshalb wird immer zunächst der Name des entsprechenden Stereotyps (hier «PAclosedLoad») angegeben, auf den ein von geschweiften Klammern umschlossener Block folgt, in dem die Eigenschaftswerte des Stereotyps eingetragen werden.

Das Beispiel-System hat eine geforderte Antwortzeit von im Mittel 8 ms, d.h. dass die Zeitspanne zwischen Eingabe in das System und Ausgabe durch das System im Schnitt die Länge von 8 ms nicht überschreiten darf. Diese Forderung wird durch den Eigenschaftswert PArespTime angegeben. Der Wert eines PArespTime ist ein PAperfValue (siehe Kapitel 1.5.2). In dem Beispiel ist PArespTime= ('req','mean',8,'ms'). Das Kürzel "req" drückt aus, dass der folgende Wert verlangt ("required") wird, das Kürzel ''mean" bedeutet in Verbindung mit der folgenden 8, dass es sich bei dem folgenden Wert um den Mittelwert handelt. Da es sich bei dem System um ein geschlossenes System handeln soll, ist eine Population zu spezifizieren, die die Anzahl der Prozesse, die in dem System zirkulieren angibt. Im Beispiel liegt die Population bei 5 Prozessen. Dieser Sachverhalt wird mit dem Eigenschaftswert PApopulation ausgedrückt, dessen Wert eine positive ganze Zahl sein muss.

Jeder Aktionszustand, der bewertert werden soll, muss mit «PAstep» stereotypisiert werden. Bei unserem Beispiel soll der einzige Aktionszustand bewertet werden. Die Spezifikation der Eigenschaftswerte des «PAstep» wird in einer an den zu bewertenden Aktionszustand annotierten Notiz vorgenommen. Da es bereits eine Notiz an diesen Aktionszustand gibt, wird die Spezifikation ebenfalls in diese Notiz eingetragen (siehe Abb. 4.17). Mit dem Parameter HUMLhost wird in HUML (über Namensgleichheit zwischen Aktionszustand und Maschinenknoten) die Verbindung zu der "CPU" hergestellt, auf der die Last bearbeitet wird. Dies ist das im obigen Verteilungsdiagramm angelegte Maschinenobjekt "CPU". Die Last, die ein mit «PAstep» markierter Aktionszustand erzeugt, wird durch den Eigenschaftswert PAdemand beschrieben. In dem Beispiel, die angenommene ("assumed", abgekürzt "assm") Verarbeitungszeit des durch den Aktionszustand beschriebenen Prozess auf einer Maschine mit Referenzgeschwindigkeit 1 ist der Mittelwert ('mean') einer exponentialverteilten Zufallsvariablen ('exponential') und beträgt 3 Millisekunden.

In dem mit «PAstep» überschriebenen Block der UML-Annotation in Abb. 4.17 ist die Beschreibung des Prozesses zu sehen.

Die Anforderung der Messung einer Menge von Leistungswerten an einem Aktionszustand wird durch die Stereotypisierung mit «HUMLresult» gekennzeichnet. Die Spezifikation muss

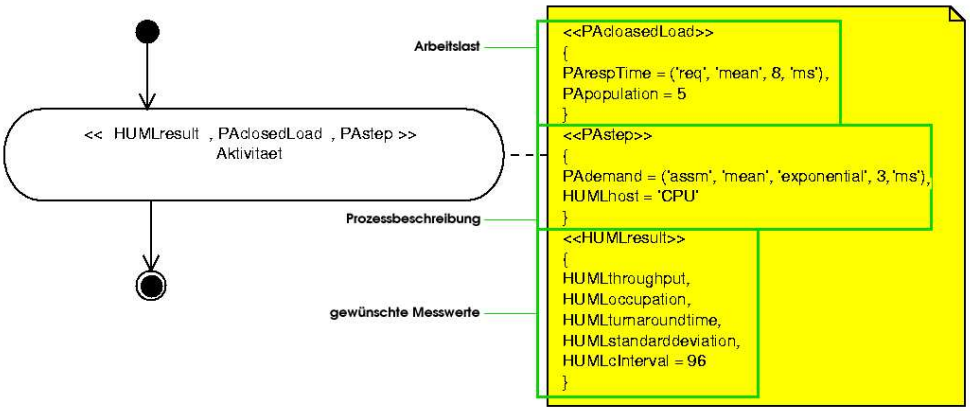

Abbildung 4.17.: erweitertes UML-Aktivitätsdiagramm für einen Aktionszustand

in einer Notiz an den zugehörigen, mit «PAstep» stereotypisierten, Aktionszustand annotiert werden. Die Messanforderungen des Beispiels sind in dem entsprechenden Block in Abb. 4.17 zu sehen. In diesem Fall sollen Durchsatz (HUMLthroughput), Belegung (HUMLoccupation, kann nur bei simulativen Lösungsverfahren ermittelt werden) und Antwortzeit (HUMLturnaroundtime) ermittelt werden. In diesem Beispiel wird ein simulativer Löser eingesetzt. Deshalb sind einige weitere Eigenschaftswerte für das Stereotyp «HUMLresult» anzugeben. Der Eigenschaftswert HUMLstandarddeviation kennzeichnet den Wunsch, dass die Standardabweichung der Messwerte berechnet werden soll. Außerdem ist es bei simulativen Lösungsverfahren notwendig, ein Vertrauensintervall anzugeben. Das Vertrauensintervall gibt die Wahrscheinlichkeit an, mit der der reale Mittelwert in einem Intervall um den durch Simulation ermittelten Mittelwert liegt. Das Vertrauensintervall bestimmt also die Genauigkeit der simulativen Lösung. Das Vertrauensintervall sollte einen Wert zwischen 95 und 99 Prozent haben, da stärker abweichende Werte die Aussagekraft der Messwerte verschlechtern. Je kleiner das Intervall (je höher also der Wert des HUMLcInterval), desto genauer sind die ermittelten Werte, die sich unter Garantie innerhalb des spezifizierten Vertrauensintervalls um ihren Mittelwert befinden.

Alle weiteren Leistungswerte, die unterstützt werden finden sie in Kapitel 1.5.5.

Nachdem beide Diagramme erstellt worden sind, werden sie in eine XMI-Datei in das Verzeichnis "Leistungsanalyse-Tools" unter dem Namen "Beispielprojekt.xmi" exportiert.

Leistungsanalyse durch HUML Im nächsten Schritt wird die Leistungsanalyse mit HUML und HIT durchgeführt. Dazu erfolgt ein Wechsel in das Verzeichnis "Leistungsanalyse-Tools" in dem HUML installiert worden ist. Durch Eingabe von "./startHuml" wird HUML gestartet.

Zunächst (bei erstmaligem Start von HUML bzw. später nach Wunsch oder Notwendigkeit) sind einige Einstellungen in HUML vorzunehmen. Wird HUML zum ersten Mal gestartet, so öffnet sich neben dem Startfenster auch das Konfigurationsfenster (Abb. 4.18). Hier sind zwei Informationen bereitzustellen. Zum Einen ist der gewünschte Projektpfad anzugeben, in dem HUML später für jedes erzeugte Projekt einen Projektordner anlegt. Zum anderen ist der Pfad zur HIT-Installation zu spezifizieren. Die Pfade des Beispiels stehen in Abbildung (Abb. 4.18).

Abbildung 4.18.: Das Dialogfenster der HUML-Konfiguration

Abbildung 4.19.: Das Dialogfenster für das Anlegen des neuen Projekts "Beispielprojekt"

Nach dem Anlegen eines neues Projekts mit dem Namen "Beispielprojekt" (Abbildung 4.19) über den entsprechenden Eintrag im Datei-Menü, kann die Leistungsanalyse entweder im "Standardmodus" oder im "Expertenmodus" durchgeführt werden.

Unabhängig vom verwendeten Modus wird die, zuvor mittels "Poseidon for UML" erstellte, XMI-Datei "Beispielprojekt.xmi" aus dem Verzeichnis "/Leistungsanalyse-Tools/" durch einen Klick auf den Knopf (XMI-Quelldatei) "Importieren" und die anschließende Auswahl der *xmi*-Datei importiert (siehe Abbildung 4.20).

Bei der Verwendung des Standardmodus, der nach der Erzeugung eines Projekts automatisch aktiviert ist, wird ein erfolgreicher Import der XMI-Quelle in der Statusleiste dokumentiert und die obere der beiden dargestellten (grauen) Lampen färbt sich grün (siehe Abbildung 4.21).

Danach kann die Leistungsanalyse durch Druck auf den Knopf "Starten" durchgeführt werden. Nach erfolgreichem Abschluss der Leistungsanalyse befindet sich das Ergebnis in Form der XMI-Datei "result.xmi" im Projektverzeichnis "/Leistungsanalyse-Tools/Projekte/Beispielprojekt".

Im Expertenmodus (siehe Abbildung 4.22) kann zusätzlich das zu verwendende Lösungsverfahren spezifiziert werden. Dazu öffnet sich nach einem Klick auf den Knopf (HI-SLANG-Datei) "Erzeugen" das Experimentfenster (siehe Abbildung 4.23).

Nach der Bestätigung mit 'OK' wird die HI-SLANG-Datei erzeugt. Mit einem Klick auf den Knopf (XMI-Zieldatei) "Erzeugen" wird die Leistungsbewertung gestartet. Anschließend wird die XMI-Zieldatei automatisch erzeugt und im Projektverzeichnis unter dem Namen "result.xmi" gespeichert.

Nach der Leistungsanalyse ist "Poseidon for UML" zu starten und die erzeugte XMI-Datei "result.xmi" zu importieren. Das fertige, um Leistungswerte ergänzte, Aktivitätsdiagramm sieht am Ende so aus wie in Abbildung 4.24.

Abbildung 4.20.: Das Auswahlfenster zum Importieren der XMI-Datei "Beispielprojekt.xmi"

Abbildung 4.21.: Das Standardfenster nach erfolgreichem Import der XML-Quelle

In der mit dem Stereotyp «HUMLresult» überschriebenen Annotation sind die Ergebnisse der durchgeführten Leistungsanalyse zu finden.
Der erste Wert des Ergebnistripels ist der gemessene Mittelwert. Der zweite und dritte Wert werden nur bei simulativen Verfahren ermittelt. Dabei gibt der zweite Wert die Standardabweichung der Ergebniszufallsvariablen des simulativen Verfahrens an, und der dritte Wert gibt die Hälfte des Konfidenzintervalls als Prozentsatz des gemessenen Mittelwertes an. Zum Beispiel bezeichnet der Wert 10 bei einem abgeschätzten Mittelwert von 200 das Intervall [180,220].
Das System, mit einer verlangten Antwortzeit von 8 s, erreicht also bei einer Belastung durch 5 Prozesse einen Durchsatz von 0.65 Prozessen/Sekunde. Die tatsächliche Antwortzeit des Prozesses "eine Aktivität" beträgt bei dieser Belastung 7.670882 ms. Also liegt die Antwortzeit des Systems unter der verlangten Antwortzeit von 8 s (angegeben durch den Eigenschaftswert PArespTime der Arbeitslast).

Abbildung 4.22.: Das Hauptfenster für Expertenmodus

Abbildung 4.23.: Das Experimentfenster zur Einstellung der HIT-Experimentparameter

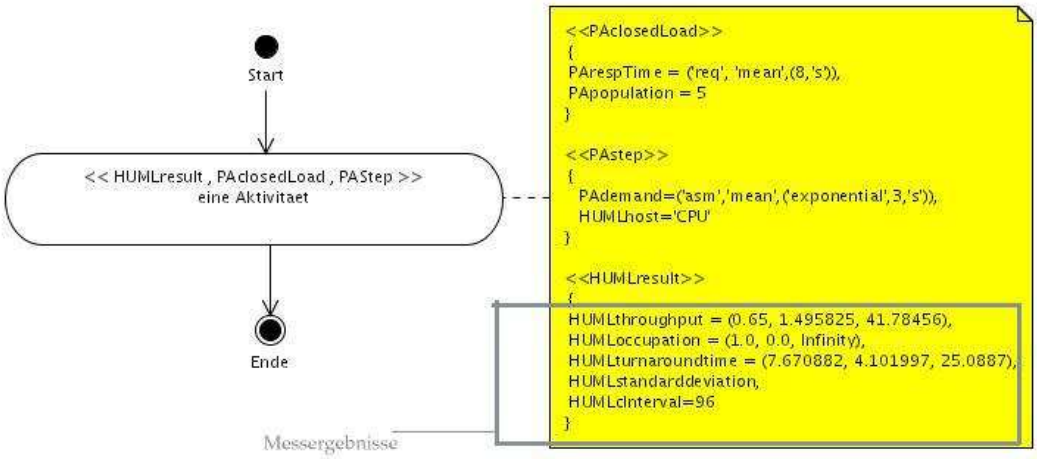

Abbildung 4.24.: erweitertes UML-Aktivitätsdiagramm für einen Aktionszustand mit Ergebnissen

5. Fazit der PG-Mitglieder

Im folgenden Kapitel sollen die persönlichen Eindrücke und gesammelten Erfahrungen der einzelnen Teilnehmer der PG438 aufgeführt werden. Dazu hat jedes Mitglied eine Stellungnahme zu der Projektgruppe abgegeben.

Jürgen Mäter (PG-Leiter)

Für mich als Veranstalter der Projektgruppe war es zunächst einmal sehr spannend, ob sich überhaupt Teilnehmer für dieses Thema finden und welche Voraussetzungen und Kenntnisse sie mitbringen würden. Es war mir von vornherein klar, dass ich keine Kenntnisse aus dem Bereich Leistungsbewertung von Rechensystemen erwarten konnte. Deshalb habe ich das einführende Kompaktseminar derart gestaltet, dass ein erster grober Überblick in dieses Gebiet erarbeitet wurde. Die Teilnehmer waren sehr motiviert. Das extern im Haus Villigst veranstaltete Kompaktseminar war ein voller Erfolg, einerseits auf fachlicher Ebene und andererseits wurde die Kennenlernphase der Projektgruppenmitglieder meiner Meinung nach sehr angenehm und positiv beeinflusst.

Ein Lehrziel, das in einer Projektgruppe vermittelt werden soll, bildet die Organisation, Leitung und Koordinierung von Projekten, also das Projektmanagement. Zwei Teilnehmer haben das Projektmanagement übernommen und gut praktiziert. Ein Projektplan wurde erstellt und als Basis für die PG- Arbeit von allen Teilnehmern akzeptiert.

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 von Rechensystemen 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.

In der anschließenden Analysephase wurde zwar motiviert eine Anforderungsdefinition sowie ein Pflichtenheft erstellt, wobei allerdings seitens der Projektgruppenmitglieder immer möglichst nur Minimalkonzepte angestrebt wurden, weshalb das Gesamtkonzept, das HUML-Paradigma, erst im zweiten Projektgruppensemester fertig gestellt werden konnte.

Bei der Erstellung des Zwischenberichtes, obwohl frühzeitig initiiert, ergaben sich die üblichen Kommunikations- und Organisationsprobleme, d.h. dass Teile nicht rechtzeitig fertig wurden, bzw. deren Qualität zu wünschen übrig ließ. Es weckte den Anschein, dass sorgfältiges Arbeiten nicht gewohnt war und sich zu sehr auf Korrekturen meinerseits verlassen wurde. Es fehlte an Einsicht, dass mindestens ein Projektgruppenmitglied den Zwischenbericht im Ganzen inhaltlich überwachen sollte. Es wurden so von den Untergruppen nicht wechselseitig konsistente Kapitel geschrieben, die erst durch viel redaktionelle Arbeiten annähernd konsistent gemacht werden mussten.

Der wesentlichste Konzeptionspart, nämlich das HUML-Paradigma, bildet den Kern der Projektgruppenergebnisse. Die Gruppe hat sich hierbei sehr schwer getan, da sie offensichtlich wissenschaftliches Arbeiten noch nicht gewohnt war. Positiv ist jedoch anzumerken, dass sie sich parallel zur Konzeption schon Gedanken über die erforderliche Umsetzung gemacht hat. Es wurde versucht, anhand eines umfangreichen Beispiels, die konzeptionellen Ideen durchzuspielen. Außerdem ist zu betonen, dass die Gruppe weltweit als Pioniere gelten, die versucht haben, UML in das HIT-Last-Maschine-Paradigma abzubilden. Von anderen Forschungsgruppen sind lediglich Lösungen für die Petri-Netz-Familie, Warteschlangennetze und ereignisdiskrete Simulatoren bekannt.

Die im zweiten Projektsemester begonnene Entwurfsphase verlief etwas schleppend, da die Projektgruppenmitglieder offensichtlich Angst vor einer umfangreichen Implementation hatten, so dass erst von einem minimalen Funktionsumfang des zu entwickelnden Leistungsmoduls ausgegangen wurde. Außerdem zeigte sich, dass nicht alle Projektgruppenmitglieder das zuvor entwickelte Konzept voll verstanden hatten. Es entstand dadurch ein erster Zeitverzug gegenüber Planungsvorgaben. Es ist positiv zu bewerten, dass die Projektgruppenmitglieder selbst erkannt haben, dass die von ihnen erstellte Entwurfsdokumentation teilweise zu knapp gehalten war, so dass sie selbst bei der anschließenden Implementierung Probleme hatten!

Die Implementierung ist größtenteils gut fortgeschritten, jedoch der Zeitverzug konnte nicht ganz aufgeholt werden. Dennoch hat die Gruppe versucht, das abschließende Fachgespräch in der letzten Vorlesungswoche zu absolvieren. Das war sehr lobenswert. Leider musste das Fachgespräch dann wegen der Erkrankung eines Referenten verschoben werden.

Das Produkt HUML hat den prototypischen Testeinsatz bestanden. Die Projektgruppe hat damit gezeigt, daß es möglich ist, ausgewählte UML-Konstrukte für eine Leistungsbewertung mittels HIT umzusetzen. Das Produkt HUML kann somit in Forschung und Lehre weiter entwickelt und auch stabilisiert werden.

Der Endbericht zeigte anfangs die gleichen Probleme auf wie der Zwischenbericht. Erst nach einer radikalen Umstrukturierung und Überarbeitung konnte er in der vorliegenden Form akzeptiert werden.

Zur Gruppenarbeit kann ich nur ein paar globale Bemerkungen machen, da ich in den Untergruppensitzungen nicht eingebunden war. Die Ergebnisse der Untergruppen waren zum Teil von sehr unterschiedlicher Qualität, was auch in den Plenumssitzungen dann nicht nur von mir kritisiert wurde. Es hat sich jedoch mal wieder gezeigt, dass die Untergruppen nicht über ihren Tellerrand hinaus und von den Produkten der anderen Untergruppen nur bis zu den gemeinsam festgelegten Schnittstellen geschaut haben, also nicht das Gesamtprodukt vor Augen hatten.

Es herrschte, soweit ich das beurteilen kann, ein recht gutes Gruppenklima. Größere Konflikte hat es wohl nicht gegeben oder wurden geschickt vor mir verborgen. Ein Highlight war sicher unsere "Weihnachtsfeier" bei Thomas P. in Düsseldorf. Nochmals herzlichen Dank an Thomas Eltern, die uns ihre Wohnung in Unordnung bringen ließen.

Im Nachhinein lieferte die Projektgruppe ein gutes Ergebnis. Ich hoffe, dass die Teilnehmerinnen und Teilnehmer für ihr weiteres Studium, insbesondere für ihre Diplomarbeiten etwas über wissenschaftliches Arbeiten gelernt haben.

Ich wünsche allen für die Zukunft alles Gute und dass sie ihr Studium gut abschließen.

Nadine Bramsiepe

Es war interessant, ein Projekt von der Einarbeitung über den Entwurf und die Implementie-

rung bis hin zum Produkttest zu entwickeln. Dadurch hat man schon während des Studiums Einblicke in die praktische Software-Entwicklung mit ihren Problemen erhalten. So mussten wir z.B. während der Implementierung feststellen, dass Teile des Entwurfs zu ungenau modelliert worden waren, und dieser Teil neu modelliert werden musste.

Desweitern lief auch die Teamarbeit nicht immer problemlos ab. Teilweise wurden zu wenig Absprachen gemacht, was zu Missverständnissen führte. Auch war der Einsatz der verschiedene PG-Mitglieder sehr unterschiedlich.

Im Großen und Ganzen hat mir die Arbeit mit der Projektgruppe gefallen, und ich habe viele Erfahrungen sammeln können.

Chang Gao

Diese Projektgruppe hat mir wirklich sehr viel positives gebracht. Sie ermöglicht mir ein interesantes Softwareprojekt mitzuentwickeln. Während dieser einjährigen Arbeit habe ich praktische Erfahrungen in der OO-Softwareentwicklung in Java, von Konzeptaufbau und Analyse der Anforderungsdefinition bis hin zur Implementierung, gesammelt. Ich habe dadurch viel neues gelernt, sowohl die Kenntnisse in HIT, Leistungsbewertung und LINUX, als auch die Art und Weise in Team zu arbeiten. Darüberhinaus konnte ich auch meine Kenntnisse in Java und UML festigen und erweitern.
Insgesamt wurde von jedem Mitglied in seinem Rahmen gute Arbeit geleistet. Somit kann das Projekt HUML am Ende pünktlich und erfolgreich abgeschlossen werden. Ich denke, mit dem erreichten Ergebnis kann die Projektgruppe zufrieden sein. Desweiteren freue ich mich auch alle Teilnehmer näher kennengelernt zu haben und bedanke mich bei allen, die mir bei Problemen warmherzig geholfen haben. Ich hoffe, dass wir nach der PG noch weiter in Kontakt bleiben können.

Esmeray Gencaslan

Das Arbeiten als Projektgruppenmitglied war für mich eine notwendige und lehrreiche Erfahrung. Das in Haus Villigst durchgeführte Seminarprogramm im ersten Semester stellte eine wesentliche Voraussetzung für ein effektives Arbeiten, da die Mitglieder sich besser kennenlernen konnten, dar. Ich konnte zudem bereits vorhandenes Wissen auffrischen und neues aneignen. Auch habe ich Probleme kennengelernt, die eine Gruppenarbeit mit sich bringt. Gegen Ende der Projektarbeit spitzte sich die Dynamik innerhalb der Gruppe fast zu. Trotzdem soll man nicht vergessen, dass sogut wie jede PG mit solch einem Stress Bekanntschaft gemacht hat. Ich möchte allen HUML-Entwicklern gratulieren und wünsche allen ein erfolgreiches und erwünschtes Berufsleben. Mein Dank geht auch an unseren Betreuer Jürgen Mäter, der immer mit Verständnis und Geduld die Gruppe auf die richtige Fährte gebracht hat.

Jiuxin Huang

Ich bin mit unserer Projektgruppe sehr zufrieden. Wir haben zusammen ein Jahr lang überwiegend schöne Zeit erlebt, dazwischen haben wir natürlich auch harte Arbeit geleistet. Die PG war gut strukturiert. In der Seminarphase haben wir die Möglichkeit gehabt uns näher kennenzulernen, und in der Praktikumphase sammelten wir die notwendigen Kenntnisse für Leistungbewertung. Wir haben ein kreatives Thema bekommen, weil ganz wenige Projekte in diesem Bereich durchgeführt werden. Im erstem Semester haben wir uns auf das Konzept konzentiert, die bis Anfang des zweiten Semesters andauerte. Deswegen habe ich zwischendurch

Sorgen gemacht, dass wir unser Zeitplan nicht rechzeitig halten können. Aber überraschenderweise haben wir unser Tempo in der Implementierungsphase beschleunigt. Darauf aufbauend habe ich gelernt, wie wichtig eine gutes Konzept und ein Projektmanagement ist. Die PG-Mitglieder waren sehr freudlich und hilfsbereit. Ich konnte von anderen viel lernen. Auch hab ich über Teamarbeit gute Erfahrung, die für ein späteres Berufsleben unabdingbar ist, sammeln können. Ich bedanke mich auch bei unserem Betreuer Jürgen Mäter, der immer den Überblick gehalten hat und uns auf die richtige Routine führte.

Thomas Kopinski

Meiner Meinung nach war die PG eine interssante Erfahrung. Trotz anfänglicher Bedenken bezueglich der Thematik, die mir voellig neu war, entwickelte sich das Projekt insgesamt positiv. Die wichtigsten zu beachtenden Aspekte sind Disziplin und Absprache. Viele Verzögerungn hätten im Verlauf des Semesters nicht sein müssen, wenn sich jeder um seine Aufgaben gekümmert hätte. Zudem lief insbesondere im ersten Semester nicht alles nach Plan, was sich letzten Endes in der äußerst späten Abgabe des Zwischenberichts wiederspiegelte. Wie sicherlich auch von anderen Mitgliedern bemerkt, ist der sorgfältige Entwurf das Wichtigste am gesamten Projekt. Selbst kleinere Änderung in diesem Bereich stellten uns vor große Planungs- und Implementierungsprobleme. Alles in Allem gehe ich nach einem Jahr mit einem positiven Eindruck aus dieser PG.

Oliver Lazar

Eigentlich weiß man nur, wie wenig man weiß. Mit dem Wissen wächst der Zweifel.
Johann Wolfgang von Goethe

Die Projektgruppe war ohne Zweifel eine Erfahrung.

Jan Papenfuß

Ich denke, daß man das Gesamtergebnis der Projektgruppenarbeit als befriedigend bezeichnen kann. Klar ist, daß auf fachlicher Seite wesentlich mehr, und auf höherem wissenschaftlichen Niveau, hätte erreicht werden können. Nun muß man allen Projektgruppenmitgliedern grundsätzlich zubilligen, daß nur wenige von ihnen wirkliche, profunde Vorkenntnisse über den Sachbereich Softwareentwurf und niemand welche über das Thema Leistungsbewertung mitbrachte und damit einbringen konnte.

Das Vorbereitungsseminar im Oktober 2003, das ich als rundum gut gelungen bezeichne, hatte zunächst darauf hoffen lassen, daß Lern- und Einarbeitungsbereitschaft bei allen Teilnehmern in so großem Maße vorhanden seien, daß die Projektgruppenarbeit zu einem vollen Erfolg würde geführt werden können. Dieser Eindruck wurde in meinen Augen aber schon während des Modellierungspraktikums gedämpft, weil der Umgang mit den Leistungsbewertungswerkzeugen HIT und HITGRAPHIC und das Verständnis der dahinterstehenden Theorien nicht von allen Projektgruppenmitgliedern in ausreichend erlernt werden konnten. Dieser Mangel hat sich m. E. leider bis zum Abschluß der Projektgruppenarbeit fortgesetzt, so daß ich bedauerlicherweise feststellen muß, daß der fachbezogene Lerneffekt bei einigen Projektgruppenmitgliedern sehr niedrig ist.

Da ich einen Mangel an Vorverstüdnis als eine wesentliche Ursache für den Mangel an Verständnis betrachte, wären nachträglich besehen einige Mitglieder unserer Gruppe gut beraten

gewesen, wenn sie vor Ableistung der Projektgruppenarbeit noch ein paar Semester mehr studiert hätten.

Die großen Unterschiede in der Lern- und Leistungsfähigkeit erzeugten in Verbindung mit dem gewählten Führungsstil Probleme. Dieser bestand nämlich in der Bildung von Kleingruppen, verbunden mit der gleichberechtigten Teilnahme aller an einer zweimal wöchentlich stattfindenden Gruppenaussprache, in der allein das fachbezogene Argument und die fachbezogene Kritik gezählt haben. Mit dieser sehr anspruchsvollen (Nicht-)Organisationsform waren manche offenbar überfordert. In den Gruppensitzungen wurden die fachlichen Unterschiede verstärkt durch charakterliche auf den Gebieten der Redebereitschaft, Initiative, Kritikfähigkeit, Schlagfertigkeit, (Selbst-)Organisationstalent und – Fleiß. Den letzteren kann man zum Glück nur einem Teilnehmer wirklich absprechen. (Daß alle anderen durchaus als fleißig betrachtet werden können, ist unter dem Strich als ein wirkliches Lob gemeint!)

Nun, die angesprochenen Unterschiede haben m. E. dazu geführt, daß zu wenige Projektgruppenmitglieder an den Problemstellungen kreativ gedacht und produktiv mitgedacht haben und sich am Ausdenken der Lösungskonzepte für die ursprünglichen und eigentlichen Projektgruppenaufgaben beteiligen konnten. Hierin ist die Ursache dafür zu sehen, daß gewisse Betrachtungsweisen des Sachgebiets und Ansätze zu den Problemlösungen von zu wenigen Projektgruppenmitgliedern, und daher z. T. zu einseitig, der Projektgruppe aufgeprägt worden sind. Deshalb konnten in bezug auf einige Teile der Aufgabenstellung nur Minimalziele erreicht bzw. von vornherein gesteckt werden. Mit etwas größerer geistiger Beteiligung hätten Sackgassen, Irrwege und methodische Grenzen besser erkannt und durchbrochen werden können.

Meine harsche Kritik am Mangel an Denken soll nicht den Eindruck erwecken, als habe die Projektgruppe nicht auch teilweise Beachtliches geleistet. In bezug auf die Übersetzung von Softwareentwürfen in Leistungsmodelle im allgemeinen und von UML-Modellen in HIT-Modelle im besonderen ist von uns Forschungsarbeit geleistet worden, die zwar nicht gerade einer Pionierarbeit gleichkommt, wohl aber den Horizont der bisherigen Forschung ein wenig erweitert. Hier ist es in Kleingruppen zu sehr fruchtbarer Zusammenarbeit Einzelner gekommen. Und auch Perspektiven zur weitergehenden Forschung sind da (meine Ideen sind mannigfaltig: Einbringung von Datenstromanalysen alternativ oder zusätzlich zur prozeßorientierten Kontrollflußanalyse, Erweiterung der Modellierungsmittel zur detaillierteren Darstellung von Softwarefunktionalitäten, Einführung von Kontrollstrukturen zur Modellierung von Auflösungen oder Umgehungen lokaler Überlastungszonen von Softwaresystemen zur Laufzeit, ...). Dieser Erfolg ist uns nicht zu nehmen und soll nicht kleingeredet werden. Bedauerlich nur, daß nicht alle Gruppenmitglieder daran teilhaben konnten.

Viel Frustration ist durch die Verteilung der mehr oder weniger dankbaren Arbeiten unter den Projektgruppenmitgliedern entstanden. Ich selbst habe keinen Grund, mich zu beklagen. Im Gegenteil betrachte ich meine eigene, weitgehende Beschäftigung mit den eher theoretischen Seiten des Forschunggebiets sowie – im Rahmen des Entwurfs und der Erstellung unseres eigenen Werkzeugs – mit der Praxis von HIT-Modellen als ein Privileg und bin mit meiner eigenen Leistung zufrieden. Auch in anderen Teilen, aus denen unser Werkzeug besteht, ist gute Arbeit geleistet worden. Diejenigen Projektgruppenmitglieder, die an diesen Teilen gearbeitet haben, haben sich Arbeiten aufgebürdet und zu einem erfolgreichen Abschluß gebracht, die mit der zuvor genannten Theorie eher weniger bis gar nichts zu tun haben. Ein großes Lob für diese Disziplin!

Die geringe Beteiligung am Voranbringen der vorbereitenden, theoretischen Forschungen durch eigene Gedankenleistung, mußten einige Projektgruppenmitglieder durch die Erledi-

gung der eher weniger geistig gewinnbringenden Aufgaben ausgleichen. Diese waren kein Spaß. Die sich dadurch ergebenden Frustrationsmomente haben die Projektgruppenarbeit in der Entwurfs- und Implementierungs- (auch hier wäre einigen anzuraten gewesen: erst noch ein paar Semester studieren!), besonders aber in der Endphase (sprich: der Verfassung des Endberichts) belastet. Hier hat die Menge an Arbeitseinsatz gegenüber derjenigen der Befriedigung überwogen. Zum Glück wurde am Ende aber alles zu einem akzeptablen Ergebnis gebracht, und es konnte einiger psychischer Druck nach weitgehend getaner Arbeit durch das hervorragende Abschlußgrillen abgebaut werden. Man hätte an unsere technischen Probleme möglicherweise auch durch mehr Sozialmaßnahmen herangehen können. Daß sich von den wenigen stattfindenden immer wieder einige Mitglieder selbst ausgeschlossen haben, ist diesen negativ anzukreiden.

Die arbeitstechnischen Gesichtspunkte in der akademischen Praxis zu erlernen – und das auf einem Weg sowohl mit wissenschaftlichen Erfolgen als auch mit Niederlagen –, betrachte ich neben dem Sachgebiet selbst ebenfalls als einen der Zwecke der Projektgruppenarbeit als solcher, und ihn hat unsere Arbeit in jedem Fall erfüllt – in positiver wie in negativer Hinsicht. Ich bin ziemlich sicher, daß jeder von uns in späteren Projekten – dann des Arbeitslebens – von ihr profitieren wird, indem er einige der Fehler dieser Projektgruppe zu meiden suchen wird, durch die positiven Erfahrungen gestärkt sein wird.

Thomas Pauli

Die Projektgruppe ist mit Sicherheit eine sehr sinnvolle Veranstaltung. Wir haben uns in ein vollkommen fremdes Thema eingearbeitet, Lösungen entwickelt und diese durch ein Software-Tool umgesetzt.
Insbesondere der Software-Entwurf war, wie zu erwarten, nicht perfekt. Allerdings war er wesentlich besser als im SoPra, so dass man auch hier von einem Lernfortschritt sprechen kann. Jedes Mitglied der Projektgruppe hat sich, meiner Meinung nach, seinen Möglichkeiten ensprechend, eingesetzt. Allerdings gab es natürlich Unterschiede bei der konstruktiven Mitarbeit. Anfängliche Differenzen bezüglich des einzusetzenden Arbeitsaufwands konnten ausgeräumt werden.
Unterm Strich muss ich sagen, dass mir die PG sehr viel Spaßgemacht hat. Die Arbeit in der Gruppe hat ganz gut funktioniert und wir haben ein, aus meiner Sicht, sehr zufriedenstellendes Ergebnis erzielt.

Marek Schekel

Im großen und ganzen war ich mit der PG recht zufrieden. Hin und wieder erwies es sich meines Erachtens als Schwierigkeit die Aufgabenstellung korrekt umzusetzen, da nicht immer ganz eindeutig war, was eigentlich das Endergebnis sein sollte welches von uns verlangt wurde. Dies brachte uns auch hin und wieder in Rückstand im Zeitplan, der aber ansonsten recht gut durch uns eingehalten werden konnte.
Es war interessant zu erfahren wie die Zusammenarbeit mit lauter verschiedenen Persönlichkeiten funktioniert, denn auch Meinungsverschiedenheiten traten hin und wieder auf. Das hat mir gezeigt, dass es nicht immer einfach ist den Willen aller unter einen Hut zu bringen.
Der Arbeitsaufwand für die PG hielt sich im Rahmen der 16 SWS, die laut der PG-Ordnung dafür vorgesehen sind. Es gab sicherlich eine Tage und Wochen wo etwas mehr gemacht werden mußte, weil der Zeitplan ja eingehalten werden wollte. Da hat sich aber auch gezeigt, dass

eine gute Arbeitsteilung und vor allem das Verteilen von Aufgaben entsprechend der Stärken der einzelnen PG-Mitglieder das gewünschte Ergebnis am Ende hervorbringt. Sicherlich hat man immer wieder Leute in einer Gruppe, die sich nicht so stark einbringen können, dies sollte man jedoch nicht zu streng betrachten, da jeder einmal in eine solche Situation kommen kann und was am wichtigsten ist: man kennt sich nur aus der PG und nicht persönlich, ein kleiner aber feiner Unterschied, weshalb man mit Urteilen über andere PG-Mitglieder nicht zu voreilig sein sollte. Sicherlich ist es schwer, aber ich persönlich vertrete den Ansatz: jeder gibt sein Bestmögliches und das Endergebnis zählt. Ich persönlich halte es sowie nicht als besonders kollegial, und schon garnicht in einer Teamarbeit, meine Teammitglieder niederzumachen oder ihnen vorzuwerfen sie wären z.B. faul. Denn man sollte niemals aus dem Auge verlieren, auch wenn man noch soviel Stolz hat und noch soviel kann, dass man nicht das Ende der Welt ist - behandle den anderen so wie du selbst behandelt werden möchtest. Was ich damit sagen will: auch der, der so handelt, wird einmal in genau die Situation seines Gegenüber kommen. In der Berufswelt tut man sich dadurch auch keinen Gefallen, denn das sind meistens diejenigen Leute, die aus der Sicht des Chefs das gesamte Klima einer ansonsten gut funktionierenden Gruppe stören und somit auch meistens als erste "gehen" dürfen. Insofern möchte ich den Nachfolger-PG's mit auf den Weg geben die Zusammenarbeit so gut wie möglich zu gestalten und oben genanntes niemals aus dem Auge zu verlieren.
Was mir persönlich die Arbeit während der PG sehr erschwert hat, ist die mangelnde Ausstattung des PC-Pools mit entsprechender Software. So war es z.B. nicht einmal möglich GIF-Dateien zu erzeugen, da aufgrund der Verwendung von OpenSource-Software kein vernünftiges und funktionsfähiges Tool verfügbar war. Auch gab es immer wieder Abstürze von notwendiger Software oder sie lief einfach nicht korrekt. So kam es vor, dass simple Arbeiten (die mit entsprechender Software, wie sie z.B. unter einem Windows System zu Verfgung steht) zu einem langen Akt wurden, weil erstmal in der zur Verfügung gestellten Arbeitsumgebung Workarounds gefunden werden mußten um dasselbe Ergebnis zu erzielen. Insgesamt empfand ich die Arbeit im PC-Pool als nicht effizient, denn es kann einfach nicht sein, dass Standardaufgaben, die vielleicht gerade mal 5 Minuten brauchen sich zu einer Stunde ausdehnen, und das nur weil die zu Verfügung gestellte Arbeitsumgebung mangelhaft ausgestattet ist. An dieser Stelle würde ich allen zukünftigen PG'lern, die dasselbe Problem haben, raten mit ihrem Laptop bzw. einer Arbeitsumgebung zu arbeiten, die ihnen alles das bietet, was sie für produktives Arbeiten benötigen - schont die Nerven und spart enorm Zeit. Insgesamt ist die PG aber zumindest aus meiner Sicht erfolgreich verlaufen und wir alle haben uns den Schein verdient.

Sebastian Schürmann

Ein Jahr PG liegen nun hinter uns. Der Anfang begann für mich mit der großen Erleichterung, doch noch eine PG zu bekommen. Denn zuerst sah es so aus, dass alle PGs bereits voll seien oder nicht zu stande kommen sollten. Ein Semester länger studieren, obwohl ich doch in der Regelstudienzeit fertig werden wollte, war keine gute Nachricht. Doch dann kam die E-Mail von Jürgen Mäter, ob ich an der PG 438 mitmachen wollte. Sofort bestätigte ich die Anfrage. Der Inhalt der PG klang ganz gut, aber leider hatte ich mich vorher nicht mit Leistungsbewertung beschäftigt. Ich begann mich einzulesen. Beim ersten Treffen bekam ich den Vortrag im Einstiegsseminar, den ich haben wollte: UML. Die bei dem Vortrag angegebene Literatur hatte ich während des SoPras im Grundstudium schon eifrig verschlungen. Nach langen Vorbereitungen begann das Seminar. Die anderen Mitglieder der PG schienen ganz nett zu sein.

Ich bin froh, dass wir uns entschlossen, das Seminar in Haus Villigst durchzuführen. Der Ort eignete sich hervorragend für die Vorträge und das Rahmenprogramm. Zwölf Vorträge über Themen, wo man sich bei vielen nicht viel darunter vorstellen kann, sind recht anstrengend. Leider kann man sich dabei nicht so viel merken, wie man möchte. Trotzdem habe ich sehr viel dabei gelernt, genauso wie beim darauf folgenden MoPra. HIT war mir vorher nicht bekannt gewesen. Glücklicherweise gab es eine systematische Einführung. Bei der Gruppenarbeit konnte man die ersten Leute näher kennenlernen. Darauf folgten viele weitere Phasen, in denen wir unser Konzept und die Software entwickelten. Die verschiedensten Arbeiten warteten auf uns. Ich stellte fest, dass ich bei weitem nicht der einzige war, für den die Leistungsbewertung relativ neu war. Aber alle hängten sich rein, lernten und erarbeiteten viel. Es war glücklicherweise kein Störenfried oder Faulpelz dabei. Einige Zeit kam es mir so vor als wenn wir die einzige PG seien, denn fast immer waren Leute von uns im PG-Rechnerpool, oft mehr als andere. Viele von uns hatten schlechte Erfahrungen im SoPra gemacht und wollten es nun besser machen. Und in der Tat kam mir die PG-Arbeit viel angenehmer und effektiver vor. Doch natürlich hatten wir auch so unsere Probleme. Nach Weihnachten erinnerte sich kaum noch jemand an die tollen Konzepte, die wir vor den Feiertagen entworfen hatten, so dass wir uns neu einarbeiten mussten. Glücklicherweise lernten wir daraus, so dass dieses Problem nach der Vorlesungsfreien Zeit nicht in dem Maße auftrat, obwohl die PG-freie Zeit viel länger war. Auch Zwischenmenschlich traten Problem auf. Glücklicherweise kamen wir alle ganz gut zusammen aus. Ich glaube, es sind keine lebenslangen Feindschaften entstanden. Aber trotzdem spielten uns die gruppendynamischen Prozesse übel mit. Eine Zeit lang hatten wir uns sogar richtig in den Haaren. Ich glaube es war nach der Weihnachtszeit, als die Arbeit selbst sehr schleppend verlief. Statt an den Problemen selbst zu arbeiten begannen wir, politische Diskussionen zu führen. Es wurde die Frage diskutiert, ob unserer sozialer Zusammenhang oder die Leistungsfähigkeit des einzelnen der einzige richtige Weg sei. Glücklicherweise überstanden wir auch diese Zeit. Zusammenfassend hat mir die PG gut gefallen. Die Leute waren nett und bemühten sich, Leistung hervorzubringen. Und ich denke, wir haben einiges vollbracht. Dafür das am Anfang niemand auf das Thema der Leistungsbewertung spezialisiert war, haben wir uns viel erarbeitet. Unser Konzept stellt in meinen Augen ein interessanter Weg dar. Und unsere Software ist trotz zwischenzeitlichen Pufferengpässen rechtzeitig fertig geworden. Ich hoffe, unsere Ergebnisse halten stand in den Augen der fachlichen Experten der Leistungsbewertung. Ich hoffe, wir konnten unseren Endbericht von den schlimmsten Formulierungen befreien. Und ich hoffe, unser PG-Vortrag wird erfolgreich. Ich bin gespannt, wem ich nach der PG wieder begegnen werde. Viele fangen wie ich nach der PG mit der Diplomarbeit an, so dass man sich an der Uni wohl eher selten sehen wird. Auf jeden Fall wünsche ich allen Mitgliedern der PG Erfolg und Glück, wo auch immer es den Einzelnen hin verschlagen mag.

Michael Stange

Diese PG hat wieder einmal gezeigt, wie wichtig ein sauberer und ordentlicher Entwurf ist. Nach dem Software-Praktikum wussten alle, dass dem so ist. Vor dem Entwurf war ich mir auch sicher, genau darauf zu achten, aber wieder einmal ist es an der Disziplin gescheitert. Diesen Fehler haben nicht alle in der PG begangen, aber ein paar. Ich kann also nur wieder einmal sagen, man unterschätzt den Entwurf immer noch.
Insgesamt denke ich, dass wir das Lernziel der PG erreicht haben. Es war eine interessante Erfahrung für so lange Zeit mit so unterschiedlichen Leuten zusammen zu arbeiten. Es hat auch an einigen Stellen kleinere Meinungsverschiedenheiten gegeben, allerdings wurden sie

auch wieder gelöst. Ich hatte perönlich Schlimmeres erwartet.
Ich kann also nur sagen, dass die PG, auf das spätere Arbeitsleben bezogen, eine der besten Veranstaltungen während des Studiums war und ich davon am meisten für später mitnehmen werde.

Reimar Twelker

Ich bin mit dem Ergebnis der Projektarbeit nicht zufrieden, da die Gruppe meiner Ansicht nach zu keiner Zeit wirklich gut gearbeitet hat und besonders zum Ende hin (gegen Ende des zweiten Semesters) eine sehr schlechte Arbeitseinstellung hatte. Die Gruppe hatte (fast) keine Vorkenntnisse im Bereich der Leistungsbewertung, sodass zu Beginn eine Menge Grundwissen und Erfahrung gesammelt werden musste. Dies erfolgte z.T. durch ein Seminar, das jedoch rückblickend nicht sonderlich sinnvoll gewesen ist, da die meisten der dort behandelten Themen entweder im Laufe der Projektarbeit keinen großen Stellenwert hatten oder sie nur für einen bis wenige Gruppenteilnehmer von unmittelbarem Interesse waren. Wir haben im Rahmen des Seminars aber einerseits Erfahrung in der Vorbereitung und Durchführung von Vorträgen gesammelt und andererseits die anderen Gruppenteilnehmer kennengelernt. Die Gruppe hätte sich daraufhin und am besten während der gesamten Projektdauer besser in das Thema der Gruppe einarbeiten müssen, da bei der Vorbereitung des Abschlussgesprächs festgestellt werden musste, dass niemand wirklich in der Lage ist, das Thema angemessen vorzustellen. Der Mangel an Fachwissen (auch meinerseits!) trug auch dazu bei, dass die Fertigstellung des Endberichts nach Ende des zweiten Semesters in der vorlesungsfreien Zeit weitergeführt werden musste, da er u.a. eine Vielzahl inhaltlicher Schwächen hatte. Dieser Teil unserer Arbeit wurde durch die schlagartige Abwesenheit eines großen Teils der PG nach Ende des Semesters erschwert.

Das erste Semester des Projekts war durch Planung und Einarbeitung bestimmt. Die Gruppenaufteilung klappte meiner Ansicht nach recht gut, die Aufgaben wurden von den meisten gut bewältigt. Allerdings litt die Arbeit unter der starken Zurückhaltung einiger Gruppenteilnehmer, die wenig bis nichts zu der Projektentwicklung im ersten Semester beigetragen haben. So etwas sollte zukünftig nicht möglich sein. Darüberhinaus beging die Gruppe den im Nachhinein kritischen Fehler, ein wichtiges Thema der Projektgruppe einem Einzelnen zu überlassen, sodass die übrigen elf von diesem Thema nur wenig verstanden. In Hinsicht auf die Korrektur des Endberichts war dies äußerst ungünstig.

Darüberhinaus war die Arbeit am Zwischenbericht der Gruppe verkrampft, da durch stark mangelnde Disziplin unnötig viele zeitraubende Korrekturen notwendig waren und der endgültige Bericht trotzdem noch immer etliche Fehler enthält. Schlimmer noch war dann die Korrektur des Endberichts, da die Gruppe aus den gemachten Fehlern nichts gelernt hatte und der Korrekturbedarf extrem hoch war. Hinzu kam, dass die Gruppe erst nach Ende des zweiten Semesters auf die gute Idee kam (gebracht wurde?), den Bericht im Ganzen zu korrigieren und auf Einheitlichkeit zu prüfen.

Im zweiten Semester wurde der Entwurf vertieft, Klassendiagramme angefertigt und die Software implementiert. Besonders ärgerlich empfand ich den extremen Aufwand bei der Dokumentation der Klassendiagramme, der meiner Ansicht nach in keinem Verhältnis zum späteren Nutzen stand, meine Teilgruppe (inklusive mir!) die Aufgabe sehr schlecht gelöst hat, so dass am Ende kein Teil der ursprünglichen Dokumentation aktuell und brauchbar war. Diese Erfahrung halte ich aber für wichtig, da sie zeigt, dass für den Entwurf eine Menge Zeit und Sorgfalt nötig ist, um später davon zu profitieren. Der Aufwand von Zeit allein ist nicht

ausreichend! Es fehlte in der Teilgruppe klar an gemeinsamen Planungen und Absprachen. Vielmehr hat jeder für sich gearbeitet. Ich habe allerdings den Eindruck, dass der Entwurf bei mindestens einer anderen Teilgruppe wesentlich besser gemacht worden ist.

Die Implementierung des Entwurfs verlief zügig und ohne größere Komplikationen. Der Produkttest war ein Ärgernis für sich, da ein Teil der Gruppe, der zunächst damit beauftragt war, innerhalb weniger Tage damit fertig geworden sein wollte. Im Nachhinein stellte sich heraus, dass diese Gruppe nichts geleistet hatte, das man als Produkttest bezeichnen kann. Die PG war daraufhin allerdings zu träge, diesen Missstand durch die Einteilung einer neuen Testgruppe schnell zu beseitigen, sodass der gesamte Produkttest nach Ende des zweiten Semesters durch eine geschrumpfte Gruppe durchgeführt werden musste.

Insgesamt bin ich mit dem Ergebnis der Gruppenarbeit unzufrieden, habe aber eine Menge gelernt und bin der Ansicht, dass die Veranstaltung Projektgruppe trotz der vielen schlechten Erfahrungen ein wichtiger Teil meines Studiums ist. Das Klima in der Gruppe war im Wesentlichen gut und die negativen Seiten von Teamarbeit sollte man kennengelernt haben, bevor man im Berufsleben damit konfrontiert wird, da sie eine Menge Stress und schlechte Stimmung erzeugen können!

Ich möchte ausdrücklich hinzufügen, dass ich Jürgen Mäter für einen sehr guten PG-Betreuer halte, da er die Gruppe im Wesentlichen arbeiten ließ und nur selten lenkend eingriff. Wir mussten selbstständig an das Thema herangehen, bekamen wenig Hilfestellungen und mussten unsere Fehler ziemlich konsequent ausbaden. Er legt (sicher zu Recht) viel Wert darauf, dass die Dokumentation in Form von Zwischenbericht und Endbericht anständig ausfällt. Es wäre allerdings wertvoll für zukünftige PGs, wenn der Betreuer (wer auch immer das ist) härter gegenüber faulen oder den Aufgaben nicht gewachsenen Gruppenmitgliedern wäre, da eine hohe Toleranz seitens des Betreuers von den arbeitenden Gruppenteilnehmern als unfair aufgefasst werden könnte und damit die Moral sinkt.

6. Ausblick

HUML ist ein Prototyp für die Übersetzung von UML-Diagrammen in HIT-Modelle, und damit für die Leistungsbewertung von durch UML-Diagrammen dargestellten Systemen. Da der Zeitrahmen der PG438 eingeschränkt ist, werden nur ausgewählte Stereotype und Eigenschaftswerte des UML-RT und einige selbstdefinierte Stereotype und Eigenschaftswerte unterstützt und somit nur eingeschränkte HIT-Modelle erstellt.

Es folgt eine unvollständige Liste von Erweiterungen, mit denen der Funktionsumfang von HUML verbessert werden kann.

- Bisher werden nicht alle Stereotype und Eigenschaftswerte des UML-RT einbezogen. Es bleibt abzuwägen, welche dieser Mittel für HUML eingesetzt werden können.

- Die Beschränkung auf Aktivitätsdiagramme ist für die Erstellung eines Prototypen, der einen Ansatz zur Leistungsbewertung von UML-Modellen mit HIT liefert, sinnvoll. Die Hinzunahme weiterer Diagrammtypen kann eine differenzierte Modellierung der Software ermöglichen. So könnte beispielsweise durch die Berücksichtigung von Klassendiagrammen ein konkreter an den Softwareentwurf gebundenes Modell erstellt werden. Damit können die einzelnen Aktivitäten mit Methoden des Klassendiagramms in Verbindung gebracht werden, und Attribute der Klassen können in die Übersetzung nach HI-SLANG einbezogen werden und von HUML bisher verwendete Zufallszahlen ersetzen, z.B. im Fall der Entscheidungsknoten und Kantenwahrscheinlichkeiten in Aktivitätsdiagrammen.

- Es fehlen Erweiterungen zur differenzierten Modellierung des Ressourcenmanagements von passiven Ressourcen, d.h. das Belegen und Freigeben von passiven Ressourcen. Bisher werden passive Ressourcen nur solange belegt, bis ein Prozess vollständig bearbeitet worden ist. Es ist vorstellbar, dass durch ein Ressourcenmanagementsystem passive Ressourcen über die Dauer mehrerer Prozesse hinweg belegt werden können.

- HUML bewertet ausschließlich komplette Modelle. Die Bewertung von einzelnen Aktionszustandspfaden ist nicht möglich. Dies kann durch einen Eigenschaftswert für das Stereotyp «PAstep» verbessert werden.

- Bisher darf nur eine Arbeitslast für das Modell angegeben werden. HUML kann verbessert werden, indem mehrere Arbeitslasten, die durch Prioritäten unterschieden werden, verarbeitet werden können. Dabei sind auch gemischte Lasten, d.h. sowohl offene und geschlossene Lasten, denkbar.

- HIT erlaubt es bei der Simulation den Startwert des Zufallszahlengenerators zu definieren (*seed*). Dieses ist bei HUML nicht vorgesehen.

- Die Verwendung der grafischen Oberfläche von HUML erleichtert den Umgang mit dem Tool. Allerdings ist eine Erweiterung des Tools um Kommandozeilenbedienbarkeit wünschenswert. Damit wäre es möglich, HUML in Skripten einzusetzen.

- Bisher läuft HUML nur auf UNIX-Betriebssystemen. Das kommt daher, dass zur Leistungsbewertung HIT verwendet wird und HIT nur unter UNIX funktioniert. HUML muss auf demselben System ausgeführt werden, auf dem auch HIT installiert ist, da bisher nur so ein Aufruf von HIT stattfinden kann. Die Schaffung eines Moduls, das den Aufruf von HIT über eine *ssh*-Verbindung ermöglicht, würde HUML plattformunabhängig machen. Dann kann das annotierte UML-Modell lokal erstellt werden, während die Leistungsbewertung extern ausgeführt wird. Anschließend muss dann die HIT-dump-Datei auf den lokalen Rechner kopiert und die Interpretation der Ergebnisse durch HUML gestartet werden.

A. Tagged Value Language (TVL)

Die *Tagged Value Language* (TVL) definiert eine formale Sprache, um *tagged values* (Eigenschaftswerte) zu spezifizieren. Ausdrücke verwenden dabei folgende Syntax: {‹tag-name› = ‹TVL-Ausdruck›}. Der TVL-Ausdruck kann ein einfaches Literal, wie eine Zahl, eine Variable oder eine arithmetische Funktion sein. Zahlen werden nur in Dezimalform angegeben, wobei Integer und Fließkommazahlen im positiven und negativen Wertebereich erlaubt sind. Beispiel: {timeout = 60}. Booleans werden nicht durch Literale ausgedrückt, sondern durch zwei vordefinierte globale Variablen $true und $false. Beispiel: isPeriodic = $true. Zeichenketten werden in einfachen Hochkommata angegeben. Beispiel: {MyString='Ich bin Text'}. Mehrere Ausdrücke können hintereinander durch Komma getrennt in Listen kombiniert werden. Beispiel: {timeout = (60, 'sec')}. Ausdrücke können auch durch vordefinierte numerische Funktionen angegeben werden, z.B. exp (arg) als Exponentialfunktion mit dem Exponenten *arg* zur Basis *e*. Für eine detaillierte Darstellung der TVL sei auf den Anhang A des UML-RTs [OMG (2003)] verwiesen.

B. Profile for Schedulability, Performance and Time (UML-RT)

Das *Profile for Schedulability, Performance and Time* [OMG (2003)], im Folgenden als UML-RT bezeichnet, ist die Standardisierung der *Object Management Group* (OMG) zu Realzeitaspekten, wozu auch die Performanzanalyse zählt. Diese Übersicht über relevante Aspekte aus dem *Profile for Schedulability, Performance and Time* bezieht sich auf die Kapitel 1, 2, 3, 4, 7 und 9 der Version 1.0 vom September 2003. Für einen tiefgreifenden Einblick in die Thematik, ist die Lektüre des UML-RTs selbst zu empfehlen.

B.1. Kapitel 1 des UML-RT: Rationale and General Principles

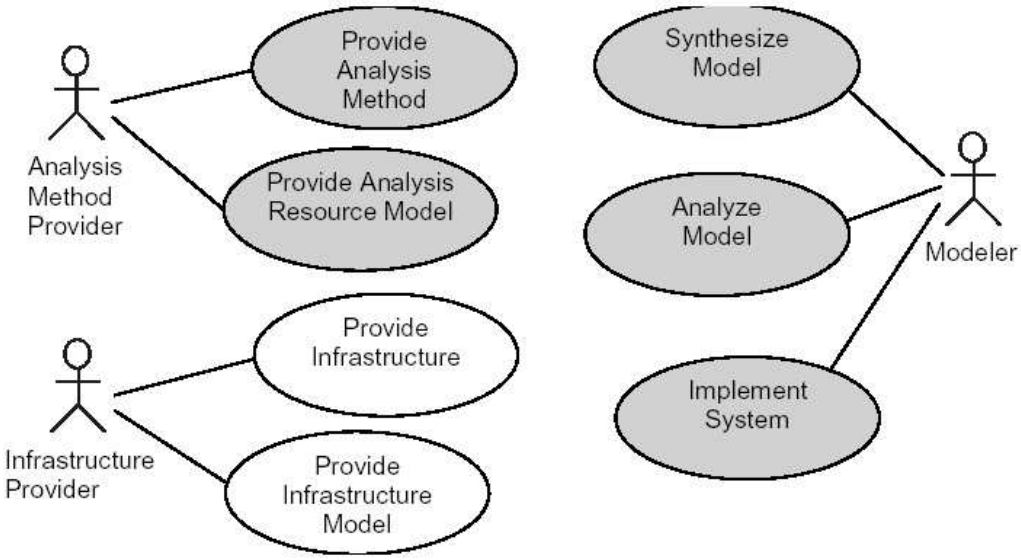

Abbildung B.1.: Anwendungsfälle für die Benutzung des UML-RTs

Im ersten Kapitel werden die generellen Prinzipien der Realzeitmodellierung mit UML aufgezeigt. Die Analysen für Performanz, wobei sowohl analytisch und simulativ als auch mittels Messungen vorgegangen werden kann, sollen möglichst früh im Softwareentwicklungsprozess eingesetzt werden können. Flexible Erweiterungen sollen den Modellierer möglichst wenig einschränken. Außerdem soll möglichst wenig Grundlagenwissen benötigt werden.
Die Benutzung des UML-RTs wird mit dem Anwendungsfalldiagramm aus Abbildung B.1 erklärt: Es gibt drei Arten von Benutzern. Der *Modellierer* modelliert ein System, analysiert seine Modelle (z.B. bezüglich Performanz) und synthetisiert seine Modelle. Der *Analysemethoden-Provider* stellt Analysemethoden und Ressourcenmodellierung zur Verfügung, während der

Infrastruktur-Provider Infrastruktur und Infrastrukturmodell liefert (wie z.B. Realzeitbetriebssystem).

B.2. Kapitel 2 des UML-RT: Approach and Structure

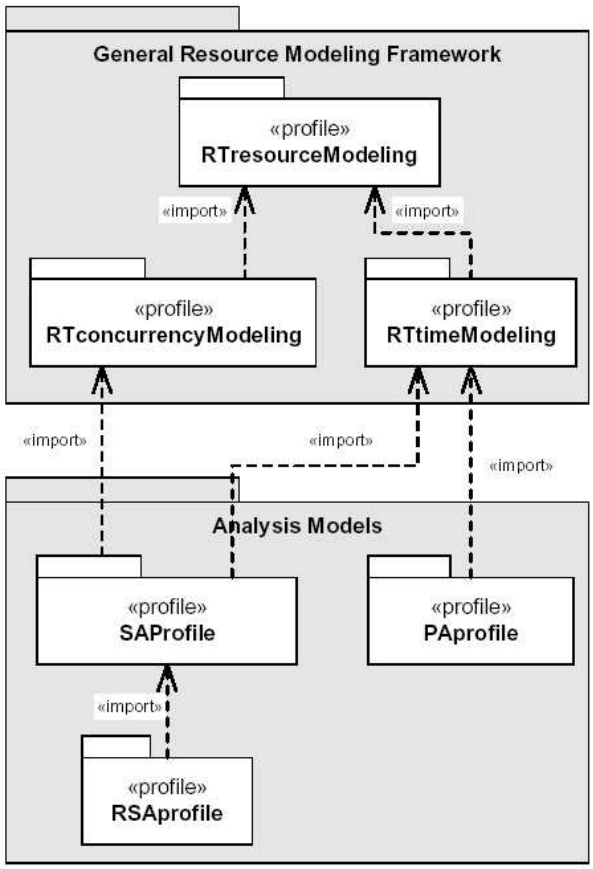

Abbildung B.2.: relevante Struktur des UML-RTs

Das zweite Kapitel beschreibt einen Ansatz der Modellierung von Realzeit, Modellanalyse und -synthese. Außerdem wird der Aufbau des UML-RTs beschrieben (siehe Abbildung B.2). In *Domain Models* zeigen Klassendiagramme die Konzepte, während im *UML View* die Realisierung mit *Stereotypen, Constraints* und *Tagged Values* beschrieben wird. Dabei wird zwischen der grundlegenden Ressourcen- und Realzeitmodellierung (Zusammenfassung der Unterpakete *RTresourceModelling, RTtimeModelling* und *RTconcurrencyModelling* zu *General Resource Modeling Framework GRM*) und den Analysemethoden *Schedulability* und *Performance* (Zusammenfassung der Unterpakete *PAprofile* und *SAprofile* zu *Analysis Models*) unterschieden, die GRM benutzen. Die Erweiterungen werden in Tabellen angegeben, wobei in einer ersten Tabelle zu einem Stereotyp die *Tags* und die zu stereotypisierenden Klassen aufgelistet sind. In einer zweiten Tabelle werden die *Tags* durch ihren Typ, ihre Multiplizität und ihren Namen im *Domain Model* näher beschrieben.

B.3. Kapitel 3 des UML-RT: General Resource Modeling

Das dritte Kapitel stellt die Grundlage für die folgenden Kapitel dar. Es wird das *General Resource Modeling Framework* GRM beschrieben, mit dem zentralen Thema *Quality of Service* (QoS).

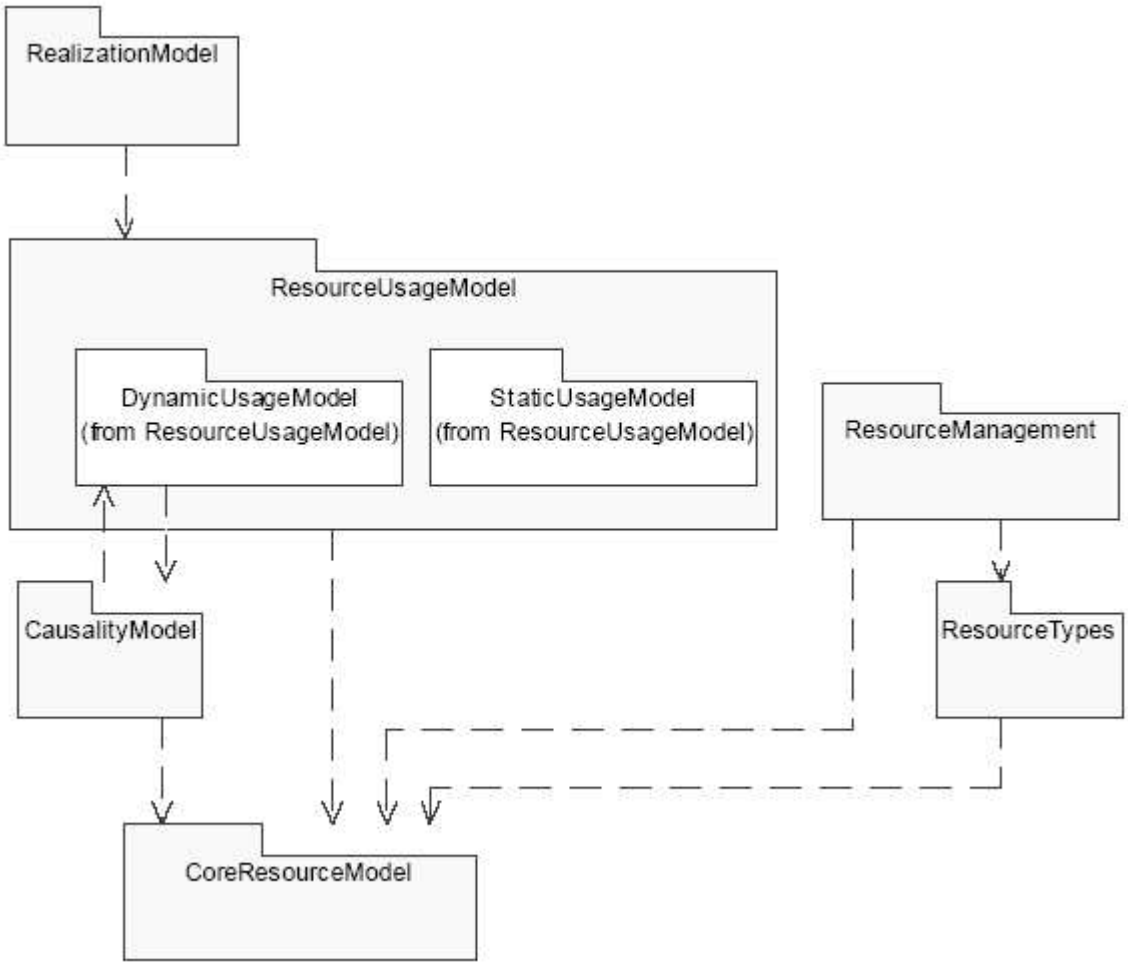

Abbildung B.3.: Struktur des GRM-Pakets

Abbildung B.3 zeigt die Struktur des GRMs, bestehend aus folgenden Unterpaketen:
Das *Core Resource Model* enthält die Basiskonzepte, wobei zwischen Instanzen und Deskriptoren unterschieden wird. Wichtig für Realzeitsysteme ist vor allem die Instanzebene. Ressourcen bieten *Resourceservices* an, die mit QoS beschrieben werden können. Im *Causality Model Package* werden die dynamischen Aspekte beschrieben. Ein Ereignis ist entweder die Erschaffung oder der Empfang einer Nachricht. Die Ausführung beim Stimulus als Methodenaufruf zwischen Objekten ist das Szenario, dessen Anfang und Ende wieder Ereignisse sind. Im *Resource Usage Model Package* wird die *Client*-Sicht der Ressource dargestellt. Ein Analysekontext besteht aus Nachfragen und Ressourcenbenutzung. Statische und dynamische Benutzung wird in Unterpaketen beschrieben. Ein *Client* ist eine statisch benutzte Instanz, der Ressourcen

und ein QoS zugeordnet sind. Bei der dynamischen Benutzung liegt hingegen ein Szenario mit aufeinanderfolgenden Schritten vor. Das *Resource Types Package* unterteilt die Ressourceninstanzen in geschützt/nicht geschützt, aktiv/passiv oder Prozessor/Kommunikation/Gerät. Für geschütze Ressourcen wird eine Zugriffsstrategie beschrieben. Das *Resource Management Package* enthält *Resource Broker* und *Resource Manager*. Das *Realization Model Package* beschreibt verschiedene Schichtmodelle: Durch *refinement* werden die Schichten detaillierter, bei *realization* wird jede Schicht von einer anderen Instanz realisiert, bei *deployment* schließlich wird in UML dem *Client* das *logical* und der Ressource das *engineering model element* zugeordnet. In UML unterscheidet man an *realize-Abhängigkeiten: code* für Programme und *deployment* für Hardwareknoten, mit einer Spezialisierung *requires*. Ist ein *Supplier* mit mehreren *Clients* verbunden bedeutet *implicit*, dass der *Supplier* mit der Kollektion von *Clients* verbunden ist, *explicit* hingegen, dass nur einer davon beteiligt ist. Bei *dynamic* kann der *Client* während der Laufzeit geändert werden, bei *static* nicht. *Deployment* kann *synchron* oder *asynchron* sein. Außerdem kann eine solche Beziehung zwischen Hardware und Software mit *replacement* markiert werden.
Folgende Stereotype werden definiert: *GRMaquire* beschreibt eine Ausführung, die eine Ressource benötigt. Dafür sind die *Tags GRMblocking* und *GRMexclServ* definiert. Das Stereotyp *GMRrelease* hat nur das *Tag GRMexclServ*. *GMRrealize* und seine Kinder *GMRcode, GMRdeploy und GMRrequires* haben das *Tag GMRmapping*. Dieses ist vom Typ *GMRmappingString*, der eine Liste von *TableEntry* in runden Klammern darstellt, welche Namen von *logical* oder *engineering elements* sind, oder *MapType* (*inclusive, exclusive static* bzw. *dynamic*), *MapLinkage* (*sync, async, replace*) oder ein *Constraint*. Das Metamodell von UML 1.4 muss an UML-RT angepasst werden. Die Klasse *ActionExecution* muss mit Assoziationen zu *Stimulus, Instance und Action* integriert werden.

B.4. Kapitel 4 des UML-RT: General Time Modeling

Das *Time Domain Model* basiert auf dem GRM und enthält vier Unterpakete (siehe Abbildung B.4).

Time Model beschreibt Zeit und ihren Wert. Die physikalische Zeit ist eine kontinuierliche, monoton fortschreitende, total geordnete Menge von Zeitpunkten. Gemessen werden können diese mittels einer Uhr diskret als *Integers* und kontinuierlich als *Real*. Eine Dauer zwischen zwei Zeitpunkten kann mittels Intervallen beschrieben werden. Das Paket *Timing Mechanism* enthält die Ressourcen *Clock* für Intervalle und *Timer* für Zeitpunkte. *QoS* sind die Stabilität der Abweichung von einer Referenzuhr, die angibt wie konstant die Rate der Zeitmessung ist, und die Rate dieser Abweichung. Die Mechanismen können gestellt, abgefragt und zurückgesetzt werden. Pause und Wiederaufnahme sind ebenfalls möglich. Das *Timed Events* Paket zeigt einen *Timed Stimulus* als Stimulus aus dem GRM zu einem bestimmten Zeitpunkt. Bei *Clocks* handelt es sich um einen *Clock Interrupt*, bei *Timern* um ein *Timeout*. Die Ankunft eines Ereignisses hingegen wird mit *Timed Event* modelliert. *Timed Actions* wiederum haben eine zeitliche Ausdehnung und daher Start- und Endzeitpunkt und ein Intervall. Das letzte Paket enthält *Time Services*. Diese Dienste sind eine *clock* und eine *Timer Factory* zusammen mit der Verwaltung für selbige.
Abbildung B.5 zeigt das grundlegende *time modelling*-Konzept.

Dieses Konzept wird folgendermaßen in UML umgesetzt. Die physikalische Zeit muss nicht modelliert werden, sondern nur deren Messung. Dies geschieht mittels des Stereotyps «RTtime»

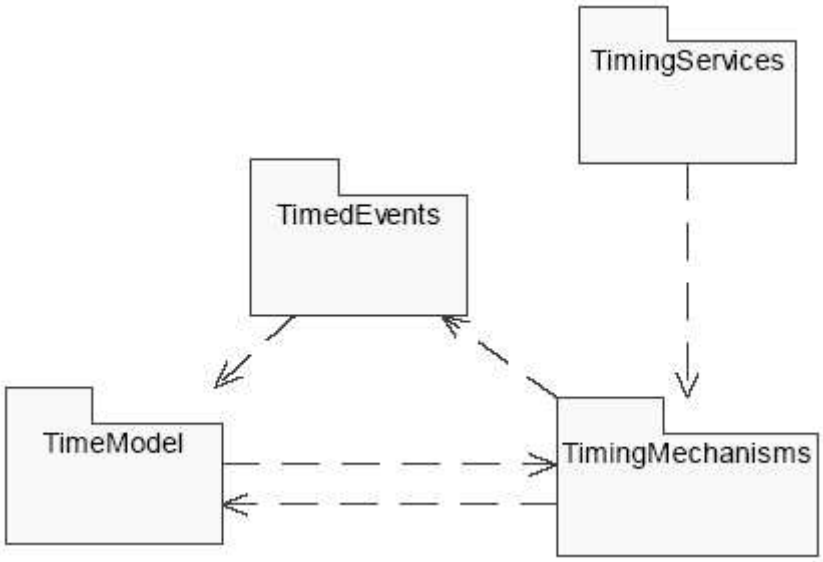

Abbildung B.4.: Struktur des *time domain models*

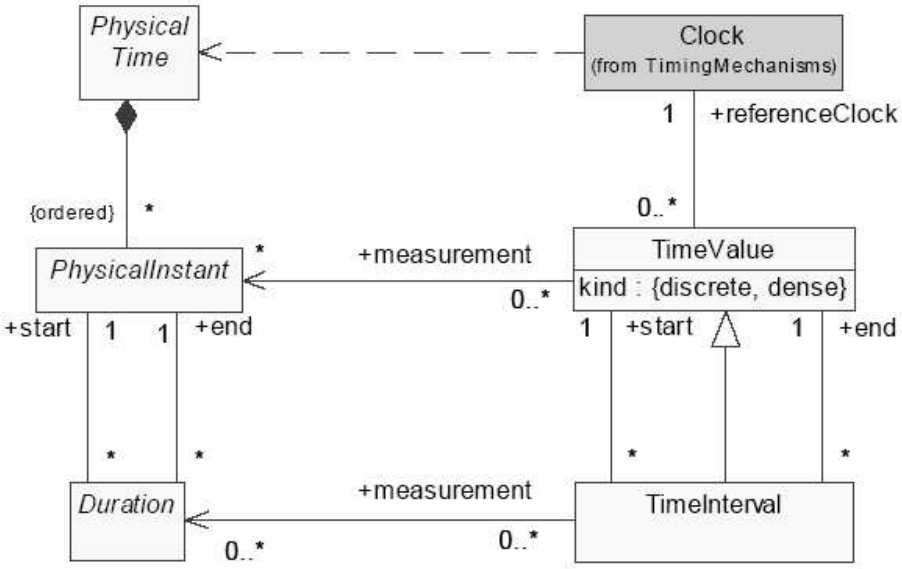

Abbildung B.5.: grundlegendes *time modelling*-Konzept

mit den *Tags RTrefClk* für die Referenzuhr und *RTkind*, das angibt, ob der Wert diskret (*discrete*) oder kontinuierlich (*dense*) ist. «RTintervall» mit den *Tags RTintStart, RTintEnd* und *RTintDuration* gibt ein Intervall mit Start- und Endzeitpunkt sowie der Länge an. «RTtimingMechanism» fasst die gemeinsamen QoS von «RTclock» und «RTtimer» in Tags zusammen. «set», «reset», «pause» und «start» sind Stereotype für Ereignisse von Zeitmechanismen. «RTaction», «RTstimulus» und «RTevent» sind Stereotype der Ereignistypen. *Tags* sind je nach Typ *RTat, RTstart, RTend* und *RTduration*. «RTclkInterrupt» und «RTtimeout»

sind die Stimuli einer *clock* bzw. eines *Timers*, die ebenfalls *RTstart* und *RTend* als *Tags* aufweisen. Eine Verzögerung «RTdelay» kann daneben auch noch *RTduration* haben. Schließlich existiert noch «RTtimeService», dessen erstellte *clocks* und *Timer* mit «RTnewClock» und «RTnewTimer» notiert werden.
Der Datentyp *RTtimeValue* wird von vielen dieser *Tags* verwendet. Er gibt in mehr oder weniger üblichen englischen Abkürzungen Zeitpunkte, Daten und Verteilungen (abgekürzt als PDF für *Probability Distribution Function*) an. Für die genaue Definition sei auf das UML-RT selbst verwiesen. Mit dem Datentyp *RTarrivalPattern* können Ankunftsraten genau spezifiziert werden, wozu auch *RTtimeValue* verwendet wird.
Das Metamodell muss neben den *Actions* auch um sogenannte *Action Execution Timing Marks* erweitert werden. Dies sind *sendTime()* und *receiveTime()* für Stimuli, sowie *startTime()* und *endTime()* für Aktionen. Diese können als *Tags* bei den entsprechenden Stereotypen «RTstimulus» und «RTation» notiert werden.

B.5. Kapitel 7 des UML-RT: Performance Modeling

Die beschriebenen Konzepte können sowohl mit Aktivitäts- als auch mit Interaktionsdiagrammen realisiert werden. Stereotype und *Tags* werden tabellarisch am Ende dieses Unterkapitels zusammengefasst. Das UML-RT verwendet für die Beschreibung von Datentypen die *Tag Value Language* (TVL), die im Anhang A beschrieben wird. Es existieren zwei komplexe Datentypen, die folgendermaßen in TVL definiert sind: *PAperfValue* ist eine Liste mit folgender Syntax: (‹source-modifier›, ‹type-modifier›, ‹time-value›). *Source-modifier* beschreibt, ob der Leistungsparameter gemessen (msr), angenommen (assm), benötigt (req) oder vorhergesagt (pred) wurde. *Type-modifier* gibt die statistische Bedeutung des Wertes an: Mittelwert (mean), Standardabweichung (sigma), k-tes Moment (k-mom, ‹k›), Maximum (max), Perzentil (percentile, ‹real›) oder Verteilung (dist). *Time-value* schließlich gibt den Wert selbst an. Der Typ von *time-value* ist *RTtimeValue* ein intuitiv verständlicher, aber umfangreicher Typ, der in Kapitel 4 (Zeitmodellierung) auf Seite 32 des UML-RTs [OMG (2003)] nachgeschlagen werden kann. Der zweite komplexe Datentyp ist *PAextOpValue*, der eine externe Operation angibt, entweder als *PAperfValue* oder in der Form: ‹Name: ›, Anzahl Wiederholungen.
Im UML-RT folgt ein Anwendungsbeispiel. Außerdem wird erwähnt, dass das Metamodell um *ActionExecution* aus Kapitel 3 erweitert werden muss.
Das gesamte Performanz-Konzept basiert grundsätzlich auf dem sogenannten *performance analysis domain model* (Abbildung B.6).

Das *Domain Model* basiert vollkommen auf GRM (*General Resource Modeling*). Die Beziehungen zwischen Performanz Modellierungs-Konzepten und entsprechenden GRM Konzepten zeigt Abbildung B.7.
Es folgt eine detaillierte Erklärung:

- **Performance Context**
  Ein *performance context* spezifiziert ein oder mehrere Szenarien, die dazu benutzt werden, unterschiedliche dynamische Situationen einschließlich Ressourcen zu untersuchen.

- **Szenario**
  Ein Szenario ist eine Sequenz von einem oder mehreren Szenario-Schritten (*Steps*). Diese *Steps* werden auf einer *Host-Ressource* ausgeführt. Die Tags *hostExecutionDemand* und

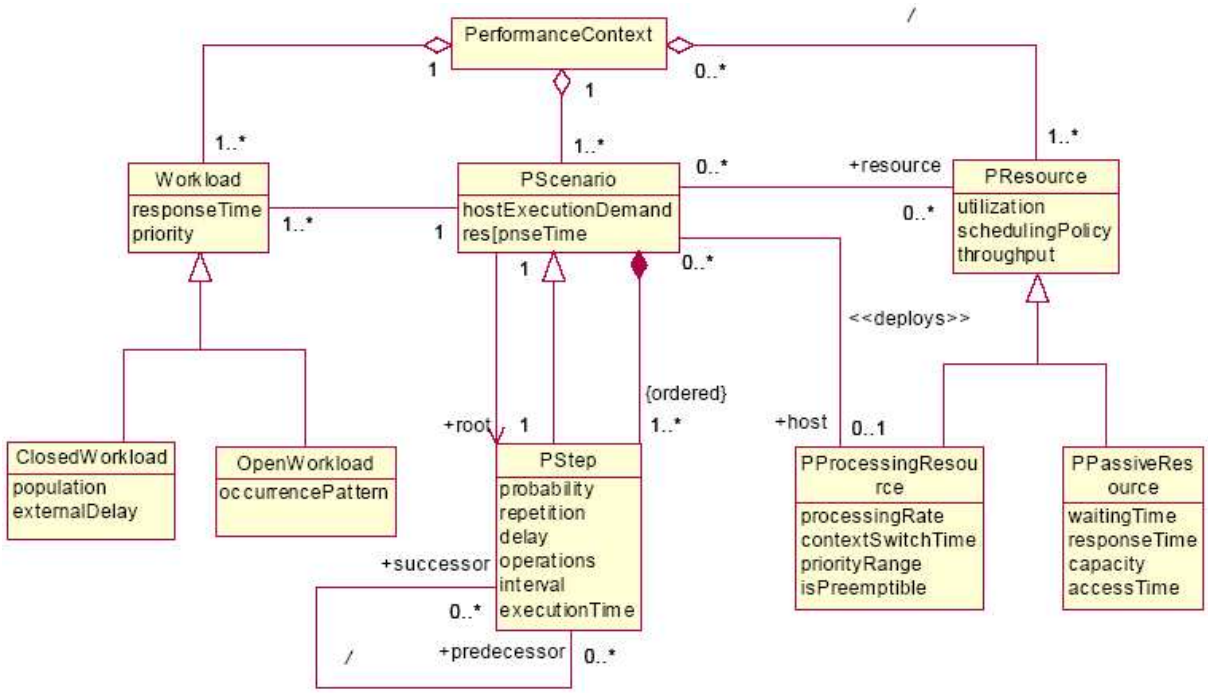

Abbildung B.6.: performance analysis domain model

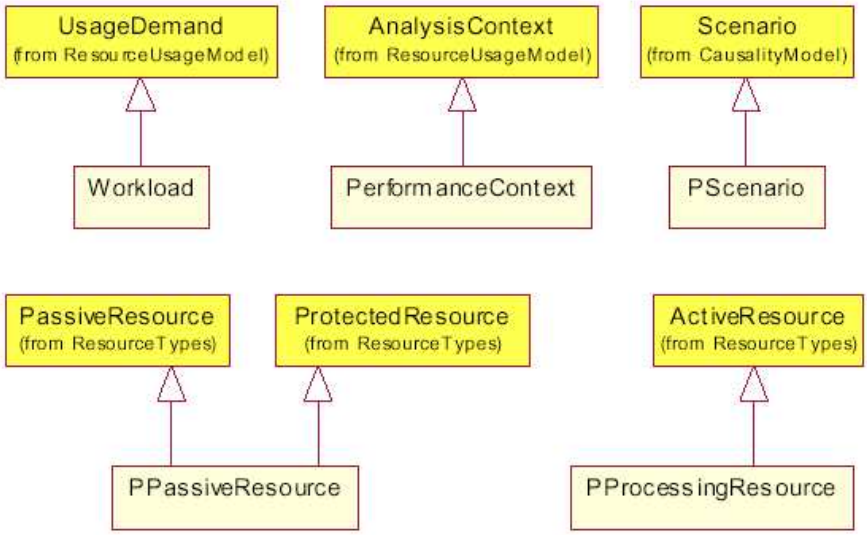

Abbildung B.7.: Beziehung zwischen Perfomance Konzepten und GRM

reponseTime sind Attribute eines Szenarios und werden in Tabelle B.1 beschrieben.

hostExecutionDemand	gibt an, wie viel Zeit für die Ausführung auf der zugewiesenen Ressource beansprucht wird
responseTime	Die gesamte benötigte Zeit, die für die Ausführung des Szenarios gebraucht wird

Tabelle B.1.: Die Eigenschaftswerte eines Szenarios

- **Step**
  Ein *Step* ist ein Ausführungsschritt, der im Allgemeinen eine endliche Ausführungszeit hat. Ein *Step* kann einen oder mehrere Nachfolge- bzw. Vorgänger-*Steps* besitzen. Ein Step kann verschiedene Attribute haben (s. Tabelle B.2).

hostExecutionDemand	gibt an, wie viel Zeit dieser *Step* für die Ausführung auf seiner zugewiesenen Ressource beansprucht (geerbt von Szenario)
delay	Wert einer Verzögerung (z.B. Warten oder Pause)
responseTime	Ausführungszeit für den Step (geerbt von Szenario)
probability	Die Wahrscheinlichkeit, dass dieser Step ausgeführt wird
interval	Das Zeitintervall zwischen den sukzessiven Wiederholungen dieses Steps
repetition	Anzahl der Wiederholungen

Tabelle B.2.: Die Eigenschaftswerte eines Step

- **Resource [abstrakt]**
  Eine abstrakte Sicht auf eine passive oder aktive Ressource, die Teil eines oder mehrerer Szenarien des *performance contexts* ist. Eine Ressource kann, wie Tabelle B.3 zeigt, durch die folgenden Attribute beschrieben werden.

utilization	repräsentiert die berechnete Auslastung der *processing resource* in Prozent
throughput	Die Rate, mit der die Ressource ihre Funktion ausführt
schedulingPolicy	Methode, mit der der Zugriff auf die Ressource kontrolliert wird

Tabelle B.3.: Die Eigenschaftswerte einer Ressource

- **ProcessingResource**
  Eine *processingResource* ist ein Baustein (z.B. ein Prozessor), ein Schnittstellen- oder

Speichergerät. Die Attribute werden in Tabelle B.4 beschrieben.

schedulingPolicy	Folgende Regeln sind vordefiniert: – FIFO = first-in-first-out – HeadOfLine = head-of-the-line or non-preemptible priorities – PreemptResume = pre-empt resume – ProcSharing = processor sharing (representing round robin) – PrioProcSharing = priority processor sharing – LIFO = last-in-first-out (geerbt von Resource)
processingRate	relativer Geschwindigkeitsfaktor für den Prozessor als Prozentsatz relativ zu einem Referenzprozessor
contextSwitchTime	Zeit für das Umschalten der Ausführung eines Szenarios zu einem anderen
priorityRange	enthält gültige Prioritäten für den Prozessor (oft abhängig vom Betriebssystem)
isPreepmtable	zeigt an, ob der Prozessor preemptiv ist, sobald die Ausführung einer Aktion beginnt

Tabelle B.4.: Die Eigenschaftswerte von ProcessingResource

- **Passive Resource**
  Eine *passiveResource* ist eine Ressource, die über einen Zugangsmechanismus geschützt wird (z.B. Semaphore). Die Attribute werden in Tabelle B.5 erläutert.

- **Workload [abstrakt]**
  Die Arbeitslast spezifiziert die Intensität der Nachfrage einer Ausführung eines bestimmten Szenarios. Die Attribute werden in Tabelle B.6 erläutert.

- **OpenWorkload**
  Eine offene Arbeitslast wird als Strom von Anfragen mit einer gegebenen Ankunftsrate modelliert (offenes System), s. Tabelle B.7.

schedulingPolicy	Folgende Regeln sind vordefiniert: – FIFO = first-in-first-out – PriorityInheritance = priority inheritance – NoPreemption = no pre-emption – HighestLockers = highest Lockers – PriorityCeiling = priority ceiling (geerbt von Resource)
capacity	Anzahl gleichzeitiger Benutzer (z.B. *counting semaphore*)
accessTime	Zeitverzug, der beim Zugriff in einem *Szenario-Step* auf eine Ressource benötigt wird.
utilization	die durchschnittliche Anzahl von gleichzeitigen Benutzern der Ressource
responseTime	die komplette Zeit von dem Moment an, wo die Ressource in Anpruch genommen wird bis zur Freigabe der Ressource
waitingTime	Die Wartezeit, bis dass die angefragte Ressource zugeteilt wird

Tabelle B.5.: Die Eigenschaftswerte einer passiven Ressource

responseTime	Der Zeitverzug zwischen dem Zeitpunkt, an dem das Szenario begonnen hat und dem Zeitpunkt, an dem das Szenario abgeschlossen wurde.
priority	Die Priorität der Arbeitslast

Tabelle B.6.: Die Eigenschaftswerte einer Workload

occurencePattern	Das Muster für die Zwischenankunftszeiten

Tabelle B.7.: Die Eigenschaftswerte einer OpenWorkload

- **ClosedWorkload**
  Eine geschlossene Arbeitslast wird durch eine feste Anzahl von Benutzern oder Jobs im System spezifiziert (geschlossenes System), s. Tabelle B.8.

population	Die Größe der Arbeitslast
externalDelay	externer Zeitverzug

Tabelle B.8.: Die Eigenschaftswerte einer ClosedWorkload

Diese Konzepte können sowohl mit Aktivitäts- als auch mit Interaktionsdiagrammen realisiert werden. Stereotype und *Eigenschaftswerte* sind in folgenden Tabellen(B.9 bis B.19) zusammengefaßt.

«PAclosedLoad»
Dieser Stereotyp modelliert eine geschlossene Arbeitslast, s. Tabelle B.10

Stereotyp	Basisklasse	Tags
«PAclosedLoad»	Message Stimulus ActionState SubactivityState Action ActionExecution Operation Method Reception	PArespTime PApriority PApopulation PAextDelay

Tabelle B.9.: Der Stereotyp PAClosedLoad

Tag	Type	Multiplicity	Domain Attribute Name
PArespTime	PAperfValue	[0..*]	Workload::responseTime
PApriority	Integer	[0..1]	Workload::priority
PApopulation	Integer	[0..1]	ClosedWorkload::population
PAextDelay	PAperfValue	[0..1]	ClosedWorkload::externalDelay

Tabelle B.10.: Die Eigenschaftswerte des Stereotyps PAClosedLoad

«PAcontext»
Dieser Stereotyp modelliert einen *performance analysis context*.

Stereotype	Basisklasse
«PAcontext»	Collaboration CollaborationInstanceSet ActivityGraph

Tabelle B.11.: Der Stereotyp *PAcontext*

«PAhost»
Dieser Stereotyp modelliert eine *processing resource*, siehe Tabelle B.13.

Stereotype	Basisklasse	Tags
«PAhost»	Classifier Node ClassifierRole Instance Partition	PAutilization PAschdPolicy PArate PActxtSwT PAprioRange PApreemptable PAthroughput

Tabelle B.12.: Der Stereotyp *PAhost*

Tag	Type	Multipl.	Domain Attribute Name
PAutilization	Real	[0..*]	Resource::utilization
PAschdPolicy	Enumeration: 'FIFO', 'HeadOfLine', 'PreemptResume', 'ProcSharing', 'PrioProcSharing', 'LIFO'	[0..1]	ProcessingResource::schedulingPolicy
PArate	Real	[0..1]	ProcessingResource::processingRate
PActxtSwT	PAperfValue	[0..1]	ProcessingResource::contextSwitchTime
PAprioRange	Integer range	[0..1]	ProcessingResource::priorityRange
PApreemptable	Boolean	[0..1]	ProcessingResource::isPreemptable
PAthroughput	Real	[0..1]	ProcessingResource::throughput

Tabelle B.13.: Die Eigenschaftswerte von PAhost

«PAopenLoad»
Dieser Stereotyp modelliert eine offene Arbeitslast, siehe Tabelle B.15.

Stereotype	Basisklasse	Tags
«PAopenLoad»	Message Stimulus ActionState SubactivityState Action ActionExecution Operation Method Reception	PArespTime PApriority PAoccurence

Tabelle B.14.: Der Stereotyp PAopenLoad

Tag	Type	Multiplicity	Domain Attribute Name
PArespTime	PAperfValue	[0..*]	Workload::responseTime
PApriority	Integer	[0..1]	Workload::priority
PAoccurence	RTarrivalPattern	[0..1]	OpenWorkload::population

Tabelle B.15.: Die Eigenschaftswerte von PAopenLoad

«PAresource»
Dieser Stereotyp modelliert eine *passive resource*, siehe Tabelle B.17.

Stereotype	Basisklasse	Tags
«PAresource»	Classifier Node ClassifierRole Instance Partition	PAutilization PAschdPolicy PAschdParam PAcapacity PAaxTime PArespTime PAwaitTime PAthroughput

Tabelle B.16.: Der Stereotyp *PAresource*

Tag	Type	Multipl.	Domain Attribute Name
PAutilization	Real	[0..*]	Resource::utilization
PAschdPolicy	Enumeration: 'FIFO', 'PriorityInheritance', 'NoPreemption', 'HighestLockers', 'PriorityCeiling'	[0..1]	PassiveResource::schedulingPolicy
PAschdParam	Real	[0..*]	PassiveResource::schedulingPolicy (z.B. bei PriorityCeiling)
PAcapacity	Integer	[0..1]	PassiveResource::capacity
PAaxTime	PAperfValue	[0..n]	PassiveResource::accessTime
PArespTime	PAperfValue	[0..n]	PassiveResource::responseTime
PAwaitTime	PAperfValue	[0..n]	PassiveResource::waitTime
PAthroughput	Real	[0..1]	Resource::throughput

Tabelle B.17.: Die Eigenschaftswerte von PAresource

«PAstep»
Dieser Stereotyp modelliert einen *Step* in einem *performance analysis scenario*, siehe Tabelle B.19.

Stereotype	Basisklasse	Tags
«PAstep»	Message	PAdemand
	Stimulus	PArespTime
	ActionState	PAprob
	SubactivityState	PArep
	Action	PAdelay
	ActionExecution	PAextOp
	Operation	PAinterval

Tabelle B.18.: Der Stereotyp *PAstep*

Tag	Type	Multipl.	Domain Attribute Name
PAdemand	PAperfValue	[0..*]	Step::hostExecutionDemand
PArespTime	PAperfValue	[0..1]	Step::responseTime
PAprob	Real	[0..1]	Step::probability
PArep	Integer	[0..1]	Step::repetition
PAdelay	PAperfValue	[0..1]	Step::delay
PAextOp	PAperfValue	[0..1]	Step::operations
PAinterval	PAperfValue	[0..1]	Step::Interval

Tabelle B.19.: Die Eigenschaftswerte von PAstep

B.6. Kapitel 9 des UML-RT: Model Processing

Unter *Model Processing* versteht man den Prozess, bei dem ein UML- Modell entweder durch eine Modell-Analyse-Technik untersucht oder durch Synthese-Methoden ergänzt wird. Ein allgemeines Konzept um Modelle zu analysieren zeigt Abbildung B.8.

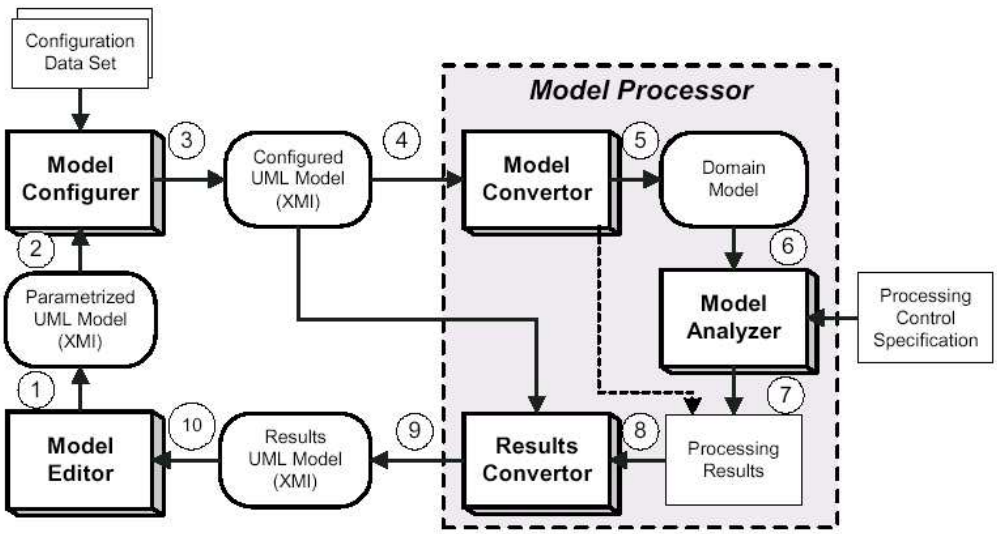

Abbildung B.8.: Allgemeines Modell Processing System

Die *Model Editor* Funktion wird dazu benutzt UML Modelle zu erzeugen und zu modi-

fizieren. Die erzeugten UML Modelle können auf die in den vorangegangenen Abschnitten beschriebene Art und Weise annotiert und parametrisiert werden. Die *Model Configurer* Funktion hat nun zur Aufgabe, das parametrisierte UML Modell zu konfigurieren, indem sie ein neues Modell bzw. Quelltext für die weitere Verarbeitung erzeugt.
Im *Model Processor* findet nun die eigentliche Analyse statt. Der *Model Convertor* kovertiert das erzeugte Modell in ein für die Analysesoftware benötigtes Format, wobei die Stereotypisierungen entsprechend im Analysewerkzeug abgebildet werden. Dabei werden auch Inkonsistenzen in den gemachten Annotationen entdeckt. Darauf folgt der *Model Analyzer*, welcher gekapselt für die Analyse selbst zuständig ist. Der *Result Convertor* wandelt nun die Analyseergebnisse so um, dass Sie wieder in einem UML-Modellierungswerkzeug zu sehen sind.

Literaturverzeichnis

[ieeesoftware 1993] *IEEE Standard for a Software Quality Metrics Methodology.* IEEE Standard 1061-1992. New York, 1993

[ANTLR] ANTLR: *http://www.antlr.org*

[Borland] BORLAND: *http://www.borland.com/together*

[Büttner u. a.] BÜTTNER, Martin ; FRICKE, Beate ; KLAASSEN, Othmar ; NOLTE, Siegfried ; SCZITTNICK, Michael ; STAHL, Harald ; WEISSENBERG, Norbert: *HI-SLANG Reference Manual for the Hierarchical Evaluation Tool HIT.* 3.6.000. Universität Dortmund, Informatik IV:

[Gentleware] GENTLEWARE: *http://www.gentleware.com*

[NanoXML] NANOXML: *http://nanoxml.sourceforge.net*

[OMG 2001] OMG: *OMG Unified Modeling Language Specification.* 1.4, September 2001. – http://www.omg.org/

[OMG 2003] OMG: *UML Profile for Schedulability, Performance, and Time Specification.* 1.0, September 2003

[PG438 2004] PG438: *Zwischenbericht der Projektgruppe 438: Entwicklung eines Performance Moduls zur Leistungsorientierten Software Entwicklung*, 2004

[Sczittnick u. a.] SCZITTNICK, Michael ; BÜTTNER, Martin ; HECK, Elke ; LANGE, Rainer ; STRÜWER, Michael ; VERWOHLT, Monika ; WYSOCKI, Christel: *HITGRAPHIC User's Guide.* 3.6.00. Universität Dortmund, Informatik IV:

Abbildungsverzeichnis

Tabellenverzeichnis