Software Engineering Methoden für die Bedienermodellierung in dynamischen Mensch-Maschine-Systemen

vorgelegt von
Dipl.-Inform. Sandro Leuchter
aus Wolfsburg

von der Fakultät für Verkehrs- und Maschinensysteme
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:
Vorsitzender: Prof. Dr.-Ing. M. Rötting
Gutachter: Prof. Dr.-Ing. L. Urbas
Gutachter: Prof. Dr. phil. M. Thüring

Tag der wissenschaftlichen Aussprache: 25. Februar 2009

Berlin 2009
D 83

Vorwort

Ich bedanke mich bei allen, die mich bei dieser Arbeit unterstützt haben. Mein besonderer Dank gilt meinem Doktorvater Herrn Prof. Urbas vom Institut für Automatisierungstechnik der Technischen Universität Dresden für die Betreuung und seinen Rat. Herrn Prof. Thüring vom Institut für Psychologie und Arbeitswissenschaft der Technischen Universität Berlin danke ich für die Übernahme des Korreferats und das der Arbeit entgegengebrachte Interesse.

Die interdisziplinäre Herangehensweise bei Bewertung und Gestaltung von komplexen sozio-technischen Systemen habe ich am Zentrum Mensch-Maschine-Systeme (ZMMS) der Technischen Universität Berlin kennengelernt. Für diese Möglichkeit danke ich besonders dem langjährigen Sprecher des ZMMS Herrn Prof. Timpe. Herrn Prof. Eyferth danke ich für die ersten Aufgabenstellungen in der kognitiven Modellierung und die Freiräume bei deren Bearbeitung. Herrn Prof. Furuta vom Cognitive Systems Engineering Lab der Universität Tokio danke ich für die freundliche Aufnahme an seinem Institut.

Die Fallstudien dieser Arbeit konnte ich in mehreren drittmittelgeförderten Projekten durchführen. Ich danke dafür den Zuwendungsgebern DAAD, DFG, JSPS und VolkswagenStiftung sowie allen Kollegen und Vorgesetzten für praktische Unterstützung und fachlichen Rat. Besonders danke ich Thomas Bierwagen, Carmen Bruder, Jeronimo Dzaack, Yorck Hauß, Marcus Heinath, Thomas Jürgensohn, Jürgen Kiefer, Martin Christof Kindsmüller, Ljudmila Nekrasova, Cornelia Niessen, Nele Pape, Dirk Schulze-Kissing, Katharina Seifert, Thilo Siegle und Oliver Späth.

Ich danke meiner Familie und meinen Freunden, speziell meiner Frau Mona und meinem Sohn Felix, die durch die geduldige Unterstützung und Verzicht diese Arbeit ermöglicht haben.

Karlsruhe, im August 2009

Kurzfassung

In der vorliegenden Arbeit werden softwaretechnische Methoden zur Bedienermodellierung in dynamischen Mensch-Maschine-Systemen untersucht und zu einem anwendungsorientierten Einsatz weiterentwickelt. Ziel ist die praktische Verwendung der Methode Bedienermodellierung zur Unterstützung der Analyse und Gestaltung von Mensch-Maschine-Interaktion im Entwicklungsprozess. Die Annahme ist, dass durch die Simulation von Benutzungstests mit Bedienermodellen anstatt der Durchführung von realen Versuchen mit Probanden *Human Factors*-bezogene Gestaltungsprobleme bei geringeren Kosten und früher im Entwicklungsprozess als bei herkömmlichen Benutzertests aufgedeckt werden können. Durch softwaretechnische Methoden sollte im Rahmen dieser Arbeit die praktische Anwendbarkeit dieser Methode verbessert werden.

Auf der Basis des Standes der Forschung in Bezug auf die Bedienermodellierung, bisher modellierte Phänomene, Anwendungsfelder und ihre Anforderungen wurden ACT-R/PM als kognitive Architektur und NGOMSL zur Aufgabenmodellierung in diesem Bereich ausgewählt. Anhand von Fallstudien, die so gewählt wurden, dass sie einen breiten Bereich potenzieller Anforderungen an die Bedienermodellierung in dynamischen Mensch-Maschine-Systemen abdecken, wurden die vorhandenen Lücken der bestehenden Theorien und Werkzeuge identifiziert und gleichzeitig neue Werkzeuge und Erweiterungen der Theorien entwickelt und im Einsatz getestet. Die folgenden Modellierungsfallstudien wurden im Rahmen dieser Arbeit durchgeführt:

- MoFL: Modellierung der kognitiven Leistungen von erfahrenen Fluglotsen in der Streckenflugkontrolle – Modellierungsschwerpunkt liegt auf Informationsaufnahme und Repräsentation der Verkehrssituation.
- DURESS: Modell der Aufgabenbearbeitung in der Mikrowelt DURESS (DUal REservoir System Simulation), einer prototypischen Prozesssimulation – Schwerpunkt ist die detaillierte Modellierung von Wahrnehmung der Informationen in der Aufgabensimulation.
- Rektifikationskolonne: Modell des Anfahrens und Regelns einer verfahrenstechnischen Anlage anhand einer Mikrowelt – Schwerpunkt ist Zeitwahrnehmung für die korrekte Aufgabenbewältigung.

Es hat sich gezeigt, dass Neuentwicklungen in Form von Frameworks und Architekturerweiterungen in folgenden Bereichen notwendig waren: Kontrollstrukturen für Multitasking und SOPs, Zeitwahrnehmung und Interaktion mit externen Aufgabenumgebungen. Weiterhin konnte die Bedienermodellierung durch die Entwicklung und Bereitstellung von Werkzeugen zur Unterstützung der konzeptuellen Modellbildung, der Formalisierung, der Modellprogrammerstellung und der Durchführung von Simulationsexperimenten unterstützt werden.

Ergebnis dieser Arbeit sind somit in den Fallstudien erprobte neue Werkzeuge und Erweiterungen von Bedienermodellierungsmethoden auf den Analyseebenen Kognition und Aufgabe in den folgenden Bereichen: Guideline (Modellierungsrationale ACT-R/PM und NGOMSL), Modellierungswerkzeug (Visualisierung, Agimap, ACT-R Shell, MT-GOMS), Framework (ACT-Com, Agimap, Planex) und Erweiterung (Timer, MT-GOMS). Alle diese neuen Entwicklungen haben Eigenschaften, die sie von anderweitig verfügbaren Arbeiten abheben und die die Bedienermodellierung in dynamischen Mensch-Maschine-Systemen effizienter und effektiver machen.

- Modellierungsrationale ACT-R/PM und NGOMSL: Auswahl der Modellierungsumgebungen und *best practice* der Bedienermodellierung werden beschrieben.
- Visualisierung: Werkzeug zur grafischen Repräsentation wesentlicher ACT-R-Modellelemente in Form einer grafischen Sprache und eines grafischen Editors.
- Agimap: Framework zur Modellierung von Wahrnehmung und Handlung mit ACT-R/PM und integrierte Entwicklungsumgebung zur Kopplung von ACT-R/PM an externe Aufgabenumgebungen über eine Abstraktionsschicht.
- ACT-R Shell: Anwendungsspezifische integrierte Entwicklungsumgebung für die Arbeit mit dem ACT-R-Modell MoFL.
- MT-GOMS: Weiterentwicklung von GOMS zu dem Zweck, die Bedienbarkeit und das Ablenkungspotenzial von *In-Vehicle Information Systems* (IVIS) effizient zu bewerten.
- ACT-Com: Middleware, die die Kopplung und Interaktion zwischen Aufgabensimulation und kognitivem Modell realisiert.
- Planex: Framework zur Verarbeitung von KLM-GOMS-artigen Aufgabenbeschreibungen in ACT-R.
- Timer: Erweiterung von ACT-R, mit der die subjektive Zeitwahrnehmung unter Berücksichtigung der Beanspruchung modelliert werden kann.

Mit ACT-R/PM, KLM-GOMS und NGOMSL wurden nützliche Architekturen bzw. Formalismen für die Bedienermodellierung identifiziert und durch neue Erweiterungen für Fragestellungen der Mensch-Maschine-Interaktion zugänglich gemacht. Durch neue Werkzeuge ist der praktische Einsatz effizienter geworden. Eine zukünftig stärkere Anwendungsorientierung beim Einsatz von kognitiven Architekturen ist abzusehen, da sie zunehmend Verbreitung außerhalb der erkenntnisorientierten Kognitionswissenschaft finden. Durch die Entwicklungen in dieser Arbeit können die Kosten eines Einsatzes reduziert werden.

Abstract

The topic of this thesis is the development of software engineering methods for user-modeling in dynamic human-machine systems and how to apply them in cognitive systems engineering. The aim of this is the practical application of user modeling for supporting analysis and design of human-machine interaction in development processes. The assumption for this approach is that through simulation tests with user models instead of conducting experiments with real subjects, human factors-related design problems could be discovered at lower cost and earlier in development. The practical applicability of this method was to be improved through new software-engineering methods.

Based on a survey of previous related work on human performance modeling in safety critical systems ACT-R/PM and NGOMSL were selected as the basis for further work. Gaps in these existing theories and tools were identified with case studies. New tools and enhancements to the theories were developed and tested also within the case studies. The following case studies were carried out in the context of this work:

- MoFL: Model of the cognitive performance of experienced air traffic controllers in en-route control – the model focuses on information acquisition and representation of the traffic situation.
- DURESS: Model of the human performance in the micro world DURESS (Dual Reservoir System Simulation), a prototypical process simulation - focus is the detailed modeling of perception of information in the external task environment.
- Rectifying Column: Model of the procedural rules of starting up the rectifying process in a micro world. Focus is on perception and estimation of time durations.

Work on these case studies showed that new developments in the form of frameworks and architectural enhancements were necessary in the following areas: control structures for multitasking and standard operating procedures, time perception and interaction with external task environments. It was also necessary to develop new tools for supporting conceptual modeling, formalization, simulation program creation and conducting of simulation experiments.

Results are case study tested new tools and enhancements of user modeling methods in the following areas: guideline (modeling ACT-R/PM rational and NGOMSL), modeling tool (visualization, Agimap, ACT-R Shell, MT-GOMS), frame-work (ACT-Com, Agimap, Planex) and extension (Timer, MT-GOMS). All these new developments are unique and make user modeling in dynamic human-machine systems more efficient and effective.

- Modeling Rationale ACT-R/PM and NGOMSL: Selection of modeling environment and best practices of user modeling are described.

- Visualization: Tool for the graphical representation of essential ACT-R model elements in the form of a graphical language and a graphical editor.
- Agimap: Framework for the modeling of perception and action with ACT-R/PM and an integrated development environment (IDE) as well as an abstraction layer for connecting cognitive models with external task environments via ACT-R/PM.
- ACT-R shell: Application-specific integrated development environment (IDE) for working with the ACT-R model MoFL.
- MT-GOMS: Enhancement of GOMS for handling multi-tasking between car driving and using in-vehicle information system (IVIS) to analyze potential distraction.
- ACT-Com: Middleware for the implementation of interaction between external task environment and cognitive simulation model.
- Planex: Framework for using KLM-GOMS-like procedure descriptions in ACT-R.
- Timer: Extension of ACT-R for subjective perception of time which also takes workload into account.

ACT-R/PM, KLM-GOMS and NGOMSL are feasable architectures and formalisms that were extended for the practical application in user modeling for dynamic human-machine systems. Their use is now more efficient because of the newly developed tools. Cognitive architectures are not only used in fundamental cognitive science but also more and more in application oriented research. The results of this work will help reducing the cost of their application.

.

Inhaltsverzeichnis

1. Einführung

1.1. Motivation und Zielsetzung der Arbeit

Es gibt Fehlersituationen in sicherheitskritischen Mensch-Maschine-Systemen, die auf eine falsche Situationswahrnehmung, -interpretation, Entscheidungen und damit eine falsche Bedienung des technischen Systems durch den Operateur zurückgeführt werden können. Diese Fehler können aus organisationalen Problemen (z.B. Personalauswahl und -entwicklung) oder einer unangemessenen Gestaltung der Arbeitsaufgabe oder der Benutzungsschnittstelle resultieren. In der Entwicklung von sicherheitskritischen Mensch-Maschine-Systemen ist der Nachweis der Beherrschbarkeit durch den Bediener bzw. die Verlässlichkeit des Systems (Giesa 2003) deshalb eine kritische Anforderung. Benutzertests sind jedoch aufwändig durchzuführen, da für statistisch fundierte Aussagen eine Anzahl von Versuchsdurchläufen gemacht und ausgewertet werden müssen. Insbesondere in verteilten Mensch-Maschine-Systemen, wo es auf die Interaktion mehrerer Bediener untereinander und mit technischen Systemkomponenten ankommt und auch das Kommunikationsverhalten vernetzter Systemteile berücksichtigt werden muss, ist der praktische Aufwand der Durchführung erheblich. Zudem müssen bereits valide Prototypen bzw. Funktionsmodelle vorliegen, an denen Benutzungstests durchgeführt werden können.

Deshalb werden im Entwurf Modelle menschlicher Eigenschaften in Simulationen benutzt. Beispielsweise werden in der Fahrzeugindustrie regelmäßig anthropometrische Menschmodelle eingesetzt, mit denen die Bedienbarkeit in Bezug auf Sichtbarkeit, Erreichbarkeit und einwirkende Kräfte simuliert werden, um den Gestaltungsprozess der Fahrgastzelle zu verbessern. In modernen komplexen Mensch-Maschine-Systemen stehen jedoch die intellektuellen Fähigkeiten der Bediener mehr im Vordergrund als die physischen. Deshalb müssen kognitive Modelle benutzt werden, um die Beherrschbarkeit solcher Systeme simulativ zu ermitteln. Diese Modelle sind in der Lage, mentale Vorgänge unter Berücksichtigung von beschränkenden Faktoren wie Zeitdruck oder begrenztem Gedächtnis zu simulieren. Für die Anwendung solcher Modelle zur Prüfung der Bedienbarkeit technischer Systeme wurde ein Ansatz vorgeschlagen, bei dem das kognitive Modell den menschlichen Bediener in einem Mensch-Maschine-System ersetzt (Bass et al. 1995, Ritter et al. 2001, Ritter et al. 2003). Damit sind simulierte Benutzertests mit geringeren Kosten, häufiger und auch früher im Entwicklungsprozess durchführbar. Daneben können diese Modelle auch als Komponente für Unterstützungssysteme, z.B. in Trainingssystemen und als Unterstützungskomponente in automatisierten Systemen angewendet werden.

Die existierenden Ansätze zur Entwicklung solcher kognitiven Modelle sind für einen anwendungsorientierten Einsatz noch nicht ausreichend. Zum einen fehlen Werkzeuge und Frameworks für ein effektives und effizientes Engineering, zum

anderen sind die existierenden Modellierungsformalismen nicht in der Lage, mit der Dynamik in solchen Mensch-Maschine-Systemen adäquat umzugehen. Probleme mit der Dynamik zeigen sich in unterschiedlichen Bereichen wie Interaktion zwischen Modell und technischem System, Multitaskingaufgabenumgebungen und Diagnose von Timing in technischen Prozessen.

Ziel dieser Arbeit ist, Software Engineering Methoden zu entwickeln, mit denen die kognitive Modellierung für den umrissenen Anwendungszweck verbessert werden kann. Das schließt sowohl die kognitionswissenschaftlichen Frameworks ein, die zur Informationsverarbeitung in dynamischen Mensch-Maschine-Systemen erweitert werden müssen, als auch die Implementierung neuer Entwicklungswerkzeuge, mit denen die Modelle effizient erstellt werden können. Die folgenden Forschungsfragen für diese Arbeit ergeben sich aus der Zielsetzung:

1) Für welche Aufgabensituation (Anwendungsdomäne) ist die Bedienermodellierung anwendbar?
2) Welche Erweiterungen sind an den kognitionswissenschaftlichen Frameworks erforderlich, um mit dynamischen Mensch-Maschine-Systemen umgehen zu können?
3) Mit welchen Entwicklungswerkzeugen kann die Bedienermodellierung unterstützt werden?

Um die Lücken in existierenden Systemen zu identifizieren, Prototypen für Werkzeuge und Erweiterungen zu entwickeln und im Einsatz zu testen, wurden Fallstudien durchgeführt, die möglichst repräsentativ den Möglichkeitsraum sicherheitskritischer dynamischer Mensch-Maschine-Systeme abdecken.

1.2. Aufbau der Arbeit

In Kapitel 2 werden Begrifflichkeiten und Konzepte, die in dieser Arbeit benutzt werden, definiert und voneinander abgegrenzt. Insbesondere werden Mensch-Maschine-Systeme, Modellierung und Simulation, Modelle des menschlichen Verhaltens in Mensch-Maschine-Systemen und Software Engineering erläutert. Dabei zeigt sich, dass für die weitere Arbeit die Unterscheidung in die zwei Analyseebenen Aufgabe und Kognition einen geeigneten Rahmen darstellt. Die Trennung der Ebenen ergibt sich daraus, dass entsprechende Methoden und Werkzeuge in zwei unterschiedlichen *Scientific Communities* entstanden bzw. aufgegriffen und weiterentwickelt wurden. Die Aufgabenanalyse hat eine starke Verankerung im Bereich Mensch-Computer Interaktion, die Analyseebene Kognition ist Gegenstand der Kognitionswissenschaft. Im Rest der Arbeit wird zwischen diesen beiden Ebenen unterschieden.

In Kapitel 3 wird der Stand der Forschung auf dem Gebiet der Modellierung und Simulation von Mensch-Maschine-Interaktion vorgestellt und bewertet. Auf der Ebene der Kognitionsanalyse werden Architekturen und Werkzeuge zur kognitiven

Simulation aus der Kognitionswissenschaft vorgestellt und bewertet. Da die Kopplung von Modell und Aufgabenumgebung sowie die Simulation der Wahrnehmung in der Aufgabenumgebung besondere Anforderungen aus der Anwendung in dynamischen Mensch-Maschine-Systemen sind, werden sie getrennt erörtert. In Abschnitt 3.3 wird dann der Stand der Wissenschaft bei der Modellierung und Simulation auf der Ebene Aufgabenanalyse vorgestellt und bewertet. Hier werden Ansätze aus der *Human Computer Interaktion*, und der *Human Factors* Forschung erörtert.

Für beide Analyseebenen werden die in der Literatur berichteten modellierten Phänomene, ihre Anwendungsfelder und die daraus resultierenden Anforderungen präsentiert. In Abschnitt 3.4 wird dann die aktuell diskutierte Synthese der Analyseebenen „Kognition“ und „Aufgabe“ anhand der Compilationsansätze und hybrider Modellierungsansätze vorgestellt. Aus der Darstellung des Standes der Forschung wird der weitere Handlungsbedarf für diese Arbeit abgeleitet.

In Kapitel 4 wird die Forschungsmethode der Fallstudie eingeführt und ihre Anwendung in dieser Arbeit begründet. Die Auswahl der Fallstudien wird anhand eines neu entwickelten Beschreibungsmodells für Aufgabenumgebungen und der Einordnung möglicher Anwendungsbereiche diskutiert. In Kapitel 5 werden dann die durchgeführten Fallstudien vorgestellt.

Die neu entwickelten Werkzeuge und Erweiterungen werden in Kapitel 6 dargestellt und ihre Eignung für den Anwendungszweck anhand des Einsatzes in den Fallstudien diskutiert. Die Arbeit endet in Kapitel 7 mit einer Zusammenfassung, Bewertung und einem Ausblick. In den Anhängen werden implementierungstechnische Details zur Umsetzung einiger Werkzeuge und Erweiterungen sowie Repräsentationsformate und Kommunikationsprotokolle dokumentiert.

2. Gegenstandsbestimmung

2.1. Mensch-Maschine-Systeme

Technische Systeme, die komplex in ihrer Struktur und dynamisch in ihrem Verhalten sind, sind oft schwierig zu steuern. Viele dieser Systeme sind zudem sicherheitskritisch. Fehler in solchen technischen Systemen haben also potenziell schwere Auswirkungen auf Gesundheit und Leben von Menschen. Beispiele für sicherheitskritische Systeme sind medizintechnische Maschinen, verfahrenstechnische Anlagen wie Kraftwerke oder chemische Fabriken und Verkehrstechnik.

Neben Vorkehrungen zur Sicherung der technischen Funktion (redundante Auslegung kritischer Komponenten, Verifikation des technischen Entwurfes und seiner Implementierung, spezielle qualitätssichernde Entwicklungsprozesse) müssen diese besonderen Risiken bei der Gestaltung der Arbeit berücksichtigt werden. Das betrifft die Benutzungsschnittstelle (Anordnung, Form und Farbe der Bedien- und Anzeigeelemente), die organisatorische Einbettung der Tätigkeit (Personalauswahl, Training, Produktions- und Effizienzvorgaben) und die Aufgabengestaltung (Grad der Automatisierung, Unterstützungssysteme, Aufgabenverteilung in einem Team, Haupt- und Nebenaufgaben).

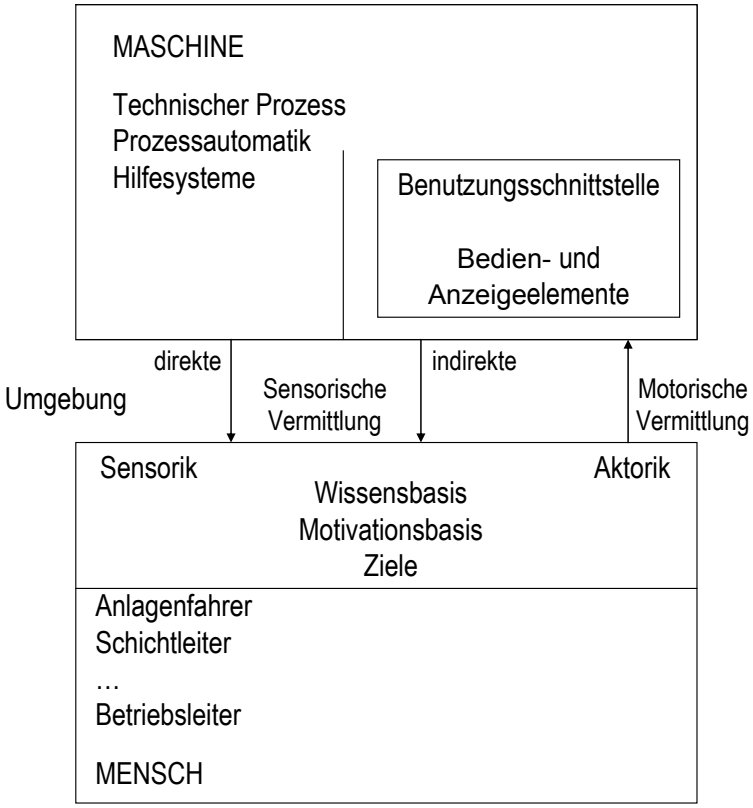


Abbildung 2-1: Struktur eines Mensch-Maschine-Systems (nach Timpe & Kolrep 2000)

Zur Betrachtung von Mensch-Maschine-Systemen in Analyse und Gestaltung wird das Strukturmodell (nach Timpe & Kolrep 2000) aus Abbildung 2-1 herangezogen. Demnach handelt es sich um ein rückgekoppeltes System, bei dem Eingriffe des

Menschen den Zustand der Maschine und damit wiederum deren Rückmeldungen an den Menschen beeinflussen. Die Interaktion geschieht sowohl direkt über die Benutzungsschnittstelle mit Bedien- und Anzeigeelementen als auch indirekt über wahrnehmbare Änderungen an nicht explizit als Anzeigeelemente geplanten Teilen des technischen Systems. Der Mensch nimmt den Zustand der Maschine über diese Kanäle wahr und greift über seine Motorik an den Bedienelementen ein. Der Zweck des Eingriffs ist die Regelung (Vorgabe einer Wirkgröße in Abhängigkeit von einem Soll-Ist-Vergleich) oder die Steuerung der Maschine (feste Vorgabe einer Wirkgröße ohne Rückkopplung).

Die Handlungen des Bedieners sind in diesem Rahmenwerk Reaktionen auf den wahrnehmbaren Zustand der Maschine und die individuellen Motive und Ziele des Bedieners. Die Ziele resultieren je nach Granularität der Betrachtung von unterschiedlichen Trägern: Anlagenfahrer, Schichtleiter, Betriebsleiter. Durch die *Kognition* des Bedieners werden Wahrnehmungen, Wissen, Motive und Ziele zu Handlungen umgesetzt. Kognition umfasst dabei alle mentalen Vorgänge, also insbesondere Wahrnehmen, Entscheiden, Problemlösen, Bewerten, Antizipieren und Lernen (Hauß & Timpe 2000). Diese kognitiven Vorgänge werden als Prozesse innerhalb der *kognitiven Architektur* des Bedieners dargestellt. Die kognitive Architektur beinhaltet die Strukturen, deren Zusammenhänge, automatische Vorgänge und Performanzeigenschaften, die für mentale Vorgänge erforderlich sind.

In einem erweiterten Modell des Mensch-Maschine-Systems werden darüber hinaus *verteilte* Mensch-Maschine-Systeme betrachtet. Es geht hierbei um mehrere Benutzer des technischen Systems, die ggf. auch disloziert sind. Entsprechend wird die Betrachtungsebene der Maschine so gewählt, dass mehrere Benutzungsschnittstellen vorhanden sind (Timpe & Jürgensohn 2000).

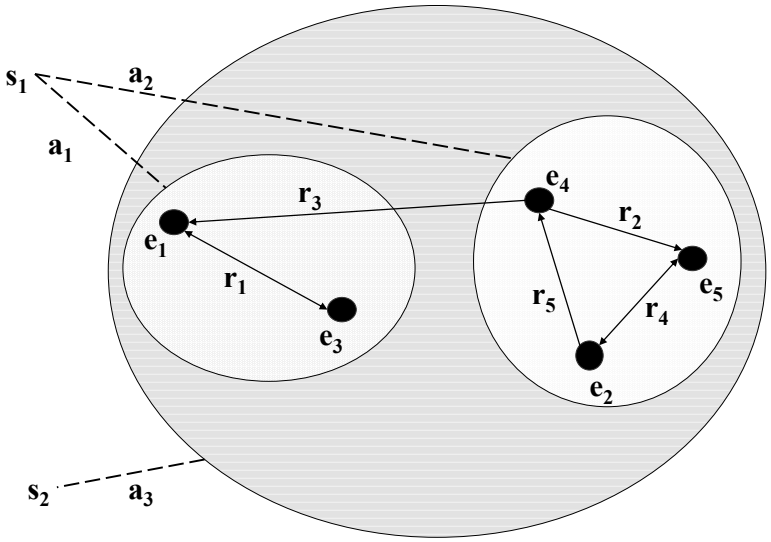
Im Rahmen dieser Arbeit wird die Mikroebene Bediener – technisches System, abgegrenzt von der Makroebene (Regierung, Rechtsprechung) und der Mesoebene (Organisation), betrachtet. Es geht ausschließlich um die direkte sensorische computervermittelte Interaktion, d.h. dass die Benutzungsschnittstelle in Form eines informationsverarbeitenden Systems realisiert ist und indirekte Wahrnehmung ausgeklammert wird. Die Kognition des Bedieners wird modelliert und simuliert. Dazu wird eine kognitive Architektur mit Langzeitgedächtnis, Arbeitsgedächtnis und Wahrnehmungskanälen als wesentlichen Strukturen angenommen. Kognitive Prozesse werden darin als Produktionssystem mit Hilfe einer hierarchischen Aufgabenanalyse modelliert. Motivation und Emotion sind nicht Gegenstand des Modells von Kognition in dieser Arbeit.

2.2. Mensch-Maschine-Systemtechnik

Durch die *Mensch-Maschine-Systemtechnik* wird versucht, Sicherheit, Effektivität und Effizienz von Mensch-Maschine-Systemen zu steigern, indem spezielle Methoden für die Analyse, Bewertung und Gestaltung des Gesamtsystems Bedie-

ner / Schnittstelle / Technik verwendet werden. Dafür wird der Systembegriff herangezogen (s. z.B. Pahl & Beitz 1993, S. 24). Ein *System Sys* besteht aus einer Menge von Elementen *E* und Relationen $R \subseteq E \times E$ zwischen den Elementen. Die Abgrenzung ist abhängig vom Betrachter *s* und einem zweckmäßigem Auswahlaspekt *a*: *Sys* = (*s*, *a*, *E*, *R*). Im Beispiel in Abbildung 2-2 ist ein Systemzusammenhang dargestellt, der aus den Elementen $E = \{e_1, e_2, e_3, e_4, e_5\}$ besteht. Zwei Betrachter s_1 und s_2 nehmen die Systemabgrenzung unterschiedlich vor: s_1 strukturiert unter den Auswahlaspekten a_1 und a_2 in zwei Subsysteme Sys_1 und Sys_1. s_2 grenzt nur ein System Sys_3 aufgrund des Auswahlaspektes a_3 ab. In der Mensch-Maschine-Systemtechnik werden Systemgrenzen regelmäßig so gezogen, dass Benutzer und Maschine als eine Einheit in einem Gesamtsystem betrachtet werden können.

Die Mensch-Maschine-Systemtechnik ist in ihren Methoden interdisziplinär. Humanwissenschaftliche Methoden, insbesondere aus Psychologie und Ergonomie werden angewendet, um die Gestaltung der Benutzungsschnittstelle und der Aufgaben an die Fähigkeiten der Bediener anzupassen. Ingenieurwissenschaftliche Methoden beziehen sich auf Entwicklungsprozesse, die Strukturierung der technischen Funktion und Unterstützungssysteme.



$Sys_1 = (s_1, a_1, E_1, R_1)$
$E_1 = \{ e_1, e_3\}$
$R_1 = \{ (e_1, e_3), (e_3, e_1) \}$

$Sys_2 = (s_1, a_2, E_2, R_2)$
$E_2 = \{ e_2, e_4, e_5\}$
$R_2 = \{ (e_2, e_4), (e_4, e_5), (e_2, e_5), (e_5, e_2) \}$

$Sys_3 = (s_2, a_3, E_3, R_3)$
$E_3 = E_1 \cup E_2$
$R_3 = R_1 \cup R_2 \{ (e_4, e_1) \}$

Abbildung 2-2: Systemformalisierungen in Sys_1/Sys_2 (gepunktet) einerseits und Sys_3 andererseits (gestreift)

2.3. Modellierung und Simulation

In dieser Arbeit werden Modellierung und Simulation in das Zentrum der Betrachtung systemtechnischer Methoden gestellt. Deshalb werden diese Begriffe in dem folgenden Abschnitt definiert und abgegrenzt. *Modelle* sind abstrakte Repräsentationen eines Teils der Realität, bei denen Unwesentliches in Bezug auf den Modellzweck weggelassen wird. Modelle stehen deshalb in Bezug zu ihrem Original und sind für einen bestimmten Anwendungszweck konzipiert. Der Bezug zwischen Modell, Original und Zweck erschließt sich aus der Interpretation des

Nutzers eines Modells. Die Interpretation ist demnach wesentlicher Bestandteil der Modellnutzung. Sie muss also durch Dokumentation oder Konventionen unterstützt werden. Durch die Verkürzung von Modellen in Bezug auf den Modellzweck können nur diese relevanten Aspekte am Modell untersucht werden. Es gibt unterschiedliche Formen von Modellen. So können reale Prozesse durch mathematische oder physikalische Modelle, die eine einfachere oder ungefährlichere Untersuchung als das Original erlauben, abgebildet werden. Modelle können qualitativ oder quantitativ ausgeprägt sein. Qualitative Modelle beinhalten Aussagen über Strukturen und Wirkzusammenhänge innerhalb des abgebildeten Originals. Bei quantitativen Modellen sind die modellierten Wirkzusammenhänge durch berechenbare Funktionen[1] ausgedrückt. Entsprechend den konkreten Berechnungsvorschriften werden relevante Eigenschaften Strukturen und andere Einflussgrößen durch numerische Parameter ausgedrückt.

Die Berechnung der Wirkzusammenhänge eines quantitativen Modells mit dem Ziel, zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind, heißt *Simulation* des modellierten Systems (VDI 3633). Im weiteren Sinne wird unter Simulation das Vorbereiten, Durchführen und Auswerten gezielter Experimente mit einem Simulationsmodell verstanden. Ein üblicher Simulationsaufbau besteht aus zwei aufeinander aufbauenden Komponenten: Ein *Simulator* enthält wiederkehrende Bestandteile einer Bandbreite quantitativer Modelle eines Gegenstandsbereiches. Ein Modell, das zusammen mit dem Simulator benutzt wird, braucht dann nur noch die für den konkreten Erkenntniszweck spezifischen Ausprägungen der zu simulierenden Situation enthalten. Der Simulator hat die Funktion eines Interpreters zur Ausführung eines Modells. Beispiele für Simulator-/Modell-Kombinationen sind

- Flugsimulation: Flugsimulator (Aerodynamik, Flugeigenschaften eines bestimmten Flugzeugmodells) und vorgeplanter Flugverlauf mit speziellen Ereignissen,
- Simulation verfahrenstechnischer Prozesse: Matlab Library (Lösung von Differentialgleichungssystemen) und Modell einer konkreten Anlage (als Differentialgleichungssystem) und
- Simulation kognitiver Modelle: Kognitive Architektur (angenommene mentale Strukturen und automatische Prozesse) und kognitives Modell (Regeln für Strategien zur Lösung eines konkreten Problems).

---

[1] In der theoretischen Informatik ist der Begriff der berechenbaren Funktion definiert als Funktion $f : M \rightarrow$, genau dann wenn es einen Algorithmus gibt, der für jeden Eingabewert $m \in M$, für den $f(m)$ definiert ist, nach endlich vielen Schritten anhält und als Ergebnis $f(m)$ liefert. In allen Fällen, in denen $f(m)$ nicht definiert ist, bricht der Algorithmus nicht ab. Eine Funktion ist also berechenbar, wenn sie programmiert werden kann.

Von den Begriffen Modell und System müssen die Konzepte Framework und Theorie unterschieden werden (Anderson 1983, S. 12ff). Ein Framework ist demnach ein systematischer Vorrat von zueinander passenden Konstrukten (Elemente und Relationen), mit denen ein Modell zum Verständnis einer Domäne aufgebaut werden kann. Eine Theorie ist ein präzises System, das in seinem Anwendungszweck und seinem Auswahlaspekt genereller als ein Modell ist. Konkrete Modelle werden aus der Theorie abgeleitet. Eine Architektur ist ein System, das Elemente und Relationen enthält, die in vielen Modellen wiederkehren. Die Architektur leitet damit die Entwicklung von Modellen, indem Teile der Architektur wiederverwendet werden. Die Architektur gibt damit die Grundstruktur und das Vokabular der Modelle vor. In Bezug auf die Simulation hat eine Architektur damit sowohl die Funktion einer Theorie und eines Simulators.

2.4. Software Engineering

Das Fachgebiet Software Engineering wurde als Reaktion auf die „Softwarekrise“ begründet (Naur & Randell 1969). Zu jenem Zeitpunkt hatte die Komplexität der Softwareprodukte einen Grad erreicht, der mit den damals verfügbaren Mitteln nicht mehr bewältigt werden konnte. Große Projekte wurden nicht beendet oder deren Ergebnis war fehlerhaft. Die Entwicklung und Produktion von Software wird seitdem in dieser Disziplin nicht mehr als Kunsthandwerk, sondern als ingenieurwissenschaftliche Tätigkeit begriffen. Die Forschung befasst sich mit der Verbesserung der Handhabbarkeit von Software inhärenter Komplexität. Dabei wurde die Technik der Programmierung analysiert und durch die Einführung strukturierter Methoden verbessert (z.B. Wirth 1971).

Eine wesentliche Idee zur Beherrschung der Komplexität ist die Dekomposition von Softwareprogrammen in *Module* mit dem Ziel der Minimierung von Abhängigkeiten (Schnittstellen) zwischen Programmteilen von Parnas (1972a, 1972b). Parnas beabsichtigt mit der Aufteilung in Module, die Entwicklungszeit durch Parallelisierung der Entwicklung zu verkürzen. Zusätzlich wurde erkannt, dass dadurch auch die Flexibilität von Softwareprodukten in Bezug auf Änderbarkeit und die Verständlichkeit der Programme erhöht werden kann, wodurch Fehlerquellen verringert werden.

Um die Nutzung von Modulen über ihre *Schnittstelle* zu vereinfachen, wurden in einigen Programmiersprachen spezielle Konstrukte für die Schnittstellenbeschreibung eingeführt. So kennen Objective-C und Java Interface-Beschreibungen, mit denen die Syntax der Aufrufschnittstelle eines Moduls spezifiziert werden kann. Eiffel wiederum hat, wenngleich auch auf Methodenebene, das *design by contract*-Prinzip eingeführt (Meyer 1992). Es erfordert eine formale Spezifikation von Vorbedingung und Nachbedingung für die Schnittstellennutzung. Dadurch wird über die Syntax hinaus auch die Semantik des Schnittstellenaufrufes genau beschrieben.

Das Prinzip der Datenkapselung (*information hiding*: Parnas 1972b), das auf der Basis des Modulbegriffs eingeführt wurde, und die daraus abgeleitete Methode des *design by contract* (Meyer 1992) zielen auf eine erhöhte *Wiederverwendbarkeit* und erleichterte Konstruierbarkeit von Softwaremodulen und dienen letztlich der Vermeidung von Fehlern bei der Nutzung fremder Module.

Beim Softwareentwurf mit Modulen hat sich gezeigt, dass das Mittel der *Objektorientierung* mit den Mechanismen zu Vererbung und Komposition eine nützliche Erweiterung des Modulkonzeptes zur Steigerung der Abstraktion darstellt, mit der Systemanalyse und Entwurf verbessert werden können (Booch 1991). Auch die Objektorientierung hat zum Ziel, Wiederverwendbarkeit von Entwicklungsartefakten (z.B. Entwurfsdokumente, Programmcode) zu erhöhen.

Die oben beschriebenen Verfahren zur Modularisierung basieren alle auf dem Prinzip der *Datenkapselung*. Seit Mitte der neunziger Jahre wird daneben auch die funktionale Zerlegung betrachtet. Diese neuartige Form der Softwareentwicklung beruht auf der Zerlegung und Komposition von *Komponenten* auf Ausführungsplattformen. Komponenten sind Module, die durch Konfiguration in einer Vielzahl von Anwendungsszenarien eingesetzt werden können. Sie werden in einer Plattform zu einer kompletten Anwendung zusammengesetzt. Durch Schnittstellen, die besonders auf die Komposition von großen Systemen ausgelegt sind, lassen sich solche Module durch Kombination und Konfiguration von Komponenten zusammensetzen. Aktuelle Forschung auf diesem Gebiet zielt darauf ab, Abhängigkeiten zwischen Komponenten zu modellieren und während der Konfiguration einer Anwendung oder zur Laufzeit zu überprüfen, um Inkompatibilitäten und Versionskonflikte zu erkennen (z.B. OSGi).

Es ist zu beobachten, dass sich die grundlegende Organisation von Softwaresystemen oft wiederholt. Eine mehrfach erfolgreich verwendete Organisation ist eine *Software-Architektur*, die den Entwurf neuer Softwaresysteme leiten sollte (Hasselbring 2006). Die Architektur lässt sich durch die Beschreibung der verwendeten Komponenten sowie deren Beziehungen zueinander und zur Umgebung beschreiben. Ebenso sind die Prinzipien, die den Entwurf und die Evolution des Systems bestimmen, Teil der Architektur.

Ein *Software-Framework* ist die Plattform aus untereinander abhängigen Softwareeinheiten (z.B. Module, Variablen, Funktionen, Klassen, Objekte), die an konkrete Anforderungen des zu entwickelnden Softwareprodukts angepasst werden. Zur Anpassung werden Schnittstellen in Form von sog. Erweiterungspunkten in Frameworks vorgesehen (Pree 1997). Beispiele für Erweiterungspunkte sind Call-Back-Funktionen, die anwendungsspezifisch ausprogrammiert werden müssen. Die Funktionen, die mit den Erweiterungspunkten („Hooks“) verbunden sind, werden dann im Betrieb vom Framework bei spezifizierten Ereignissen aufgerufen.

Das Software Engineering beschäftigt sich zudem auch mit Werkzeugen, die die Softwareentwicklung und -produktion unterstützen. So sind integrierte Entwicklungsumgebungen (IDE), neuartige Compiler, Zwischen- und Repräsentationssprachen von Interesse. Insbesondere die Visualisierung von Software z.B. mit der UML (*unified modeling language*: Jacobsen et al. 1998) hat einen wesentlichen Beitrag zur Verstehbarkeit und Kommunizierbarkeit von Software geleistet.

Ein weiteres Prinzip, das im Software Engineering angewendet wird, ist die Identifikation und Dokumentation von Design Patterns (Gamma et al. 1995). Das sind abstrakte Konstellationen von Software-Entitäten (Objekte, Komponenten etc.) für vorgegebene Problemklassen, die aus der Analyse erfolgreicher Software-Architekturen isoliert wurden. Als wiederverwendbare Lösungen für wiederkehrende Problemklassen mit nachgewiesener Qualität stellen sie inzwischen einen allgemein anerkannten Kanon guter Programmiertechniken dar.

Weitere Themen des Software Engineering beziehen sich auf das Management von Software-Entwicklung. Hier wird insbesondere für Entwicklungsprozesse erforscht, wie mit hoher Verlässlichkeit und planbar eine hohe Produktqualität erreicht werden kann. Ebenso sind die Reife von Software entwickelnden Organisationen und die Messung der Qualität von Software Artefakten wichtige Zielstellungen. Schließlich sind Erhebung und Formulierung von Anforderungen an zu entwickelnde Softwaresysteme, die Unterstützung der Abbildung von Anforderungen auf Funktionen und Architekturelemente und deren geplante Änderungen während der Produktlebenszeit im Fokus des Software Engineering. Dabei geht es z.B. um die Nachverfolgung der Auswirkungen von Änderungen von Anforderungen auf Änderungen an Software-Artefakten

In dieser Arbeit werden aus Sicht des Software Engineerings Software-Architekturen für Benutzungsschnittstellen und Software-Frameworks von Simulationssystemen dargestellt. Die Erweiterungen an existierenden Modellierungs- und Simulationssystemen erfolgen mit dem Ziel, die Wiederverwendbarkeit, Flexibilität und Verständlichkeit zu fördern, um eine effizientere Arbeit zu ermöglichen und Fehler zu minimieren. Dabei werden Zielsetzungen und Lösungsprinzipien wie die Identifikation von Design Patterns aus dem Software Engineering übernommen und an die Erfordernisse der Benutzermodellierung angepasst. Zur Unterstützung der Entwicklung wurden zudem neue Werkzeuge entworfen und im praktischen Einsatz getestet.

2.5. Modelle menschlichen Verhaltens in Mensch-Maschine-Systemen

Die Simulation des Verhaltens technischer Systeme ist nicht Gegenstand dieser Arbeit. Es geht hingegen um die quantitative Modellierung von Nutzereigenschaften, Benutzungsschnittstelle und ihrer Bedienung, um damit die Mensch-Maschine-Interaktion zu simulieren. Die Simulation dieser Interaktion wird dann im

Entwicklungsprozess von Mensch-Maschine-Systemen angewendet (s. Urbas et al. 2005). Die dafür verwendeten Modelle decken jeweils nur einen Teilaspekt menschlichen Verhaltens bei der Interaktion mit technischen Systemen ab. Man kann daher die folgenden Klassen von Modellen für Nutzereigenschaften bzw. Mensch-Maschine-Interaktion abgrenzen:

- Anthropometrische Modelle: Bei diesen Modellen wird die physiologische Beschaffenheit der Bediener berücksichtigt. Insbesondere werden durchschnittliche Längen, Winkel, Kräfte und Beschleunigungen, denen der Mensch bei der Bedienung ausgesetzt wird, simuliert (Schmidtke 1976).
- Regler-Mensch-Modelle: Der Modellierungsansatz beruht auf der mathematischen Repräsentation von Regelungsverhalten in hochdynamischen Mensch-Maschine-Systemen. Dazu werden Differentialgleichungen zur Darstellung der Invarianz von zeitlicher Veränderung benutzt. Kognitive Verhaltenskomponenten, die das menschliche Regelungsverhalten beeinflussen, werden durch Störgrößen angenähert (Jürgensohn 2000).
- Aufgabenmodelle: Hier wird die Arbeitsaufgabe und ihre Abarbeitung in Form einer hierarchischen Spezifikation von erforderlichen Aktivitäten auf kognitiver, motorischer und der Wahrnehmungsebene und deren Bedingungen formalisiert. Die konkrete Form der Benutzungsschnittstelle wird bei der Formulierung der Aufgabenabarbeitung berücksichtigt.
- Konzeptuelle Modelle: Diese Klasse von Modellen beschreibt *Human Factors* in Mensch-Maschine-Systemen mit Hilfe von Aufgabenmodellen auf der Basis qualitativer Modelle menschlichen Verhaltens (z.B. fertigkeits-, regel- und wissensbasiertes Verhalten nach Rasmussen 1983) und anhand von Fehlerwahrscheinlichkeiten für bestimmte Handlungen (z.B. THERP – Technique for Human Error Rate Prediction, Swain & Guttmann 1983).
- Kognitive Modelle: Diese Modelle haben zum Ziel, ein Abbild mentaler Strukturen und Vorgänge darzustellen (Opwis & Spada 1994). Auf der Basis von Methoden der symbolischen KI werden kognitionswissenschaftlich fundierte Programme formuliert, mit denen die mentalen Modelle von Benutzern simuliert werden können.
- Wissensmodelle: In der Informatik werden Repräsentationen des Benutzerwissens verwendet, um nutzeradaptive Benutzungsschnittstellen zu implementieren. Diese Modelle speichern in Faktenform, welche Konzepte des technischen Systems, der Benutzungsschnittstelle und der Aufgabendomäne dem Benutzer bekannt sind und welche Ziele er aktuell verfolgt und passen die Darstellung oder Dialoge der Software daran an (Kobsa & Wahlster 1989).

In dieser Arbeit werden folgende in der Literatur uneinheitlich verwendete Begriffe für konkrete Ausprägungen dieser Modellklassen benutzt: Menschmodell (anthropometrische Modelle), Benutzermodell (Wissensmodell), Bedienermodell

(Regler-Mensch, Aufgaben-, konzeptuelle oder kognitive Modelle der Bedienung von Mensch-Maschine-Systemen).

Im Rahmen dieser Arbeit werden die anthropometrischen und regelungstechnisch motivierten Ansätze zur Bedienermodellierung nicht berücksichtigt, weil sie nicht in der Lage sind, Fragestellungen zu beantworten, die mit Wissensstrukturen zu tun haben. Insbesondere die Simulation von Verhalten in verteilten Mensch-Maschine-Systemen, bei hoher Eigendynamik im technischen Prozess und *Human Factors* Problemstellungen wie die Simulation von *Situation Awareness* und *Cognitive Workload* ist jedoch abhängig von der Modellierung solcher Wissensstrukturen (Kindsmüller et al. 2004).

2.6. Entwicklungsprozesse und Modellierung und Simulation in Mensch-Maschine-Systemen

Im folgenden Abschnitt werden Modellierung und Simulation als Methoden im Lebens- und Entwicklungsprozess der Mensch-Maschine-Systemtechnik dargestellt. Der Lebenszyklus von Systemen kann in die folgenden sechs Phasen eingeteilt werden (VDI 2221):

1) Systemvorstudie
2) Systementwicklung
3) Systemherstellung
4) Systemeinführung
5) Systembetrieb
6) Systemwechsel

Insbesondere die Entwicklungsphase wird aufgrund ihrer Komplexität anhand eines Vorgehensmodells gestaltet. Um eine zielgruppenkonforme und angemessene Funktionalität zu erreichen, ist eine interdisziplinäre Entwicklung mit ständiger Partizipation der Zielgruppe nützlich. In der Mensch-Maschine-Systemtechnik wird daher oft der parallel-iterative Entwicklungsprozess eingesetzt (Timpe & Kolrep 2000). Abbildung 2-3 zeigt dieses Vorgehensmodell. Der Entwicklungsprozess sieht eine phasenorientierte, parallele Bearbeitung von Aspekten der Mensch-Maschine-Interaktion und der funktionalen Entwicklung der Technik vor. Die Integration des technischen Entwicklungsstranges (Schritte „Funktionsanalyse“ und „Realisierung technischer Komponenten“) und des humanwissenschaftlichen Entwicklungsstranges (Schritte „Aufgabenanalyse“ und „Gestaltung GUI, Organisationsentwicklung“) erfolgt durch die Bewertung unterschiedlicher Lösungsansätze bereits in frühen Phasen der Systementwicklung.

Modelle und Simulationen der Mensch-Maschine-Interaktion können hier in unterschiedlichen Phasen eingesetzt werden. Zum einen werden bereits in frühen Phasen Aufgabenmodelle zur Planung der Benutzungsschnittstelle und der Systemfunktion eingesetzt (s. z.B. John & Kieras 1997). Zum anderen helfen hier

Simulationen, um die Aufgabenteilung zwischen Mensch und Maschine festzulegen (s. z.B. Kraiss 1998). Während der Entwicklung können durch Simulationen der Mensch-Maschine-Interaktion analog zu Benutzertests alternative Entwürfe für die Benutzungsschnittstelle getestet werden (s. z.B. Leuchter & Urbas 2004c). Im Betrieb können Benutzermodelle verwendet werden, um adaptive Benutzungsschnittstellen zu realisieren oder den Zustand der Benutzer vorherzusagen (s. z.B. Onken et al. 2001). Schließlich können bei der Einführung neuer Systeme Simulationen in Trainings der zukünftigen Benutzer eingesetzt werden (s. z.B. Urbas & Leuchter 2002).

Die Simulation der Mensch-Maschine-Interaktion stellt eine Aktivität in entsprechenden Vorgehensmodellen der Mensch-Maschine-Systemtechnik dar. Diese Aktivität gliedert sich wiederum nach einem Vorgehensmodell auf. Abbildung 2-4 zeigt ein Vorgehensmodell zur Simulation (in Anlehnung an Lugner & Bub 1990), das so in der Mensch-Maschine-Systemtechnik angewandt werden kann. Das Modell eines Systems wird aufgrund eines konkreten Bedarfs, der Problemstellung, entwickelt. Mit Hilfe einer Architektur bzw. einer Theorie wird über ein konzeptionelles Modell, das möglicherweise auch nur implizit bei der Formulierung eines konkreten formalen Modells vorhanden ist, ein quantitatives Modell abgeleitet. Simulationsexperimente sollen einen Einblick in das Benutzungsverhalten unter bestimmten gegebenen Bedingungen ermöglichen. Dazu muss eine ablauffähige Version dieses Modells z.B. in Form eines Computerprogramms entwickelt werden, mit dem in Simulationsläufen Ergebnisse auf der Basis wirklichkeitsanaloger Prozesse generiert werden können. Es handelt sich um einen rückgekoppelten Prozess, bei dem die Überprüfung und Interpretation der Ergebnisse eines Arbeitsschrittes mögliche Änderungen bedingen. Insbesondere durch die Auswirkungen der Simulationsergebnisse auf die Problemstellung ist die Modellierung in der Regel ein zyklischer Prozess.

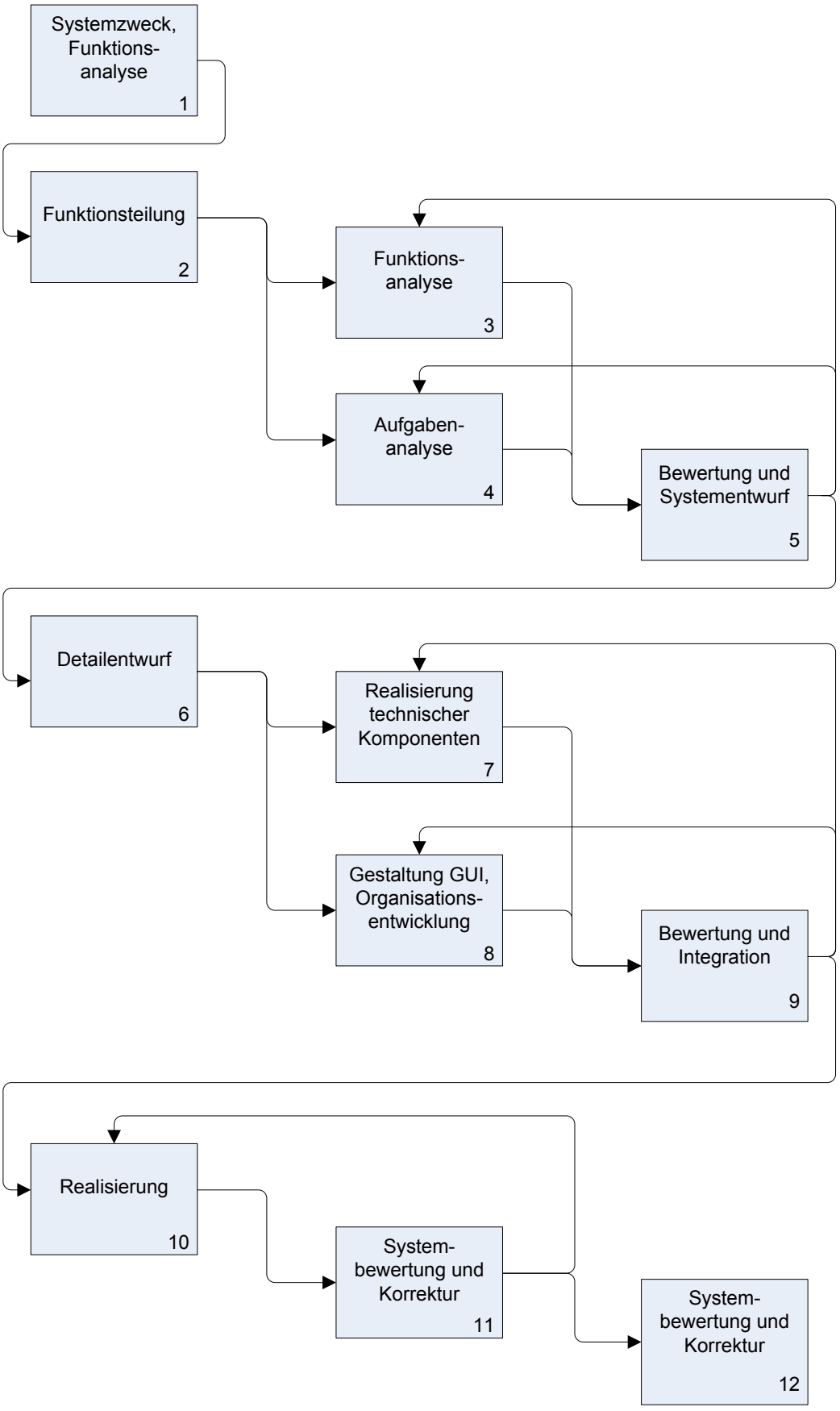


Abbildung 2-3: Phasenschema parallel-iterative Entwicklung von Timpe & Kolrep 2000 nach Leuchter & Urbas 2004a

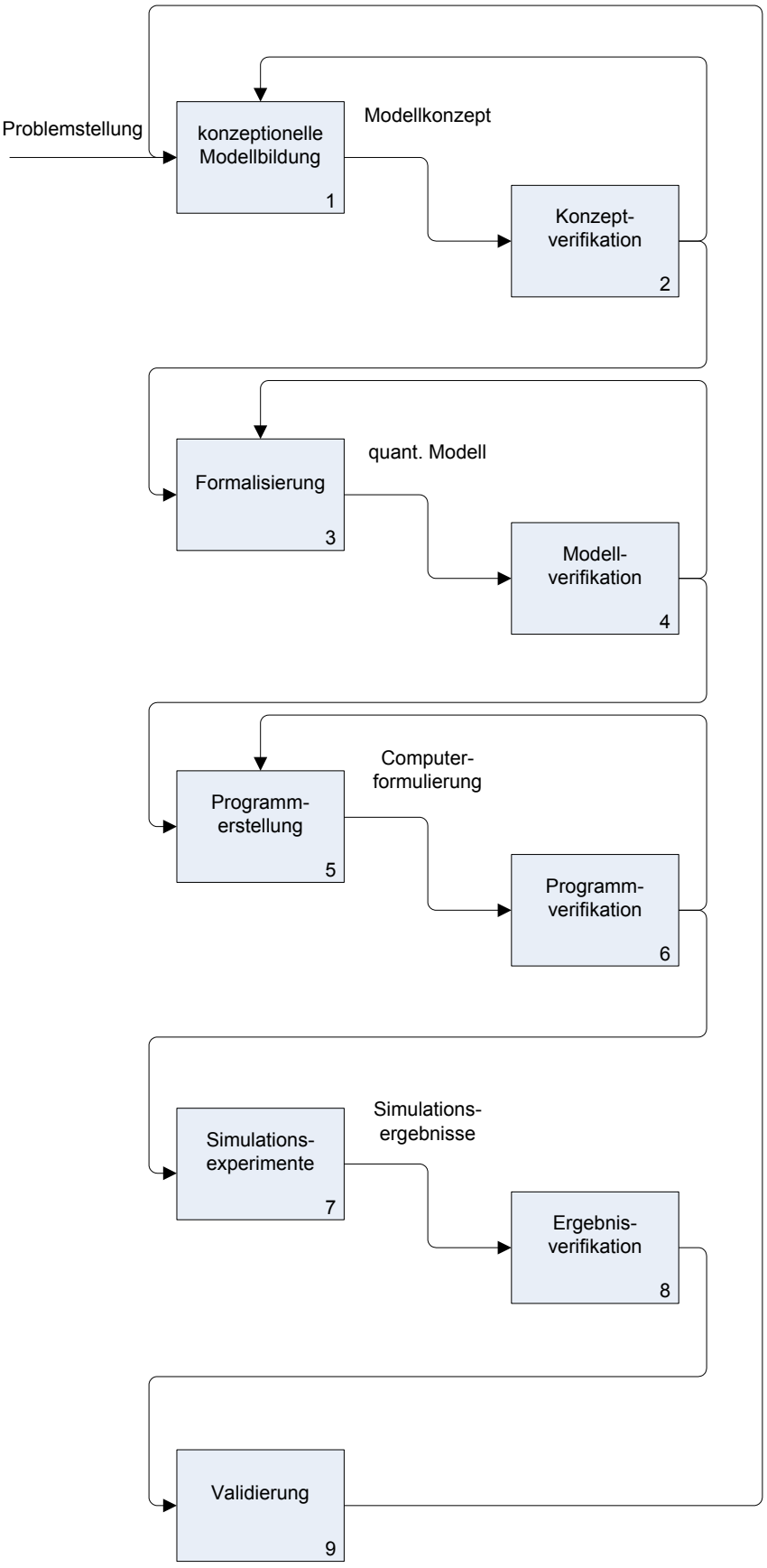


Abbildung 2-4: Prozess für Modellbildung und Simulation (in Anlehnung an Lugner & Bub 1990)

2.7. Wirtschaftlichkeit von Modellierung und Simulation in Mensch-Maschine-Systemen

Durch den Einsatz eines GOMS-Aufgabenmodells (s. 3.3.5) konnte in einem konkreten Fall vor Einführung eines neuen Telefonvermittlungssystems ein Designfehler, der zu geschätzten Kosten von 1,89 Mio USD pro Jahr geführt hätte, vorhergesagt und vermieden werden (Gray et al. 1993). Der Aufwand für die Erstellung und Validierung eines kognitiven Modells ist bei komplexen Aufgaben jedoch groß. Neue Werkzeuge, wie die in dieser Arbeit beschriebenen Systeme verringern zwar den Aufwand, jedoch müssen Kosten und Nutzen von Modellierung und Simulation alternativen Methoden gegenübergestellt werden.

Der Einsatz von Modellierung und Simulation während der Entwicklung geschieht, um Benutzungsprobleme zu identifizieren. Inzwischen kann eine Reihe von Phänomenen, die in Usability-Evaluationen von Computersimulationen oder Prototypen von technischen Systemen und ihren Benutzungsschnittstellen interessant sind, als wahrnehmungsnaher kognitiver Prozess auf einem niedrigen Abstraktionsniveau simuliert werden (s. z.B. Byrne 2001):

- Wechsel der visuellen Aufmerksamkeit zwischen Elementen der Benutzungsschnittstelle,
- Abruf von Gedächtnisinhalten, die für die Interpretation von Elementen der Benutzungsschnittstelle und ihres Zustandes erforderlich sind (beschleunigter oder verzögerter Abruf, Verwechseln, Vergessen) und
- Auffinden von Elementen der Benutzungsschnittstelle bzw. Abruf von deren Position aus dem Gedächtnis (beschleunigter oder verzögerter Abruf, Verwechseln, Vergessen oder Neu-Suchen).

Usability-Evaluationen mit kognitiven Modellen sind Software-Simulationen von Benutzungstestreihen (Simulator als *surrogate user*: Ritter et al. 2001). Interindividuelle Unterschiede können durch Parametrierung der kognitiven Architektur und des Modells, intraindividuelle Unterschiede bei der Bedienung einer Benutzungsschnittstelle durch statistische Überlagerung subsymbolischer Parameter erreicht werden.

Idealtypisch müssten mit simuliertem und realem Benutzertest dieselben Benutzungsprobleme bei unterschiedlichem Untersuchungsaufwand aufgedeckt werden. Der Geltungsbereich der Simulationen ist jedoch nicht so groß. Dafür können Gründe für Probleme, die modellgestützt entdeckt werden, durch eine Analyse der Simulationsergebnisse und des zugrunde liegenden Modells sehr genau verstanden werden. Im Gegensatz zu realen Benutzungstests sind ausführbare Spezifikationen von technischen Systemen und ihren Benutzungsschnittstellen, wie sie bei der modellbasierten Entwicklung bereits früh im Entwicklungsprozess anfallen, ausreichend. Simulierte Benutzungstests können also schon früher in der Entwicklung durchgeführt werden. Die simulierten Benutzungstests sind mit wenig

Aufwand (ggf. sogar automatisiert analog zu build-Tests im Software-Engineering) immer wieder im Entwicklungsprozess durchführbar (Leuchter & Urbas 2004c). Dadurch lässt sich eine hohe Produktqualität in der Entwicklung sicherstellen.

Die Kosten für die Erstellung von kognitiven Modellen sinken zwar durch den Einsatz neuer ingenieurmäßiger Werkzeuge (Urbas et al. 2005). Der Aufwand für die Modellierung ist jedoch immer noch in Relation zum Aufwand von Benutzertestreihen zu sehen. Der Modellierungsansatz wird in solchen Kostenrechnungen immer dann besonders gut abschneiden, wenn Benutzertests sehr aufwändig zu konfigurieren oder durchzuführen sind. Eine Simulation einer Benutzertestreihe ist daher immer dann relativ günstig, wenn verteilte Mensch-Maschine-Systeme entwickelt werden, die von mehreren Bedienern gesteuert werden oder die aus vielen interdependenten technischen Teilsystemen bestehen.

Ein Beispiel für solche komplexen Mensch-Maschine-Systeme ist die Flugsicherung. Der Nutzen des Fluglotsenmodells MoFL (s. 5.1) liegt hierbei in der Möglichkeit, vorhandene Parameter eines neuen Flugsicherungskonzeptes systematisch zu variieren und den simulierten Fluglotsen mit diesem System umgehen zu lassen. Dabei können in schneller Folge eine große Anzahl möglicher Szenarien durchgespielt werden. Es ist so mit verhältnismäßig geringem Aufwand möglich, einen sinnvollen Bereich der Parameterauslegungen zu generieren, an dessen Rändern genauere Untersuchungen mit zukünftigen Bedienern durchgeführt werden können (Jürgensohn et al. 2000).

Während des Betriebs von Mensch-Maschine-Systemen ist die Simulation der Benutzung eine Möglichkeit zur Implementierung von Unterstützungssystemen. Kognitive Modelle können zur Online-Simulation des Bedienerverhaltens eingesetzt werden (Onken et al. 2001). Die Simulation eines Modells läuft dann entweder anstatt des Bedieners ab und simuliert dessen Interaktion mit dem technischen System anhand der wahrnehmbaren Situation des technischen Systems, oder sie läuft neben dem Bediener mit und baut dessen vermutetes mentales Modell auf, um Unterstützungssysteme oder Trainings an den vermuteten mentalen Zustand des Benutzers anzupassen. Eine alternative Vorgehensweise zur Vorhersage des mentalen Zustandes des Benutzers bieten Bayes'sche Netze (Künzer et al. 2004), die jedoch in komplexen dynamischen Mensch-Maschine-Systemen nicht einsetzbar sind, denn die den Bedieneraktionen folgenden Systemzustände sind in der Komplexität ihrer Wechselwirkungen nicht a priori bestimmbar.

Ein weiterer möglicher Einsatz von kognitiven Modellen ist als eine Taxonomie für *Human Factors*-bezogene Fehler zur Benutzung als Retrievalschlüssel in Berichts- und anderen Dokumentationssystemen. Konzeptionelle Modelle werden bereits länger erfolgreich zur Unterstützung in der Fehleranalyse eingesetzt. Busse & Johnson (1999) beschreiben z.B. ein kognitives Modell einer konkreten Fehlersituation beim Landeanflug eines Piloten auf eine falsche Runway mit Hilfe der

kognitiven Architektur ICS (*interacting cognitive subsystems*: Barnard & May 1999). Bei Unfällen mit sicherheitskritischen Systemen sehen gesetzliche Vorschriften in vielen Bereichen eine ausführliche Analyse des Unfalls vor (z.B. beim Betrieb von Kernkraftwerken, in der Luftfahrt und im Seeverkehr). *Human Factors*-bezogene Fehler werden hierbei mit konzeptionellen Modellen identifiziert und nachvollzogen. Es entstehen also keine zusätzlichen Kosten, um eine solche Analyse durchzuführen, wenn ein solches Modell als Speicherschlüssel verwendet werden soll. Anders ist die Situation mit Vorfällen in sicherheitskritischen Systemen, die sich nicht als Unfall auswirken. Solche Vorfälle werden in vielen Organisationen gesammelt und zum organisationalen Lernen verwendet. Hier entstünde ein erheblicher Mehraufwand, denn bei den üblichen Systemen (z.B. in der norwegischen Off-Shore Industrie) werden keine umfangreichen Analysen angestellt, deren Ergebnisse so verwertet werden könnten. Es geht hier vielmehr um eine multifaktorielle Beschreibung der Problemsituation und nicht um eine Beschreibung der etwaigen kognitiven Ursache, um in erneuten ähnlichen Problemsituationen schnell vergangene Fälle mit ihren Lösungen zu finden.

Zusammenfassend lässt sich feststellen, dass Modellierung und Simulation dann ein gutes Kosten-Nutzenverhältnis gegenüber alternativen Methoden haben, wenn

- Adaptive Assistenz in dynamischen komplexen Mensch-Maschine-Systemen implementiert werden soll.
- Benutzungstests in frühen Phasen durchgeführt werden können. Dies ist dann der Fall, wenn Modell und Simulator per se bereits vorhanden sind oder Werkzeuge zur Transformation von Spezifikationen der Benutzungsschnittstelle vorhanden sind.
- Verteilte Mensch-Maschine-Systeme vorliegen, bei denen Benutzungstests extrem aufwändig sind
- Ein besonders hohes Risiko durch einen Unfall besteht, d.h. wenn der potenzielle Schaden sehr hoch ist.
- Der Zugriff auf interne Größen der Simulation erforderlich ist, um Rückschlüsse auf das Experimentalergebnis zu ziehen.
- Ein Mensch-Maschine-System vorliegt, bei dem die Kontrolle der Versuchsbedingungen in der Simulation deutlich leichter ist oder Benutzungstests als alternative Methoden hohe ökologische Kosten hätten, da trotzdem eine hohe Frequenz von (simulierten) Benutzungstests möglich ist.

2.8. Modellierungsebenen

Die Modellierung von Mensch-Maschine-Interaktion muss abhängig vom Modellzweck auf mehreren Ebenen mit unterschiedlichen Methoden und Werkzeugen erfolgen (Pew 2007). Auf der am feinsten aufgelösten Ebene werden die kognitiven Vorgänge analysiert (*cognitive band*: Newell 1990). Hier stehen

Modelle von Wahrnehmung, Gedächtnis und Handlungsregulation im Fokus. Modelle auf dieser Ebene erlauben Aussagen über kognitive Vorgänge wie die Auswahl alternativer Problemlösestrategien oder die Nutzung mentaler Ressourcen (Anderson 2002). Aus diesen Informationen lassen sich *Human Factors*-Konstrukte ableiten. Dadurch sind prospektive Aussagen zu Bedienfehlern aufgrund zu hoher kognitiver Beanspruchung oder mangelnder *Situation Awareness* möglich (Ritter et al. 2003). Es werden inter- und intraindividuelle Unterschiede mit unterschiedlichen Techniken simuliert. Der Geltungsbereich solcher Modelle liegt tendenziell im Bereich von Simulationen mit mehreren Minuten Dauer, wobei der Auflösungsgrad im Bereich von 1/10 Sekunden liegt. Der Aufwand bei der Erstellung solcher Modelle ist sehr hoch, denn es werden kleinste Phänomene modelliert.

Auf der nächsthöheren Analyseebene werden Aufgaben als Teilaufgaben und deren Zusammenhänge modelliert (*rational band*: Newell 1990). Hier wird von der „kognitiven Hardware“ der Bediener abstrahiert, die die Aufgaben erfüllen. Demzufolge werden keine inter- und intraindividuellen Unterschiede in der Leistung simuliert. Die zeitliche Auflösung der Modelle ist deutlich gröber als auf der kognitiven Analyseebene. Die Auflösung liegt im Sekundenbereich. Demzufolge können die sinnvoll simulierbaren Abschnitte von Interaktionsverhalten länger sein und komplexere Aufgaben beinhalten als auf der kognitiven Analyseebene. Der Aufwand für die Modellierung ist geringer als auf der kognitiven Analyseebene (Pew 2007). Aufgabenmodelle können zudem teilautomatisiert aus formalen Spezifikationen des technischen Systems abgeleitet werden (Hamacher et al. 2002b). Außerdem sind die Anforderungen an das Vorwissen des Modellierers deutlich geringer (John & Kieras 1997).

Zwischen der kognitiven Analyseebene und der Aufgabenanalyseebene gibt es eine Wechselwirkung: Die Aufgabenstruktur bestimmt wesentlich die kognitiven Vorgänge bei der Aufgabenbearbeitung und umgekehrt bestimmt die kognitive Architektur die potenzielle Leistungsfähigkeit des Menschen bei der Bearbeitung (Anderson 2002). Gegenwärtig werden Compilationstechniken (s. 3.4.1) diskutiert, die die eine direkte Verknüpfung dieser beiden Analyseebenen erlauben (Ritter et al. 2006a): Eine Aufgabenanalyse wird mit einer Theorie der Aufgabenabarbeitung teilautomatisiert in ein kognitives Modell transformiert. Dadurch kann mit einem Aufgabenmodell – und somit mit vergleichsweise geringem Aufwand – eine Simulation auf der kognitiven Analyseebene – also mit vergleichsweise hohem Detaillierungsgrad – durchgeführt werden.

3. Stand der Forschung

3.1. Bedienermodellierung

Es gibt verschiedene Methoden und Werkzeuge zur Bedienermodellierung. Sie decken unterschiedliche Aspekte der Mensch-Maschine-Interaktion ab. Die eine Standardmethode, die in jedem Anwendungsfall zum besten Resultat in Bezug auf Effizienz und Effektivität führt, gibt es nicht. Tatsächlich existiert ein Trade-Off zwischen Modellierungsaufwand und Simulationsgenauigkeit (Pew 2007): Einige Methoden erlauben die Modellierung auf einem hohen Abstraktionsniveau – und damit einen vergleichsweise geringeren Modellierungsaufwand, mit ihnen ist aber keine genaue Simulation der Interaktion möglich. Die Modellierungsmethoden, die auf einem niedrigen Abstraktionsniveau aufsetzen, können dagegen den gesamten Bereich der Interaktionsmodellierung abdecken (z.B. ACT-R: Anderson 2002), der Aufwand zum Erlernen der Anwendung der Methoden und bei der Durchführung ist jedoch hoch (Ritter et al. 2006a).

Ziel dieses Kapitels ist, die verfügbaren Methoden und Werkzeuge zur Bedienermodellierung vorzustellen, in ein Schema einzuordnen und miteinander zu vergleichen, um dadurch die geeignetsten Methoden und Werkzeuge zur Bedienermodellierung in dynamischen Mensch-Maschine-Systemen und gegebenenfalls Bedarf zur Weiterentwicklung zu identifizieren, damit der Trade-Off zwischen Aufwand und Genauigkeit günstiger wird.

Die Struktur dieses Kapitels orientiert sich an der oben getroffenen Untergliederung in die kognitive Analyseebene und die Aufgabenanalyseebene, sowie deren Synthese (s. 2.8). Dieser Unterteilung können alle Methoden und Werkzeuge zur Bedienermodellierung zugeordnet werden. Das Ordnungsschema orientiert sich damit an einer im Fachgebiet gängigen Untergliederung, die z.B. auch von Pew (2007) verwendet wird.

Nach einer Einführung in die Grundlagen der Bedienermodellierungsmethoden auf der kognitiven Analyseebene werden in Abschnitt 3.2.2 wichtige kognitive Architekturen und Werkzeuge zur kognitiven Simulation vorgestellt. Deren Auswahl ist durch den vermuteten Nutzen für die Bedienermodellierung in dynamischen Mensch-Maschine-Systemen gesteuert. Im folgenden Abschnitt wird dann berichtet, welche Phänomene auf dieser Ebene erfolgreich modelliert wurden. Dabei wird wiederum auf für den Zweck dieser Arbeit relevante Phänomene fokussiert. Aus den vorgestellten Anwendungsbeispielen werden anschließend die vertretenen Anwendungsfelder und ihre Anforderungen an Methoden und Werkzeuge zur Bedienermodellierung herausgearbeitet. Da die Kopplung von Bedienermodell und Aufgabenumgebung und die Möglichkeit zur Modellierung der Wahrnehmung von Elementen der Benutzungsschnittstelle besondere Herausforderungen auf dieser Modellierungsebene darstellen, wird der Stand der Forschung auf diesem Gebiet

gesondert betrachtet. Der Abschnitt endet mit einer bewertenden Zusammenfassung der Methoden und Werkzeuge auf der Ebene der Kognitionsanalyse.

Der folgende Abschnitt zur Modellierung und Simulation auf der Ebene Aufgabenanalyse beginnt mit einer Darstellung der diesem Ansatz zugrundeliegenden Theorie der hierarchischen Aufgabenanalyse. Dann werden ausgewählte Methoden zur Aufgabenanalyse und -modellierung aus den Bereichen *Human-Computer Interaction* und Software Engineering vorgestellt. Neben diesen Ansätzen gibt es weitere Methodenfamilien, die gesondert dargestellt werden: *Human Reliability Analysis*, *Task Networks* und die GOMS-Familie von Aufgabenmodellierungsansätzen stellen jeweils eigene Gruppen von Methoden dar, die untereinander hohe Ähnlichkeit in Annahmen und Vorgehen haben. Die eindeutige Zuordnung von Modellierungsmethoden zu diesen drei Familien ergibt sich aufgrund des dort jeweils gewählten Ansatzes (Pew 2007). Wie im vorigen Abschnitt zur Analyseebene Kognition werden abschließend modellierte Phänomene, Anwendungsfelder und ihre Anforderungen und eine bewertende Zusammenfassung auf dieser Analyseebene vorgestellt.

Die Darstellung des Standes der Technik endet mit einer Zusammenfassung, in der die Synthese der beiden Analyseebenen „Kognition“ und „Aufgabe“ anhand aktueller Arbeiten zu Compilationsansätzen und hybriden Modellierungsansätzen vorgestellt werden. In einer abschließenden Gesamtzusammenfassung des Kapitels zum Stand der Technik werden diese beiden Herangehensweisen bewertet und der Handlungsbedarf bei der Weiterentwicklung von Methoden und Werkzeugen zur Bedienermodellierung aufgezeigt.

3.2. Modellierung und Simulation auf der Ebene Kognitionsanalyse

3.2.1. Kognitive Modellierung

Eine Grundannahme der Kognitionswissenschaft ist die Repräsentierbarkeit von mentalen Vorgängen als Informationsverarbeitung in Symbolsystemen. Nach Newell (1990, S. 76ff) haben Symbolsysteme die folgenden Elemente:

- Symbole: Muster, die Entitäten oder Zusammenhänge repräsentieren
- Speicher: Menge von Strukturen aus Symbolen
- Operationen: Prozesse mit Symbolstrukturen als Input und Output
- Interpretation: Prozesse, die Operationen ausführen, mit Symbolstrukturen als Input

Symbolsysteme bestehen also aus Elementarsymbolen und formalen Regeln zur Bildung und Transformation von Symbolstrukturen. Ein *physical symbol system* ist solch ein System, wenn es eine materielle Implementierung hat (Strube 1996). Die Realisierung in Bezug auf das Substrat ist dabei beliebig (z.B. Neuronen oder Transistoren). Die Strukturen und Prozesse von *physical symbol systems* können

also unterschiedlich implementiert werden, z.B. als Computerprogramm oder nach Annahme der Kognitionswissenschaft als mentale Vorgänge im Gehirn des Menschen. Die *physical symbol system hypothesis* besagt, dass ein materielles Symbolsystem als Implementierung eines Symbolsystems die notwendige und prinzipiell auch ausreichende Bedingung für Kognition ist (Newell & Simon 1976, Newell 1980, Anderson 2007). Deshalb gelten Computerimplementierungen von Symbolsystemen als legitimes Mittel, um kognitive Vorgänge zu modellieren und zu simulieren.

In der Kognitionswissenschaft werden mehrere Frameworks zur Formalisierung kognitiver Vorgänge im Sinne eines Symbolsystems benutzt. Neben künstlichen neuronalen Netzen sind Produktionssysteme (Post 1943) eine weit verbreitete Basis zur Implementierung von Symbolsystemen. Produktionssysteme bestehen aus einem Satz von Produktionsregeln und einem Speicher mit Symbolen. Ein Interpreter wendet die Produktionsregeln nach einem vorgegebenen Formalismus auf den Symbolspeicher an. Die Produktionsregeln haben einen Bedingungsteil, der sich auf den Zustand des Speichers bezieht, und einen Aktionsteil, der ausgeführt wird, wenn die Regel vom Interpreter ausgeführt wird („feuert“).

Im Bedingungsteil kann je nach verwendetem Formalismus ausgedrückt werden, dass bestimmte Muster von Symbolstrukturen oder einzelne Symbole im Speicher vorhanden sein müssen oder nicht vorhanden sein dürfen. Mit logischen Verknüpfungen und Quantoren können dafür in manchen Produktionssystemformalismen komplexe Ausdrücke formuliert werden. Im Bedingungsteil können Platzhalter als Variablen benutzt werden. Von einer Produktionsregel können Instanzen gebildet werden, indem alle Variablen im Bedingungsteil gebunden werden. Bei der Bindung einer Variablen wird ihr Platzhalter durch den konkreten Wert eines passenden Symbols ersetzt. Hat eine Produktionsregel im Bedingungsteil keine Variablen ist sie gleichzeitig ihre einzige Instanz.

Im Aktionsteil von Produktionsregeln können Funktionen aufgerufen werden, mit denen neue Symbole im Speicher erzeugt, Symbole zu neuen Strukturen zusammengefasst und Symbole oder Symbolstrukturen modifiziert oder gelöscht werden können. Bei manchen Formalismen zur Formalisierung von Produktionssystemen können im Aktionsteil auch Funktionen aufgerufen werden, mit denen die Arbeitsweise des Interpreters verändert werden kann oder Aktionen in einer externen Umwelt außerhalb des Produktionssystems veranlasst werden können. Im Aktionsteil können die Variablen aus dem Bedingungssteil der Regel benutzt werden. In einer Produktionsregelinstanz sind diese Variablen an dieselben Werte wie im Bedingungsteil gebunden.

Der Aktionsteil einer Produktionsregel wird ausgeführt, wenn die Produktionsregel feuert. Regeln werden vom Interpreter, der Ablaufumgebung des Produktionssystems, während der Laufzeit zum Feuern ausgewählt. Eine

Produktionsregel kann zum Feuern ausgewählt werden, wenn ihr Bedingungsteil zutrifft. Der Interpreter in Produktionssystemen läuft in einem Verarbeitungszyklus nach einem Formalismus in drei Schritten ab:

1) *Matching*: Alle Produktionsregeln, deren Bedingungsteil zu diesem Zeitpunkt erfüllt werden kann, werden ausgewählt. Für jede ausgewählte Produktionsregel werden alle möglichen Instanzen mit der aktuellen Konfiguration des Symbolspeichers gebildet. Diese Instanzen bilden die Konfliktmenge.
2) Konfliktresolution: Je nach Interpreter werden aus der Konfliktmenge eine oder mehrere Produktionsregelinstanzen ausgewählt und zu Feuern markiert. Dafür werden unterschiedliche Verfahren benutzt. In einigen Produktionssystemen wird hier in jedem Verarbeitungszyklus genau eine Instanz ausgewählt, in anderen einige oder sogar alle. Es gibt unterschiedliche Verfahren, nach denen Instanzen ausgewählt werden:
   - *Most specific first*: die Instanz, bei der die Wahrscheinlichkeit des Zutreffens am geringsten ist
   - *Most general first*: die Instanz, bei der die Wahrscheinlichkeit des Zutreffens am größten ist
   - *Most recently used*: eine Instanz einer Produktionsregel, bei der die Zeit seit der letzten Nutzung am geringsten ist, bei mehren Instanzen dieser Regel, die bei der am meisten Variablen dieselbe Bindung hatten
   - *Least recently used*: eine Instanz einer Produktionsregel, bei der die Zeit seit der letzten Nutzung am größten ist, bei mehren Instanzen dieser Regel, die die am meisten Variablen eine andere Bindung hatten
   - Zufällige Auswahl: indeterministische Auswahl einer Produktionsregelinstanz
   - Einbeziehung von subsymbolischen Parametern, die während der Verarbeitung angepasst werden (s. z.B. ACT-R); beispielsweise die Zeit die zur Bindung der Variablen bei der Instanzbildung benötigt wird (komplexere Symbole werden hier langsamer gebunden, oft benutzte Symbole können schneller gebunden werden, oft benutzte Produktionen binden Variablen schneller etc.)
   - Komplexe: eine Kombination aus den vorstehenden Verfahren
3) Ausführung: Die im zweiten Schritt zum Feuern markierten Produktionsregelinstanzen werden ausgeführt – die Produktionen feuern. Der Aktionsteil wird dabei mit den genauso wie im Bedingungsteil gebundenen Variablen ausgeführt. Wenn mehrere Instanzen zum Feuern markiert sind, feuern die Produktionen alle gleichzeitig („parallel“).

Bei einigen Formalismen für Produktionssysteminterpreter ist es möglich, dass Produktionsregeln mit nur teilweise zutreffendem Bedingungsteil *matchen* und ihre Instanzen zur Konfliktmenge hinzugefügt werden („*partial matching*“). Der

Konfliktresolutionsalgorithmus berücksichtigt dann den Grad der Übereinstimmung: Je geringer der Anteil an zutreffenden Bedingungsteilen in einer Produktion beim *partial matching* ist, desto unwahrscheinlicher wird, dass sie im Konfliktresolutionsschritt zum Feuern ausgewählt wird.

Mit unterschiedlichen Konfliktresolutionsstrategien können verschiedene Verhaltensweisen in einem Produktionssystem implementiert werden. So kann ein System entweder in den immer gleichen Bahnen arbeiten (*most recently used*) oder eine große Bandbreite von Verhalten zeigen (*least recently used*).

Damit ist eng der Grad der Zielorientierung bei der Verarbeitung verbunden. Ein Symbolsystem hat zu einem Zeitpunkt möglicherweise ein Ziel, auf das die Verarbeitung hinstrebt (z.B. Problemlösung). Produktionssysteme können dazu Ziele als Symbolstrukturen repräsentieren und wie andere Symbole mit Produktionsregeln in die Verarbeitung im Bedingungsteil und im Aktionsteil manipulieren. Dadurch kann Unterzielbildung und hierarchische Aufgabenzerlegung in Produktionssysteme modelliert werden.

Man unterscheidet entsprechend zwei Modi bei der Verarbeitung in Regelsystemen: die daten- und die zielorientierte Regelanwendung. Bei der datenorientierten Verarbeitung wird der Zustand des Speichers mit den aktuell anwendbaren Produktionsregeln vorwärtsverkettend transformiert – hierbei werden die Heuristiken der Konfliktresolution und Zielstrukturen als Hilfestellung benutzt – während bei der zielorientierten Verarbeitung Rückwärtsverkettung benutzt wird und ausgehend von einem Zielzustand (z.B. wie in Prolog mit dem Resolutionskalkül von Robinson) auf die anzuwendenden Regeln zurückgeschlossen wird.

Unter der Annahme, dass die *physical symbol system hypothesis* zutrifft und menschliche Kognition durch solche Berechnungsvorgänge abbildbar ist, muss die Architektur des menschlichen Verarbeitungssystems die elementaren Mechanismen beinhalten, die den kognitiven Prozessen zugrunde liegen (Krems 1996). Eine solche kognitive Architektur ist demnach eine mögliche Implementierung von einem Symbolsystem.

Newell (1990) fordert darüber hinaus die Formulierung von *unified theories of cognition*. Das sind implementierte Symbolsysteme bzw. kognitive Architekturen, mit denen eine Vielzahl kognitiver Phänomene simulierbar ist. Sie beinhalten dazu sowohl die Theorie als auch einen Formalisierungsmechanismus in Form einer spezialisierten Programmiersprache. Anforderungen an solche kognitive Architekturen sind, dass sie insbesondere Mechanismen für die Interaktion mit der Umgebung, für den Erwerb und die Repräsentation von Wissen und für das Entscheiden haben (Newell 1990). Solche kognitiven Architekturen beinhalten auch theoretisch begründete Einschränkungen der Freiheitsgrade bei der Formalisierung. Sie bieten damit einen einheitlichen konzeptuellen und technischen Rahmen, der zwischen unterschiedlichen Anwendungen konstant bleibt (Krems 1996). Es gibt

mehrere kognitive Architekturen, die Anwärter auf den Titel einer *unified theory of cognition* sind. Insbesondere Soar und ACT-R sind potenzielle *unified theories of cognition*.

Existierende Architekturen von kognitiven Modellen realisieren jedoch unterschiedliche Ansprüche an ihre Allgemeingültigkeit. So gibt es kognitive Modelle, die auf der Basis einer anwendungsspezifischen Theorie über Symbolsysteme mit einer *general purpose* Programmiersprache implementiert werden („ad hoc Modelle“). Andere Modelle bedienen sich einer *unified theory of cognition* und derer Programmiersprache. Im Folgenden werden Vor- und Nachteile dieser beiden Ansätze erörtert.

Ad hoc Modelle sind vergleichsweise einfach zu formalisieren und zu implementieren. Der Modellierer muss sich keinen Einschränkungen des Modellierungsformalismus' unterwerfen. Die Verwendung von *general purpose* Programmiersprachen hat den Vorteil, dass Expertise leicht verfügbar ist. Jedoch müssen die Basismechanismen zur Implementierung des Symbolsystems immer wieder neu formalisiert werden. Würden entsprechende Code-Anteile in wiederverwendbare Libraries ausgelagert werden, bedeutete dies einen ähnlich hohen Pflegeaufwand wie bei einer *unified theory of cognition*: Bei Änderungen an den Libraries müssen alle früheren damit implementierten Modelle noch laufen. Die Kommunizierbarkeit von Modellen ist eingeschränkt, da die Notation einer *general purpose* Programmiersprache keine spezifischen Konstrukte für die Modellierung kognitiver Prozesse vorsieht. Ein Anschluss an externe Prozesse und Funktionen ist mit *general purpose* Programmiersprachen leicht zu erreichen.

Kognitive Modelle mit einer *unified theory of cognition* als Architektur sind aufwändiger zu implementieren. Durch die Beschränkung auf die Mechanismen der vorgegebenen kognitiven Architektur entsteht ein Implementierungsoverhead. Die Beschränkungen der kognitiven Architektur unterstützen jedoch auch die Formalisierung dadurch, dass sie durch Vorgaben leiten. Die kognitive Architektur liefert Basismechanismen und Eigenschaften nicht nur aus dem direkt modellierten Bereich, sondern geht darüber hinaus. So könnten Lernmechanismen der kognitiven Architektur zur Beantwortung von Fragen nach der Erlernbarkeit einer Gestaltungslösung genutzt werden, obwohl der ursprüngliche Modellzweck in der Analyse ihrer Bedienbarkeit lag. Die Verwendung einer *unified theory of cognition* orientierten Architektur ermöglicht die Nutzung zukünftiger Verfeinerungen und Erweiterungen der Architektur, die von einer Community beigetragen werden. Modellbereiche sind möglicherweise eher wiederverwendbar. Die Beschränkungen einer solchen Architektur wirken sich jedoch auch auf die umsetzbaren Modelle aus: So ist man auch im Modell an die in der Architektur adressierten kognitiven Mechanismen und Strukturen gebunden (bspw. Beschränkung auf Problemlösen, Probleme bei der Kopplung an externe Simulatoren, Integration in Simulationssysteme). Die Verwendung einer komplexen kognitiven Architektur

bedingt zudem auch eine Ausbildung in den kognitionswissenschaftlichen Prinzipien, die ihr zugrunde liegen.

Kindsmüller et al. (2004) unterscheiden *ad hoc* und *unified theory of cognition* basierte kognitive Modelle in Bezug auf die prospektive Überprüfung von Gestaltungsalternativen bei der Mensch-Maschine-Interaktion: Die letzteren sind langlebigere Modelle, denn viele Annahmen über grundlegende kognitive Strukturen und Mechanismen sind indirekt über die kognitive Architektur in die Entwicklung eingeflossen. In den Anwendungsdomänen, in denen Produktlebenszyklen lang sind und wo der evolutionäre Schritt zwischen Produktgenerationen klein ist, kann die prädiktive Kraft von den langlebigeren Modellen mit umfangreicher kognitiver Architektur genutzt werden. Beispiele für solche Anwendungsdomänen sind die Flugführung, die Flugsicherung, und Prozessführung in Leitwarten. Ad hoc Modelle sind in diesem Sinne kurzlebig. Sie haben einen geringeren Geltungsbereich, der sich nur auf eine konkrete Gestaltungsalternative beziehen kann, für die sie entworfen wurden. In Anwendungsbereichen, in denen die technischen Produkte rasch wechseln und sich auch Interaktionsprinzipien fortwährend ändern, wie z.B. bei Mobiltelefonen und PDAs, ist eine fallweise Neuentwicklung von kognitiven Modellen erforderlich. Die Entwicklung von ad hoc Modellen zur Evaluierung solcher Produkte ist dann mit dem geringst möglichen Aufwand verbunden.

Der Einsatz von aufwändiger anzuwendenden kognitiven Architekturen – ob mit dem Anspruch einer *unified theory of cognition* oder nicht – als Basis für Simulationen mit kognitiven Modellen verspricht Vorteile gegenüber der in einigen Bereichen weniger aufwändigen ad hoc Modellierung mit einer *general purpose* Programmiersprache:

- Fundierung auf kognitionswissenschaftlich begründete Strukturen und Mechanismen, die validierte Konstrukte beinhalten, wodurch eine höhere Prädiktionskraft erwartet werden kann, also ein geringerer Aufwand für das Erheben von Parametern besteht.
- Hinter kognitiven Architekturen steht jeweils eine Community, die neue Erkenntnisse auf der Basis der Architektur gewinnt und Ergebnisse im Rahmen der Architektur zur Verfügung stellt.
- Insbesondere im Bereich der Gedächtnisleistungen und der Wahrnehmung gibt es in aktuellen Architekturen viele funktionierende Bausteine.

Im Folgenden werden einige auf Produktionssystemen basierende kognitive Architekturen und andere Werkzeuge zur kognitiven Modellierung vorgestellt, die für die Modellierung von Bedienerverhalten auf der kognitiven Analyseebene in Frage kommen. Dabei werden die Elemente eines Symbolsystems auf kognitionswissenschaftliche oder implementierungstechnische Strukturen abgebildet:

- Symbole sind in der Kognitionswissenschaft Chunks. Chunks sind die Gedächtniseinheiten, die sich gegenseitig referenzieren können. Chunks können

auch andere Chunks enthalten. In der Implementierung mit Mitteln der künstlichen Intelligenz sind Chunks Fakten bzw. deklarative Wissenseinheiten.

- Speicher: In der Kognitionspsychologie gibt es unterschiedliche Modelle für das Gedächtnis. In den meisten Modellen gibt es jedoch einen Bereich, der Arbeitsgedächtnis genannt wird. Im Arbeitsgedächtnis stehen kurzfristige Informationen, die gegenwärtig durch mentale Prozesse verarbeitet werden. Durch Lernprozesse können Informationen von hier ins Langzeitgedächtnis übertragen werden. Manche Modelle sehen daneben noch sensorische Kurzzeitspeicher vor. In der technischen Realisierung mit Mitteln der Künstlichen Intelligenz ist das Arbeitsgedächtnis ein Speicher für deklarative Wissenseinheiten und Fakten.
- Produktionsregel: In der Kognitionswissenschaft steht neben dem deklarativen Wissen das prozedurale. Potenziell sind alle Wissenseinheiten sowohl prozedural als auch deklarativ formalisierbar. Bei verbalisierbarem Wissen geht man von einer deklarativen Codierung und bei nicht-verbalisierbarem Wissen von einer prozeduralen Codierung aus. Prozedurales Wissen wird in Form von Produktionsregeln formalisiert. Produktionsregeln werden oft als Inhalt des Langzeitgedächtnisses angesehen. Es handelt sich um hoch automatisiertes gelerntes Wissen.
- Operation: Operationen werden in den folgenden Systemen durch Produktionsregeln implementiert.
- Interpretation: Interpretation ist nicht durchgängig in den folgenden Systemen vorhanden. In manchen Systemen werden diese Elemente des Symbolsystems Metakognition oder Metaregeln genannt. Deren Interpretation wird durch Produktionsregeln implementiert.

3.2.2. Kognitive Architekturen und Werkzeuge zur kognitiven Simulation

Es gibt eine große Anzahl kognitiver Architekturen und Werkzeuge zur kognitiven Simulation. Darunter sind viele Forschungsprototypen, die teilweise zum Erkenntnisgewinn in eng begrenzten Forschungsgebieten entwickelt wurden. Es musste für diese Arbeit demzufolge eine Auswahl von Systemen getroffen werden, die die anwendungsorientierte Modellierung im Bereich der Mensch-Maschine-Interaktion unterstützen. Im Bereich der wehrtechnischen Forschung in den USA gibt es dazu einige öffentlich zugängliche Gegenüberstellungen von Methoden und Werkzeugen zur Bedienermodellierung, die hier als Informationsgrundlage dienen:

- Pew & Mavor (1998) stellen die Systeme ACT-R, COGNET, EPIC, HOS, Micro SAINT, MIDAS, OMAR, SAMPLE, Soar als Kandidaten für militärische Human Factors Simulationsanwendungen vor.

- Schoelles (2002) bezieht sich bei seiner Betrachtung explizit auf kognitive Architekturen in dynamischen Umgebungen und vergleicht für diesen Anwendungszweck EPIC, Soar und ACT-R.
- Ritter et al. (2003) unterscheiden zwischen psychologisch orientierten Architekturen (ACT-R, Soar, EPAM, neuronalen Netzwerken, PSI, COGENT), dem Sim Agent Toolkit und Engineering Modellen (APEX, SMOC und COCOM).
- Gluck & Pew (2005) konnten im AMBR-Programm die Systeme ACT-R, COGNET/iGEN, EASE (Elements of ACT-R/Soar/EPIC) und DCOG (Distributed Cognition Framework) ausführlich vergleichen.

Nach der Gesamtgliederung dieses Kapitels gehören die Systeme Micro SAINT, MIDAS (und OMAR als direkter Verwandter), APEX, SMOC und COCOM zur Ebene der Aufgabenmodellierung. Diese Systeme werden deshalb unten im Abschnitt 3.3 noch einmal aufgegriffen und hier nicht weiter betrachtet.

Bei allen oben aufgeführten Surveys waren EPIC, ACT-R und Soar in den Vergleich aufgenommen worden. Diese Systeme werden deshalb in dieser Arbeit vorrangig untersucht. Für die weitere Filterung der Kandidaten wurden die Proceedings der etablierten Konferenzreihe *International Conference on Cognitive Modeling* (ICCM) herangezogen. Daraus geht hervor, dass von den weiteren Kandidatensystemen COGNET/iGEN (und damit auch dessen Vorgänger HOS), COGENT und PSI in der Forschung verwendet werden. Von PSI sind jedoch keine Anwendungen im Bereich Mensch-Maschine-Interaktion bekannt. Die besonderen Fähigkeiten von PSI im Bereich der Modellierung von Motivation und Emotion spielen zudem in dieser Arbeit keine Rolle. Demzufolge werden von jenen Systemen nur COGNET/iGEN und COGENT in die Betrachtung aufgenommen. Das System OCCS wird wegen der Verwendung einer Wissenskomponente daraus in einer der unten beschriebenen Fallstudie (s. 5.2) zusätzlich beschrieben.

In den nächsten Abschnitten folgen Beschreibungen der architekturiellen Annahmen, Strukturen und Verarbeitungsprozesse von EPIC, ACT-R, Soar, COGNET/iGEN, COGENT und OCCS.

EPIC

EPIC (*executive process — interactive control*) ist eine kognitive Architektur. Sie hat zum Ziel, Modelle von Operator-Leistungen in neuen Arbeitsumgebungen, die mit Multimodalität und Mehrfachaufgaben zurechtkommen müssen, ingenieurmäßig beschreibbar zu machen. Dabei sollen Resultate ökonomisch erzielt werden können.

Die Architektur von EPIC gruppiert Motor- und Wahrnehmungsprozessoren um einen kognitiven Kern (s. Abbildung 3-1). Sie interagieren mit einer simulierten Aufgabenumgebung. Die einzelnen Komponenten dieser Architektur sind: Kognitiver Prozessor mit Produktionsregelinterpreter und Arbeitsgedächtnis als Symbolspeicher, Wahrnehmungsprozessoren (visueller Prozessor, auditiver

Prozessor, taktiler Prozessor) und Motorprozessoren (manuell, okular, vokal), die mit einer simulierten Aufgabenumgebung interagieren (Kieras et al. 1997, Kieras & Meyer 1997).

Der kognitive Prozessor ist ein regelbasierter Interpreter. Er besteht aus einem Produktionsregelinterpreter und einem Arbeitsgedächtnis in Form eines Symbolspeichers, auf das die Produktionen zugreifen. Der Interpreter ist eine modifizierte Version des in der Programmiersprache LISP implementierten *Parsimonious Production System*. Das Produktionssystem ist parallel, d.h. alle Produktionsregelinstanzen in der Konfliktmenge feuern gleichzeitig. Deshalb ist kein Konfliktresolutionsalgorithmus vorgesehen. Die Begründung für dieses Verhalten liegt in der Annahme, dass es ausreiche, die Verarbeitung durch eine realistische Beschränkung der Eingabe- und Ausgabekanäle zu begrenzen, die die Verarbeitung innerhalb des kognitiven Prozessors triggern sollen. Empirische Befunde von Meyer et al. (1995) belegen das in einigen Anwendungsbereichen. Ein Verarbeitungszyklus dauert in EPIC 50 ms, wobei Simulationen auch in einem stochastischen Modus ausgeführt werden können, so dass die Dauer eines Zyklus' dann um diesen Mittelwert schwankt. Es gibt einige besondere Funktionen, die Produktionen in ihrem Aktionsteil ausführen können. So gibt es eine Zufallsfunktion „guess“, die aus einer Liste zufällig einen Wert bestimmt, der an eine Variable gebunden wird. „time-stamp“ bindet die aktuelle Zyklusnummer an eine bestimmte Variable. Mit „increment“ können Variablen hochgezählt werden. „delay-count down“ verwaltet das Warten für eine bestimmte Anzahl von Zyklen, indem eine Variable entsprechend in jedem Zyklus manipuliert wird.

Es gibt kein Aktivationskonzept im Symbolspeicher von EPIC. Auch eine Vergessensfunktion ist im kognitiven Prozessor nicht vorgesehen. Die Kapazität des Arbeitsgedächtnisses in EPIC ist unbeschränkt. Es unterteilt sich in unterschiedliche Partitionen, die jedoch nicht vom Produktionssysteminterpreter vorgegeben werden, sondern durch Konvention aufgeteilt werden. Das Arbeitsgedächtnis wird in visuelle, auditive und taktile Partitionen aufgeteilt, die die Wahrnehmungen der entsprechenden Prozessoren speichern. Zusätzlich gibt es ein Motor-Arbeitsgedächtnis, in dem der aktuelle Zustand der Motorprozessoren, z.B. ob die Hand gerade in Bewegung ist, vermerkt wird. Zwei besondere Teile des Arbeitsgedächtnisses werden *amodal* genannt, weil sie Informationen enthalten, die nicht direkt aus den anderen Prozessoren gewonnen werden. Der *control store* enthält Zielstrukturen und Repräsentationen von Schritten, wie innerhalb von Prozeduren Ziele erreicht werden. Produktionen können, anders als z.B. in Soar-Modellen, die Zielstrukturen direkt manipulieren. Der andere amodale Speicher innerhalb des Arbeitsgedächtnisses ist dafür vorgesehen, Schlüsse von Regeln oder andere Zwischenergebnisse, die aus der Arbeit von Produktionen entstehen, zu speichern.

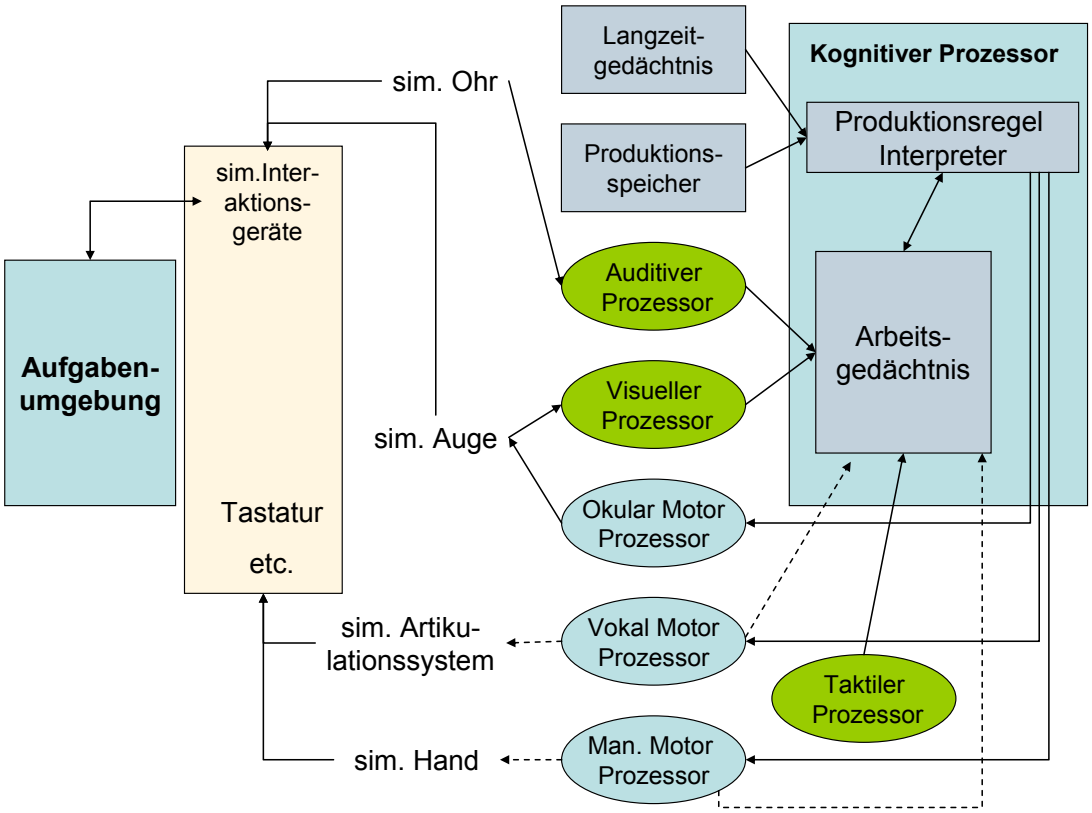


Abbildung 3-1: Architektur von EPIC: Mehrere Wahrnehmungs- und Motorprozessoren verbinden einen kognitiven Prozessor mit einer simulierten Aufgabenumgebung (nach Kieras & Meyer 1997)

Die Wahrnehmungsprozessoren lesen Informationen über den aktuellen Zustand der simulierten Aufgabenumgebung aus und speichern entsprechende Informationen in entsprechenden Teilen des Arbeitsgedächtnisses.

Visueller Prozessor: Das Modell des Auges in EPIC beinhaltet eine Retina, die bestimmt, welche Art sensorischer Informationen über Objekte in der Aufgabenumgebung abhängig vom Abstand eines Objektes auf der Retina zur Mitte der Fovea an den kognitiven Prozessor gemeldet werden. Dazu werden in EPIC drei Bereiche mit unterschiedlichen Wahrnehmungsfähigkeiten definiert: Fovea (Radius 1°), Parafovea (Radius 10°) und Peripherie (Radius 60°). Der erkannte Inhalt wird als Repräsentation der Objekte und ihrer Eigenschaften in den entsprechenden Bereich des Arbeitsgedächtnisses übertragen. Zur symbolischen Verarbeitung visueller Informationen werden der visuelle Sensorikprozessor (VSENP) und der visuelle Wahrnehmungsprozessor (VPERP) benutzt. Die Aufgabe des VSENP ist, visuelle Ereignisse zu filtern und mit der entsprechenden Verzögerung die physikalischen Informationen an den VPERP zu senden. Der VPERP fügt entsprechende Repräsentationen im Arbeitsgedächtnis ein und sendet Motoranweisungen an den „*Ocular Motor Processor*“ (s. Abbildung 3-1).

Auditiver Prozessor: Auditive Informationen aus der simulierten Aufgabenumgebung werden über den auditiven Prozessor in den kognitiven

Prozessor in EPIC übertragen. Die Informationen gelangen als Repräsentation einer Sequenz von auditiven Ereignissen (z.B. Sprache) oder als einzelne Repräsentationen von auditiven Ereignissen (z.B. Alarmglocke) in den entsprechenden Teil des Arbeitsgedächtnisses. Diese Repräsentationen verblassen nach einer bestimmten Zeit.

Taktiler Prozessor: Der taktile Prozessor verarbeitet Informationen über Bewegungs-Feedback von Effektor-Organen, die für die Koordination bei parallelen Aufgaben wichtig sein können. Er ist sehr einfach aufgebaut und reicht entsprechende Symbole aus der Aufgabenumgebung verzögert an den entsprechenden Teil des Arbeitsgedächtnisses weiter.

Motorische Prozessoren: Der kognitive Prozessor sendet ein Kommando, das aus dem symbolischen Namen für den gewünschten Bewegungstyp und dessen relevanten Parametern besteht, an einen Motorprozessor, der darauf mit einer entsprechenden Zeitcharakteristik eine Bewegung in der simulierten Aufgabenumwelt ausführt. Der Zustand eines Prozessors, also ob gerade eine Bewegung ausgeführt wird oder nicht und in welcher Phase der Bewegungsausführung er sich befindet, wird im Arbeitsgedächtnis repräsentiert. Es gibt Motorprozessoren für die Bewegung des Auges, der Hände und zum Sprechen, die alle parallel arbeiten (s. Abbildung 3-1).

Manueller Motorprozessor: Der manuelle Motorprozessor beinhaltet unterschiedliche Bewegungs-*styles*, wie z.B. das Drücken einer Taste, wenn sie unter dem Finger ist, Bewegungen der Hand an eine bestimmte Stelle und das horizontale Bewegen über einer Tastatur.

Okularer Motorprozessor: Die Aufgabe des okularen Motorprozessors ist die Bewegung des Auges. Es werden Saccaden zu einem visuellen Objekt erzeugt. Blickbewegungen können entweder gewollt sein oder ungewollt passieren. Ungewollte Blickbewegungen können bewirken, dass ein Objekt zur Mitte der Fovea zentriert wird oder Reflexbewegungen auslösen. Gewollte Augenbewegungen werden vom kognitiven Prozessor veranlasst. Dazu können folgende Kommandos an den okularen Motorprozessor gesandt werden: *move*, *fixate*, *preposition*, *prepare*, *enable*/*disable centering*/*reflex*.

Vokaler Motorprozessor: EPIC verwendet ein einfaches System, um Sprechen zu simulieren. Äußerungen sind Phrasen, die durch Symbole repräsentiert werden.

Aufgabenumgebung: Ein implementiertes EPIC-Modell besteht aus einem LISP-Prozess. Teil dieses LISP-Prozesses ist die simulierte Aufgabenumgebung, die die Reaktionen der Umgebung auf Eingaben und von der Umgebung initiierte Ereignisse verwaltet und erzeugt. Die Aufgabenumgebung interagiert mit den Wahrnehmungs- und Motorprozessen über simulierte Ein-/Ausgabegeräte, die auch als LISP-Funktionen mit entsprechenden Eigenschaften, wie Verzögerungen, implementiert werden müssen. Die eigentliche Interaktion des Benutzermodells mit den simulierten

Ein-/Ausgabegeräten geschieht über den Austausch von vorher definierten Symbolen, die entsprechend interpretiert werden müssen. So wird bei einer Ausgabe eines simulierten Lautsprechers ein Symbol, das den Ton oder einen gesprochenen Satz repräsentieren kann, von der simulierten Aufgabenumgebung übermittelt. Die Lautsprecherfunktion muss die Länge der Darstellung des Symbols kennen und entsprechende Signale, die wiederum als Symbole repräsentiert werden, nach der dazugehörigen Verzögerung kontinuierlich an den auditiven Prozessor senden. Die Funktion, die diesen Wahrnehmungsprozessor implementiert, muss wiederum eine Spezifikation dieser Symbole haben, die entsprechende Aktionen auslösen, um passende Einheiten im entsprechenden Teil des Arbeitsgedächtnisses des kognitiven Prozessors zu speichern.

EPIC ist gedacht als System, mit dem Benutzerverhalten zur vorhersagenden Bewertung neuer Mensch-Maschine-Systeme simuliert werden kann (Kieras et al. 1997). Es dient nur sekundär der psychologischen Modellbildung (Meyer et al. 1995).

ACT-R

ACT-R (Adaptive Control of Thought: Anderson 1993, Anderson & Lebiere 1998) ist eine in der Kognitionswissenschaft verbreitete kognitive Architektur. Sie basiert auf einem Produktionssystem. Die Konfliktresolution und die Repräsentation der Chunks sind besonders elaboriert, um die Eigenschaften der mentalen Beschränkungen in der Informationsverarbeitung abzubilden. Die Theorie und die Implementierungsumgebung der kognitiven Architektur ACT-R werden kontinuierlich weiterentwickelt. Alle Elemente der Architektur sind empirisch abgesichert. Eine Abwärtskompatibilität für Modelle zwischen unterschiedlichen Versionen der Architektur wird jedoch nicht angestrebt.

In ACT-R wird zwischen einer symbolischen und einer subsymbolischen Schicht unterschieden. Die symbolische Schicht wird durch Chunks und Produktionen beschrieben. Auf der subsymbolischen Schicht wird die Verwendung von Chunks und Produktionen benutzt, um Parameter anzupassen, die den Konfliktresolutionsmechanismus beeinflussen.

In ACT-R werden Chunks als Datenstrukturen mit Namen, Typ und Slots im Arbeitsgedächtnis gespeichert. Slots haben einen Namen. Sie enthalten Symbole, Zahlen oder Referenzen auf andere Chunks. Durch Referenzierung zwischen Chunks wird ein semantisches Netz aufgebaut. Die Semantik der Relationen ergibt sich aus dem Namen der Slots, die die Referenzen beinhalten. Der Typ eines Chunks bestimmt die Slots, die er hat. Dazu gibt es Chunk-Typ-Vereinbarungen, in denen Slots spezifiziert werden. Wie in der objekt-orientierten Programmierung stehen Chunk-Typen in einer singulären hierarchischen Vererbungshierarchie. Speziellere Chunk-Typen erben alle Slots ihres Eltern-Typs. Die Verwendung der Generalisierung ist bei Tests auf den Typ eines Chunks möglich.

```
CHUNK44                CHUNK42                  CHUNK99
   ISA SUCCESSOR          ISA Addition             ISA type
   NUMBER 3               ADDEND1 3                SLOT1 CHUNK44
   SUCCESSOR 4            ADDEND2 1                SLOT2 'symbol
                          SUMME NIL
```

Quelltext 3-1: Notation von Chunks in ACT-R

Das Beispiel in Quelltext 3-1 zeigt drei Chunks `CHUNK44`, `CHUNK42` und `CHUNK99`. `CHUNK44` ist vom Typ `SUCCESSOR`, `CHUNK42` vom Typ `Addition` und `CHUNK99` vom Typ `type`. `CHUNK44` hat zwei Slots: `NUMBER` mit dem numerischen Wert `3` und `SUCCESSOR` mit dem numerischen Typ `4`. Mit diesem Chunk wird in diesem Beispiel der Fakt repräsentiert, dass 4 der Nachfolger von 3 ist. `CHUNK42` hat drei Slots. Zwei mit numerischen Werten, und `SUMME` hat den Wert NIL, was ausdrückt, dass hier (noch) kein Wert zugewiesen wurde. Dieser Chunk repräsentiert in diesem Beispiel die Additionsaufgabe „3+1“, die hier noch nicht gelöst wurde. `SLOT1` von `CHUNK99` hat eine Referenz auf `CHUNK44` als Wert, und `SLOT2` hat ein Symbol als Wert.

Ein herausgehobener Chunk im Arbeitsgedächtnis hat eine spezielle Rolle. Mit ihm wird das aktuelle Ziel repräsentiert. Das Ziel steuert die Informationsverarbeitung. Welcher Chunk das aktuelle Ziel repräsentiert, kann mit Produktionen geändert werden. Bis ACT-R 5 gibt es daneben eine hierarchische Zielstruktur, die mit dem Zielstapel (*Goalstack*) repräsentiert wird. Der Goalstack dient dazu, zukünftige Ziele zu verwalten. Er wird verwendet, um die Teilzielbildung zu unterstützen. Dabei wird das aktuelle Ziel in mehrere Teilziele zerlegt, die auf dem Zielstapel gespeichert werden („*push*“). Sie liegen damit über dem aktuellen Ziel auf dem Goalstack. Das oberste Ziel ist jeweils das aktuelle Ziel. Wenn dieses Ziel abgearbeitet ist, also das Teilproblem, das es repräsentiert, gelöst wurde, wird dieses Ziel vom Zielstapel entfernt („*pop*“) und das darunterliegende wird das aktuelle Ziel. Mit der Stapel-Struktur kann eine Tiefensuche im Zielraum mit Backtracking realisiert werden.

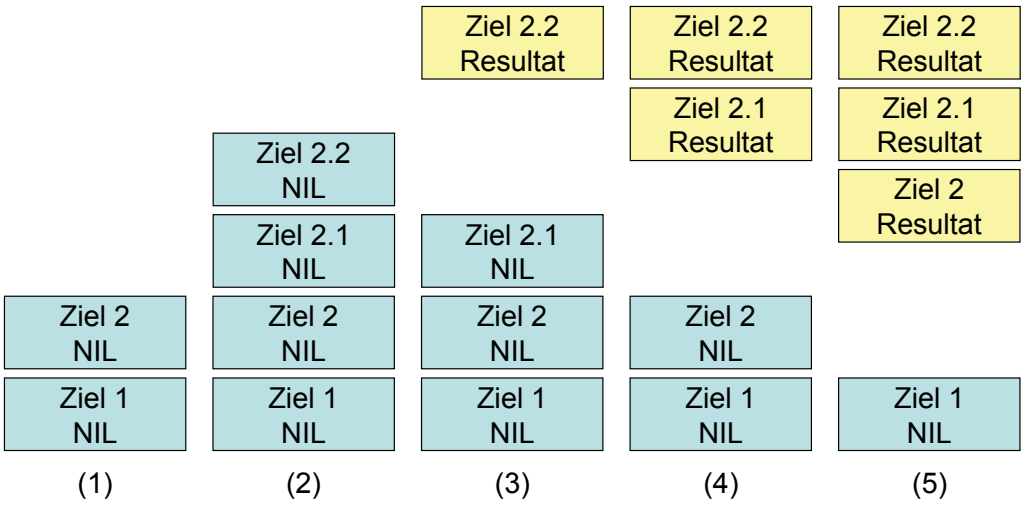


Abbildung 3-2: Beispiel für Tiefensuche mit dem Goalstack in ACT-R

In Abbildung 3-2 ist ein Beispiel für die zielgerichtete Verarbeitung mit dem Goalstack dargestellt. Es wird jeweils der Zielstapel zu fünf nachfolgenden Zeitpunkten (1-5) gezeigt. Die Chunks, die noch ungelöste Ziele repräsentieren, haben „NIL“ als Markierung, um zu verdeutlichen, dass sie einen Slot haben, in dem später das Ergebnis gespeichert wird. Chunks, die gelöste Teilziele repräsentieren, sind nicht mehr auf dem Zielstapel, aber noch im Arbeitsgedächtnis (in der Abbildung oberhalb der Zielstapel). Sie sind mit „Resultat“ markiert, um auszudrücken, dass ihre Slots die Teilproblemlösung enthalten.

1) Im ersten Zeitpunkt liegen auf dem Goalstack die beiden Ziele „Ziel 1“ und „Ziel 2“. „Ziel 2“ ist das aktuelle Ziel, denn es liegt oben auf dem Zielstapel. Bei der Verarbeitung von „Ziel 2“ stellt sicher heraus, dass das Problem, das von dem Ziel repräsentiert wird, in zwei kleinere Teilprobleme zerlegt werden muss. Dafür werden zwei neue Chunks gebildet, die als Ziel interpretiert, diese Teilprobleme lösen sollen. Beide Chunks „Ziel 2.1“ und „Ziel 2.2“ werden auf dem Goalstack abgelegt.
2) Damit wird das nun oberste Element auf dem Zielstapel „Ziel 2.2“ das aktuelle Ziel. Es wird bearbeitet und im Folgenden das Teilproblem gelöst. Die Lösung wird im Ziel-Chunk vermerkt.
3) Damit kann „Ziel 2.2“ vom Goalstack entfernt werden. „Ziel 2.1“ wird das aktuelle Ziel. Es wird bearbeitet und das Teilproblem gelöst.
4) Dann wird das gelöste „Ziel 2.1“ vom Zielstapel entfernt und das ursprüngliche „Ziel 2“ wird wieder aktuelles Ziel. Nun kann aufgrund des Arbeitsgedächtnisses, das nun die erarbeiteten Teillösungen in den Chunks „Ziel 2.1“ und „Ziel 2.2“ enthält, das übergeordnete Problem zu „Ziel 2“ gelöst werden. Das „Ziel 2“ enthält nun die Problemlösung und wird vom Zielstapel entfernt.
5) Somit kommt das nachfolgende Problem mit dem „Ziel 1“ in der Informationsverarbeitung an die Reihe.

Produktionen testen im Bedingungsteil die Existenz von Chunks im Arbeitsgedächtnis. Dafür müssen Chunks anhand ihres Typs (auch generalisiert) und durch die Werte ihrer Slots spezifiziert werden. Es müssen nicht alle Slots spezifiziert werden. Daneben sind spezielle Tests möglich (Negation, bis Version 3 auch größer als, kleiner als bei numerischen Werten und Listenoperationen bei Listenwerten). Slotwerte können an Variablen gebunden werden und in folgenden Spezifikationen und im Aktionsteil verwendet werden. Der erste Test im Bedingungsteil bezieht sich immer auf das aktuelle Ziel. Optional kann ein ACT-R eigener Mechanismus zum partiellen Matching von Chunks verwendet werden. Hier führen auch nur teilweise zutreffende Spezifikationen von Chunks zu einem erfolgreichen Abruf.

Im Aktionsteil werden Chunks im Arbeitsgedächtnis geändert. Änderungen beziehen sich auf Slots. Die geänderten Werte können konstant vorgegebene Symbole, Referenzen oder numerische Werte sein oder aus im Bedingungsteil

gebundenen Variablen abgerufen werden. In diesem Fall werden Slotwerte von einem Chunk in einen anderen kopiert. Es können auch Referenzen auf Chunks an Variablen gebunden und damit im Aktionsteil in Slots von zu modifizierenden Chunks geschrieben werden. Es können auch neue Chunks erzeugt oder existierende gelöscht werden. Außerdem kann das aktuelle Ziel umgesetzt werden oder der Goalstack modifiziert werden (*pop*, *push*).

```
(P PlusNull                          (P PlusEins
   =GOAL>                               =GOAL>
     ISA Addition                         ISA Addition
     ADDEND1 =num                         ADDEND1 =num1
     ADDEND2 0                            ADDEND2 =num2
     SUMME NIL                            SUMME NIL
 ==>                                      =RETRIEVAL1>
   =GOAL>                                   ISA SUCCESSOR
     SUMME =num                             NUMBER =num1
)                                           SUCCESSOR =val1
                                         =RETRIEVAL2>
                                           ISA SUCCESSOR
                                           NUMBER =val2
                                           SUCCESSOR =num2
                                     ==>
                                         =GOAL>
                                           ADDEND1 =val1
                                           ADDEND2 =val2
                                     )
```

Quelltext 3-2: ACT-R Produktionen für eine einfache rekursive Additionsstrategie

Das Beispiel in Quelltext 3-2 zeigt die Informationsverarbeitung mit Produktionen in ACT-R. Es gibt zwei Produktionen `PlusNull` und `PlusEins`. Beide Produktionen können benutzt werden, wenn das aktuelle Ziel vom Typ `Addition` ist. Die Produktion `PlusNull` wird benutzt, wenn der Slot `ADDEND2` des aktuellen Additionsziels `0` ist. In diesem Fall ist das Ergebnis gleich `Addend1`. Um diesen Wert zu übertragen wird er im Bedingungsteil an die Variable `=num` gebunden. An die Variable `=GOAL` wird das aktuelle Ziel gebunden, um es im Aktionsteil modifizieren zu können. Im anderen Fall, wenn die Produktion `PlusEins` benutzt wird, werden neben dem aktuellen Ziel zwei Chunks aus dem Arbeitsgedächtnis abgerufen, um Fakten über Nachfolger bei natürlichen Zahlen zu benutzen. Die rekursive Strategie, die damit realisiert wird, ist `Addend1` in jedem Schritt um eins hochzuzählen und `Addend2` dabei um eins zu verkleinern. Der Rekursionsanker ist der Fall, wenn `Addend2` null ist. In diesem Fall wird die Produktion `PlusNull` feuern.

Neben diesen theoriegebundenen Konstrukten bietet die Implementierungsumgebung von ACT-R auch Zugriff auf das „Wirtssystem“ CommonLISP. Daher kann

LISP-Code ausgeführt werden, um komplexe Bedingungen zu formulieren oder Seiteneffekte im Aktionsteil auszulösen.

Die wesentlichen subsymbolischen Parameter sind die Aktivierung von Chunks und die Stärke von Produktionen. Sie werden während der Verarbeitung aufgrund des Zugriffes auf Chunks und Produktionen angepasst. Die Aktivierung eines Chunks gibt an, mit welcher Wahrscheinlichkeit er im nächsten Produktionszyklus in einer Produktionsinstanz verwendet werden wird. Analog gibt die Stärke einer Produktion an, mit welcher Wahrscheinlichkeit sie im nächsten Zyklus feuern wird.

Die Chunk-Aktivierung wird im semantischen Netz des Arbeitsgedächtnisses propagiert. Sie setzt sich daher aus den folgenden beiden Komponenten zusammen: Die *Base-Level Activation* drückt die Relevanz des Chunks an sich aus. Die *Context Activation* bezieht sich auf den Zusammenhang mit anderen Chunks – insbesondere in Hinblick auf das gerade gegebene Ziel. Der aktuelle Kontext wird vom aktuellen Ziel und den Chunks gebildet, die vom aktuellen Ziel referenziert werden. Alle Chunks des aktuellen Kontextes sind Quelle von Aktivierung für alle sie referenzierenden Chunks. Diese *Context Activation* wird zur *Base Level Activation* jedes Slots addiert.

In der Formel 3-1 steht A_i für die Aktivierung des Chunks i. B_i ist dessen *Base Level Activation*. Die Context Activation wird in der Summe über alle Chunks j des Kontextes ausgedrückt. Dabei wird die so zu vergebende Aktivierung (ein ACT-R-Parameter) über die Anzahl der Chunks im Kontext normiert. Dieser Wert W_j wird mit der Interassoziationsstärke S_{ji} zwischen den Chunks j und i gewichtet. Die Interassoziationsstärke ist ein numerischer Parameter des semantischen Netzwerkes.

$$A_i = B_i + \sum_j W_j \times S_{ji} \qquad (3\text{-}1)$$

Das Beispiel in Abbildung 3-3 zeigt die Bildung des aktuellen Kontextes und die Propagation von *Context Activation*. Der aktuelle Kontext wird durch das aktuelle Ziel bestimmt. Im Beispiel ist das aktuelle Ziel ein Chunk vom Typ Multiplikation, der die beiden Faktoren und das Ergebnis als Slots hat. Im gegebenen Fall werden die Chunks `drei` und `zwei` als Faktoren referenziert. Das Ergebnis ist noch unbekannt. Damit hat der Ergebnis-Slot den Wert `NIL`.

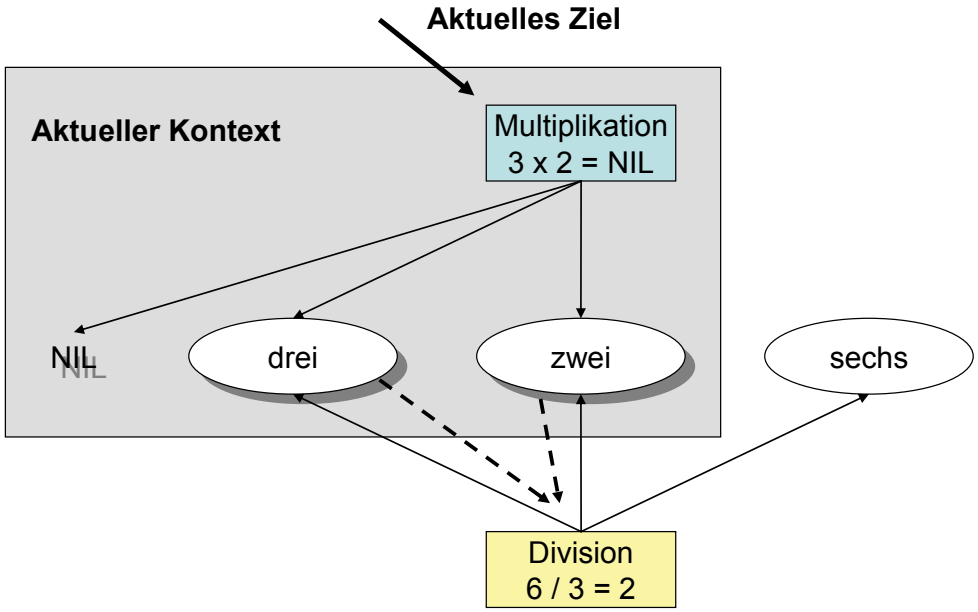


Abbildung 3-3: Beispiel für den aktuellen Kontext als Quelle von Aktivierung

Die Chunks `drei` und `zwei` des aktuellen Kontextes sind Quelle von Kontextaktivierung für den Ziel-Chunk „6 / 3 = 2“ (gestrichelte Pfeile), der ein bereits gelöstes Ziel bzw. einen Fakt repräsentiert, denn er referenziert die beiden Chunks `drei` und `zwei`. Tatsächlich lässt sich das Multiplikationsproblem leicht lösen, wenn das Ergebnis des Divisionsproblems bekannt ist.

Die Aktivierung beeinflusst in der ACT-R Theorie, wie schnell ein Chunk aus dem Arbeitsgedächtnis abgerufen werden kann. Die Produktionsinstanzenbildung erfolgt in ACT-R konzeptionell parallel. Die erste Produktionsinstanz, die bei der Konfliktresolution zur Verfügung steht, feuert. Daher hat die Aktivierung von Chunks einen großen Einfluss auf ihre Anwendung in Produktionen.

Der Konfliktresolutionsmechanismus berücksichtigt neben der Aktivierung noch den erwarteten Gewinn einer Produktion. Der Gewinn errechnet sich aus mehreren Parametern. Der Ausdruck 3-2 beinhaltet die Berechnungsvorschrift für den erwarteten Gewinn der Produktion p.

$$P_p \times \; ? - \; ?_p \qquad (3\text{-}2)$$

P_p ist die erwartete Wahrscheinlichkeit, dass die Produktion p für das aktuelle Ziel benutzt wird und feuert. G ist ein Maß für den Wert des Ziels. C_p ist ein Maß für die Kosten das aktuelle Ziel mit der Produktion p zu erreichen (die Zeit, die zur Bildung der feuernden Produktionsinstanz erforderlich ist). Alle Werte werden als Zeit ausgedrückt. Die Wahrscheinlichkeit ergibt sich aus der vergangenen Verarbeitung des Produktionssystems.

Allgemein kann man erkennen, dass die Kosten einer Produktion umso höher sind, je komplexer ihre Bedingung ist, da dann mehr Zeit zum Abruf der Chunks erforderlich ist und die Produktionsinstanz erst später gebildet werden kann.

Das Arbeitsgedächtnis von ACT-R kennt keinen direkten Vergessensmechanismus. Entweder werden Produktionen benutzt, um explizit im Aktionsteil Chunks zu löschen oder die Aktivierung wird interpretiert, um den zugreifbaren Teil des Arbeitsgedächtnisses zu identifizieren. Ein Zugriff auf Chunks ist dann nur noch über einer Aktivierungsschwelle möglich. Um diesen Mechanismus zu nutzen, muss die *Base Level Activation* von Chunks durch Lernen anhand von Formel 3-3 angepasst werden.

$$B_i(t) = \log \sum_{j=1}^{n} (t - t_j)^{-d} \tag{3-3}$$

Im Prinzip wird die *Base Level Activation* B_i von Chunk i bei jedem erfolgreichen Zugriff j auf den Chunk erhöht. t_j ist der Zeitpunkt des erfolgreichen Zugriffs, so dass weiter zurückliegende Zugriffe für die *Base Level Activation* zum aktuellen Zeitpunkt einen geringeren Einfluss haben als kürzlich zurückliegende.

Neben dem Lernen als Anpassung subsymbolischer Parameter bietet ACT-R auch Lernmechanismen auf der symbolischen Ebene. Dabei werden neue Produktionen erzeugt („*production compilation*“: Taatgen & Lee 2003), die den erfolgreichen Gebrauch mehrerer Chunks in einem Zusammenhang codieren und so in der Zukunft schneller abrufbar machen.

ACT-R liegt in mehreren Versionen als Theorie und Modellierungsumgebung vor. Im Rahmen dieser Arbeit wurden Fallstudien mit den Versionen 3, 4 und 5 durchgeführt. Die Unterschiede zwischen diesen Versionen werden hier kurz erörtert.

In ACT-R 3 gibt es keine Beschränkungen über die Art und Struktur von Produktionen. Im Bedingungsteil sind beliebig komplexe Abfragen des Arbeitsgedächtnisses möglich und im Aktionsteil können auch Slots von Chunks geändert werden, die nicht aktuelles Ziel sind. Als Slot-Werte sind neben numerischen Werten, Symbolen und Chunk-Referenzen auch Listen erlaubt. Dementsprechend gibt es Prädikate, um Listeneigenschaften zu testen.

In ACT-R 4 wurden diese Freiheiten eingeschränkt, um eine bessere Prädiktionskraft von Modellen zu erreichen. Abbildung 3-4 zeigt das in ACT-R 4 umgesetzte Verarbeitungsprinzip. Die ganze Verarbeitung gruppiert sich um das aktuelle Ziel. Nur hier können Slots geändert werden. Es kann nur ein Chunk aus dem Arbeitsgedächtnis abgerufen werden, um ihn in die Verarbeitung des Ziels einzubeziehen. Neue Chunks im Arbeitsgedächtnis können nur abgearbeitete Ziele werden.

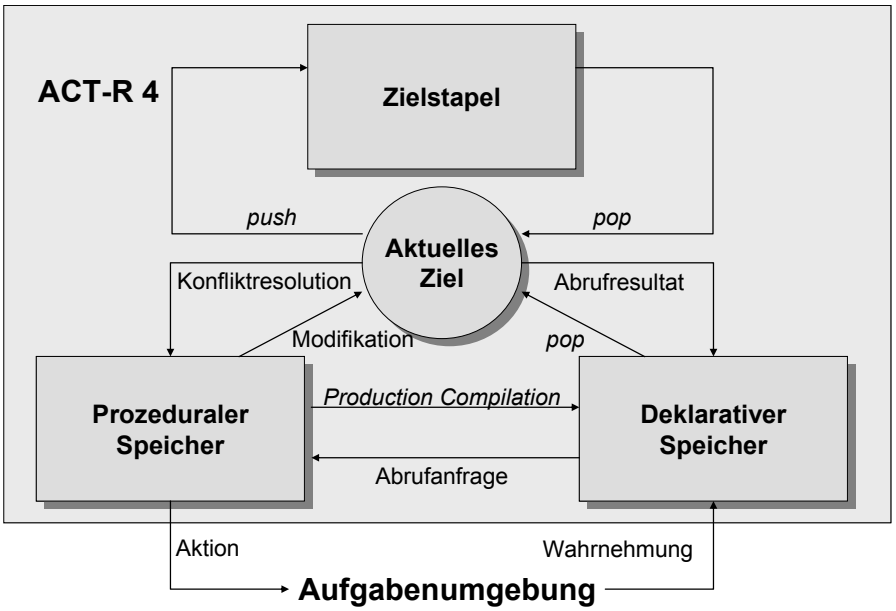


Abbildung 3-4: Architektur und Kontrollfluss in ACT-R 4 (nach Anderson & Lebiere 1998)

Im Umfeld von ACT-R 4, aber nicht als integraler Bestandteil, wurde die Erweiterung PM (*Perceptual Motor,* Byrne & Anderson 1998) veröffentlicht. Angelehnt an die Architektur von EPIC wird hier erstmals für ACT-R eine Schnittstelle zur Ankopplung an externe Aufgabenumgebungen auf der Ebene von Wahrnehmung und motorischer Handlung angeboten.

In ACT-R 5 wurde PM in den Standardumfang der Architektur integriert. Dafür wurde ein Buffer-Konzept eingeführt, das parallel arbeitende Verarbeitungsmodule voneinander trennt. Die Buffer kommunizieren über Kommandos miteinander. So ist nun auch der Abruf aus dem Arbeitsgedächtnis über Buffer-Kommandos nötig. In ACT-R 5 wurde darüber hinaus der Goalstack fallengelassen, um eine anwendungsspezifische und flexible Zielverarbeitung zu unterstützen.

Soar

Soar (State, Operator, Action, and Result: Laird et al. 1987, Newell et al 1993) ist eine kognitive Architektur, bei der Wissensverarbeitung auf Suche in Problemräumen mit einem Produktionssystem abgebildet wird. Das Konzept der Problemräume beschreibt Problemlösen als Transformation von Zuständen, die Situationen beschreiben. Transformationen geschehen über die Anwendung von Operatoren, die Situationselemente modifizieren. Ausgehend von einem Startzustand wird die Kombination von Operatoranwendungen gesucht, die zu einem von mehreren gewünschten Zielzuständen führen. Diese kombinatorische Aufgabe wird durch unterschiedliche heuristische Suchverfahren gelöst.

Auch in Soar wird wie in ACT-R zwischen deklarativem Wissen (Chunks im Arbeitsgedächtnis) und prozeduralem Wissen in Form von Produktionen unterschieden. Der Zustand ist dabei charakterisiert durch Chunks, die als Objekte

mit Attribut/Wert-Paaren im Arbeitsgedächtnis repräsentiert werden, und Operatoren, die auch durch Objekte repräsentiert werden.

Der Arbeitsspeicher mit den Chunks ist in Standard-Soar nicht beschränkt, Vergessen wird nicht modelliert. Eine unterschiedliche Aktivierung der Chunks ist nicht vorgesehen.

In Soar wird prozedurales Wissen in Form von Produktionen repräsentiert. Mit Produktionen werden dabei sowohl die Operatoranwendung als auch interne Verarbeitungsschritte umgesetzt. Im Aktionsteil von Produktionen kann nicht nur das deklarative Wissen im Arbeitsgedächtnis geändert werden, sondern auch die Verarbeitungskontrolle beeinflusst werden. Die Transformation eines Ausgangszustandes in einen Folgezustand geschieht in fünf Schritten:

- *proposal:* Mit Produktionen, deren Bedingungsteil in der aktuellen Situation erfüllt ist, werden Operatoren zur potenziellen Ausführung markiert.
- *comparison:* Mit spezialisierten Produktionen wird bewertet, wie nützlich die vorgeschlagenen Operatoren in der aktuellen Situation sind. Dazu werden Präferenzen für die einzelnen Operatoren vergeben („*acceptable*“, „*reject*“, „*better/worse*“, „*best*“, „*worst*“, „*indifferent*“, „*require*“, „*prohibit*“).
- *selection:* Die Architektur wählt aufgrund der beiden vorangegangenen Schritte einen Operator aus, der ausgeführt werden soll.
- *application:* Der ausgewählte Operator wird ausgeführt, indem Produktionen angewendet werden. Dadurch wird das Arbeitsgedächtnis geändert oder eine Aktion in der externen Aufgabenumgebung ausgeführt.
- *termination:* Mit spezialisierten Produktionen wird festgestellt, ob die Anwendung des ausgeführten Operators beendet ist.

Dieses Schema beinhaltet eine modellspezifische Form der Konfliktresolution. Außerdem ist sowohl eine parallele Ausführung von Produktionen (zum Operatorvorschlag und -bewertung) als auch eine sequentielle Ausführung der Operatoren möglich. Kontroll- und anderes Meta-Wissen kann in den *comparison*-Produktionen spezifiziert werden.

Wenn in einer Situation Wissen zur Auswahl eines geeigneten Operators fehlt und deshalb der Vorschlag eines Operators misslingt, wird in Soar ein neuer Problemraum zur Erzeugung des fehlenden Wissens kreiert. Durch die Lösung eines solchen Teilproblems wird neues Wissen über die aktuelle Situation erzeugt. Wenn ein so erzeugtes Unterproblem gelöst wurde, kann zum ursprünglichen Problemraum zurückgekehrt werden. Dort steht dann für die weitere Bearbeitung neues situationsspezifisches Wissen zur Verfügung, mit dem ein Operator ausgewählt werden kann. Es gibt vier unterschiedliche Situationen, in denen ein neuer Problemraum angelegt wird („*impass*“):

- Es wurde kein Operator vorgeschlagen.

- Es wurden Operatoren vorgeschlagen, aber es kann keiner ausgewählt werden, weil keine Bewertungsproduktion feuern kann.
- Es wurden Operatoren vorgeschlagen und mehrere werden gleich bewertet (keine eindeutige Präferenz eines Operators).
- Es wurden Operatoren vorgeschlagen und einer ausgewählt, aber es gibt keine anwendbare Produktion zu seiner Umsetzung.

Jeder Problemraum hat einen Anfangszustand und mindestens einen Zielzustand. Das Ziel im neuen Problemraum ist abhängig von der Art der Situation, die zu seiner Bildung geführt hat. Die Unterproblemräume werden auf einem Kontrollstack verwaltet. Es ergibt sich dadurch ein Kontrollschema, das analog zu der Teilzielbildung in ACT-R und dem dortigen Goalstack ist.

In dynamischen Situationen mit Multitasking-Anforderungen kann jedoch nicht genau ein Zielzustand spezifiziert werden. Daher müssen mehrere parallele Ziele in eigenen Problemräumen verwaltet werden (Aasmann 1995).

Soar-Modelle können mit einem einheitlichen Mechanismus („*Chunking*“: Laird et al. 1986) Produktionen lernen. Dabei wird die Lösung eines *Impass* mit der Elaboration im Unter-Problemraum in einer Produktion codiert. Dafür werden Heuristiken benutzt, um signifkante Chunks bzw. Chunk-Eigenschaften der Ausgangssituation (vor dem *Impass*) im Bedingungsteil und der Zielsituation (nach dem *Impass*) im Aktionsteil zu speichern. Durch die neu gelernte Produktion wird eine Abkürzung realisiert, die die Elaboration im Unter-Problemraum spart, indem die neue Produktion das Auftreten des *Impass* verhindert.

In Soar ist eine Interaktion mit der Aufgabenumgebung möglich. Zum Einen gibt es ein API, das über C bzw. mit SWIG-Bindings angesprochen werden kann. Per SWIG können beliebige externe Programmumgebungen angebunden werden. Verbreitet ist die Kopplung von Soar und Tcl (TclSoar). Daneben steht mit SML (Soar Community o.J.) eine XML-basierte verallgemeinerte Schnittstelle zur Kopplung mit externen Aufgabenumgebungen zur Verfügung. Sie stellt einen Inputlink und einen Outputlink als Chunks im Arbeitsgedächtnis zur Verfügung. Mit API und SML sind jedoch keine Wahrnehmungs- oder Motorikmechanismen verbunden, sondern es handelt sich um reine Datenaustauschschnittstellen.

COGNET/iGEN

COGNET ist eine kognitive Architektur mit Simulationsumgebung, die mit dem Ziel einer praktischen Anwendung entwickelt wurde (Zachary et al. 1997, Zachary et al. 2005). Dazu werden insbesondere leistungsbeschränkende Faktoren in kognitiven Agenten modelliert und simuliert. Das sind physiologische Faktoren (z.B. beschränkte Reichweiten, Stärke und Fähigkeiten der Beweglichkeit), Wahrnehmungsfaktoren (Beschränkungen der Sinneswahrnehmungen), mentale Faktoren (Gedächtniseigenschaften wie Vergessen und fehlerhafte Entscheidungen

unter Zeitdruck) und psychologische Faktoren wie Stress, Ermüdung und sogar Angst) (nach iGEN *User's Reference Guide*).

COGNET basiert wie EPIC auf einer Architektur mit Wahrnehmung-, Motor- und kognitivem Prozessor die auf einem Symbolspeicher als Repräsentation von Gedächtnisstrukturen mit einem Produktionssystem arbeiten. Wahrnehmung-, Motor- und kognitiver Prozessor arbeiten voll parallel. Wahrnehmungs- und kognitiver Prozessor haben unabhängig voneinander Zugriff auf dieselben Gedächtnisstrukturen. Beschränkungen menschlicher kognitiver Leistungen werden in COGNET durch begrenzte sensumotorische Ressourcen und Metakognition umgesetzt. iGEN ist eine integrierte Entwicklungsumgebung für das Erstellen, Debuggen und Integrieren von COGNET-Modellen.

COGNET-Modelle beinhalten im Symbolspeicher deklarative Wissensstrukturen. Das sind Chunks auf einem Blackboard – einer Struktur, die Fakten speichert und von den Verarbeitungskomponenten der kognitiven Architektur parallel zugegriffen werden kann.

Prozedurales Wissen wird vom kognitiven Prozessor ausgeführt. Die Regeln, die das prozedurale Wissen ausmachen, modifizieren Chunks auf dem Blackboard und können Aktionen in der externen Aufgabenumgebung durch Kommandos an den Motorprozessor auslösen.

Der Interpreter BATON (*the Blackboard Architecture for Task-Oriented Networking*) ist die Regelmaschine, die das prozedurale Wissen anwendet, die Regeln ausführt und die Aktionen in den anderen Komponenten der kognitiven Architektur initiiert. Mit dieser Verarbeitungskomponente wird Informationsverarbeitung und logisches Schließen in COGNET-Modellen simuliert.

Neben dem prozeduralen und dem deklarativen Wissen gibt es weitere Expertisestrukturen in COGNET-Modellen. Für die metakognitive Verarbeitung werden eigene Chunks auf einem getrennten Blackboard und Produktionsregeln benutzt. Für den Motorprozessor wird Wissen über Aktionen gespeichert.

Die prozedurale Verarbeitung geschieht mit Strategien anhand von „Goal – subgoal – operator"-Strukturen. Die Abarbeitung einer Strategie ist abhängig vom gegenwärtigen Ziel, das den gewünschten Endzustand der Aufgabenbearbeitung repräsentiert. Mit COGNET/iGEN können wie in EPIC Multitasking-Aufgaben modelliert werden. Dazu bietet COGNET/iGEN ein Konzept von nebeneinander gültigen Aufgaben (*cognitive tasks*), die mit Produktionsregeln nachgebildet werden. Folgende Eigenschaften machen diese Aufgaben aus:

- Aufmerksamkeitsfokus: Der kognitive Prozessor ist zu jedem Zeitpunkt mit höchstens einem Chunk, der aktiven Aufgabe, beschäftigt.
- Musterbasierte Aufmerksamkeitskontrolle: Jeder Aufgabe ist ein Informationsmuster im Gedächtnis-Speicher zugeordnet. Das Vorhandensein solch eines Musters triggert die Aufgabe und gibt ihr den Aufmerksamkeitsfokus.

- Jede Aufgabe berechnet dynamisch ein Maß ihrer relativen Priorität anhand der aktuellen Situation. Die Aufgabe mit der höchsten Wichtigkeit wird aktiv.
- Aktive Aufgaben können aus dem Aufmerksamkeitsfokus geraten, wenn eine wichtigere Aufgabe zur Bearbeitung ansteht.
- Aufgaben ändern den Gedächtnisspeicher mit Operatoren.
- *Dämonen* sind Produktionsregeln, die durch Wahrnehmungen getriggert werden. Sie ändern den Gedächtnisspeicher wie Aufgaben.
- Aufgaben können wie Produktionen mehrere Instanzen haben, wenn ihr zugeordnetes Muster mit unterschiedlichen Chunks im Gedächtnisspeicher erfüllt werden kann.
- Aufgaben können anhand von Bedingungen über Muster im Speicher entscheiden, selbst inaktiv zu werden. Dann wird eine andere Aufgabe aktiviert.

Modelle in der COGNET-Architektur haben also als Bestimmungselemente Aufgaben, Aktionen, das Blackboard, Dämonen und Methoden. In Abbildung 3-5 ist ein COGNET-Modellausschnitt in iGEN, der integrierten Entwicklungsumgebung zu COGNET, mit diesen Elementen gezeigt. Das Modell simuliert die Kontrolle von Flugbewegungen in einer simulierten Flugsicherungsaufgabe aus dem Vorhaben AMBR (Modell von Zachary et al. 2005, eigener Screenshot). Die parallelen Aufgaben sind hier `accept_incoming_aircraft`, `welcome_incoming_aircraft` etc. Jede Aufgabe hat einen Trigger, der sie in einer neu gebildeten Instanz aktiviert. Dämonen wie `update_ac` sorgen für eine Abbildung der wahrgenommenen Situation in der simulierten Aufgabenumgebung im deklarativen Speicher. So kann die Wahrnehmung eines einfliegenden Flugzeugs einen Dämon triggern, der dies als Ereignis auf dem Blackboard repräsentiert. Das repräsentierte Ereignis triggert dann beispielsweise die Aufgabe `accept_incoming_aircraft` in einer Instanz mit diesem Ereignis. Falls andere wichtigere Aufgaben aktiv werden, wird diese neue Aufgabe zurückgestellt. Die Aufgabe `accept_incoming_aircraft` hat zwei Ziele (`accept_incoming_aircraft` und `update_mental_model`), die mit einer Reihe von Aktionen (`Find_specific_track_on_Display` etc.) erfüllt werden können.

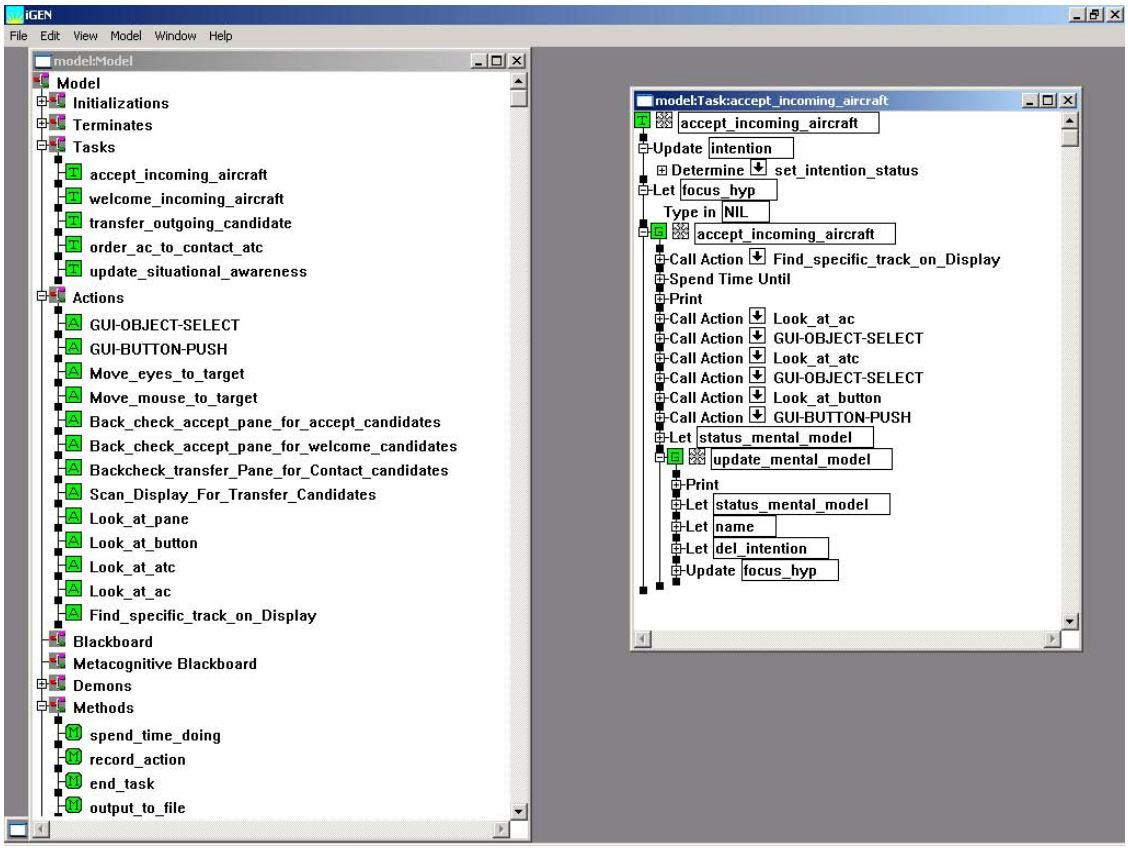


Abbildung 3-5: Screenshot von iGEN mit Ausschnitt aus dem Fluglotsenmodell aus dem AMBR-Vorhaben

iGEN beinhaltet eine Reihe von Hilfsmitteln für die Entwicklung von Modellen und ihre Simulation. So können beispielsweise Teilmodelle für untereinander unabhängige Aufgabenbereiche gebildet werden. COGNET-Modelle können auch in externe Anwendungen eingebunden werden. Dazu existieren C++-Wrapper.

COGNET ist ein geeignetes Werkzeug für die Analyse von Mensch-Maschine-Systemen. So ist z.B. Multitaskingverhalten leicht modellierbar. COGNET hat allerdings im Vergleich mit anderen kognitiven Architekturen einen geringen Anspruch an Allgemeingültigkeit. Es werden nur wenige Bestimmungsstücke einer *unified theory of cognition* modelliert. So enthält der Motorprozessor beispielsweise keine ergonomischen Regeln über Beschränkungen, um einen Effekt in der Umwelt auszulösen. Diese Funktion wird an die Schnittstelle zwischen iGEN und simulierter Aufgabenumgebung delegiert. Es wird hauptsächlich reaktives Verhalten im Gegensatz zu Problemlösen berücksichtigt. Lernmechanismen sind erst seit kurzer Zeit Teil der COGNET-Architektur. Sie wurden für eine spezialisierte Aufgabe im AMBR-Vorhaben eingebracht (Zachary et al. 2005).

Die angestrebte Analyseebene von COGNET-Modellen ist flexibel. Sie liegt im Bereich von Sekunden bis Stunden.

COGENT

COGENT (*cognitive objects in a graphical environment*) ist ein Baukastensystem, mit dem Annahmen über unterschiedliche kognitive Architekturen in Produktionssystemen implementiert werden können (Cooper et al. 1998, Cooper 2002). Vorgänger des Systems ist Sceptic (Cooper & Farringdon, 1993), eine Spezifikationssprache für kognitive Theorien, in der z.B. Soar reformuliert wurde. COGENT beschreibt keine *unified theory of cognition*, sondern definiert viele unterschiedliche Versatzstücke, die in ad hoc Modellen eingesetzt werden können. Diese Versatzstücke können unterschiedlich parametriert und miteinander verbunden werden. COGENT-Modelle bestehen aus einem Netzwerk von solchen „Kognitionsobjekten“, die untereinander über Kanten Nachrichten austauschen (s. Abbildung 3-6).

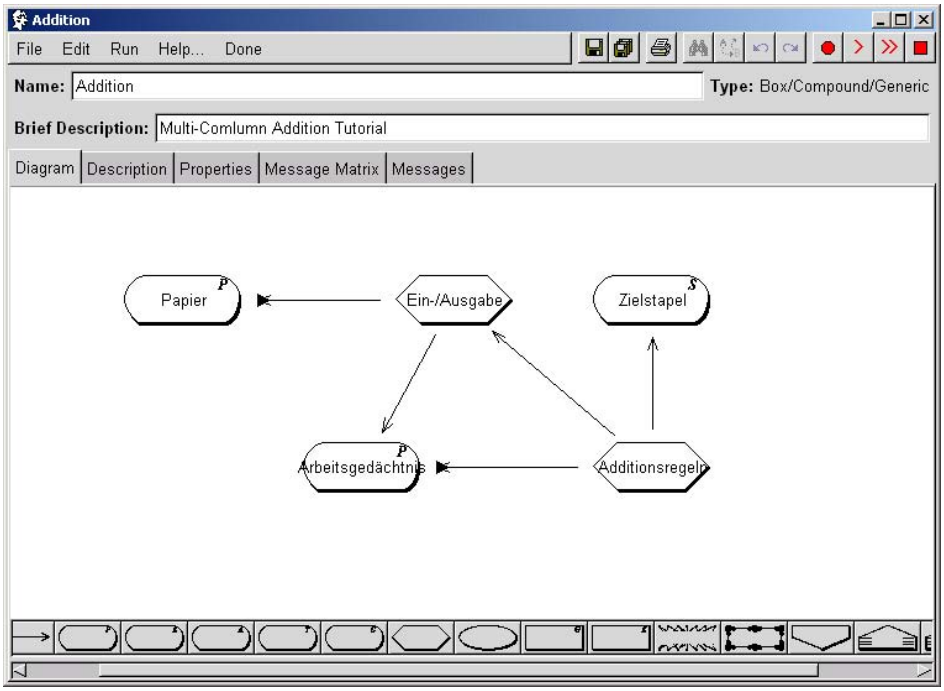


Abbildung 3-6: Beispiel eines COGENT Box/Arrow Modells mit zwei propositionalen Puffern, einem Stack-Puffer und zwei Prozessen

Um einen Nachrichtenfluss zu initiieren, gibt es Datenquellen, in denen eine Liste von Nachrichten steht, die in ihrer Reihenfolge an Boxen gesandt werden, die mit einer Kante zu dieser Quelle verbunden sind. Ebenso gibt es Senken zur Aufnahme von Simulationsergebnissen.

Puffer dienen als Speicher für deklaratives Wissen. Über die Kanten zu solchen Pufferobjekten können Nachrichten zur Speicherung einer neuen deklarativen Einheit gesandt werden. Kanten von Pufferobjekten dienen zum Abruf von gespeicherten Informationen. COGENT bietet als Framework unterschiedliche Ausprägungen von Puffern. Sie können parametriert werden, so dass sie sich unterschiedlich verhalten (Kapazität, Vergessenscharakteristik, Zugriffsmechanismen beim Matching). Deshalb sind COGENT Puffer dazu geeignet, unterschiedliche Typen von Informationen zu speichern. Sie sind sowohl für die kurzfristige Speicherung, wie im

Arbeitsgedächtnis, als auch für große Wissensbasen, wie im Langzeitgedächtnis, passend zu konfigurieren. Propositionale Puffer sind der Speichertyp in COGENT, der am flexibelsten eingesetzt werden kann. Es gibt keine Restriktionen über die Art von Entitäten, die sie enthalten können. Die syntaktischen Formen der Einträge sind Prolog-Prädikate. Sie haben unterschiedliche Eigenschaften, die die Behandlung von Einträgen steuern. Analoge Puffer sind spezialisierte COGENT Puffer, die grafische Repräsentationen enthalten und manipulieren können. Sie können dazu benutzt werden Modelle, die auf bildlichen Prozessen basieren, zu formalisieren.

Die prozedurale Verarbeitung innerhalb eines Modells erfolgt über Prozess-Boxen, die Produktionsregeln enthalten. Innerhalb eines Modells kann es mehrere Prozesse geben. Abhängig von den Eingabewerten, also Nachrichten, die von Kanten in eine Prozess-Box hineingehen, werden Regeln ausgewählt, die wiederum Nachrichten über Kanten, die aus der Prozess-Box hinausweisen, an Folgeboxen senden. Intern werden die Regeln als Prolog-Terme repräsentiert.

COGENT bietet darüber hinaus die Möglichkeit, symbolische und konnektionistische Komponenten in einem Modell zu koppeln. Dazu können Netzwerk-Boxen benutzt werden. Eine Netzwerk-Box repräsentiert die Simulation eines Perzeptron-Netzes, also eines zweischichtigen Künstlichen Neuronalen Netzwerkes, das einen numerischen Eingabevektor auf einen Ausgabevektor abbildet. Solch ein Netzwerkobjekt kann mit der Nachricht `train(InV, OutV)` entweder mit der Hebb'schen oder der Delta-Regel trainiert werden. Als Aktivierungsfunktionen kann entweder eine lineare oder eine Sigmoidal-Funktion benutzt werden.

COGENT ist in erster Linie ein Werkzeug für Kognitionswissenschaftler, die eigene Modelle simulieren wollen. Der Vorteil von COGENT ist, dass eine kognitionsspezifische Anwendungsschicht über Prolog gebildet wird.

OCCS

Operator Cognitive Crew Cognitive Simulation (OCCS) ist ein Framework für die ad hoc Modellierung im Bereich von Leitwarten in prozesstechnischen Anlagen mit der Programmiersprache Prolog. Es wurde am Cognitive Systems Engineering Lab der Universität Tokio entwickelt und im Bereich der Analyse und prospektiven Gestaltung kerntechnischer Anlagen angewendet und validiert. Die zugrundeliegende kognitive Architektur wurde in Prolog implementiert. Sie beschreibt schwerpunktmäßig nichtmonotone Schlussfolgerungsmechanismen und die Verarbeitung unsicheren Wissens. Das *Step Ladder Modell* zur Informationsverarbeitung von Rasmussen (1986) liefert die Grundlage für OCCS (s. Abbildung 3-7, links). In diesem Rahmenmodell wird das Bedienverhalten in Anlagen als gestufter Analyseprozess interpretiert. Von der Wahrnehmung von Anlagenparametern wird auf den Zustand des technischen Prozesses der Anlage geschlossen. In einem weiteren Planungsprozess wird dann ein Aktionsziel identifiziert und ausgeführt. Durch Lernen

sind für manche Aufgaben Abkürzungen in diesen Prozessen möglich. Zielgerichtetes Verhalten wird in OCCS analog zum Rahmenmodell von Rasmussen auf regel- und planbasierte Informationsverarbeitung abgebildet. Die Verarbeitungsprozesse unterscheiden sich jedoch insbesondere in der Analyse, in der Hypothesenbildung und -bestätigung (s. Abbildung 3-7, rechts).

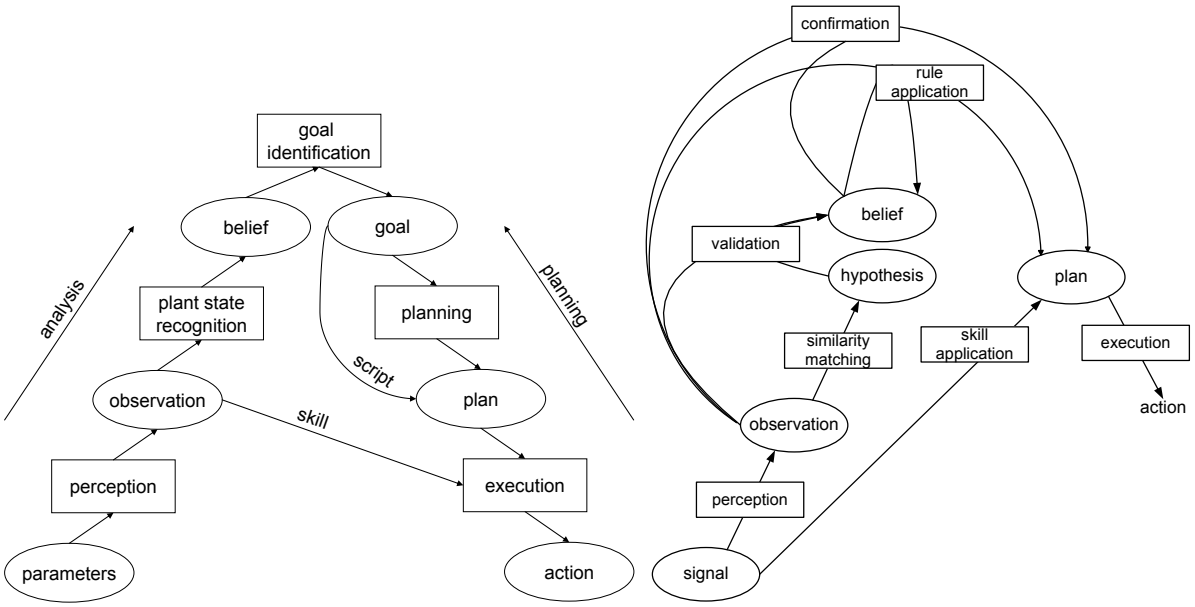


Abbildung 3-7: *Step Ladder Model* (links, nach Rasmussen 1986) und Adaption in OCCS (rechts, nach Furuta & Kondo 1993)

Die Signalerkennung wird durch eine Fuzzy-Komponente realisiert. In den Prolog-Regeln wird mit Symbolen zu linguistischen Variablen der verwendeten Signale und einer numerischen Fuzzy-Zugehörigkeit gearbeitet, die auch als Konfidenzmaß interpretiert werden kann. Die Wahrnehmungskomponente kommuniziert über Interprozesskommunikation mit dem Kraftwerksimulator oder einer anderen Mikrowelt (Kanno et al. 2003). Die anderen Systemfunktionen kommunizieren über ein Blackboard (s. Abbildung 3-8). Die Komponenten der Systemarchitektur von OCCS sind Sensoren, *Planner*, Inferenz und Wirkkomponenten („*Executor“)*. *Inferenzkomponente* und *Planner* benutzen eine anlagenspezifische Wissensbasis. Die Informationsverarbeitung erfolgt regelbasiert. Die eingesetzte Logik ist nichtmonoton, denn sie arbeitet mit Default-Annahmen. Ein Truth-Maintenance-System („TMS“), das mit der zentralen Blackboardkomponente integriert ist, sorgt dafür, dass Folgerungen und Zwischenergebnisse revidiert werden, die auf als falsch erkannten Default-Annahmen oder veralteten Signalen beruhen.

OCCS ist zur prospektiven Simulation von Beanspruchung in vorgegebenen Aufgabensituationen eingesetzt worden. Dazu wurde die Aktivität in den einzelnen Komponenten überwacht und zu einem Profil verdichtet (Abbildung 3-9): VP: *Visual Perception Score* (Zugriff auf visuell wahrnehmbare Anlagenparameter über Sensor-Modul, ohne auditiv wahrnehmbare Signale wie Alarme), IA: *Inference Application*

Score (Aktivität der *Inferenzkomponente*: Regelanwendung, impliziert Aktivität des Arbeitsgedächtnisses), KR: *Knowledge Retrieval Score* (Aktivität von *Inferenzkomponente* oder *Planner*, die sich auf die *Wissensbasis* beziehen), BC: *Belief Creation Score* (Aktivität auf *Blackboard*), TM: *Truth Maintenance Score* (Revision von Zwischenschlüssen und Annahmen, impliziert Aktivitäten im Arbeitsgedächtnis), AS: *Action Scheduling Score* (Aktivität initiiert durch *Planner*, impliziert Beanspruchung des Arbeitsgedächtnisses), PW: *Physical Work Score* (*Executor* führt geplante Prozedur aus, impliziert Aktivität der Wirkkomponenten).

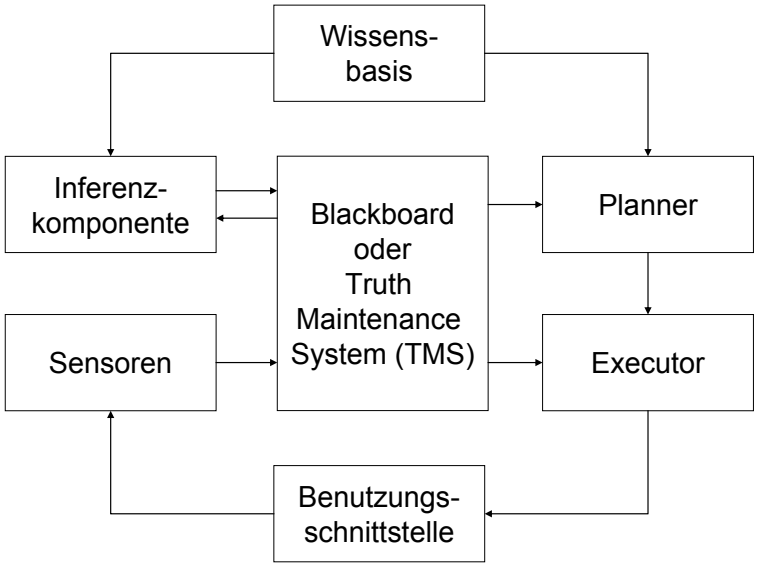


Abbildung 3-8: Blackboard-basierte Systemarchitektur von OCCS

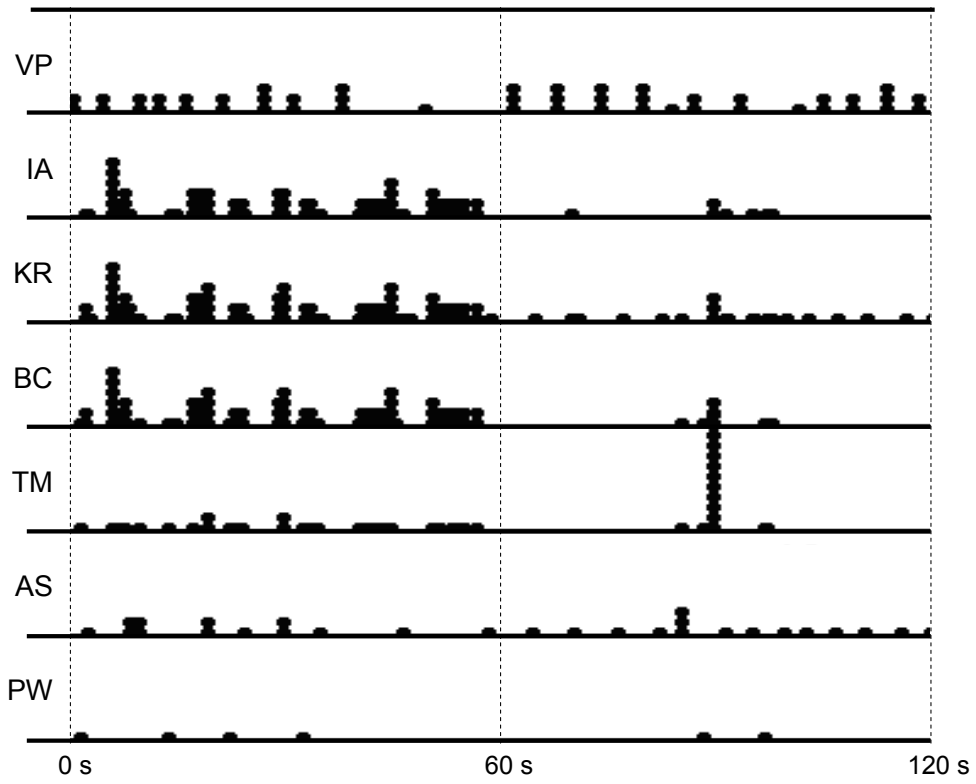


Abbildung 3-9: Beispiel für ein OCCS-Workload-Profil in einer aufwändig zu erkennenden Fehlersituation in einem Druckwasserreaktor (Furuta et al. 1995)

Das Beispiel in Abbildung 3-9 zeigt das abgeleitete Beanspruchungsprofil für eine Situation, in der eine Fehlerdiagnose durchgeführt werden muss. In der ersten Minute sind vor allem IA, KR und BC belastet. Nach ca. 100 Sekunden stellt sich aber heraus, dass eine bestimmte Annahme über die Fehlerursache nicht aufrecht zu halten ist. Daher ist in dieser Situation der TM Score besonders hoch, weil Schlüsse revidiert werden müssen.

OCCS ist in einer modifizierten Version auch zur Bewertung von Leitwartenanordnungen eingesetzt worden (Furuta et al. 1999). Dazu wurde die Simulation der Mensch-Maschine-Schnittstelle und das Sensormodul um ein räumliches Virtual Reality Modell erweitert. Dadurch wird die Verdeckung von im Raum angeordneten Komponenten und Bedienerbewegungen zur ungehinderten Informationsaufnahme simuliert. Es wurden jeweils vier Operateure in einer Warte modelliert, die unterschiedliche Aufgaben und Arbeitsanweisungen haben. Softwaretechnisch wird das durch eine Kopplung getrennter Blackboards und unterschiedlicher Wissensbasen mit den anderen Informationsverarbeitungskomponenten realisiert. Diese OCCS-Version wurde zum Vergleich der Operateursinteraktion und ihrer Bewegungsmuster in drei prototypischen Leitwarten mit unterschiedlich angeordneten Instrumenten angewendet.

OCCS ist ein Modellierungsframework, dessen Ansatz sich von dem kognitiver Architekturen wie ACT-R/PM, unterscheidet: Die Granularität der OCCS-Modelle ist gröber, dafür wird die physische Umwelt mit berücksichtigt. OCCS hat nicht den Anspruch eine *unified theory of cognition* darzustellen, sondern bildet einen kleineren Umfang von Phänomenen ab. Es ist aber insofern eine kognitive Architektur, dass konkrete Modelle für unterschiedliche Bediener und unterschiedliche Leitwarten durch unterschiedliche Wissensbasen erstellt werden können, die alle auf denselben Satz von Regelverarbeitungsmechanismen zurückgreifen.

3.2.3. Modellierte Phänomene

Die Simulation kognitiver Vorgänge soll herangezogen werden, um Arbeitsaufgaben und die Gestaltung der dazu verwendeten Mensch-Maschine-Schnittstelle zu analysieren. Dazu müssen empirisch beobachtbare kognitionswissenschaftliche und ergonomische Konstrukte herangezogen und auf messbare Simulationsparameter abgebildet werden. Die grundlegende Idee dazu ist, in einer Simulation das kognitive Modell in einer Arbeitsumgebung Aufgaben bearbeiten zu lassen. Zur Laufzeit der Simulation oder im Anschluss müssen vorher definierte Maße der internen Verarbeitung aus Protokolldaten und Logfiles abgelesen und mit den interessierenden Konstrukten abgeglichen werden. Die folgenden Konstrukte haben Bedeutung für die arbeitswissenschaftliche Bewertung von Mensch-Maschine-Systemen.

Beanspruchung

Das ergonomische Konzept „Beanspruchung“ kann unterschiedlich operationalisiert werden. Ein verbreitetes Maß für Beanspruchung ist der NASA Task Load Index (Hart & Staveland 1988). Hier wird Beanspruchung auf sechs Skalen durch subjektive Bewertung erhoben: *physical demand*, *mental demand*, *temporal demand*, *performance*, *effort* und *frustration*.

Ein Beispiel für eine Operationalisierung der Skalen aus Daten einer Simulation mit einem kognitiven Modell ist das vereinfachte Flugsicherungsmodell mit COGNET/iGEN aus dem AMBR-Vorhaben (Zachary et al. 2005). Hier wurde für jede Dimension des NASA Task Load Index eine Entsprechung in COGNET definiert, gemessen und mit empirischen Daten abgeglichen. Folgende Mappings zwischen Skalen und COGNET/iGEN-Konstrukten wurde benutzt:

- *physical demand*: Es wurde die Selbstwahrnehmung des Modells in Bezug auf die Aktivität des Motor-Systems herangezogen. Dabei wurden die Motoraktionen nach dem Grad der angenommenen bewussten Kontrolle gewichtet. Als Operationalisierung wurde die über die Szenariolaufzeit normierte Summe der so gewichteten Ausführungszeiten für Motoraktionen benutzt.
- *mental demand*: Zur Operationalisierung dieser Dimension wurde die Selbstwahrnehmung der kognitiven Komplexität der im Szenario anstehenden

Aufgaben benutzt. Für jede Aufgabe wurde ihre angenommene kognitive Komplexität aus der durchschnittlichen Anzahl ihrer internen Ziele und der Methodenaufrufe während ihrer Bearbeitung ermittelt. Dieses Komplexitätsmaß einer Aufgabe wurde mit der Anzahl ihrer Vorkommen im Szenario multipliziert. Die über die Szenariolänge normierte gewichtete Summe wurde für die Simulation der Dimension *mental demand* verwendet.

- *temporal demand*: Als negativer Indikator für die zeitliche Beanspruchung wurde die gemessene *idleness* des laufenden Modells herangezogen. Das Modell kann sich im Zustand *idle* befinden, wenn die Verarbeitungsregeln keine Schlussfolgerungs- oder Handlungserfordernis feststellen. Die *idle*-Zeiten werden summiert und über die Szenariodauer normiert. Die Negation dieses Wertes wird zur Vorhersage der Dimension *temporal demand* benutzt.
- *performance*: Das COGNET/iGEN Fluglotsenmodell für AMBR konnte Auslassungsfehler machen. Weiterhin konnte das Modell solche Fehler bemerken und korrigieren. Die Anzahl dieser erkannten Fehler wurde protokolliert und negiert als Maß für *performance* herangezogen.
- *effort*: Die Summe der Arbeitszeit im Szenario wurde verwendet, um *effort* auszudrücken. Die Arbeitszeit bezieht sich auf die Aktivität der Motor-Kanäle. Die visuelle Motorik und das in dieser Aufgabe häufige „Scannen“ des Radarschirmes werden nicht berücksichtigt. Die Summe der Arbeitszeit wird normiert über die Szenariolänge als Maß für *effort* benutzt.
- *frustration*: In der COGNET-Architektur können multiple Aufgaben anliegen. Jeweils eine davon wird zu einem Zeitpunkt bearbeitet. Durch Wahrnehmungen und metakognitive Prozesse können jedoch inaktive Aufgaben höher priorisiert werden und die gerade aktive Aufgabe unterbrechen. Die Anzahl dieser Unterbrechungen wird protokolliert und normiert über die Szenariolänge als Maß für *frustration* verwendet.

Das Workload-Profil (s. Abbildung 3-9) aus dem OCCS-System von Furuta et al. (1995) ist ein anderes Maß für Beanspruchung. Das hier vorgestellte System hat den Vorteil ein direktes Mapping auf ein allgemein gebräuchliches Maß zu benutzen. Daher gibt es validierte Messmethoden. Dadurch ist es möglich, die COGNET/iGEN-Operationalisierung anhand eines empirischen Vergleichs von Modellvorhersagen für ein Szenario und Operateursbewertungen für dasselbe Szenario zu bewerten. Außer im Bereich *effort* ergab sich bei dieser experimentellen Überprüfung im AMBR-Vorhaben über unterschiedliche Szenarien eine signifikante Korrelation von Modellvorhersagen und Versuchspersonen-Antworten. Besonders hoch war die Güte der Vorhersage für die Workload-Dimension *performance*.

Situation Awareness

Unter dem Begriff *Situation Awareness* wird das Wissen eines Operateurs über die Situation, die er kontrolliert, gefasst. Die Situation setzt sich aus Objekten und ihren Relationen zusammen. Endsley (1995) unterteilt *Situation Awareness* in drei Ebenen:

- Ebene 1: Wissen über den aktuellen Zustand der Objekte.
- Ebene 2: Wissen über die Bedeutung der Objekte für die Situation.
- Ebene 3: Wissen über den zukünftigen Zustand der Objekte.

Für die Bewältigung von Kontrollaufgaben in komplexen Systemen sind die Ebenen eins und zwei von Bedeutung. In dynamischen Systemen kann zudem Ebene drei für die Aufgabenstellung notwendig oder hilfreich sein.

Im Bereich der Flugführung ist eine Komponente von *Situation Awareness* die *Mode Awareness* des Piloten. Unter *Mode Awareness* wird das Wissen über den Zustand des Autopiloten und der Flugparameter, die darauf Einfluss haben, verstanden. Der Autopilot kann sich zu jedem Zeitpunkt abhängig von der Flugphase, vorherigen Eingaben des Piloten und bestimmter Flugparameter wie Geschwindigkeit, Höhe, Beschleunigung in einem von mehreren Zuständen befinden.

Lüdtke (2005) stellt ein erfolgreich validiertes kognitives Modell für einen Piloten bei der Autopilotenbedienung vor, das eine bestimmte Fehlerklasse, die zu mangelnder *Mode Awareness* führt, vorhersagt. Die Situation, in der der *Mode Awareness* Fehler auftreten kann, ist der Steigflug mit Autopiloten bei einer Piper Cheyenne PA42IIIA. Hier kann es unter bestimmten Bedingungen zu einem sog. indirekten Moduswechsel des Autopiloten kommen. Das kognitive Modell sagt einen falschen Interaktionsschritt mit dem Autopiloten voraus, der daraus resultiert, dass der Pilot glaubt, der Autopilot befinde sich noch in einem anderen Modus als tatsächlich der Fall.

Das Modell enthält Heuristiken zur Vorhersage von Abkürzungen in Interaktionsprozeduren auf der Basis der ISP-DL Theorie (*Impasse Success Problem Solving Driven Learning*: Möbus et al. 1994). Dahinter steht die Beobachtung „gelernter Sorglosigkeit“. Menschen lernen demnach Sorglosigkeit, wenn sie Handlungen wiederholt ohne die Berücksichtigung von eigentlich vorgesehenen Sicherheitsvorkehrungen durchführen, ohne dass die Fehlersituation eintritt, die eigentlich durch die Sicherheitsvorkehrung verhindert werden sollte.

Das kognitive Modell enthält demzufolge einen entsprechenden Lernmechanismus. Das Modell ist ein ad hoc Modell, das in Prolog implementiert wurde. Es hat eine Interaktionsschnittstelle zur Aufgabenumgebung, die in diesem Fall aus dem MS Flugsimulator besteht.

Wahrnehmung und motorische Fähigkeiten

Einige kognitive Architekturen enthalten Regeln über Begrenzungen der motorischen Fähigkeiten des Menschen. Es werden jedoch im Vergleich zu den Menschmodellen

aus der Ergonomie und Fabrikplanung (s. Naumann & Rötting 2007) nur sehr stark abgegrenzte Bereich modelliert. Insbesondere die Aktorik am Büroarbeitsplatz ist im Fokus der Architekturen EPIC und ACT-R/PM. Beide Architekturen verfolgen ein modulares Konzept, in dem Motorprozessoren als Träger der kognitiven Prozesse für Motorik modelliert werden. Sie führen Motorprogramme aus. Empirisch belegte Erkenntnisse über das Timing beim wiederholten Abruf eines Motorprogramms werden dabei berücksichtigt. Auch andere ergonomische Gesetzmäßigkeiten wie Fitt's Law sind implementiert.

Die Architektur OCCS ist hingegen nicht auf einen Büroarbeitsplatz als Interaktionsumfeld ausgelegt, sondern auf Leitwarten in Kernkraftwerken. Dementsprechend werden hier andere Gesetzmäßigkeiten von Wahrnehmung und Motorik modelliert. Insbesondere die Verdeckung von Ableseinstrumenten durch Operateure in der Leitwarte wurde in einer Simulation zur Optimierung des Layouts einer Anlage simuliert (Furuta et al. 1999).

Bei den menschlichen Eigenschaften der Wahrnehmung legen alle kognitiven Architekturen einen Fokus auf die visuelle Wahrnehmung. EPIC und ACT-R simulieren die visuelle Wahrnehmung auf der Ebene der Blickbewegungen (Anderson et al. 1997). Die perzeptuelle Komponente von ACT-R ist jedoch noch Gegenstand der Diskussion. Salvucci (2001) hat ein eigenes visuelles System implementiert, dass dann für die Kraftfahrzeugführung angewendet wurde (Salvucci 2006).

Neben dem visuellen System wurden auch die Effekte von unterschiedlichen akustischen Umgebungsreizen auf eine synthetische Radarkontrollaufgabe z.B. in EPIC modelliert (Kieras et al. 2001).

Ermüdung und Strategienauswahl

Ermüdung hat sowohl Einfluss auf Wahrnehmungsprozesse als auch auf Entscheidungsverhalten und Strategieauswahl. Jongman (1998) stellt ein Modell zur Simulation von Ermüdung auf der Basis von Hockeys (1997) *compensatory control model* am Beispiel des Sternberg-Tasks vor. Dabei benutzt sie den Aktivierungsmechanismus von ACT-R als Analogon zu Ermüdung. Je nach Ausmaß der zur Verfügung stehenden Gesamtaktivierung kann so zwischen zwei Strategien, die aus jeweils drei Produktionen bestehen, differenziert werden. Die Strategien unterscheiden sich nach der Ausprägung des Trade-Offs zwischen zeitlichem Aufwand und Genauigkeit bei der Aufgabenerfüllung. Über den Aktivierungslevel wird so die Unterscheidungsfähigkeit zwischen gerade relevanten (vom aktuellen Ziel referenzierten) und aktuell weniger wichtigen Chunks geregelt. Darüber können nach Hockeys Theorie vorherzusagenden Effekte in Bezug auf Reaktionszeit und Genauigkeit bei der Aufgabenerfüllung simuliert werden.

Lernen

Lernen ist eine Eigenschaft, die in kognitiven Architekturen als eingebauter Prozess vorhanden ist. Am Beispiel von ACT-R kann leicht demonstriert werden, welche Arten von Lernen unterstützt werden. Auf der symbolischen Ebene können Chunks und Produktionen gelernt werden. Gelernte Chunks sind encodierte Wahrnehmungen aus der Aufgabenumgebung oder inferierte Fakten. Zum Lernen von Produktionen bietet ACT-R 4 den „*production compilation*“ Mechanismus. Dabei werden zwei Produktionen die wiederholt nacheinander feuern zu einer neuen verschmolzen. Auf der subsymbolischen Ebene werden Parameter angepasst, die die Konfliktresolution beeinflussen. Das sind insbesondere die Produktionsstärke, und die Aktivierung der Chunks.

Lernen ist jedoch auch ein aufgabenspezifischer Prozess. Ein ACT-R Modell von Taatgen und Wallach demonstriert einen aufgabenspezifischen Lernprozess in der Aufgabenumgebung „Sugar Factory“. Sugar Factory ist eine Aufgabensimulation, in der Versuchspersonen die Produktion von Zucker (sp) durch die Zuweisung von Ressourcen (w) kontrollieren müssen. Die mathematische Abbildung von Ressourcen auf Produktionsoutput ist konterintuitiv und besitzt eine stochastische Komponente ($sp_t = 2w_t - sp_{t-1}$ + Zufallswahl aus {1, 0, 1}).

Das Modell von Taatgen & Wallach (2002) speichert dazu episodische Chunks über Input-Output Verhältnisse. Das Modell benutzt dann den Mechanismus des *partial matching*, der ein inhärenter Bestandteil der ACT-R Architektur ist, um gelernte Fakten über die beobachteten Zustände des Systems abzurufen. Die abgerufenen Beobachtungen werden dann vom Modell an die aktuelle Kontrollaufgabe angepasst. Taatgen und Wallach vergleichen die Entwicklung der Kontrollleistung des Modells mit Versuchspersonen und kommen zu sehr ähnlichen Leistungen beim Lernen des Systemverhaltens.

Arbeitsgedächtnis

Insbesondere die kognitive Architektur ACT-R enthält viele Annahmen über die Struktur und Leistung des Arbeitsgedächtnisses. Anderson et al. (1998) haben die ACT-R Theorie bspw. angewendet, um eine Reihe von Effekten bei *serial recall*, *free recall* und implizitem Gedächtnis zu simulieren. Dazu wird die Aktivation der Chunks im Arbeitsgedächtnis durch unterschiedliche Prozesse manipuliert.

Die in ACT-R modellierten Eigenschaften und Beschränkungen des Arbeitsgedächtnisses werden auch für die Simulation im Bereich *Human Factors* eingesetzt. So kann das Verwechseln von unterschiedlichen Bedienprozeduren, die als Liste von Einzelschritten als Chunks im Arbeitsgedächtnis repräsentiert sind, über unterschiedliche Priming Effekte simuliert werden (Schoppek & Boehm-Davis 2004).

3.2.4. Anwendungsfelder und ihre Anforderungen

Kognitive Modellierung wurde bereits zur Simulation von Bedienerverhalten in einer Reihe von Anwendungsfeldern eingesetzt. Die Anforderungen dieser Anwendungen sind unterschiedlich. Den größten Einfluss auf die Struktur der resultierenden Modelle hat die Aufgabencharakteristik.

Man kann reaktive und deliberative Aufgaben unterscheiden. Bei reaktiven folgt auf einen detektierbaren Stimulus direkt eine vorhersagbare Reaktion. In deliberativen Aufgaben muss die Art der Reaktion anhand einer Wissensbasis mit Regeln gefolgert werden. Frühere Stimuli können die Entscheidung beeinflussen. Daneben haben Ausmaß und Art der Dynamik einer Aufgabenumgebung einen wesentlichen Einfluss auf die Struktur eines kognitiven Modells.

Im folgenden Abschnitt werden Aufgabenumgebungen und dazugehörige kognitive Modelle erörtert.

Kraftfahrzeugführung

Obwohl die Kraftfahrzeugführung keine hierarchische Top-Down-Aufgabenstruktur hat, gibt es doch eine Reihe von Unteraufgaben, die alle zu einem gegebenen Zeitpunkt „aktiv“ sein können. Aasmann (1995) nennt diese Unteraufgaben, bzw. die Ziele, die mit ihnen erfüllt werden, *homeostatisch*. Es beschreibt ein kognitives Modell in Soar, mit dem Kontrolle und Koordination zwischen diesen Unteraufgaben in einer Fahrsimulation modelliert werden. Cnossen (2000, p. 40) und Salvucci (2006) verfolgen einen ähnlichen Ansatz. Sie beschreiben für ihre jeweiligen Autofahrermodelle, die beide in ACT-R umgesetzt sind, folgende wesentlichen homeostatischen Unteraufgaben zur Fahrzeugführung: Geschwindigkeitskontrolle, Lenken, visuelle Suche und Distanzhalten.

Die Kraftfahrzeugführung ist eine reaktive Aufgabe. Es handelt sich um stark automatisierte Abläufe. Dabei handelt es sich um eine hochdynamische Aufgabenumgebung, weil der Zustand des Fahrzeugs und die Umwelt sich ständig und teilweise unvorhergesehen ändern.

Flugführung

Wegen der Einführung von *fly by wire* und einem hohen Automatisierungsgrad im modernen *glass cockpit* erfüllen Piloten hauptsächlich eine Überwachungsaufgabe. Die Aufgabe ist durch fest vorgegebene und trainierte Prozeduren vorgegeben. In einem ACT-R-Modell der Führung einer Predator-Drohne wurden diese Prozeduren für mehrere Manöver direkt als Produktionsregeln umgesetzt (Gluck et al. 2003, Gluck et al. 2007). Die Werte der Anzeigen wurden als Attribute des Goal-Chunk repräsentiert.

In einem validierten ACT-R-Modell der Flugführung eines Verkehrsflugzeuges wurden die Prozeduren hingegen als verkettete Chunkliste repräsentiert, so dass die einzelnen Schritte verwechselt werden können (Schoppek et al. 2001).

Das Prolog-Modell zur Mode Awareness bei der Flugführung von Lüdtke et al. (2002) ist oben beschrieben.

Die Anforderungen aus der Dynamik der Aufgabensituation bei der Flugführung sind gering. D.h. Änderungen vollziehen sich langsam und sind aufgrund der Eigenschaften des Flugzeugs und des Wetters vorhersehbar. Es handelt sich um typische Regelungsaufgaben. Aufgrund der hohen Standardisierung, der Automatisierung und des Trainingsstandes der Piloten handelt es sich um eine reaktive Aufgabe.

Flugsicherung

In der Flugsicherung überwachen Lotsen die Flugbewegungen im Luftverkehr an Radarschirmen. Obwohl Routen und Geschwindigkeiten im Voraus geplant sind, um gefährliche Annäherungen zu vermeiden, kann es aufgrund von Abweichungen und äußeren Einflüssen zu potenziell gefährlichen Annäherungen kommen, die Fluglotsen antizipieren und durch Anweisungen an die Piloten verhindern müssen.

Neben dem selbstentwickelten kognitiven Modell MoFL (s. 5.1) gibt es weitere Modelle der kognitiven Vorgänge bei Fluglotsen. Taatgen (2002) beschreibt ein ACT-R-Modell, mit dem Lernen in einer vereinfachten Fluglotsenaufgabe analysiert wurde. In einem ACT-R-Modell, das im Rahmen des AMBR-Programms entwickelt wurde, wurde die Bearbeitung einer anderen synthetischen Flugsicherungsaufgabe unter besonderer Berücksichtigung der Wahrnehmungsaspekte simuliert (Lebiere 2005).

Die Anforderungen aus der Dynamik der Aufgabensituation sind hoch. Es gibt eine Vielzahl von Elementen, die sich eigenständig bewegen und jederzeit von ihrem erwarteten Verhalten abweichen können. Aufgrund des hohen Antizipations- und Planungsanteils bei der Arbeit handelt es sich um eine deliberative Aufgabenumgebung.

Eine ähnliche Aufgabe besteht in einem Führungsinformations- und Waffeneinsatzsystem (FüWES) an Bord eines Kriegsschiffes. In einer Radarumgebung mit zusätzlichen Informationen und Unterstützungssystemen muss die Umgebung des Schiffes auf der Wasseroberfläche und am Himmel beobachtet werden. Dargestellte Flugzeuge und Boote müssen identifiziert werden und es muss die Unterscheidung in Freund, Feind und neutral getroffen werden. Im Falle eines Angriffs muss ein Waffensystem aktiviert werden. Scott & Marshall (1998) beschreiben ein ACT-R-Modell, das die Aufgaben an einem simulierten FüWES erfüllen kann. Der Bericht zeigt aber, dass das Modell nicht vollständig in ACT-R implementiert werden konnte. Insbesondere die Zig-Zag Methode zur

Aufgabenkontrolle und zum Multitasking musste außerhalb von ACT-R mit LISP implementiert werden.

Interaktive Computerspiele

Interaktive Computerspiele sind für die kognitive Modellierung eine herausfordernde Aufgabenumgebung. Insbesondere in der Ausprägung von *Multiplayer First Person Shootern* und *Real Time Strategy Spielen* besitzen sie eine hohe Dynamik. Änderungen der Situation ergeben sich ohne Zutun des Spielers. Aufgaben existieren sowohl auf einer reaktiven Ebene (bei der Steuerung) als auch auf einer deliberativen (bei der Missionserfüllung).

Quake-Bot ist ein Forschungsprototyp von Laird (2001). Hier wurde ein kognitives Modell in Soar an eine simulierte Aufgabenumgebung angebunden, die aus dem *Multiplayer First Person Shooter* Quake besteht. Das Soar-Programm ersetzt in diesem Versuchsaufbau einen gegnerischen Spieler durch Simulation von dessen Handlungen. Eine besondere Eigenschaft dieses simulierten Spielers ist seine Fähigkeit, gegnerische Aktionen aus Selbstreflektion und den Aktionen des Spielers zu antizipieren, indem die eigenen Fähigkeiten und Verhaltensregeln auf beobachtete Opponenten übertragen werden.

Im Bereich der *Real Time Strategy Spiele* hat sich inzwischen sogar ein Wettbewerb im Sinne einer Herausforderung an die Wissenschaft etabliert. Dazu gibt es die Open Source Aufgabenumgebung ORTS (*open real time strategy*). Die Aufgabe ist, einen simulierten Spieler in dieser Welt bestehen zu lassen. Ein Beispiel für ein kognitives Modell, das mit RTS interagieren kann, ist SORTS (Xu 2006). In diesem kognitiven Modell in Soar werden besonders Planung als deliberatives Element und Multitasking zwischen mehreren homeostatischen Zielen als reaktives Element modelliert. Multitasking wird über eine Repräsentation von Aufgaben als Chunks im Arbeitsgedächtnis umgesetzt. Eine zentrale regelbasierte Komponente entscheidet, welche gerade zurückgestellte Aufgabe als nächstes abgearbeitet werden soll.

Prozesskontrolle in Leitwarten

Es gibt kognitive Modelle zur Kontrolle vereinfachter chemischer und energetischer Prozesse in ACT-R (Wallach 1996, Taatgen & Wallach 2002), die die Steuerung durch Abruf von Chunks modellieren. Außerdem gibt es eine Vielzahl von Modellen zur Steuerung von Kernkraftwerken in OCCS, mit denen Abläufe und Gestaltung von Leitwarten optimiert wurden (Shu & Furuta 2005, Furuta & Kondo 1993, Furuta et al. 1995).

Im Gegensatz zu den anderen bis hier beschriebenen Aufgabenumgebungen ändert sich die Struktur der Aufgabensituation bei der Prozesskontrolle in Leitwarten nicht. Es handelt sich immer um einen Prozess, der in einer Wirkstruktur in einem Prozessleitsystem abgebildet wird. Das mentale Modell der Bediener orientiert sich

an diesen Strukturen. Ungeachtet dessen wird auch durch das Anfahren und Abschalten von Anlagenteilen und die Möglichkeit, Prozesseinheiten unterschiedlich miteinander zu verschalten, eine Dynamik in der Aufgabe erzeugt. Viele chemische und energetische Prozesse sind aufgrund der komplizierten Dynamik der Prozessgrößen und die teilweise sehr großen Zeitkonstanten schwer zu steuern. Die Aufgabe ist deliberativ, da die Prozessdynamik anhand des eigenen mentalen Modells des Prozesses vom Wartenfahrer beurteilt werden muss.

Schiffsführung

Kognitive Modelle aus der Schiffsführung sind nicht bekannt. Es gibt jedoch ein validiertes ad-hoc Modell in LISP, das die Aufgabenbearbeitung eines Approach-Offiziers auf einem U-Boot simuliert (Gray et al. 1997, Kirschenbaum & Gray 2000). In dem Modell wird die Aufgabe als Problemlösen mit Schemata repräsentiert. Dabei werden sequentiell Unterziele zur Problemzerlegung erzeugt.

Die Anforderungen aus der Dynamik sind in der Schiffsführung gering. Normalerweise bewegen sich andere Schiffe mit geringer Geschwindigkeit auf vorhersehbaren Bahnen. Die Zeitkonstanten sind jedoch sehr groß, was die Regelung erschwert. Aufgrund von Strömungen und anderen äußeren Faktoren, die bei der Aufgabenbearbeitung berücksichtigt werden müssen, handelt es sich um eine deliberative Aufgabe.

3.2.5. Kopplung und Wahrnehmung

Eine wesentliche Herausforderung bei der Anwendung von kognitiven Architekturen für die Simulation von Mensch-Maschine-Interaktion ist die Möglichkeit der Interaktion zwischen kognitivem Modell und einer externen Aufgabenumgebung (Bass et al. 1995). Der kognitive Prozess muss zur Wahrnehmung den Aufmerksamkeitsfokus steuern, um einen bestimmten Bereich der Aufgabenumgebung zu erfassen. Dabei wird die komplexe Umwelt von einem Abbildungsprozess in eine symbolische Repräsentation encodiert. Die encodierten Wahrnehmungen werden mit Chunks als Teil des Arbeitsgedächtnisses repräsentiert. Dadurch stehen sie für eine weitere Verarbeitung mit Produktionen bereit.

Die Architektur zur Kopplung von externer Aufgabenumgebung und kognitiver Architektur von Kieras & Meyer (1997) in EPIC war besonders einflussreich. In ihr wurden viele ergonomische Gesetzmäßigkeiten zur Wahrnehmung bis hinunter auf die Ebene der Blickbewegungen implementiert und sie ist modular aufgebaut. Architektur und Spezifikationen der Wahrnehmungs- und Motor-Prozessoren aus EPIC wurden deshalb erfolgreich in andere kognitive Architekturen transferiert, die vorteilhafter in der kognitiven Verarbeitung sind. ACT-R/PM (Byrne & Anderson 1998), EPIC-Soar (Chong 1995) und EASE (Elements of ACT-R, Soar, and EPIC: Chong & Wray 2005) beinhalten das EPIC-Konzept zu Wahrnehmung und Motorik.

In den meisten Fällen liegt die Aufgabenumgebung in Form einer Software vor, auf die per Interprozesskommunikation (z.B. TCP/IP) zugegriffen werden muss. Die Aufgabenumgebung kann ihrerseits Komponenten zur Ankopplung eines externen Prozesses oder seiner Simulation enthalten. Mit dem kognitiven Modell muss dann in plausibler Weise auf die Aufgabenumgebung zugegriffen werden. Dazu werden unterschiedliche Ansätze verwendet. Die Software, die die Aufgabenumgebung darstellt, kann entweder als Teil des Prozesses ablaufen, im dem auch das kognitive Modell als Simulation läuft, oder als eigenständiger Prozess auf demselben oder einem entfernten Rechner, der über eine Netzwerkverbindung angeschlossen ist. Es gibt auch komplexe verteilte Simulationen, wo eine Vielzahl kognitiver Modelle als software-agentenbasierte Komponenten mit anderen Simulatoren technischer Systeme und der Umwelt über CORBA miteinander kommunizieren (Kanno et al. 2006).

Im ersten Fall greift die kognitive Architektur auf eine interne Repräsentation der grafischen Benutzungsoberfläche zu, in der deren aktueller Zustand (Widgets, mit Eigenschaften wie z.B. Position, angezeigter Inhalt) gespeichert ist. Die Systeme EPIC, ACT-R, OCCS und APEX (s. 3.3.6, S. 77) bieten diese Art der Kopplung mit einer Aufgabenumgebung. Diese Systemarchitektur hat mehrere Probleme:

- EPIC, ACT-R und APEX basieren auf LISP als Programmsystem, OCCS auf Prolog. Diese Programmiersprachen werden selten benutzt, um Anwendungssysteme und grafische Benutzungsoberflächen zu programmieren. Eine existierende Aufgabenumgebung, von der oft auch kein Quellcode vorliegt, kann deshalb nicht ohne Weiteres mit einem kognitiven Modell gekoppelt werden.
- Es gibt mit diesem Ansatz keine Möglichkeit, Aufgabenumgebung und kognitives Modell zeitlich zu synchronisieren.
- Der Ressourcenbedarf für die Simulation eines kognitiven Prozesses und eines technischen Prozesses, mit dem das kognitive Modell interagieren soll, kann eine zu hohe Rechenlast für einen Rechner bewirken. In diesem Fall können beide Simulationen nicht auf mehrere Rechner verteilt werden.

Diese Art der Kopplung ermöglicht es aber, auf der Seite der kognitiven Architektur plausible wahrnehmungspsychologische Prozesse in der Schnittstellendefinition abzubilden, da die Art des Zugriffes auf die Aufgabenumgebung vollständig von der Architektur kontrolliert wird.

Im anderen Fall, wo die Aufgabenumgebung als eigenständiger Prozess auf demselben oder einem entfernten Rechner abläuft, muss ein anwendungsspezifisches Protokoll definiert werden, mit dem auf die Aufgabenumgebung und die Elemente ihrer grafischen Benutzungsoberfläche zugegriffen werden kann. Über ein so zu definierendes Interface muss mit einer Middleware, die auf beiden Seiten zu implementieren ist, auf die wahrnehmbaren Elemente der Aufgabenumgebung zugegriffen werden und die Elemente als Chunks

encodiert werden. In diesem Fall ist es eine Aufgabe der anwendungsspezifischen Protokolldefinition, plausible wahrnehmungspsychologische Prozesse abzubilden. Soar, COGNET/iGEN und MIDAS (s. 3.3.6, S. 76) verwenden diesen Ansatz. In Soar wird dieser Ansatz sogar mit unterschiedlichen Kopplungskomponenten verfolgt.

Robot-Soar (Laird et al. 1993) ist einer der ersten Versuche, Soar mit einer externen Aufgabenumgebung zu koppeln. Für Soar gibt es inzwischen mehrere Middlewares für die Integration mit anderen Systemen. Ein Beispiel ist die Soar-Tcl Erweiterung, mit der Tcl als Integrationsumgebung zur Anbindung externer Prozesse benutzt wird (Lonsdale & Ritter 2000). Dadurch ist es auf der technischen Ebene einfach, Aufgabenumgebungen anzubinden. Eine andere Anwendung ist Soar-MODSAF, womit eine Kopplung von Soar an eine große Plattform für die verteilte militärische Simulation angebunden wurde (Jones et al. 1999). Auch neuere Schnittstellen wie SML (Soar Community o.J.) verwenden diesen Schnittstellenansatz.

Bei der Auswahl dieser beiden Kopplungskonzepte für eine Simulations-anwendung muss zwischen der Erfordernis der wahrnehmungspsychologischen Genauigkeit der Informationsaufnahme und dem Portierungsaufwand für eine Aufgabenumgebung bzw. dem Erfordernis komplexen Simulation eines technischen Prozesses abgewogen werden. Ein Anhaltspunkt ist die Skala der untersuchten Zeitabläufe. Liegen sie im Bereich mehrerer Minuten statt Sekunden, ist die wahrnehmungspsychologische Genauigkeit i.A. weniger wichtig als wissensbasierte und deliberative Fähigkeiten eines kognitiven Modells.

Ein wichtiges Merkmal der Kopplung ist, wie Wahrnehmung und Encodierung modelliert werden. Welche analogen Eigenschaften der Aufgabenumgebung werden wahrgenommen, wie werden Features daraus extrahiert und repräsentiert? Die Ansätze, die in z.B. in ACT-R und EPIC verwendet werden, gehen von einer vorhergehenden Codierung der Eigenschaften der Aufgabenumgebung z.B. anhand von Bildschirmpositionen und einem direkten Mapping von Darstellung und Symbol aus. Aktuelle Forschungsansätze gehen darüber hinaus. So werden visuelle Eigenschaften der Bildschirmdarstellung in einem interaktiven Realzeitspiel von Wintermute (2006) benutzt, um eine Feature Map der wahrgenommenen Situation aufzubauen, die zum Auffinden von Objekten in einem Soar-Modell herangezogen wird.

Ritter et al. (2000) und Amant & Riedl (2001) gehen sogar noch weiter, indem sie fordern, die Extraktion von Symbolen vollständig durch eine visuelle Interpretation der Szene auf einem Computerbildschirm mit Bildverarbeitungsmethoden umzusetzen. Dieser Ansatz hat den Vorteil, dass beliebige Programme als Aufgabenumgebung an kognitive Modelle angeschlossen werden können, ohne dass sie um eine Kopplungsschnittstelle zu erweitern sind. Jedoch ist auch diese Art der Integration stets aufgabenabhängig. Dieser Ansatz wurde bereits erfolgreich, jedoch

mit großem Aufwand, in interaktiven Computerspielen (Shah et al. 2003) und der Kraftfahrzeugführung in einem Simulationsprogramm (Ritter et al. 2006b) eingesetzt.

Ein ähnlicher Ansatz wird mit dem Tool *Simulated Hand and Eye – SHE* (Misker et al. 2001) verfolgt. SHE ist eine ACT-R/PM spezifische Middleware, mit der kognitive Modelle mit einer Klasse von Windows-Programmen (programmiert in Visual Basic, Visual C++ und Java AWT) auf dem Level von Standard-Widgets interagieren können. Dazu wird ein vom kognitiven Modell unabhängiger Prozess benutzt, der auf die Widgets der zu bedienenden Windows-Software zugreifen kann. Dieser Prozess wird über TCP/IP an das kognitive Modell angebunden.

3.2.6. Zusammenfassung der Ebene Kognitionsanalyse

Der wesentliche Nutzen der Erörterung von Methoden und Werkzeugen zur Modellierung und Simulation auf der Ebene der Kognitionsanalyse ist im Rahmen dieser Arbeit, das Anwendungspotenzial zu identifizieren und eine Auswahl von Methoden im Rahmen der Anwendung für die Lösung von Problemen im Bereich dynamischer Mensch-Maschine-Systeme zu ermöglichen. In Tabelle 3-1 ist eine bewertete Gegenüberstellung der Werkzeuge und Architekturen zur kognitiven Analyseebene zusammengefasst.

EPIC hat für diesen Anwendungszweck interessante Fähigkeiten beim Umgang mit Dynamik. Es ist damit jedoch nur möglich, Bedienung in reaktiven Aufgabenumgebungen nachzubilden. EPIC hat zudem als Werkzeug und Modellierungsumgebung noch nicht die erforderliche Reife erreicht. Ein wesentlicher Grund dafür liegt in der geringen praktischen Verbreitung. EPIC ist deshalb für die Anwendung in der Bedienermodellierung in dynamischen Mensch-Maschine-Systemen nicht geeignet.

ACT-R hat eine breite Nutzerbasis in der kognitionswissenschaftlichen Grundlagenforschung, es wird zunehmend in der anwendungsorientierten Forschung eingesetzt. ACT-R kann insbesondere bei deliberativen Aufgabenumgebungen eingesetzt werden, für viele Verarbeitungsprozesse und kognitive Strukturen gibt es Human Factors relevante Beschränkungen. Die Bereiche Multitasking und Interaktion mit externen Aufgabenumgebungen werden noch unzureichend unterstützt. Es erscheint aber möglich, ACT-R an diesen Stellen auch mit Hilfe der großen Nutzergemeinde weiterzuentwickeln. Deshalb ist ACT-R ein Kandidat für die Anwendung in der Bedienermodellierung in dynamischen Mensch-Maschine-Systemen.

Soar hat eine große Verbreitung gefunden. Es gibt zudem Schnittstellen auf technischer Ebene, um mit Aufgabenumgebungen zu interagieren. Soar ist eine kognitive Architektur, in der viele Heuristiken zur Produktionsauswahl eingesetzt werden. Es ist möglich, komplexes Verhalten vor allem in wissensbasierten Aufgabenumgebungen zu simulieren. Soar beinhaltet jedoch nur sehr wenige Human Factors Beschränkungen und ist deshalb für den angestrebten Zweck nicht geeignet.

Tabelle 3-1: Bewertete Gegenüberstellung der Werkzeuge und Architekturen zur kognitiven Analyseebene

System	Community	Anwendungs-bereich	Handling von Dynamik	Interaktion mit Aufgaben-umgebung
EPIC	einige Anwender in HCI, Kognitionspsychologie	nur reaktive Aufgaben	möglich (Multitasking)	Wahrnehmung und Aktion vorhanden, kein technisches I/F
ACT-R	breit, Grundlagenforschung Kognitionswissenschaft, zunehmend anwendungsorientiert	insb. deliberative Aufgaben, Problemlösen, Gedächtnis	schwierig	Wahrnehmung und Aktion vorhanden, begrenztes technisches I/F
Soar	breit, anwendungsorientiert KI	insb. wissensbasierte Aufgaben	schwierig	technisches I/F vorhanden
COGNET/ iGEN	nur Fa. CHI, Anwendungen für US Militär	Standard operating procedures, keine deliberativen Aufgaben	möglich	Wahrnehmung und Aktion und technisches I/F vorhanden
COGENT	schmal, allg. Psychologie	keine Aufgabenumge-bungen	schwierig	nein
OCCS	nur University of Tokyo, CSE Lab	nur Leitwarten (standard operating procedures)	möglich	begrenztes Modell für Wahrnehmung und Aktion, technisches I/F vorhanden

COGNET/iGEN hat keine breite Benutzerbasis, wird aber kommerziell von der Firma CHI für militärische Anwendungen erfolgreich eingesetzt. Es ist nur möglich die Abarbeitung von *Standard Operating Procedures* – auch mit Schnittstellen zu externen Aufgabenumgebungen – zu simulieren, deliberative Aufgabenumgebungen können mit dem System nicht modelliert werden. Für den angestrebten allgemeineren Zweck kann das System deshalb nicht verwendet werden.

COGENT hat nur eine schmal Nutzergruppe, die das System hauptsächlich im Bereich der psychologischen Forschung einsetzt. Spezielle Funktionen zur Modellierung von Interaktion und zum Umgang mit Dynamik sind nicht bekannt. Da COGENT als Baukasten nicht eine einheitliche Theorie beinhaltet, wird die Modellierung nicht durch Beschränkungen geleitet. Deshalb ist COGENT nicht für den angestrebten Anwendungszweck geeignet.

OCCS wird als Forschungsprototyp nur an der University of Tokyo eingesetzt, um *Standard Operation Procedure* (SOP)-basierte Interaktion in Leitwarten zu modellieren. OCCS kann mit dynamischen Aufgabenumgebungen interagieren, dabei ist ein Modell für Wahrnehmung und Aktion vorhanden. Aufgrund der geringen Verbreitung, der schlechten Dokumentation und dem speziellen Anwendungsgebiet wird OCCS als Basis für die Bedienermodellierung in dynamischen Mensch-Maschine-Systemen nicht weiterverfolgt.

Aufgrund dieser Bewertung wird ACT-R als Modellierungsbasis weiter verfolgt. Konzeption, Realisierung und Erprobung von ACT-R Erweiterungen zum Umgang mit Dynamik und zur Verbesserung der Interaktionsfähigkeit mit externen Aufgabenumgebungen sind Gegenstand dieser Arbeit.

3.3. Modellierung und Simulation auf der Ebene Aufgabenanalyse

Die Spezifikation der Aufgaben in einem Arbeitssystem ist der Ausgangspunkt für eine benutzerzentrierte Analyse und Gestaltung des Mensch-Maschine-Systems. Der Begriff der Aufgabe bezeichnet in diesem Zusammenhang ein Ziel in der Arbeitsumgebung, das unter bestimmten Bedingungen durch den Benutzer zu erreichen ist (Leontiev 1972 nach Leplat 1988). Die Aufgabenanalyse liefert die prozedurale Beschreibung des Arbeitssystems.

3.3.1. Hierarchische Aufgabenanalyse

Bei der hierarchischen Aufgabenanalyse wird die betrachtete Aufgabe in Teilaufgaben – und diese wiederum in ihre Teilaufgaben – zerlegt, um auf eine geeignete Analyseebene zu kommen (Annett & Duncan 1967). An einem Punkt der Zergliederung entspricht die betrachtete Aufgabe gerade einer auf der gewählten Betrachtungsebene atomaren Aktivität des Bedieners. Aktivitäten zur Erfüllung der Aufgabe im Kontext interaktiver informationstechnischer Systeme ist eine Reihe von Dialogschritten oder Bedienaktionen, die vom Ausgangszustand zum Aufgabenziel führen.

Abbildung 3-10 verdeutlicht ein Modell der Aufgabenanalyse und des Entwurfszyklus' (nach Bösser & Melchior 1992). Dabei wird von der konkreten Welt mit ihrer Arbeitsaufgabe, den realen Interaktionseinrichtungen und den hier vorgegebenen Benutzungsprozeduren abstrahiert. Die Arbeitsaufgabe der konkreten Welt wird analysiert. Das Ergebnis der Analyse ist ein Modell der Aktivitäten, die der Bediener in seiner Aufgabenumgebung durchführt. Sie werden in einem nächsten Schritt, der Aufgabenanalyse, strukturiert, in Sequenzen angeordnet und Zielen zugeordnet, die durch ihre Durchführung erreicht werden. Das Ergebnis ist das Aufgabenmodell.

Dieses Analyseergebnis dient im Syntheseschritt dazu, die Mensch-Maschine-Interaktion zu optimieren. Dazu können die Benutzungsprozeduren angepasst werden, so dass die Sequenz der Bedienschritte aufgabenangemessener wird, oder die Interaktionselemente der Aufgabenumgebung können so angepasst werden, dass die Bedien- und Wahrnehmungsaktivitäten optimiert werden.

Bei der Durchführung der Aufgabenanalyse muss jedoch immer zwischen der vorgegebenen Aufgabe (in der abstrakten Welt) und der tatsächlich durchgeführten Aufgabe (in der konkreten Welt) unterschieden werden. Gegebenfalls besteht eine Diskrepanz zwischen diesen beiden Modellen. Deshalb muss bei der Aufgabenanalyse immer sowohl ein empirischer als auch ein gestalterischer Ansatz parallel verfolgt werden.

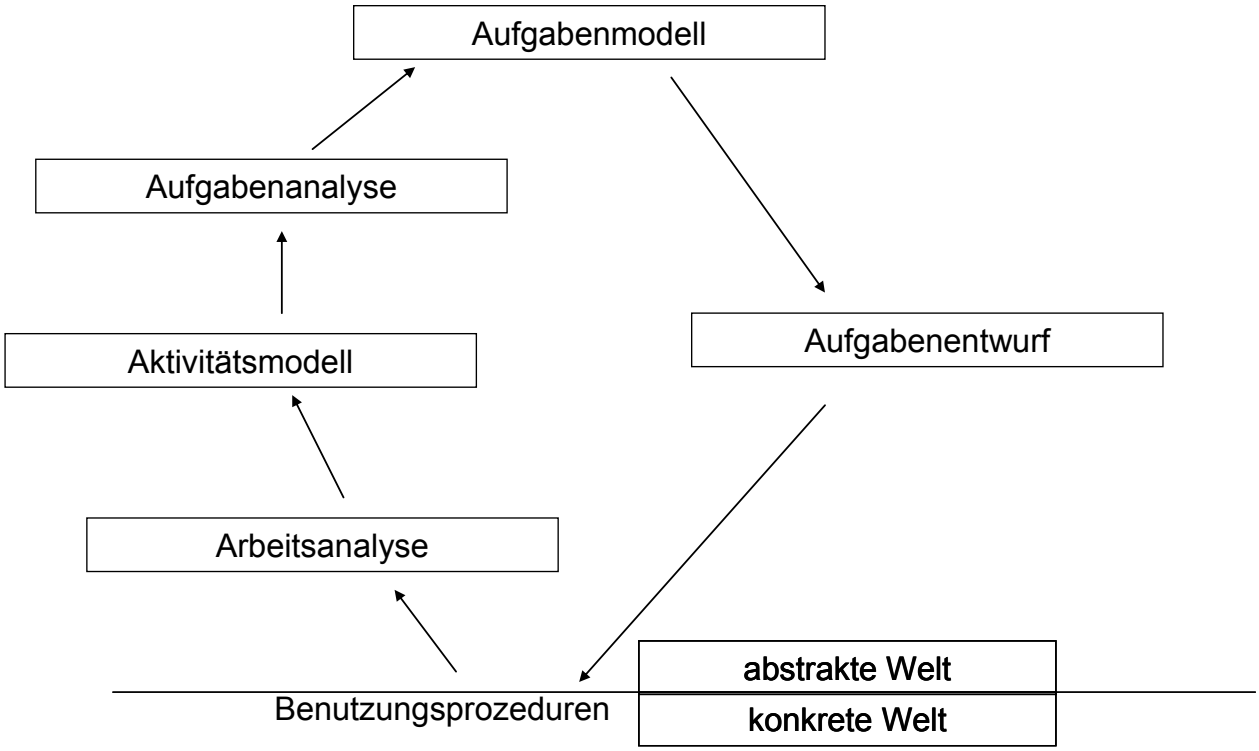


Abbildung 3-10: Aufgabenanalyse und Entwurfszyklus nach Bösser & Melchior (1992)

3.3.2. Methoden zur Aufgabenanalyse und -modellierung

In den folgenden Abschnitten werden Methoden zur Aufgabenanalyse und -modellierung erörtert. Sie basieren alle auf einer hierarchisch gegliederten prozeduralen Beschreibung von Aufgaben. Die Methoden erheben den Anspruch, das Aufgabenmodell innerhalb dieses Analyserahmens unabhängig von Benutzerklasse und Aufgabendomäne zu erheben und zu beschreiben. Dafür benutzen sie eine allgemein gebräuchliche Taxonomie für die Hauptaufgabentypen:

- Planung: Erarbeiten einer Sequenz von Aktionsschritten vom Ausgangszustand zum Zielzustand
- Kommunikation: Nachrichtenaustausch zwischen unterschiedlichen Teilsystemen zur Aufgabenbearbeitung
- Management: Auswahl, Koordination, Durchführung und Kontrolle von Aktionsschritten zur Aufgabenbearbeitung und Zielerreichung
- Navigation: Prozess der Kontrolle und Steuerung eines Objekts der Aufgabenumgebung

Bei der hierarchischen Aufgabenanalyse wird die Aufgabenbearbeitung durch eine rekursive Zerlegung in Teilaufgaben bis hin zu nicht mehr zergliederbaren Aktivitäten erreicht. Die Art der Aufgabe gibt die Sequenzierung der Teilaufgabenbearbeitung vor und welche Teilmenge der Unteraufgaben zu erfüllen ist. Die Aufgabenbearbeitung wird als Sequenz von Aktivitätsschritten repräsentiert. Die Aufgabe selber ist jedoch unabhängig von der Sequenz, z.B. weil es alternative Lösungswege gibt.

Die mögliche Aufgliederung und die Sequenzierung der Aufgabenbearbeitung können mit unterschiedlichen Formalismen beschrieben werden. Neben den Methoden der GOMS-Familie (s. 3.3.5) werden die folgenden Modellierungsmethoden und Repräsentationssprachen für die hierarchische Aufgabenanalyse benutzt: CLG, TAKD, CCT, TAG, ETAG, CTT, UML, K3, BPMN und AMBOSS (chronologisch sortiert).

CLG – Command Language Grammar

Für die Aufgabenbearbeitung in kommandobasierten Dialogsystemen wurde die *Command Language Grammar* Methodik (CLG: Moran 1981) entwickelt. Hier werden vier unterschiedliche abstrakte Ebenen der Aufgabe unterschieden. Auf der abstraktesten Ebene wird die Aufgabe mit den externen Anforderungen zur ihrer Bearbeitung beschrieben. Auf der Ebene der Semantik wird die zur Bearbeitung zu verwendende Anwendungslogik anhand ihrer Objekte und der dazugehörenden Methoden repräsentiert. Auf der nächsten Ebene wird die Syntax des Dialogsystems mit den zu verwendenden Kommandos betrachtet. Auf der konkretesten Ebene werden dann physikalische Methoden zur Interaktion von Benutzer und Dialogsystem beschrieben.

TAKD – Task Analysis by Knowledge Description

Die Methode *Task Analysis by Knowledge Description* (TAKD: Johnson et al. 1984, Diaper 2001) basiert auf der Beschreibung von Wissenseinheiten, die zur Problemlösung benötigt werden. Die Anwendung der Methode erfolgt in vier Schritten. 1) Erzeugen der Aufgabenbeschreibung, 2) Identifizieren des erforderlichen Wissens in Form von Objekten und Aufgaben, 3) Klassifizierung der Wissenseinheiten in generischen Aktionen und Objekten und 4) Repräsentation der Aufgabe mit einer *knowledge representation grammar* (KRG), einer formalen Zuordnung von Ziel und Aktionen.

CCT – Cognitive Complexity Theory

Die *Cognitive Complexity Theory* (CCT: Kieras & Polson 1985) basiert ebenso wie GOMS (s. 3.3.5) auf einer Zergliederung in Ziele. Die Methoden werden anhand der Auswahlregeln in einem Produktionssystem repräsentiert. Dadurch entsteht ein ablauffähiges Aufgabenbearbeitungsmodell. CCT ist daher nicht nur ein Formalismus, sondern gleichzeitig ein analytisches Simulationswerkzeug, mit dem eine algorithmische Abschätzung von Lernzeiten, Ausführungszeiten und dem Grad des Wissenstransfers zwischen unterschiedlichen Aufgaben möglich ist. Die kognitive Komplexität eines Bediensystems ist analog zur Anzahl der zur Beschreibung seiner Bearbeitung erforderlichen Produktionsregeln.

TAG – Task Action Grammars

Task Action Grammars (TAG: Payne & Green 1986) ist eine Methode zur Formalisierung der Interaktion mit kommandobasierten Systemen. TAG stellt eine Metasprache zur Formulierung von Task Languages bereit, mit denen konkrete Aufgabenbeschreibungssprachen ausgedrückt werden können. Dabei werden Interaktionsobjekten und Operatoren zu deren Manipulation beschrieben.

ETAG – Extended Task-Action Grammar

Die *Extended Task-Action Grammar* Methode (ETAG: Tauber 1990) ist eine Erweiterung von TAG. Im Gegensatz zu TAG wird in ETAG eine Verfeinerung der Abstraktionsebenenhierarchie aus der CLG bei der Definition der Grammatik eines Interaktionsystems verwendet. Dazu wird eine *user's virtual machine* (UVM) mit Konzepten, Attributen, Relationen, Funktionen und Ereignissen der Aufgabenwelt definiert, mit der dann basale Aufgaben, bzw. die entsprechenden grammatikalischen Produktionsregeln, definiert werden.

CTT – ConcurTaskTrees

ConcurTaskTrees (CTT: Paternò 1999) sind grafische Modelle von hierarchisch zerlegten Aufgaben. Im Gegensatz zu den früheren Ansätzen ist es möglich, temporale Abhängigkeiten zwischen der Ausführung von Aktivitäten in parallelen Teilaufgaben auszudrücken. Die folgenden Spezifikationen von temporalen Abhängigkeiten sind möglich: unabhängige Nebenläufigkeit, Auswahl, Nebenläufigkeit mit Informationsaustausch, geordnete Unabhängigkeit, Deaktivierung, Ermöglichung, Ermöglichung mit Informationsaustausch, Unterbrechung mit Wiederaufnahme, Iteration, optionale Aufgabe, Rekursion. Dadurch lassen sich komplexe Wirkzusammenhänge zwischen Teilaufgaben ausdrücken. *ConcurTaskTrees Environment* (CTTE: Mori et al. 2002) ist ein grafisches Werkzeug zum Modellieren mit CTT und zur Analyse von CTT-Modellen. Alternative Aufgabenmodelle können in CTTE anhand eines Satzes von vorgegebenen Indikatoren (in erster Linie die Anzahl einiger Modellelemente) verglichen werden und es können quantitative Simulation zur Aufgabenabarbeitung durchgeführt werden.

UML – Unified Modeling Language

Die *Unified Modeling Language* (UML: OMG 2000) ist eine Notation für objektorientierte Systemanalysen und -entwürfe. Neben anderen gibt es Zustands- und Aktivitätsdiagramme, die für die Aufgabenmodellierung verwendet werden können. Mit UML werden jedoch vorwiegend Systemabläufe und nicht das Verhalten, die Ziele und Aufgaben von Benutzern formalisiert (Bomsdorf & Szwillus 2002). Die dafür notwendigen Erweiterungen können jedoch in Form von UML-Stereotypen modelliert werden.

K3 – Koordination, Kooperation, Kommunikation

Koordination, Kooperation, Kommunikation (K3: Foltz et al. 2000) dient der qualitativen Modellierung von Kooperations-, Kommunikations- und Koordinationsprozessen in Teams. K3 baut auf UML-Aktivitäts- und -Zustandsdiagrammen auf und erweitert sie um Blobs (Container für Aktivitäten), abgeschlossene und optionale Aktivitäten, Koppelelemente zur synchronen Zusammenarbeit zwischen mehreren modellierten Rollen und Informationen. Damit ist es möglich, komplexe kooperative Arbeitsprozesse zu beschreiben. Die Bedienung interaktiver Systeme tritt hier in den Hintergrund.

BPMN – Business Process Modeling Notation

Die *Business Process Modeling Notation* (BPMN: OMG 2006) ist eine grafische Notationssprache für Geschäftsprozessmodelle. Sie ist von der OMG standardisiert. Geschäftsprozessmodelle können als quantifizierte *multi-user* Aufgabenmodelle angesehen werden. Sie bieten den Vorteil, dass Simulationswerkzeuge zur Analyse von Ressourcennutzung und Durchlaufzeiten verfügbar sind. In BPMN werden Geschäftsprozesse in Form von Events, Aktivitäten und Gateways modelliert. Die Informationsflüsse sind anhand von „*swim lanes*“ den handelnden Akteuren zugeordnet. Geschäftsprozessmodelle in BPMN können automatisch in BPEL (Business Process Execution Language) transformiert werden. BPEL kann direkt in entsprechenden Ablaufumgebungen ausgeführt werden. Dazu müssen die Aktivitäten und andere Elemente der Geschäftsprozesse algorithmisch vorliegen und entsprechende Aufrufschnittstellen z.B. in Form von Web-Services haben.

AMBOSS

AMBOSS (Mistrzyk & Szwillus 2008) ist ein Softwarewerkzeug, mit dem die Aufgabenbearbeitung in sicherheitskritischen Systemen grafisch modelliert werden kann. Dazu werden die folgenden Konzepte anhand einer graphbasierten Struktur repräsentiert: `Task`, `Connection`, `Riskfactor`, `Barrier`, `Object`, `Timevalues`, `Role`, `Message` und `Controlobject`. Aufgaben werden dabei hierarchisch untergliedert. Mehrere Akteure können als Problemlöser in unterschiedlichen Rollen modelliert werden. Die Problemlösung kann auch asynchron erfolgen. Nachrichtenaustausch zur Koordination oder Kooperation zwischen Akteuren kann repräsentiert werden. Mit AMBOSS ist ein Export des Aufgabenmodells in einem nicht standardisierten XML-Format möglich. Daneben können unterschiedliche Reports zur Analyse und Dokumentation des Aufgabenmodells erzeugt werden. AMBOSS enthält eine Simulationskomponente, mit der Durchlaufzeiten ermittelt werden können.

3.3.3. Human Reliability Analysis

Unter dem Begriff *Human Reliability Analysis* (HRA) wird eine Reihe von Verfahren zusammengefasst, die die Aufgabenabarbeitung durch Operateure mit Fehlerwahrscheinlichkeiten für bestimmte Tätigkeitstypen zusammenbringen. Mit ihnen soll zusammen mit *Probabilistic Risk Assessments* (PRA), also der Analyse Ausfallwahrscheinlichkeit technischer Komponenten in Anlagen und Maschinen, die Gesamtfehlerwahrscheinlichkeit von sicherheitskritischen Mensch-Maschine-Systemen analysiert werden. PRA und HRA basieren auf Ereignisbäumen, mit denen Aktivitäten und äußere Einflüsse, die ein Auftreten eines Fehlers ermöglichen und die Fehlerpropagation beeinflussen, repräsentiert werden.

Eine verbreitete HRA-Methode ist die *Technique for Human Error Rate Prediction* (THERP, Swain & Guttmann 1983). Die Analyse mit THERP läuft in fünf Schritten ab:

1) Identifikation der zu analysierenden Systemfehler (Systemfunktionen, die durch menschliche Fehler beeinflusst werden können)
2) Aufgabenanalyse in Bezug auf die identifizierten Systemfehler
3) Schätzung der Fehlerwahrscheinlichkeiten (menschliche Fehler)
4) Analyse der Auswirkungen menschlicher Fehler auf das Gesamtsystem (Integration in übergeordnete PRA), Schwachstellen identifizieren
5) Ableitung ergonomischer Gestaltungsempfehlungen

Dazu werden Ereignisablaufdiagramme benutzt. Sie stellen in einem binären Baum Aktivitäten der Aufgabenanalyse als Knoten dar. Jeder Knoten hat genau zwei Nachfolger: Zum einen eine im Erfolgsfall folgende Aktivität, zum anderen eine im Fehlerfall folgende Aktivität. Anhand der einzelnen Wahrscheinlichkeiten wird für jeden Weg durch den Baum die bedingte Wahrscheinlichkeit gemessen und als Erwartungswert auf einen normierten Zeitraum hochgerechnet (z.B. Ereignisse pro Jahr).

Die THERP stellt Tabellen mit basalen Aktivitäten (in GOMS-Benennung Operatoren, s. 3.3.5) zur Abschätzung der Fehlerwahrscheinlichkeiten zur Verfügung. Dabei wird eine anwendungsspezifische Taxonomie von Aktivitäten verwendet. Zumindest für den Bereich der Leitwartenbedienung in Kernkraftwerken und für die Flugführung liegen solche Tabellen vor.

THERP ist sehr gut dokumentiert, partiell verifiziert und breit akzeptiert (Reichart 1992). Wesentliche Nachteile der Methode sind jedoch die Unabhängigkeit der Betrachtung der einzelnen Aktivitäten und die ungenügende Berücksichtigung von leistungsbestimmenden Einflussfaktoren, die insbesondere die kognitiven Aspekte der menschlichen Verlässlichkeit betreffen.

Eine weitere Methode zur HRA ist die *Cognitive Reliability and Error Analysis Method* (CREAM, Hollnagel 1998). Sie unterscheidet sich von THERP durch die stärkere Betonung von kognitiv beanspruchenden Aktivitäten (z.B. diagnostizieren,

planen und problemlösen), wie sie in modernen Arbeitsumgebungen eine immer größere Rolle spielen.

Die kognitiven Aktivitäten nehmen deshalb in der CREAM-eigenen Taxonomie von menschlichen Fehlern und Fehlerursachen, auf der die Analysen beruhen, einen größeren Raum ein als bei THERP. Mit CREAM ist demzufolge eher eine Analyse der kognitiven Aspekte der ergonomischen Gestaltung von sicherheitskritischen Mensch-Maschine-Systemen möglich. Dies schließt im Gegensatz zu THERP explizit auch die Kontextabhängigkeit der meisten kognitiven Aktivitäten mit ein.

Es gibt spezialisierte Fassungen der CREAM für bestimmte Anwendungsdomänen (z.B. in der Fahrzeugführung: Driver Reliability and Error Analysis Method – DREAM und in der Schiffsführung: Bridge Reliability and Error Analysis Method – BREAM).

Die Analyse von kognitiv geprägten Aktivitäten in CREAM basiert auf dem *Contextual Control Model* (COCOM: Hollnagel 1993), einer Weiterentwicklung von *Cognitive Simulation Model of Human Decision Making and Behavior in Accident Management of Complex Plants* (COSIMO: Cacciabue et al. 1992). In COCOM werden kognitive Leistungen durch unterschiedliche Kontrollmodi gekennzeichnet. Es gibt den strategischen Modus (Plangenerierung), den taktischen (Abarbeitung von Prozeduren), den opportunistischen (reaktive Abarbeitung von Aktivitäten, die zu wahrgenommenen Reizen passen) und *scrambled* (zufällige Auswahl von Aktivitäten). Das Modell beinhaltet eine Abbildung einer Vielzahl kognitiv geprägter Tätigkeiten auf diese Kontrollmodi. Mit der Theorie, die COCOM zugrunde liegt, werden Modus-Übergänge erklärt. Wechsel zwischen Modi werden im Wesentlichen durch Veränderungen in der Aufgabenumgebung und nicht durch Informationsverarbeitung des Operateurs ausgelöst. Es werden aber die Erfolgswahrscheinlichkeit einer Aktivität, die zur Verfügung stehende Zeit und die Anzahl parallel anstehender Teilaufgaben berücksichtigt.

Hollnagel (1993) postuliert als Ziel für die Modellierung mit COCOM die Generierung minimaler bereichsspezifischer Modelle: „A minimal model is a representation of the main principles of the control and regulation that are established for a domain – as well as of the capabilities and limitations of the controlling system“. Obwohl der Begriff der Kognition und des kognitiven Modells für COCOM verwendet wird, handelt es sich deshalb hierbei lediglich um weniger detaillierte konzeptuelle Modelle (Definition s. Abschnitt 2.5), denn die Vorhersage von Abläufen auf der kognitiven Analyseebene steht nicht im Fokus dieser Methode.

3.3.4. Task Networks

Micro Saint (Bloechle & Schunk 2003), das darauf aufbauende IMPRINT (Mitchell 2000) sowie SANE Toolkit (Bösser & Melchior 1990), basieren auf der Simulation diskreter Ereignisse. Sie entstammen anwendungsorientierten Zielstellungen, z.B. bei der Simulation von logistischen Prozessen. Dieser Modellierungsansatz wurde

auf Aufgabennetzwerke („Task Networks“) übertragen. Eine Aufgabe wird in voneinander abhängige Teilaufgaben zerlegt, zu deren Bearbeitung begrenzte kognitive Ressourcen erforderlich sind. *Human Factors*-Aspekte werden in IMPRINT durch die Modellierung von Ressourcen in Anlehnung an das Modell von Wickens (1984) simuliert. Regelgeleitetes Verhalten wird durch bedingte Verzweigungskonstrukte zwischen Aufgabenteilnetzwerken abgebildet. In Micro Saint Sharp werden stochastisch gesteuerte, symbolisch fallunterscheidende und Aufgaben parallelisierende Verzweigungen unterstützt (Bloechle & Schunk 2003).

Daneben werden unterschiedliche Klassen von Petri-Netzen zur Analyse von Task Networks eingesetzt. Petri-Netz-Formalismen sind automatentheoretische Konstrukte, mit denen nebenläufige Prozesse modelliert werden können. Im grundlegendsten Fall besteht ein Petri-Netz aus *Stellen* und *Transitionen*. Es gibt eine vorgegebene Anzahl von *Marken*. Die Marken befinden sich an Stellen. Dabei haben Stellen eine begrenzte Kapazität zur Aufnahme von Marken. Eine Marke kann über eine Transition zwischen zwei Stellen verschoben werden, wenn die Transition schaltet. Transitionen schalten, wenn Bedingungen über die Füllung verbundener Stellen erfüllt sind. Ein Petri-Netz ist demnach ein gerichteter azyklischer Graph. Dieser Formalismus kann benutzt werden, um Arbeitsprozesse zu modellieren. Sie können z.B. in der Geschäftsprozessmodellierung zur Analyse verwendet werden. Konkrete Geschäftsprozesse werden dafür durch *Workflownetze* beschrieben. Das sind endliche Petri-Netze, mit ausgezeichneter Anfangsstelle und Endstelle, bei denen alle anderen Netzelemente auf Pfaden zwischen Anfang und Ende liegen (Martens 2004). Durch diese Formalisierung ist die Analyse mit der Graphentheorie möglich. Martens (2004) definiert auf dieser Basis eine Reihe von Eigenschaften, die Geschäftsprozesse aufweisen können und die für deren Implementierung wichtig sind. Die Stellen können z.B. zur Typisierung von Teilaufgaben herangezogen werden. Der Formalismus der als Workflownetze spezialisierten Petri-Netze lässt sich unter Angabe von Zusatzinformationen in andere formale Notationen von Geschäftsprozessen wie BPEL4WS transformieren.

Andere Erweiterungen von Petri-Netzen können verwendet werden, um *Human Factors*-orientierte Aufgabennetz-Modelle zu erstellen. Hierfür werden insbesondere *zeiterweiterte Petri-Netze* verwendet. Hierbei verbrauchen Transitionen beim Schalten eine Zeitdauer, die auch stochastische Eigenschaften haben kann. Ezzedine & Kolski (2005) verwenden für die Modellierung von kognitiven Aktivitäten in unterschiedlichen Situationen bei der Verkehrsüberwachung und -steuerung von Eisenbahnzügen *Objekt Petri-Netze*. Das sind Petri-Netze, bei denen die Marken wiederum Petri-Netze sind. Damit können nebenläufige Teilaufgaben modelliert werden: Das Petri-Netz auf der obersten Ebene wird zur Modellierung des Managements der Multitasking Aufgabe benutzt. Die Unter-Netze repräsentieren jeweils eine parallele Teilaufgabe. Der Objekt Petri-Netz Formalismus erlaubt die Definition unterschiedlicher Synchronisationsmechanismen.

Werther & Lorenz (2003) verwenden *eingefärbte zeitbehaftete Petri-Netze*. Dieser Formalismus führt eine Unterscheidbarkeit von Marken ein (z.B. anhand von Farben). Transitionsbedingungen können dadurch zwischen der unterschiedlichen Zugehörigkeit von Marken zu den verwendeten Markenmengen unterscheiden. Dies erlaubt ihnen die Einführung von Ressourcenbeschränkungen ähnlich dem Modell von Wickens (1984) bei der Beschreibung menschlicher Informationsverarbeitungsprozesse. Durch die Anwendung formaler Analysemethoden können charakteristische Eigenschaften wie die zielgerichtete Handlungsausführung oder der Umfang der Ressourcenbeschränkung in konkreten Arbeitsaufgaben auf einer quantitativen Ebene überprüft werden. Das Modell wurde dazu anhand empirischer Befunde aus Mikrowelt-Experimenten parametriert (Werther 2006).

3.3.5. GOMS-Familie von Aufgabenmodellierungsansätzen

Besondere Bedeutung hat die Familie der GOMS-Methoden zur Aufgabenmodellierung, denn diese Methoden haben eine große Verbreitung gefunden. Der Grund für die große Verbreitung dieser Methoden ist die Verfügbarkeit von Werkzeugen zur Modellierung und zur Simulation, die hohe Aussagekraft von GOMS-Aufgabenmodellen, da sie in Simulationen mit hohem Prädiktionspotenzial verwendet werden können, die leichte Erlernbarkeit der Methodik und der Notation und die gut dokumentierte theoretische und empirische Fundierung.

GOMS steht für „*goals, operators, methods, and selection rules*“ (Card et al., 1983). Die Familie der GOMS-Modellierungswerkzeuge basiert auf einer hierarchischen Aufgabenanalyse und der produktionssystem-basierten Modellierung der Aufgabe. Aufgaben auf allen Ebenen werden durch Ziele („*goals*“), die zur Erledigung der Aufgabe zu erfüllen sind, charakterisiert. Alternative Methoden („*methods*“) werden benutzt, um ein gegebenes Ziel zu erreichen. Um zwischen Zielerreichungsmethoden zu wechseln, werden Regeln („*selection rules*“) formalisiert, die aufgrund wahrgenommener Rückmeldungen oder anderer Bedingungen eine Methode auswählen. Eine Methode besteht dann aus einer vorher festgelegten Sequenz nicht mehr weiter zerlegbarer Teilhandlungen („*operators*“). Operatoren beschreiben sowohl motorische Aktivitäten (z.B. Button drücken etc.) als auch kognitive Prozesse (z.B. Erinnern eines Wertes). In Abhängigkeit von dem Modellierungsziel können sie auch nicht weiter aufgelöste komplexere Handlungen (z.B. Sprechen eines Satzes) zusammenfassen.

Operatoren werden Katalogen entnommen. Für alle Operatoren müssen im Vorfeld Ausführungszeiten und Fehlerraten experimentell bestimmt werden. Die Simulation der atomaren Aktionen, die durch Operatoren repräsentiert werden, ermöglicht auf der Basis eines hinterlegten kognitiven Modells wie MHP (*Model Human Processor*) oder EPIC (S. 28) die Vorhersage der Bedienbarkeit und Erlernbarkeit der Aufgabe. APEX (S. 77) als spezielles *Human Factors*-Analysewerkzeug beinhaltet darüber hinaus noch ein räumliches Modell der Aufgabe-

numgebung. Die Interaktion mit Elementen der Aufgabenumgebung kann so detailliert simuliert werden.

Die GOMS Methoden eignen sich für die Modellierung der Bedienung interaktiver Geräte (Hamacher & Hähnel 2001). Der Modellierer legt die Methoden mit ihren Operatorsequenzen, die Auswahlregeln und Ziele während der Modellerstellung fest (Kieras 1999). Der Formalismus ist vergleichsweise schnell zu erlernen, Modelle werden anhand von manuell erstellten Aufgabenanalysen oder vorliegenden formalen Spezifikationen des interaktiven Gerätes erstellt (John & Kieras 1997, Hamacher et al. 2002b). Das Ergebnis von Simulationsläufen mit GOMS-Modellen sind Kennwerte über die Erlernbarkeit einer Aufgabe (Länge der Methoden), die erwarteten Ausführungszeiten und Bedienfehlerraten (Kieras et al. 1995). Nimmt man diese Kennwerte als Maße für Effizienz und Effektivität, ist somit eine partielle Evaluation der Gebrauchstauglichkeit möglich (Hamacher et al. 2002a).

GOMS-Aufgabenanalysen sind auf unterschiedlichen zeitlichen Detaillierungsebenen sinnvoll. Auf der Ebene der übergeordneten Aufgabe handelt es sich um Zeitdauern von Tagen bis zehn Minuten. Die übergeordnete Aufgabe wird in Unteraufgaben auf der Minutenebene zerlegt (*Subtask*). Diese Teilaufgaben werden in einer *kognitiven Aufgabenanalyse* weiter auf der Ebene von zehn Sekunden dauernden *Unit Tasks* untergliedert. Auf einer noch feineren Ebene können für manche Analysezwecke *Mikrostrategien* im Sekundenbereich identifiziert werden.

John & Kieras (1997) nennen vier unterschiedliche Varianten von GOMS-Analysetechniken, die sich auch im Detaillierungsgrad hinsichtlich der zeitlichen Auflösung unterscheiden:

- CMN-GOMS: Die von Card, Moran & Newell (1983) vorgeschlagene Methode zur Aufgabenanalyse anhand einer hierarchischen Strukturierung in Ziele und Unterziele sowie die prozedurale Formulierung von Methoden, Operatoren und Auswahlregeln bildet den Kern der GOMS-Methoden. Mit CMN-GOMS können Ausführungszeiten von konkreten Abläufen vorhergesagt werden. Die Analyse ist auf den Detaillierungsgrad von *Unit Tasks* ausgerichtet.
- KLM: Das *Keystroke Level Model* ist eine vereinfachte GOMS-Fassung von Card et al. (1983). Es berücksichtigt nur Operatoren in Form von Tastatureingaben und Mausbewegungen. Mit einem Satz von einfachen Heuristiken werden mentale Operatoren ermittelt. Ziele, Methoden und Auswahlregeln werden nicht verwendet. Im Gegensatz zu CMN-GOMS ist KLM nur in einfachen Aufgabenumgebungen mit flacher Kontrollstruktur einsetzbar.
- NGOMSL: Natural GOMS Language (Kieras 1988) bringt den an Produktionsregeln orientierten Modellierungsansatz von CCT und den dort als Kontrollstruktur verwendeten Goalstack mit CMN-GOMS zusammen. So beinhalten die für NGOMSL vordefinierten Methoden interne mentale Operatoren, die CCT-Mechanismen wie Zugriffe auf das Arbeitsgedächtnis oder

Unterzielbildung für den GOMS-Formalismus repräsentieren. NGOMSL beinhaltet darüber hinaus einen Modellierungsprozess mit Heuristiken über Methodenkomplexität, Unterzielverarbeitung und die Verwendung mentaler Operatoren. NGOMSL erlaubt neben einer Vorhersage von Ausführungszeiten auch die Prognose von Fehlerwahrscheinlichkeiten und von Lernzeiten.

- CPM-GOMS: CPM-GOMS (John & Gray 1995) beinhaltet ein Ressourcenmodell. Operatoren werden parallel verfügbaren kognitiven, perzeptuellen und motorischen Ressourcen zugeordnet. Operatoren können parallel ausgeführt werden, wenn sie unterschiedliche Ressourcen belegen. Zwischen Operatoren werden wie in einem Netzplan logische Abhängigkeiten definiert, so dass die *Critical Path Method* zur Analyse von minimalen Ausführungszeiten bzw. zur optimalen Parallelisierung der Operatorsequenzen in CPM-GOMS benutzt werden kann. Der Detaillierungsgrad von CPM-GOMS-Modellen liegt im Bereich der Mikrostrategien.

Eine weitere GOMS-Methode ist PDL bzw. GOMS+ (Freed & Remington 2000). Aufbauend auf CPM-GOMS erlaubt sie die Spezifikation von Parallelität und Unterbrechbarkeit von Operatoren und Methoden in Multitaskingumgebungen mit zusätzlichen Konstrukten.

Alle GOMS-Methoden haben die Eigenschaft, dass mit ihnen nur hochautomatisiertes reaktives Verhalten beschrieben werden kann. Problemlösen oder leistungsbestimmende Einflüsse können in GOMS-Analysen nicht erfasst werden. Operatorenkataloge liegen für eine große Bandbreite von Aufgabenumgebungen vor.

3.3.6. Simulationsumgebungen

Im folgenden Abschnitt werden Simulationsumgebungen vorgestellt, mit denen quantitative Analysen der Mensch-Maschine-Interaktion aus Aufgabenmodellen abgeleitet werden können: MHP-GOMS, GLEAN, MIDAS und APEX (chronologisch sortiert).

MHP-GOMS

GOMS-Modelle sind mit entsprechenden Interpretern simulierbar. CMN-GOMS berücksichtigt als Simulator die einfache kognitive Architektur *Model Human Processor* (MHP, Card et al. 1983). In MHP sind einige ergonomische Gesetzmäßigkeiten, die für den erfassten reaktiven Typ von Aufgaben relevant sind, enthalten. So wird bspw. bei der Berechnung des Gesamttimings der Aufgabenabarbeitung Fitt's Law berücksichtigt.

Neben einer Reihe einfacher Editoren mit integriertem Simulator (z.B. Beard et al. 1996, Hamacher et al. 2002a, Wandmacher 2002) gibt es spezielle Werkzeuge, mit denen die Aufgabenerfassung und Modellierung für KLM vereinfacht werden soll.

Mit dem Werkzeug CRITIQUE (Hudson et al. 1999) können Logfiles der Interaktion von Benutzern mit grafischen Benutzungsoberflächen analysiert werden, um KLM Modelle zu erstellen. Dabei werden Interaktionsereignisse auf entsprechende Interaktionsoperatoren für Tastatur oder Maus abgebildet. Für die Verwendung mentaler Operatoren wird folgende Heuristik benutzt: Wenn Benutzer mit einem neuen Objekt der Anwendung zu arbeiten beginnen oder wenn sie die Interaktionsart (z.B. Tastatur oder Maus) mit einem Objekt ändern, wird ein neuer mentaler Operator erzeugt.

CogTool (John et al. 2004) ist ein weiteres Werkzeug zur automatischen Erzeugung von KLM-Modellen aus Interaktionslogfiles. CogTool ist eine Erweiterung von Macromedia Dreamweaver, mit der HTML-Prototypen und Storyboards von interaktiven Anwendungen analysiert werden können. Dazu demonstriert der Modellierer die Aufgabenbearbeitung innerhalb der Dreamweaver-Umgebung. Diese Interaktion wird aufgezeichnet und mit ähnlichen Methoden wie in CRITIQUE als Zustandsgraph repräsentiert. Dieses Aufgabenmodell wird nach ACT-Simple (s. 3.4.1, S. 88) übersetzt und ist so für ACT-R als Simulator zugänglich.

Liu et al. (2006) stellen Queueing Network-Model Human Processor (QN-MHP) vor, das als Komponente im Softwareprodukt *ProModel Optimization Software Suite* verwendet wird. QN-MHP ist eine Erweiterung des MHP um komplexe Wahrnehmung, Problemlösungs- und Handlungskomponenten, deren Teilaufgaben in der Art von CPM-GOMS in einem komplexen Abhängigkeitszusammenhang untereinander stehen. QN-MHP ermöglicht das Scheduling parallel anstehender Teilaufgaben mit einem einfachen Priorisierungsschema. QN-MHP wurde zur Simulation von Doppelaufgaben in der Kraftfahrzeugführung eingesetzt.

GLEAN

GLEAN (*GOMS Language Evaluation and ANalysis*: Kieras et al. 1995, Kieras 1999) ist eine Simulationsumgebung für GOMS-Aufgabenmodelle. Sie verwendet GOMSL, eine formalisierte Fassung von NGOMSL, zur Aufgabenbeschreibung. Der Simulator GLEAN enthält eine vereinfachte Version von EPIC (s. 3.2.2, S. 28) als kognitive Architektur.

GLEAN beinhaltet mit dem vereinfachten EPIC-Kern auch einen kognitiven Prozessor. Die Kontrolle der Verarbeitung erfolgt in diesem kognitiven Prozessor anhand eines Goalstacks. Das sehr einfache Arbeitsgedächtnismodell ist als *Hashmap* organisiert: Der GOMSL-Operator `store` speichert einen Wert anhand eines vorzugebenden Schlüssels (*tag*) ab. Andere GOMSL-Operatoren können auf Werte, die im Arbeitsgedächtnis gespeichert sind, durch Angabe eines entsprechenden *tags* zugreifen. Zusätzlich kann der GOMSL-Operator `delete` verwendet werden, um Werte aus dem Arbeitsgedächtnis zu löschen.

Zur Simulation des Mensch-Maschine-Systems muss neben dem Aufgaben-modell und der kognitiven Architektur, die für die Vorhersage des Benutzerverhaltens

verwendet werden, auch das Verhalten des technischen Systems berücksichtigt werden. Dazu enthält GLEAN eine Schnittstelle und Dummy-Implementierungen in C++ (GLEAN3, Kieras 1999), um einen *Device Behaviour Simulator* anzubinden. Der *Device Behaviour Simulator* wird benötigt, um das Timing des technischen Systems abzubilden. Darüber hinaus beinhaltet er auch eine Repräsentation des Zustandes des technischen Systems, so dass Systemantworten aus vorangegangener Interaktion des Benutzers abhängig sein können.

Zur Benutzung von GLEAN ist neben diesen beiden Teilsimulationen auch eine Auswahl von *Benchmark Tasks* vorzugeben. Das sind spezifische Aufgabeninstanzen in Form von Methodensequenzen, anhand derer die Aufgabenbearbeitung mit den korrekten Ausführungszeiten simuliert werden. *Benchmark Tasks* sollen unabhängig von der konkreten Realisierung der Benutzungsschnittstelle formuliert werden, damit ein simulativer Vergleich alternativer Schnittstellen (in Form unterschiedlicher *Device Behaviour Simulatoren*) möglich ist.

GLEAN3 ist als CommonLISP-Code mit einer systemspezifischen *Foreign Interface* zu *Device Behaviour Simulatoren* für akademische Zwecke erhältlich.

MIDAS

MIDAS (*Man Machine Interactive Design and Analysis System*: Smith & Tyler 1997) ist ein Modellierungssystem, das von der NASA zur Bewertung von Cockpit-Designs und Luftfahrtprozeduren entwickelt wurde (Corker 2000). MIDAS-Modelle bestehen aus der Simulation der Aufgabenumgebung und einem oder mehreren Bedienermodellen. Mit den simulierten Aufgabenumgebungen können auch ergonomische Aspekte untersucht werden. Zur Ausgestaltung der Aufgabenumgebungen können CAD-Daten verwendet werden, mit denen die jeweiligen Arbeitsplätze der Operateure modelliert werden.

Im Bedienermodell gibt es deklaratives und prozedurales Wissen. Das Bedienermodell interagiert mit der Simulation der Aufgabenumgebung. Interaktion wird im Operatormodell über ein Perzeptionsmodul realisiert. Es arbeitet auf der Ebene der Simulation von Blickbewegungen. Neben dem visuellen Kanal ist auditive Wahrnehmung über das Perzeptionsmodul möglich. Durch diese Art der Wahrnehmung werden Elemente der deklarativen Wissensbasis (UWR – *updateable world representation*) geändert. Zur Simulation der Interaktion mit einer externen Aufgabenumgebung gibt es weiterhin ein Motormodul, das auf der Basis anthropotechnischer Modelle Eingriffe in der Aufgabenumgebung bewirken kann.

Deklaratives Wissen wird als semantisches Netz von Objekten mit Attributen repräsentiert. Die Relationen zwischen den Objekten haben eine Gewichtung, die die semantische Nähe zwischen den repräsentierten Konzepten widerspiegelt. Eine exponentiell von der Zeit abhängige Vergessensfunktion löscht Objekte aus dem Speicher des Arbeitsgedächtnisses.

Prozedurales Wissen wird als Verknüpfung zwischen einem zu spezifizierenden Umgebungsereignis und der passenden Aktivität des Operateurs formalisiert. Für die Aktivität können zusätzlich Vorbedingungen vereinbart werden, die vor deren Aktivierung geprüft werden. Folgeaktivitäten können mit Regeln getriggert werden. Falls in einer Situation keine Regel zutrifft oder falls eine Abweichung des Zustandes der Weltrepräsentation von einem zuvor definierten Sollzustand wahrgenommen wird, wird ein deliberativer Entscheidungsprozess gestartet, durch den die Situation geklärt wird.

Geplante Handlungen werden in einem Warteschlangenmodell verwaltet. Die Warteschlangen werden von einem Steuerungsmodul (*Scheduler*) nach einem definierten *task load-model* abgearbeitet. Der *Scheduler* berücksichtigt *Multitaskingmerkmale*, wie Unterbrechbarkeit, Priorität von Tasks und erwartete Dauer der Ausführung von Handlungen.

Das System MIDAS wurde für zivile und militärische Anwendungen in Luft- und Raumfahrt entwickelt und ist nicht frei verfügbar.

APEX

APEX (*Architecture for Procedure Execution*: Freed 1998a, Remington et al. 2003) ist eine Simulationsumgebung für die Analyse von Mensch-Maschine-Systemen. Aufbau und Zielsetzung sind ähnlich zu MIDAS. Das System besteht aus einem Benutzermodell mit *Ressource Library*, der *Action Selection Architecture* und einer *Procedure Library* sowie der Aufgabensimulation.

In der *Ressource Library* sind Beschränkungen der Interaktionsmöglichkeiten zwischen Benutzermodell und Aufgabensimulation in Form von Sinneskanälen und motorischen Modalitäten repräsentiert. Die Interaktion zwischen Aufgabenumgebung und Benutzermodell wird über aufgabenspezifisch zu spezifizierende Funktionen abgebildet.

Im Gegensatz zu MIDAS können vorliegende Komponenten für das Umgebungsmodell z.B. CAD-Objekte nicht übernommen werden, sondern müssen in das APEX-Programm über ein LISP-API eingebunden werden.

Die *Procedure Library* enthält Handlungsprozeduren, die bei bestimmten Wahrnehmungen in der simulierten Aufgabenumgebung von der *Action Selection Architecture* getriggert werden. Alle getriggerten Prozeduren werden dann parallel abgearbeitet. Ein Scheduler übernimmt die Priorisierung, Auswahl und Wechsel zwischen parallel anstehenden Prozeduren. Während der Abarbeitung der anstehenden Aufgaben können neue Wahrnehmungen in der simulierten Umwelt weitere Prozeduren triggern.

Die Handlungsprozeduren sind auf der Basis von CPM-GOMS in PDL (*Procedure Definition Language*, Freed & Remington 2000) formuliert. Für das Scheduling gibt

es zusätzliche Informationen über Unterbrechbarkeit von Prozeduren, Überlappbarkeit von Schritten und zur Priorisierung von Verarbeitungsschritten.

Das System APEX ist frei verfügbar. APEX wurde bisher insbesondere eingesetzt, um eine Simulation der Informationsverarbeitungsbegrenzungen von Fluglotsen in eine sehr große verteilte Simulation der Kontrolle des US-amerikanischen Luftraums einzubringen (Lee et al. 2005).

3.3.7. Modellierte Phänomene

Mit den Ansätzen zur Analyse auf der Ebene der Aufgaben können nicht so detaillierte Phänomene betrachtet werden wie auf der Ebene der Kognitionsanalyse. Der Vorteil dieser Ansätze ist jedoch eine bessere praktische Anwendbarkeit. So ist der Ausbildungsbedarf für den Einsatz dieser Modelle wesentlich geringer und auch der Aufwand der Analyse und die Modellierung selber sind geringer.

Im folgenden Abschnitt werden die Phänomene vorgestellt, die trotz des geringeren Aufwandes analysiert werden können: Bedienbarkeit der Aufgabe, Ausführungszeiten, Fehler, Lernzeiten und Beanspruchung.

Bedienbarkeit der Aufgabe

Die Bedienbarkeit der Aufgabe ist eine Qualität des interaktiven Systems, dessen Bedienung die Aufgabe darstellt. Die Bedienbarkeit ist von den Aufgabeneigenschaften abhängig. Die Aufgabeneigenschaften sind jedoch unabhängig von der Leistung des Benutzers. Diese Qualität kann also ohne Informationen über den Benutzer und dessen Eigenschaften alleine aus dem Aufgabenmodell abgelesen werden.

Dazu wird u.a. die *Abdeckung* des Aufgabenmodells herangezogen (John & Kieras 1994): In allen GOMS-Methoden können jedem modellierten Ziel, das ein Benutzer bei der Bedienung des interaktiven Systems haben kann, die Ziele gegenübergestellt werden, die zu seiner Erreichung beitragen. Ein Aufgabenmodell hat eine volle Abdeckung, wenn zu jedem Ziel mindestens eine Methode gefunden werden kann.

Die *funktionelle Konsistenz* (Kieras 1988) ist eine Aufgabeneigenschaft, die ebenso die Bedienbarkeit einer Aufgabe beeinflusst. Es handelt sich jedoch nicht um einen numerischen Wert, sondern um eine qualitative Einschätzung. Sie wird in NGOMSL-Modellen ermittelt, indem Methoden und die Wissensstrukturen, die zu ihrer Bearbeitung erforderlich sind gegenübergestellt werden. Funktional konsistente Aufgaben sind solche, bei denen ähnliche Methoden zur Erfüllung ähnlicher Ziele benutzt werden. Das betrifft u.a. auch den strukturellen Aufbau von Methoden. Kieras (1988) gibt eine Reihe einfacher Heuristiken zur Ermittlung der Ähnlichkeit an.

Zu diesem Bereich zählen auch *Zielverarbeitungsfehler*, die aus einer funktional inkonsistenten Aufgabengestaltung resultieren. Gray (2000) stellt dazu eine

Taxonomie von *push*- und *pop*-Fehlern auf. Ein Beispiel für eine in diesem Sinne konsistente Aufgabengestaltung ist die Funktionsweise von Geldautomaten: Das Ziel ist aus Sicht des Benutzers Bargeld zu erhalten. In NGOMSL wird dieses Ziel als erreicht gekennzeichnet, wenn der Automat das Geld ausgibt. Wenn zur Abarbeitung der Methode dann noch weitere Schritte erforderlich sind (wie z.B. das Entnehmen der ec-Karte), handelt es sich um eine ungünstige Methodenstruktur, denn die letzten Schritte könnten vom Benutzer vergessen werden, da sein Ziel (Bargeld abheben) bereits erreicht ist. Deshalb ist die Aufgabengestaltung beim Geldautomaten so, dass Bargeld erst am Ende ausgegeben wird und das Benutzerziel damit auch als letzter Schritt erreicht ist.

Ausführungszeiten

Die Ausführungszeiten von Aufgaben hängen von der Sequenz der Operatoren ab, die zur Aufgabenbearbeitung benutzt werden, und von deren einzelnen Ausführungszeiten. Die a-priori Schätzung von Ausführungszeiten ist der originäre Einsatzzweck von allen Techniken der GOMS-Familie und von zeitbehafteten Aufgabennetzen.

Die Prädiktionskraft von GOMS wurde im Projekt Ernestine untersucht (Gray et al. 1993). Als Anwendung wurden zwei unterschiedliche Interaktionsentwürfe für Telefonbedienerarbeitsplätze in einer Telefonvermittlungszentrale verglichen. Es handelt sich dabei um komplexe Interaktionsaufgaben. Für beide Varianten wurden Aufgabenanalysen mit CPM-GOMS erstellt. Eine Variante war bereits implementiert und in Benutzung. Sie stellte die Baseline dar. Der andere Entwurf war augenscheinlich effizienter in der Bedienung. Die Frage war aber, wie exakt die GOMS-Modellvorhersagen wären. Die Modellsimulation und experimentelle Validierung zeigten jedoch längere Ausführungszeiten als nach Augenschein vermutet. Mit Hilfe des CPM-GOMS-Modells war eine Erklärung des Ergebnisses möglich. Die Validierung hat eine hohe Übereinstimmung der modellgestützt vorhergesagten mit den experimentell ermittelten Ausführungszeiten gezeigt.

Ein weiteres Beispiel für die erfolgreich überprüfte Vorhersage von Operatorsequenzen und Ausführungszeiten ist die Anwendung von CMN-GOMS auf Textverarbeitung (Card et al. 1983). Mit einem aufgabenmodellbasierten Vorgehen können Ausführungszeiten nur für trainierte Benutzer mit bereits stark automatisierten Handlungen vorhergesagt werden. Die kognitiven Prozesse, die bei ungeübten Benutzern bei der Auswahl von Bearbeitungsprozeduren und bei der Planung der Aufgabenbearbeitung ablaufen, können auf dieser Analyseebene nicht erfasst werden.

Fehler

Mit Aufgabenmodellen können menschliche Fehler auf unterschiedlichen Ebenen modelliert werden. GOMS ist beispielsweise zur Modellierung und Analyse von

fehlerhaftem Bedienverhalten in internetbasierten E-Commerce Anwendungen modifiziert worden (Wood 2000). Dazu wurde GLEAN um ein Modell der Fehlerbehandlung erweitert. Es werden hier fünf Zustände der Fehlerbehandlung bei der Aufgabenbearbeitung unterschieden (Wood & Kieras 2002):

1) *Error*: Der Bediener macht einen Fehler.
2) *Detection*: Der Benutzer wird der fehlerhaften Situation gewahr.
3) *Identification*: Der Benutzer identifiziert den Fehlertyp.
4) *Correction*: Der Benutzer korrigiert den Fehlereffekt.
5) *Resumption*: Der Benutzer führt die Aufgabe, die zum Fehler geführt hat, weiter.

Dazu wurde die GOMSL-Notation in GLEAN erweitert. Jede Methode erhält als neue Eigenschaft eine Spezifikation einer Fehlerkorrekturprozedur. Die Fehlerkorrekturprozedur wird ausgeführt, wenn eine Fehlersituation eintritt. Mit ihr wird die Fehlerbehandlung modelliert. Fehlersituationen können sich sowohl auf die Interaktion mit der Aufgabenumgebung beziehen (wenn beispielsweise eine Fehlermeldung vom Dialogsystem ausgegeben wird) als auch auf die interne Aufgabenabarbeitung (wenn beispielsweise die Kapazität des Arbeitsgedächtnisses nicht ausreicht, um einen früher gespeicherten Wert abzurufen).

Mit diesem Schema ist es möglich, Fehlersituationen in der Aufgabenmodellierung in Form einer Ausnahmebehandlung zu berücksichtigen. Die Vorhersage interner Fehler ist jedoch abhängig von der kognitiven Architektur des Simulationswerkzeuges. Im Fall von GLEAN ist das die vereinfachte Fassung von EPIC.

Die Schätzung von Fehlerwahrscheinlichkeiten auf unterschiedlichen Ebenen und von Teiltätigkeiten in ihrer Verkettung ist die originäre Aufgabe von HRA. Dadurch wird eine Gesamtfehlerwahrscheinlichkeitsschätzung im Rahmen der PSA (*probabilitic safety assessment*) möglich. Diese Verfahren werden regelmäßig im Bereich der Analyse von Kernkraftwerken eingesetzt. Daneben gibt es Beispiele für die Anwendung in chemischen Anlagen, der Flug- und Schiffsführung.

Eine Vorhersage von Timingfehlern, also nicht ausreichender Zeit für die Abarbeitung parallel anstehender Teilaufgaben, ist insbesondere mit zeitbehafteten Aufgabennetzwerken (IMPRINT, zeitbehaftete Petri-Netze) möglich. Eine solche Analyse ist bei den Techniken der GOMS-Familie nicht so einfach durchzuführen, da Aufgabenwechsel nicht gut modellierbar sind. Mit CPM-GOMS kann dies auf der Ebene der Mikrostrategien gemacht werden.

Lernzeiten

Mit CCT und NGOMSL bzw. GLEAN sind die Vorhersage von Lernzeiten und dem Wissenstransfer zwischen ähnlichen Aufgaben möglich. Die zugrunde liegenden Modelle sind anhand einer Reihe von Experimenten validiert.

Die Vorhersage der Lernzeiten stützt sich in NGOMSL auf die Anzahl der Schritte einer Methode. Die zum Lernen einer Methode erforderliche Zeit wird als linear abhängig von der Anzahl der Operatoren angenommen. Dabei können aber nur Operatoren berücksichtigt werden, die selber bereits gelernt sind. Eine Aussage über Lernzeiten von Methoden mit Operatoren, die noch nicht gelernt sind, ist mit NGOMSL nicht möglich.

Ein Beispiel für die in Hinblick auf die Vorhersage von Lernzeiten und Wissenstransfer erfolgreiche Aufgabenmodellierung beschreiben Lee et al. (1989). Anhand der Aufgabenmodellierung für die Benutzung eines digitalen Oszilloskops werden die Lernzeiten für unterschiedliche Benutzungsarten korrekt vorhergesagt. Ebenso wird der Wissenstransfer zwischen strukturell ähnlichen Benutzungsarten, der durch eine verkürzte Lernzeit operationalisiert wird, korrekt geschätzt.

Beanspruchung

Die Aufgabenmodellierung ermöglicht eine Zuordnung von Teilaufgaben zu Verarbeitungsressourcen. Ansätze wie CPM-GOMS, APEX oder zeitbehaftete Aufgabennetzwerke erlauben die Simulation der Ressourcenbelegung durch parallel ausführbare Teilaufgaben des Benutzers. Die Analyse erfolgt immer auf der Ebene der Mikrostrategien. Aus der Belegung der Ressourcen wird die kognitive Beanspruchung abgeleitet.

Dazu werden beispielsweise in APEX unterschiedliche Arten von Randbedingungen zwischen Teilaufgaben formuliert. Darüber steuert dann der Interpreter die Parallelisierung und Vermaschung der Teilaufgaben in einem Planungs- bzw. Schedulingprozess. Remington et al. (2003) unterscheiden drei unterschiedliche Arten von Randbedingungen, die in APEX verwendet werden können:

1) Logische Randbedingungen: Teilaufgaben, die von der erfolgreichen Beendigung logisch vorhergehender anderer Teilaufgaben abhängen, können erst beginnen, wenn alle vorhergehenden Teilaufgaben abgeschlossen sind und deren Ergebnisse vorliegen.
2) Ressourcen Randbedingungen: Teilaufgaben, die auf dieselbe nicht-teilbare Ressource zugreifen wollen, können nicht überlappend ausgeführt werden. Durch eine Priorisierung wird entschieden, welche Teilaufgabe zuerst ausgeführt wird.
3) Schlupfzeitnutzungsausschluss: Die Schlupfzeit ist beim Scheduling die Zeit, innerhalb derer eine Aktivität verschoben werden kann. Bei einigen kognitiven Operatoren wird ein Schlupfzeitnutzungsausschluss angenommen. Das bedeutet, dass sie trotz kurzzeitig zur Verfügung stehender Ressource nicht angewendet werden sollen, sondern erst wenn mehrere in einem Zusammenhang stehende Teilaktivitäten abgeschlossen sind, die die Ressource noch belegen werden.

In CPM-GOMS können die Verarbeitungsressourcen verwendet werden, die in MHP als kognitiver Architektur von CPM-GOMS definiert sind. MHP beinhaltet in der

Standardfassung drei parallel arbeitende Prozessoren. Das sind der Wahrnehmungsprozessor, der motorische Prozessor und der kognitive Prozessor. Die Prozessoren sind über Gedächtnisspeicher verbunden. So gibt es voneinander unabhängig ein visuelles und ein akustisches Arbeitsgedächtnis. Als getrennte Ressourcen stehen also die visuelle Wahrnehmung, die akustische Wahrnehmung, die kognitive Verarbeitung und die Motorik zur Verfügung. In MHP werden noch mehrere motorische Modalitäten unterschieden, die je nach Anwendung auch noch als unabhängige Ressourcen voneinander unterschieden werden können: Blickbewegung, Handbewegung und Sprechäußerung.

3.3.8. Anwendungsfelder und ihre Anforderungen

Die Aufgabenmodellierungsansätze wurden erfolgreich in unterschiedlichen Anwendungsdomänen eingesetzt.

Bürosoftware

Der Begriff „Bürosoftware“ soll als Anwendungsdomäne kennzeichnen, dass es sich um einen einzelnen Büroarbeitsplatz mit einem PC als Arbeitsgerät handelt. Die Software, die gemeint ist, nimmt nicht Online-Daten auf, die zur Kontrolle eines externen Prozesses genutzt wird, sondern wird ausschließlich vom Benutzer gesteuert. Beispiele für solche Anwendungen sind Textverarbeitung und die Recherche in Online-Hilfen oder Informationssystemen.

Insbesondere die Aufgabenmodellierung mit Techniken der GOMS-Familie wurde hier erfolgreich eingesetzt. John & Kieras (1997) nennen die Analyse vom Textverarbeitungssystem Xerox Star in den frühen 80er Jahren mit KLM, die zum endgültigen Design der Maus geführt hat. KLM wurde demnach auch erfolgreich zur Analyse der Gestaltungsalternativen für Telefonverzeichnisse bei Bell und für die Anpassung eines CAD-Systems an eine neue Einsatzumgebung verwendet.

Der Charakter dieser Aufgabendomäne erlaubt im Zusammenhang mit Methoden der Aufgabenanalyse nur eine Prüfung der Dialogstrukturen und -schritte. Insbesondere mit KLM können ohne großen Aufwand entsprechende Analysen durchgeführt werden. Aber auch Lernzeiten können mit NGOMSL vorhergesagt werden. Für diese Aufgabendomäne liegen Kataloge mit den relevanten Operatoren vor.

Command and Control

Unter der Anwendungsdomäne *Command and Control* (C2) werden Mensch-Maschine-Systeme subsumiert, bei denen die Aufgabe in der Kontrolle und Steuerung von autonomen Einheiten besteht. Dabei gibt es immer eine Anzeigeeinheit, die mit einer gewissen Ungenauigkeit und zeitlichen Verzögerung relevante Daten zu den autonomen Einheiten darstellt. Beispiele für C2-

Aufgabenumgebungen sind die Flugsicherung, militärische Führungssysteme und Einsatzleitstellen.

Für die Analyse von C2-Systemen kann auf der Ebene der Aufgabenmodellierung der Ansatz der Aufgabennetze verwendet werden. So sind z.B. farbige Petri-Netze zur Vorhersage des Timings und der Ressourcenbelastung der Lotsen in der Flugsicherung erfolgreich eingesetzt worden (Werther 2006).

Methoden der GOMS-Familie wurden ebenfalls erfolgreich bei der Analyse der Flugsicherung verwendet. So haben Freed et al. (1998a) APEX benutzt, um den Entwurf eines Flugsicherungssystems in Bezug auf mögliche Bedienfehler und auch Lernzeiten simulativ zu verbessern.

NGOMSL wurde zur Analyse eines Führungs- und Waffeneinsatzsystems (FüWES) auf einem Kriegsschiff der US-Navy benutzt. Es handelt sich um ein Radarsystem, das Umgebungsverkehr auf dem Wasser und in der Luft darstellt. Die Tracks müssen ausgewertet und als feindlich, neutral oder zu den eigenen Kräften gehörig erkannt werden. Im Fall einer Bedrohung dient das FüWES zur Steuerung der Bekämpfung. Mit NGOMSL wurden mögliche fehlerhafte Aufgabenunterbrechungen für diese Aufgabenumgebung simuliert (Santoro et al. 2000). Später wurde dieser Ansatz zur Modellierung von Teams in Einsatzführungszentren weiterentwickelt.

Auch KLM wurde für die Analyse eines C2 eingesetzt (*Space Operations Database System*, John & Kieras 1997). Ein weiteres Beispiel für die Anwendung von CPM-GOMS ist die Telefonvermittlung von NYNEX (Gray et al. 1993).

Obwohl C2-Systeme eine inhärente Dynamik haben und jede zu kontrollierende autonome Einheit der Aufgabenumgebung eine parallele Aufgabe darstellt, sind die Methoden der GOMS-Familie und der Aufgabennetze gut anwendbar. HRA ist jedoch im Bereich der C2-Systeme schwierig einzusetzen, weil die autonomen zu kontrollierenden Einheiten eine hohe Eigendynamik besitzen und damit einen großen Einfluss auf das Gesamtsystemverhalten haben. Deshalb ist es schwierig Fehlerbäume a-priori aufzustellen.

Prozesskontrolle in Leitwarten

Die Prozesskontrolle in Leitwarten hat eine andere Aufgabencharakteristik. Die zu kontrollierenden Aufgabenelemente in Form von Prozessvariablen hängen voneinander ab und ändern sich nicht autonom. Bei der Prozesskontrolle in Leitwarten existieren zudem in den meisten Fällen *Standard Operation Procedures* (SOPs), die die Aufgabenbearbeitung beschreiben. SOPs stellen eine Form der Aufgabenmodellierung dar. Insofern ist die Aufgabenmodellierung Teil des Entwurfs eines solchen Arbeitssystems. SOPs haben jedoch präskriptiven Charakter und können nicht benutzt werden, um Simulationen durchzuführen.

SOPs sind jedoch die Grundlage für die Aufstellung von Fehlerbäumen, die in HRAs bzw. PSAs benutzt werden. Insbesondere für die Kontrolle von Leitwarten von Kernkraftwerken sind PSAs vorgeschrieben. Auch im Bereich der chemischen Prozesstechnik gibt es PSAs.

Auch die Ansätze der GOMS-Aufgabenmodellierung wurden bei der Prozesskontrolle in Leitwarten angewendet. John & Kieras (1997) beschreiben eine NGOMSL-Analyse eines Produktionsplanungs- und -steuerungssystems und eine NGOMSL-Analyse eines Unterstützungssystems in einer Kernkraftwerkleitwarte.

Flugführung

Auch in der Flugführung liegen strikte SOPs vor, die die erforderlichen Flugführungsprozeduren komplett beschreiben. Diese SOPs erlauben eine Analyse mit Methoden der Aufgabenanalyse. Besonders wichtig ist die Beanspruchung von Piloten in kritischen Situationen. Dazu wurden bereits Aufgabennetzwerke, insbesondere IMPRINT, eingesetzt. Mitchell (2000) beschreibt beispielsweise WinCrew zur Analyse der kognitiven Beanspruchung von Helikopter-Piloten bei unterschiedlichen Gestaltungsalternativen. Daneben gibt es auch Beispiele für aufgabennetzwerkbasierte Analysen zur Beanspruchung bei Kampfpiloten (Craig et al. 2002). Auch MIDAS ist für Analysen solcher Aufgabenumgebung eingesetzt worden (Corker 2000).

Mit GOMS sind solche Aufgaben nicht so gut beschreibbar, weil eine Analyse der Beanspruchung auf der Ebene der Aufgabenanalyse die Modellierung der Mikrostrategien bedingt. Für eine praktikable Modellierung von Mikrostrategien sind SOPs in der Flugführung jedoch zu komplex und umfangreich. Modellierungstechniken aus dem Bereich der Aufgabennetze ermöglichen hier jedoch eine höhere Abstraktion, so dass auch umfangreiche Aufgaben modelliert werden können und in der Simulation Aussagen über die Beanspruchung gemacht werden können.

Kraftfahrzeugführung

Eine Aufgabenmodellierung der Kraftfahrzeugführung ist aufgrund der hohen externen Dynamik und der reaktiven Natur der Aufgabe nur schwer möglich. Eine Modellierung müsste auf der Ebene der Mikrostrategien ansetzen. Dafür ist die gesamte Aufgabe jedoch zu komplex und umfangreich. Aufgrund der externen Dynamik der Aufgabensituation handelt es sich bei der Kraftfahrzeugführung um eine Aufgabe mit vielen parallelen Teilaufgaben. Das Multitasking bzw. Scheduling zwischen diesen Teilaufgaben ist mit den Aufgabenmodellierungsansätzen nicht modellierbar.

Aufgabenmodellierung wird in diesem Bereich jedoch bei der Bedienung von *In Vehicle Information Systems* (IVIS) eingesetzt. Mit IVIS wird nicht die Kraftfahrzeugführung als solche unterstützt, sondern übergeordnete Aufgaben wie die Navigation oder unabhängige Aufgaben wie Telekommunikation oder die

Auswahl von Unterhaltungsdiensten. Die Steuerung von IVIS im Kraftfahrzeug kann zumindest im Stand, also ohne die Berücksichtigung der Fahraufgabe, mit GOMS Modellen analysiert werden (Hamacher & Hähnel 2002). Dazu müssen Operatoren, die zum jeweiligen Bedienungskonzept passen, separat erhoben werden (z.B. zentraler Dreh-/Drückschalter an der Mittelkonsole oder Taster am Lenkrad).

3.3.9. Zusammenfassung auf der Ebene Aufgabenanalyse

Der wesentliche Nutzen der Erörterung von Methoden und Werkzeugen zur Modellierung und Simulation auf der Ebene der Aufgabenanalyse besteht im Rahmen dieser Arbeit darin, das Anwendungspotenzial zu identifizieren und eine Auswahl von Methoden im Rahmen der Anwendung für die Lösung von Problemen im Bereich dynamischer Mensch-Maschine-Systeme zu ermöglichen. In Tabelle 3-2 ist eine bewertete Gegenüberstellung der Werkzeuge und Methoden zur Aufgabenanalyseebene zusammengefasst. Ihre Auswahl resultiert aus dem dokumentierten Einsatz für die anwendungsorientierte Interaktions- und Bedienermodellierung in technischen Systemen (Abschnitte 3.3.7 und 3.3.8).

KLM wird in der angewandten HCI-Gestaltung u.a. bei C2-Systemen verwendet. Ein Einsatz bei dynamischen Aufgabenumgebungen und die Kopplung mit externen Aufgabenumgebungen sind jedoch nicht vorgesehen. KLM hat nur einen sehr begrenzten Anwendungsbereich, ist aber sehr leicht zu verwenden. Deshalb kommt KLM für die Bedienermodellierung auf der Ebene einfacher Aufgaben in Betracht.

NGOMSL/GLEAN wird für die HCI-Gestaltung bei der Prozesskontrolle in Leitwarten und C2-Systemen verwendet. Handling von Dynamik und Multitasking sind nicht möglich, für die Kopplung steht ein *Foreign Interface* zu *Device Behaviour Simulatoren* zur Verfügung. Aufgrund der guten Verbreitung und leichten Handhabbarkeit im angestrebten Einsatzgebiet ist NGOMSL trotzdem als Kandidat für einen geeigneten Formalismus zur Bedienermodellierung auf dieser Analyseebene zu betrachten.

CPM-GOMS/APEX wird vereinzelt für die HCI-Gestaltung bei der Prozesskontrolle in Leitwarten, C2-Systemen und der Kraftfahrzeugführung verwendet, ist aber gegenwärtig mehr als Forschungsgegenstand denn als Werkzeug zu sehen. Handling von einfachen Multitaskinganforderungen ist bei hohem Modellierungsaufwand möglich, für die Kopplung mit externen Aufgabenumgebungen stehen Schnittstellen bereit. CPM-GOMS ist somit ein Kandidat für die Bedienermodellierung auf der Ebene der Aufgabenanalyse.

MIDAS wird bei NASA und US Militär im Bereich der Human Factors Analyse z.B. bei der Flugführung eingesetzt. Mit dynamischen Aufgabenumgebungen kann MIDAS durch die Modellierung von Metawissen umgehen. Das System wurde bereits in großem Maßstab an operationelle Simulatoren angebunden. Leider ist MIDAS jedoch nicht frei verfügbar.

Tabelle 3-2: Bewertete Gegenüberstellung der Aufgabenmodellierungsmethoden und Werkzeuge zur Aufgabenanalyseebene

Aufgaben-modellierungs-methode bzw. Werkzeug	Community	Anwendungs-bereich	Handling von Dynamik	Interaktion mit Aufgaben-umgebung
KLM	Verwendung in angewandter HCI-Gestaltung	Bürosoftware, C2	nicht möglich	nicht möglich
(N)GOMSL/ GLEAN	Verwendung in angewandter HCI-Gestaltung	Prozesskontrolle in Leitwarten, C2, Bürosoftware	nicht möglich	*Foreign Interface* zu *Device Behaviour Simulatoren*
CPM-GOMS/APEX	vereinzelte Verwendung in angewandter HCI-Gestaltung, Forschungs-gegenstand	Prozesskontrolle in Leitwarten, C2, Büro-software, Kraftfahrzeug-führung	schwer möglich	CommonLISP intern, externe Kopplung per LISP
MIDAS	Anwendung bei NASA, US Militär	Flugführung	möglich, Metawissen	umfangreiche Simulatoren angebunden
Aufgabennetze	Grundlagenforschung, zunehmend anwendungsorientiert	Flugführung, C2	bei Objekt Petri-Netzen möglich	keine Anbindung
HRA	Anwendung in PSA	Prozesskontrolle in Leitwarten	schwer möglich	keine Anbindung

Aufgabennetze und HRA können im Gegensatz zu den oben beschriebenen Systemen nicht in online-Simulationen verwendet werden, wo sie mit externen Aufgabenumgebungen gekoppelt werden müssten, sondern sind für den analytischen offline-Gebrauch konzipiert. Außerdem ist mit diesen Ansätzen die Modellierung wissensbasierten Verhaltens im Gegensatz zu MIDAS und den GOMS-Methoden (N)GOMSL und CPM-GOMS nicht möglich. Sie sind deswegen für den angestrebten Anwendungszweck nicht geeignet.

Für die Bedienermodellierung auf der Ebene Aufgabenanalyse sind also KLM für einfache sowie (N)GOMSL und CPM-GOMS für wissensbasierte Aufgaben als prinzipiell geeignete und verfügbare Systeme zu erörtern. Da auf der Ebene der Kognitionsanalyse ACT-R ausgewählt wurde und die Interaktion der Analyseebenen bei der Bedienermodellierung anzustreben ist, ist (N)GOMSL besser geeignet als CPM-GOMS, weil (N)GOMSL und ACT-R eine ähnliche Kontrollstruktur mit Zielverarbeitung aufweisen. Mit (N)GOMSL können zudem aufgrund des einfacheren Formalismus' komplexere Aufgaben als bei CPM-GOMS modelliert werden. Der in (N)GOMSL noch fehlende Umgang mit Dynamik wird in dieser Arbeit durch Erweiterungen verbessert und die Beschränkung von KLM auf einfache Aufgaben wird durch Einbettung in ACT-R überwunden (s.u.).

3.4. Synthese der Analyseebenen „Kognition“ und „Aufgabe“

Aus der Darstellung der vorangegangenen Abschnitte zum Stand der Forschung der modellgestützten Analyse auf der Ebene der Kognition und zur Aufgabenmodellierung ist ersichtlich, dass beide Ansätze Vorteile haben, die durch Nachteile erkauft werden: Die Aufgabenmodellierung ist vergleichsweise leicht einsetzbar. Das psychologische Grundlagenwissen ist in die Werkzeuge integriert. Bei der Anwendung sind nur begrenzte Kenntnisse dieser Grundlagen erforderlich. Dafür ist die Effektivität der kognitiven Modellierung größer. Es können sehr viel weitergehende Aussagen über mentale Prozesse gemacht werden, die sich auch in der Bearbeitung von Aufgaben in dynamischen Mensch-Maschine-Systemen auswirken. Beispiele für solche kognitiven Vorgänge sind Problemlösen, Lernen, individuelle Leistungsunterschiede, Ermüdung.

Deshalb ist gegenwärtig bei einigen Forschungsgruppen ein Ziel, die Vorteile beider Ansätze in *High-level Behavior Representation Languages* miteinander zu verbinden (Ritter et al. 2006a). Gegenwärtig gibt es dafür zwei methodische Wege: *Compilationsansätze* und *hybride Modellierungsansätze*.

3.4.1. Compilationsansätze

Bei den Compilationsansätzen wird ein Aufgabenmodell mit einem Übersetzer in ein Modell für eine kognitive Architektur transformiert. Dabei wird psychologisches Hintergrundwissen über die adäquate Umsetzung von Teilaufgaben auf der Ebene von *Unit Tasks* in kognitive Modellelemente verwendet, das im Übersetzer hinterlegt sein muss. Es gibt bereits mehrere Umsetzungen dieses Konzepts.

ACT-Simple

ACT-Simple (Salvucci & Lee 2003) ist eine KLM-artige Sprache für einfache Aufgabenmodelle, die mit ACT-R Makros formuliert wird. Dafür gibt es eine Reihe grundlegender perzeptueller, motorischer und kognitiver Kommandos, die auf Produktionsregeln in ACT-R abgebildet werden. Die Befehle sind: `move-hand`, `move-mouse`, `click-mouse`, `press-mouse`, `release-mouse`, `press-key`, `speak`, `look-at`, `listen` und `think`. Einige dieser Kommandos haben noch zusätzliche Parameter.

Aus jedem dieser ACT-Simple-Kommandos wird bei der Expansion dieser CommonLISP-Makros eine ACT-R Produktion generiert (`look-at` wird zu zwei Produktionen expandiert). Das Kommando `think` kann jedoch nicht zu dem dahinterstehenden kognitiven Vorgang expandiert werden, sondern hier wird eine „leere“ Produktion generiert, die eine definierte Zeitspanne wartet. Die Produktionen sind über dasselbe Ziel verknüpft, in dem eine Zustandsvariable über alle KLM-Schritte iteriert wird. Es handelt sich also um eine lineare ununterbrechbare Verkettung von Interaktionsschritten.

ACT-Simple wird in CogTool verwendet, um Analysen von Prototypen von Benutzungsschnittstellen in Form von Storyboards durchzuführen (John et al. 2004). Die ACT-Simple Statements werden direkt in einer modifizierten Variante von ACT-R ausgeführt. Die Modifikationen betreffen die Wahrnehmungs- und Motorikmodule, die durch die Änderungen direkt auf das Storyboard in CogTool zugreifen.

G2A

Bei G2A (*GOMS to ACT-R*, Amant & Ritter 2004) werden GOMSL-Modelle nach ACT-R 5.0 transformiert. Ritter et al. (2006b) berichten von einer Effektivitätssteigerung der Modellierung einer Aufgabe von einer Stunde in GOMSL gegenüber 50 Stunden in ACT-R.

Da der GOMS-Formalismus einfacher als der von ACT-R ist, gibt es Mehrdeutigkeiten bei der Transformation eines GOMSL-Konstruktes nach ACT-R. Alle Möglichkeiten für Operatortransformationen werden von G2A generiert, so dass im Nachhinein eine Auswahl der passenden Transformationsart erforderlich ist. G2A variiert dafür systematisch das Mapping von GOMS-Operatoren zu ACT-R-Produktionen.

Die Kontrolle wird wie in ACT-Simple über ein Ziel bewerkstelligt, dessen Slot von den generierten Produktionen entsprechend manipuliert wird, so dass die GOMS-Sequenz nachgebildet wird. Da GOMSL komplexere Kontrollstrukturen mittels eines Goalstacks zulässt, werden die dazu erforderlichen Verarbeitungsschritte ebenfalls in G2A nachgebildet.

ACT-Fly

Schoppek & Boehm-Davis (2004) verwenden eine Compilationstechnik, um ein ACT-R 4.0 Modell aus einem NGOMSL-Aufgabenmodell für eine Flugführungsaufgabe zu generieren. Ziel ist es, Fehlersituationen in der Flugführungsaufgabe zu simulieren. Der Aufgabe entsprechend liegt der Fokus auf der Modellierung von Entscheidungsprozessen mit einer hierarchischen Struktur im Gegensatz zum ACT-Simple Ansatz, der ohne Kontrollstruktur auskommt und für die Simulation von Interaktionsaufgaben geeignet ist.

Das resultierende Modell kann über eine Schnittstelle mit einer Flugsimulation interagieren. Es kann dabei jedoch nicht flexibel auf asynchrone Ereignisse in der dynamischen Aufgabenumgebung reagieren, sondern nur einen aus mehreren vorhandenen ausgewählten Plan abarbeiten und dabei in synchroner Weise mit der simulierten Aufgabenumgebung interagieren. Zur Repräsentation und Verarbeitung des Plans wurde eine flexiblere Kontrollstruktur eingeführt als in den anderen beschriebenen Ansätzen. Dazu wird das NGOMSL-Modell mit den Elementen Methode, Schritte, Operatoren in einer speziellen Chunk-Struktur repräsentiert. Der Chunk, der die Methode repräsentiert, hat einen Slot mit einem Verweis auf den ersten Schritt der Methode. Die Schritte einer Methode sind als Chunks repräsentiert,

die in einer verketteten Liste organisiert sind. Jeder Schritt-Chunk hat einen Verweis auf einen Chunk, der den dazugehörenden Operator repräsentiert. Der Operator hat eine direkte Realisierung als Produktion, die während der Verarbeitung ausgeführt wird. Der Methoden-Chunk hat außerdem noch einen Verweis auf den gerade auszuführenden Schritt. Das Ziel hat Verweise zur aktuell zu bearbeitenden Methode und zum aktuellen Operator-Chunk. Mit diesem Schema ist es möglich mit den subsymbolischen Features von ACT-R Auslassungen von Schritten zu modellieren, wenn Operatoren eine hohe Aktivierung durch häufige Benutzung haben.

Dieser Ansatz wird hier deshalb zu den Compilationsansätzen gezählt, weil ein Hilfswerkzeug in MS-Excel existiert, mit dem NGOMSL-Methoden-Definitionen halbautomatisch in dieses Verarbeitungsschema übertragen werden können.

Herbal

Herbal ist eine Werkzeugkette, die eine eigenständige High-Level Behavior Representation Language verwendet (Cohen et al. 2005). In der Herbal-Sprache, die auch in einer grafischen IDE unterstützt wird, wird der Prozess der Programmierung anhand einer expliziten Klassenhierarchie unterstützt (Ritter et al. 2006a). Hier werden insbesondere die Soar-Konzepte Zustand, Operator, *Elaboration*, *Impass*, Bedingung, Aktion und Arbeitsgedächtnis repräsentiert. Die Zielsprache von Herbal ist Soar. In der integrierten Entwicklungsumgebung werden entsprechende Soar-Agenten in eine Ausführungsumgebung geladen. Durch die Herbal-Umgebung wurde im Vergleich mit der Soar-Modellentwicklung in einem kontrollierten Experiment eine Beschleunigung um den Faktor drei beobachtet (Morgan et al. 2005 nach Ritter et al. 2006a).

HLSR

HLSR (*High Level Symbolic Representation Language*, Jones et al. 2006) ist eine abstrakte formal Sprache, mit der die kognitive Modellierung vereinheitlicht werden soll. Dazu wurden die Strukturen und Prozesse, die in mehreren kognitiven Architekturen vorkommen, erfasst und als Modellierungselemente z.B. in Form von Logiktabellen in HLSR bereitgestellt. Gegenwärtig wird die Sprache und ein dazugehörender Compiler entwickelt, mit dem aus hochsprachlichen HLSR-Spezifikationen kognitive Modelle in ACT-R und Soar generiert werden können.

3.4.2. Hybride Modellierungsansätze

Hybride Modellierungsansätze verbinden im Gegensatz zu den Compilations-techniken Elemente der Aufgabenanalyse mit kognitiven Modellen in einer einzigen Simulation.

IMPRINT + ACT-R

Ein hybrider Ansatz zur Modellierung auf Aufgabenebene und auf Kognitionsebene ist die Verbindung von IMPRINT mit ACT-R (Craig et al. 2002, Lebiere et al. 2002). Dieser Ansatz wurde zur Simulation von Kampfpiloten verwendet. Dazu wurde ein Aufgabennetzwerk in IMPRINT an bestimmten kritischen Stellen mit ACT-R „in die Tiefe“ simuliert. Hierzu wird am Beginn solch einer Detailsimulation als Teil eines übergeordneten Aufgaben-Netzwerkes ein ACT-R-Modell mit dem gegenwärtigen mentalen Zustand geladen, dann einige Sekunden in hoher Granularität simuliert, also z.B. unter Berücksichtigung von Beschränkungen des Arbeitsgedächtnisses, bis das Detailmodell eine bestimmte Aktion auslöst, die wiederum an die übergeordnete Abarbeitung des Aufgabennetzwerkes zurückgespeist wird. Die Detailsimulation mit dem ACT-R-Modell wird dann beendet und die übergeordnete Simulation mit dem Aufgabennetzwerk geht weiter. Die Detailmodellierung in ACT-R hat einen hohen Modellierungsaufwand. Die Netzwerksimulation ermöglicht aber eine Skalierung des Modellierungsansatzes (Lebiere et al. 2003).

Ansatz von Kontogiannis (2005)

Kontogiannis (2005) stellt ein weiteres hybrides Modell vor. Dabei wird ein Aufgabennetzwerk mit einem kognitiven ad hoc Modell verbunden. Damit wird eine flexiblere Verarbeitung mit dem Ziel gewährleistet, das Aufgabennetzwerk beispielsweise an sich ändernde Aufgabenanforderungen anzupassen. Dazu werden eingefärbte Petri-Netze verwendet, die über eine API mit dem kognitiven Modell verknüpft sind. Das kognitive Modell ist in der Lage, mit regelbasierter Informationsverarbeitung Entscheiden, Multitasking und Fehlerbehandlung durchzuführen. Das kognitive Modell besteht aus zwei Modulen zur Aufmerksamkeitssteuerung und unterschiedlichen Gedächtnisspeichern, in denen z.B. zurückgestellte parallel anstehende Teilaufgaben oder Pläne abgelegt werden. Mit diesem kognitiven Modell können Kennwerte wie Beanspruchung oder Ermüdung online, d.h. unter Berücksichtigung der aktuellen Situation in der simulierten Aufgabenumgebung erhoben werden.

3.4.3. Zusammenfassung der Ebene Synthese „Kognition“ und „Aufgabe“

Compilations- und hybride Ansätze sind nützliche Erweiterungen für die anwendungsorientierte Bedienermodellierung, da sie durch die Abstraktion von Prozessen die Effizienz (auf der Aufgabenebene) der Modellierung mit hoher Effektivität (auf der Ebene Kognition) der Simulation vereinen. Die verfügbaren Prototypen haben jedoch noch keinen hohen Reifegrad erreicht, sondern sind experimentell und teilweise sogar anwendungsspezifisch in der Art wie die Ebenen zusammenspielen (hybride Modellierungsansätze).

In dieser Arbeit wurde ein neuer Compilationsmechanismus entwickelt, mit dem die Modellierung der Interaktion von Modell bzw. simuliertem Bediener und

Aufgabenumgebung besonders effizient realisiert werden kann. Eine solche Abstraktionsschicht ist für ACT-R noch nicht vorhanden gewesen.

Weiterhin muss erörtert werden, ob die auf den anderen Analyseebenen bereits ausgewählten Modellierungssysteme ACT-R und KLM bzw. (N)GOMSL geeignet synthetisiert werden können.

ACT-R und (N)GOMSL sind in den verfügbaren Systemen G2A und ACT-Fly synthetisiert. Beide wurden jedoch in dieser Arbeit nicht verwendet, weil sich in den Fallstudien keine Erfordernis dazu ergeben hatte. Es hat sich gezeigt, dass in realen Aufgabenumgebungen die Aufgabenmodelle für eine Compilation zu weit vom kognitiven Modell entfernt sind. NGOMSL ist demnach nur in der Modellkonzeption nützlich.

ACT-R und KLM werden in ACT-Simple verschmolzen. KLM-Beschreibungen werden in ACT-Simple nach ACT-R transformiert. Die Anwendbarkeit ist jedoch auf sehr einfache Abläufe beschränkt. Im resultierenden ACT-Simple Modell kann der Ablauf aufgrund der Zielstruktur dann auch nicht unterbrochen werden – Multitasking ist so nicht möglich. In dieser Arbeit wird folglich eine andere Synthese von ACT-R und KLM erarbeitet, die Multitasking-fähig ist. Dieses System hat sich in der Bedienermodellierung als nützlich erwiesen.

4. Darstellung der Methodik

Tabelle 4-1 stellt eine Einschätzung von Freed et al. (1998a) über die Einsetzbarkeit von unterschiedlichen Methoden zur Bewertung von Mensch-Maschine-Systemen dar. Benutzertests können erst anhand existierender Systeme, deren Prototypen oder Mockups durchgeführt werden. Insbesondere bei verteilten Mensch-Maschine-Systemen sind Benutzertests nur sehr aufwändig durchzuführen. Dafür haben sie eine hohe Effektivität, da ein hoher Anteil der vorhandenen Benutzungsprobleme identifiziert werden kann. Guidelines wie z.B. *Style Guides* und *Cognitive Walkthroughs*, also Expertenbeurteilungen, sind zwar früh im Entwicklungsprozess einsetzbar, haben aber eine eingeschränkte Aussagekraft insbesondere in komplexen und interaktiven Aufgabenumgebungen. Durch den Einsatz von Simulatoren kann hingegen auch in komplexen Aufgabenumgebungen ein relativ großer Nutzen erzielt werden. Ein weiterer Vorteil der simulationsgestützten Analysemethode liegt in der im Vergleich mit den anderen Methoden höheren Geschwindigkeit, mit der Designalternativen während des Entwicklungsprozesses halbautomatisiert bewertet werden können.

Tabelle 4-1: Eignung unterschiedlicher Methoden zur Analyse von Mensch-Maschine-Systemen nach Freed et al. (1998a)

Methode[2]	Zeitpunkt im Entwicklungsprozess	Kosten von Designänderungen	Kosten des Methodeneinsatzes	Eignung für anspruchsvolle Umgebungen	Effektivität der Methode
Benutzertests	spät	–	–	+	+
Anwendung von Guidelines	früh	+	+	–	–
Cognitive Walkthrough	früh	+	+	–	0
Bedienermodellierung und Simulation	früh	+	0	+	0

Das Ziel dieser Arbeit ist, Simulation auf der Basis von Bedienermodellierung im Vergleich zu den anderen Methoden attraktiver zu machen. Dazu sollen sowohl die

[2] + = niedrige Kosten, 0 = mittlere Kosten, – = hohe Kosten,
+ = hohe Eignung, – = geringe Eignung,
+ = hohe Effektivität, 0 = mittlere Effektivität, – = niedrige Effektivität

Kosten des Einsatzes solcher Simulationen verringert werden, indem die Bedienermodellierung vereinfacht wird, als auch die Effektivität der Simulation erhöht werden, indem der Einsatzbereich auf Analysebereiche ausgedehnt wird, in denen solche Simulationen bisher nicht anwendbar waren.

Um dieses Ziel zu erreichen, werden in dieser Arbeit anhand von drei Fallstudien konkrete Lücken identifiziert, die die existierenden Werkzeuge und Methoden für die anwendungsorientierte Bedienermodellierung in dynamischen Mensch-Maschine-Systemen haben. Die Methode der Fallstudie beinhaltet in der Software Engineering Forschung die Beobachtung, Beschreibung und Analyse eines interessierenden Aspekts der Softwareentwicklung an einem konkreten Beispiel aus der Praxis z.B. in Form von Machbarkeitsstudien, Technologiepotenzialabschätzungen oder Werkzeugerprobungen (Höfer & Tichy 2007). Fallstudien wurden bereits zur Bewertung von Methoden und zur Ableitung von neuen Anforderungen an zukünftige Entwicklungen in der Bedienermodellierung benutzt (Ritter et al. 2000). Die Erkenntnisse, die aus Fallstudien gewonnen werden, haben eine hohe ökologische Validität, da jeder Fall in seinem gesamten Einbettungskontext betrachtet wird und dessen reale Umstände berücksichtigt werden. Beobachtungen sind jedoch schwerer zu generalisieren als bei kontrollierten Experimenten, in denen der Wirkzusammenhang von unabhängigen auf abhängige Variablen leichter nachgewiesen werden kann. Nichtsdestotrotz ist die Fallstudienmethode im Software Engineering die am häufigsten eingesetzte Forschungsmethode, um professionelle Softwareentwicklung wissenschaftlich zu untersuchen (Höfer & Tichy 2007).

Innerhalb der Fallstudien werden auch im Rahmen dieser Arbeit neu entwickelte softwaretechnische Methoden und Werkzeuge erprobt und auf ihre Eignung hin untersucht, mit denen die identifizierten Lücken geschlossen werden sollen. Die Auswahl des Falls ist von entscheidender Bedeutung für die Aussagekraft der abgeleiteten Erkenntnis (Eisenhardt & Graebner 2007). Die wesentlichen Anforderungen an einen guten Fall sind dessen Relevanz für eine praktische Fragestellung, eine möglichst große Generalisierbarkeit, die durch die Auswahl eines prototypischen Vertreters einer ganzen Klasse gleichartiger Fallsituationen erreicht wird, und die Beobachtbarkeit von relevanten Parametern, aus deren Ausprägung Rückschlüsse auf den zu untersuchenden Sachverhalt geschlossen werden können.

Die Zielsetzung dieser Arbeit ist, möglichst breit die Probleme im Bereich der Bedienermodellierung für dynamische Mensch-Maschine-Systeme zu identifizieren und Lösungen dafür zu entwickeln. Dazu müssen die Fallstudien hier so gewählt werden, dass sie ein großes Spektrum der Herausforderungen der Bedienermodellierung für dynamische Mensch-Maschine-Systeme abdecken. Die Fallstudien müssen also im Sinne einer zu definierenden Metrik ausgewogen verteilt werden.

Als wesentliche Hindernisse beim Einsatz der Bedienermodellierung auf der Ebene der Kognition und Aufgabenanalyse sind die Dynamik und Komplexität der Situation und die Abbildung der Interaktion zwischen Aufgabenumgebung und Bedienersimulation anzusehen. Aufbauend auf ergonomischem Basiswissen wird ein Framework für die Beschreibung solcher Eigenschaften von Aufgabenumgebungen definiert. Anhand dieses Frameworks wird dann eine Reihe von Anwendungsdomänen zur Auswahl der Fallstudien charakterisiert. Unabhängig von Dynamik und erforderlichen Interaktionsmodi dieser Anwendungsdomänen ist auch der Zweck der Modellierung ein wesentlicher Faktor, der die Art der Bedienermodellierung mitbestimmt. Die Fallstudien werden so ausgewählt, dass anhand dieser Bestimmungsstücke eine ausgewogene Verteilung erreicht wird. Durch dieses Vorgehen wird eine breite Abdeckung des möglichen Problemraums erzielt.

4.1. Framework für die Charakterisierung der Domänen

Mit dem Framework sollen wesentliche Merkmale von Mensch-Maschine-Systemen charakterisiert werden, mit denen Komplexität und Dynamik von Aufgabenumgebungen beschrieben werden können. Abbildung 4-1 zeigt die Elemente dieses Frameworks. Das Framework besteht aus den Bereichen Situation, Aufgabenmodell, kognitives Anforderungsprofil und Mensch-Maschine-Interaktion.

Situation: Die *Aufgabenumgebung* beinhaltet ein *System* mit *Elementen* und *Relationen* zwischen Elementen. Sowohl Elemente als auch Relationen des Systems können *Aufgabenobjekte* darstellen. Das sind Entitäten der Situation, die für die Bearbeitung der Aufgabe relevant sind. Aufgabenobjekte haben einen Zustand, der sich während der Aufgabenbearbeitung ändern kann. Beispiele für zustandsbehaftete Aufgabenobjekte sind in verfahrenstechnischen Anlagen Prozessvariablen mit ihrem Wert oder in der Flugsicherung Luftfahrzeuge mit ihrem Rufzeichen und ihrer Höhe. Ein Beispiel für ein Aufgabenobjekt, das aus einer Relation resultiert, ist ein antizipierter Konflikt zwischen zwei Luftfahrzeugen mit seinem vermuteten Ort und seinem geschätzten Zeitpunkt.

Aufgabenmodell: Die Aufgabenumgebung besteht aus einer Reihe von *Aufgaben*, die in Teilaufgaben zergliedert sein können. Obwohl sich Aufgaben grundsätzlich aus dem System ableiten, haben sie einen stärkeren Bezug zur Aufgabenumgebung, denn die konkrete Ausprägung des Aufgabensatzes resultiert aus der konkreten Gestaltung der Aufgabenumgebung. Jede Aufgabe besitzt ein Ziel, das mit ihrer Bearbeitung erfüllt werden soll. Aufgaben werden in *Operationen* zergliedert. Die weitergehende Taxonomie aus GOMS mit Methoden, Schritten, Operatoren und Auswahlregeln wird in diesem Zusammenhang nicht verwendet, weil sie für die Beschreibung von Komplexität und Dynamik von Aufgabensituationen keinen Mehrwert bietet.

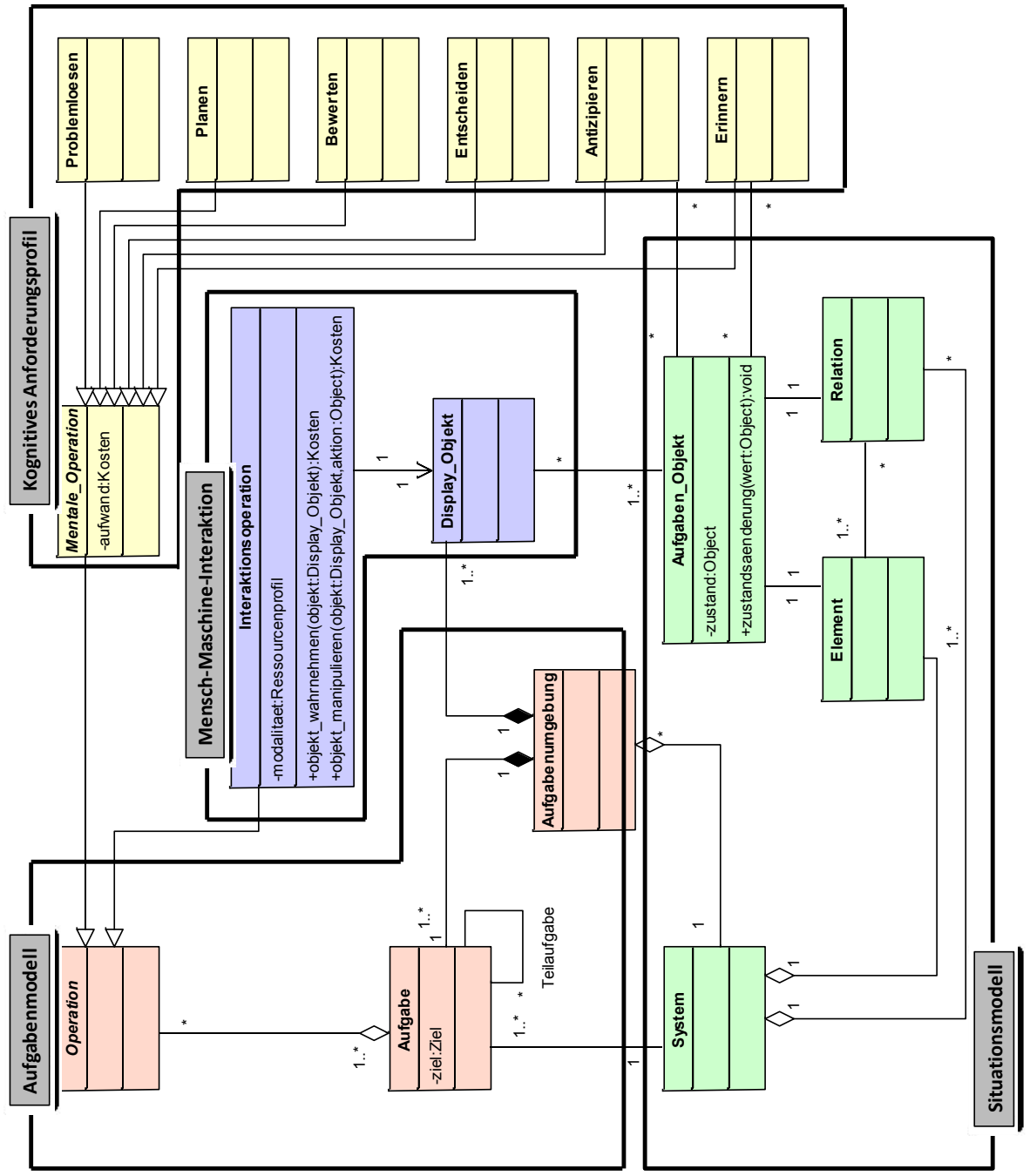


Abbildung 4-1: Framework zur Beschreibung von Dynamik und Komplexität in Aufgabenumgebungen als UML-Klassendiagramm

Kognitives Anforderungsprofil: Operationen sind in diesem Framework ein abstrakter Sammelbegriff für *mentale Operationen* und *Interaktionsoperationen*. Mentale Operationen sind *Problemlösen*, *Planen*, *Bewerten*, *Entscheiden*, *Antizipieren* und *Erinnern* (entsprechend der in Abschnitt 2.1 eingeführten Systematik). Entsprechend

der Aufwandskosten der mentalen Operationen wird eine kognitive Last bei der Aufgabenbearbeitung erzeugt. Denn die mentalen Operationen beanspruchen den Bediener durch ihren Verarbeitungsaufwand. Die konkreten mentalen Operationen sind unterschiedlich komplex. Diese Operationen sind zwar nicht unabhängig voneinander (der Vorgang der Antizipation beinhaltet bspw. das Erinnern), die Abhängigkeiten sind jedoch für den angestrebten Modellierungszweck – die Beschreibung von Komplexität und Dynamik von Aufgabensituationen – nicht relevant und werden hier nicht erfasst.

Mensch-Maschine-Interaktion: Interaktionsoperationen entsprechen KLM-Operatoren. Sie beziehen sich auf eine konkrete Gestaltung der Aufgabenumgebung. Dabei beschränkt sich dieses Beschreibungsframework auf computervermittelte bzw. anzeigenbasierte Mensch-Maschine-Schnittstellen. Mensch-Maschine-Systeme, in denen die Körperwahrnehmung des technischen Systems durch den Bediener zur Diagnose des Systemszustandes eine wesentliche Rolle spielt („body impact“), werden von diesem Schema nicht erfasst. In der konkreten Gestaltung der Aufgabenumgebung werden die Aufgabenobjekte in *Displayobjekten* abgebildet. Displayobjekte umfassen im Rahmen dieses Frameworks neben der Anzeige des Aufgabenobjektes mit seinem aktuellen Zustand auch Bedienelemente, mit denen das Aufgabenobjekt beeinflusst werden kann. In der Gestaltung der Aufgabenumgebung ist es möglich ein Aufgabenobjekt durch mehrere unterschiedliche Displayobjekte anzuzeigen. Analog zu den Aufwandskosten der mentalen Operationen haben die Interaktionsoperationen Kosten, die sich aber auf ein komplexeres Ressourcenprofil beziehen können. In viel größerem Maß als bei den mentalen Operatoren ist der Aufwand für die Interaktionsoperationen abhängig von früheren Interaktionsschritten, denn je nach Gestaltung der Mensch-Maschine-Schnittstelle können unterschiedlich weite Mausbewegungen oder z.B. im Fall von Prozessleitsystemen sogar Seitenwechsel erforderlich sein. Deshalb liefern die Methoden der Interaktionsoperationen einen entsprechenden Wert zurück.

Assistenzfunktionen und Automatisierung werden in diesem Framework nicht als eigenständige Elemente modelliert. Stattdessen werden sie als Teil des Systems angesehen, und ihre Funktionsweise wird in der Aufgabengestaltung berücksichtigt. Hoch automatisierte Systeme unterscheiden sich im Rahmen dieses Frameworks deshalb von weniger automatisierten durch eine tendenziell geringere Anzahl mentaler und Interaktionsoperationen.

4.1.1. Komplexität

In diesem Framework können nun wesentliche Eigenschaften von Mensch-Maschine-Systemen beschrieben werden. Die Komplexität eines Systems wird oft anhand der Anzahl der Elemente beschrieben. Daneben tragen die Relationen zwischen den Elementen eines Systems besonders zu dessen Komplexität bei

(Urbas & Leuchter 2002). Im Rahmen des vorgestellten Frameworks werden diese beiden Größen auf die Zahl der Aufgabenobjekte der Aufgabenumgebung abgebildet.

4.1.2. Dynamik

Die Ansätze der Regelungstechnik zur Beschreibung von Systemdynamik sind geschlossen formalisiert. Sie eignen sich jedoch nicht für die Klassifizierung von Aufgabenumgebungen, um eine breite Palette von Problemen der Bedienermodellierung zu identifizieren (Leuchter & Urbas 2003, 2004b). Stattdessen werden im Folgenden für die Modellierung Merkmale der Dynamik von Aufgabenumgebungen entwickelt, die hier zur Unterscheidung genutzt werden können.

Dynamik bezieht sich auf die Änderung des Zustands von Aufgabenobjekten und auf die Änderung der Menge der Aufgabenobjekte. Beispiele für dynamisches Verhalten in der Flugsicherung sind das Einfliegen neuer Luftfahrzeuge in den zu kontrollierenden Sektor (neue Aufgabenobjekte) und die Änderung der Flughöhe eines Luftfahrzeuges im Anflug auf einen Flughafen (Änderung des Zustands eines Aufgabenobjekts). Die folgenden Dynamikeigenschaften haben einen Einfluss auf Modellierungsansätze:

- Die Modellierung der Situationsrepräsentation ist abhängig davon, ob die Menge der Aufgabenobjekte konstant (strukturinvariante Aufgabenumgebung) ist oder nicht (strukturell dynamische Aufgabenumgebung).
- Für die Modellierung der Verarbeitungsprozesse ist es ein wesentlicher Unterschied, ob Änderungen im System der Aufgabenumgebung *kontinuierlich oder diskret* sind.
- Die erforderliche Leistungsfähigkeit der Simulation ist abhängig von der *Geschwindigkeit* der Änderungen. Sie bezieht sich bei kontinuierlicher Dynamik auf die Größe der Änderung und bei diskreter Dynamik auf die Anzahl der Änderungsereignisse pro Zeiteinheit.
- Für die Modellierung der Kontrolle ist die *Vorhersehbarkeit* der Änderungen des Zustandes von Aufgabenobjekten entscheidend. Sie ist abhängig vom Grad der *Eigendynamik* des Systems, deren Auswirkung sich in der Größe des *Prädiktions-horizontes* zeigt. So wird der Prädiktionshorizont z.B. von der Wahrscheinlichkeit externer Störungen beeinflusst (Urbas & Leuchter 2002).
- Der Grad der *Verzögerung* ist ein Merkmal, das die Modellierung der Überwachung beeinflusst. Das betrifft sowohl die Verzögerung, die zwischen der Änderung von Aufgabenobjekt und der Aktualisierung vom dazugehörenden Displayobjekt auftreten kann, als auch Totzeiten, die die erwartete Zustandsänderung eines Aufgabenobjektes nach einer Interaktionsoperation verzögern.

Die Kontrollstruktur einer Aufgabenumgebung ist sowohl abhängig von der Situation als auch von der Aufgabenstruktur. Sie ergibt sich aus den Zielen, die während der Aufgabenbearbeitung verfolgt werden.

Es gibt unterschiedliche Arten von Zielen. Aasmann (1995) unterscheidet die folgenden Dimensionen:

- Erreichbar vs. homeostatisch: Ziele, die erreichbar sind, werden im Laufe der Aufgabenbearbeitung gelöst. Homeostatische Ziele haben einen steuernden Charakter, sie werden nicht erreicht, sondern bleiben erhalten, um so die Aufgabenbearbeitung zu steuern.
- Top-level vs. Unterziel: Ziele können von ihrer Art her auf der obersten Ebene angesiedelt sein, dann haben sie eine direkte Entsprechung in der Struktur der Aufgabenumgebung, oder es kann sich um Unterziele handeln, die aufgrund der Bearbeitung eines Problems entstehen und temporär sind.
- Langzeit vs. Kurzzeit: Ziele können lange bestehen bleiben, auch wenn es sich um erreichbare Ziele handelt.
- Multiple top-level vs. single top-level: Die Kontrollstruktur einer Aufgabenumgebung kann sich auf ein einzelnes oder aber mehrere parallele top-level Ziele beziehen.

Bei der Bedienermodellierung in dynamischen Mensch-Maschine-Systemen ist die Behandlung von Multitasking, also dem Umgang mit mehreren parallelen homeostatischen und Langzeitzielen, eine Anforderung, die aus der Kontrollstruktur der Aufgabenumgebung erwächst. Die parallelen Ziele beziehen sich entweder auf mehrere Hauptaufgaben die verfolgt werden müssen oder mehreren Aufgabenobjekten die die Situation ausmachen. Die Art der möglichen Behandlung von Multitasking ist abhängig von der Dynamik der Aufgabenumgebung. Sie bestimmt, welche Verfahren zum Scheduling und zur Priorisierung der parallel anstehenden Verarbeitungserfordernisse verwendet werden müssen, ob im laufenden Betrieb umgeplant werden muss und ob die Aufgabenbearbeitung unterbrechbar ist.

4.1.3. Interaktion in reaktiven oder deliberativen Aufgabenumgebungen

Die Aufgabenumgebungen unterscheiden sich auch im Hinblick auf die Art der Benutzungsschnittstelle und die Interaktion des Bedieners mit ihr. Die Bedienermodellierung wird aufgrund der daraus resultierenden Anforderungen an die Systemintegration von Simulatoren für Bedienermodell und Aufgabenumgebung beeinflusst.

In *reaktiven* Aufgabenumgebungen besteht ein Großteil der Aufgabenbearbeitung darin, auf direkt wahrnehmbare Reize mit einer fest zugeordneten Aktion zu reagieren. *Deliberative* Aufgabenumgebungen enthalten einen größeren Anteil von kognitiv beanspruchenden Aufgaben wie planen, bewerten, entscheiden. Rasmussen

(1983) definiert dazu ein dreistufiges System, mit dem Verhalten zur Aufgabenbewältigung auf den zunehmend deliberativen Ebenen *Skills*, *Rules* und *Knowledge* beschrieben werden können.

Im Sinne des Frameworks in Abbildung 4-1 ist der Anteil mentaler Operationen in deliberativen Aufgabenumgebungen größer und in reaktiven der Anteil der Interaktionsoperationen. In reaktiven Aufgabenumgebungen hat daher die konkrete Gestaltung der Interaktionsschnittstelle in Bezug auf die Wahrnehmbarkeit von Interaktionselementen tendenziell einen größeren Einfluss auf die Bearbeitung der Aufgaben als in deliberativen Aufgabenumgebungen, wo die Aufgabengestaltung tendenziell die größere Auswirkung hat. Deshalb sind die Anforderungen an Methoden und Werkzeuge zur Bedienermodellierung im Hinblick auf die Erfassung von Wahrnehmung und manueller Interaktion in reaktiven Aufgabenumgebungen höher.

So sollten in diesem Fall die physiologischen und wahrnehmungspsychologischen Eigenschaften der angesprochenen Sinnes- und Handlungsorgane berücksichtigt werden. Deshalb können insbesondere reaktive Aufgabenumgebungen nach den beanspruchten Wahrnehmungs- und Handlungsmodalitäten klassifiziert werden. In deliberativen Aufgabenumgebungen kann Interaktion oft als abstrakter Informationsaustausch zwischen technischem System und Bediener modelliert werden.

Es gehört zu den Anforderungen einer Reihe von Aufgabenumgebungen, ein aktuelles Bild der Situation zu haben. Komplexität und Dynamik der Aufgabenumgebung bestimmen, in welchem Maß der Bediener dazu in der Lage ist. Es gibt zwei Strategien wie Bediener zu solch einem nutzbaren Situationsbild kommen können. Entweder werden die Menge der Aufgabenobjekte und ihr jeweiliger Zustand direkt aus der Aufgabenumgebung anhand von Displayobjekten abgelesen oder Bediener entwickeln eine mentale Repräsentation der Situation, auf die sie durch einen gelegentlichen Abgleich mit Displayobjekten der Aufgabenumgebung aktuell halten. Bei hohem Aufgabendruck oder sehr aufwändigen Interaktionsoperationen kann es günstiger für den Bediener sein, viele Elemente der Situation im Gedächtnis zu behalten und den aktuellen Zustand aus dem früher wahrgenommenen Anfangszustand und Wissen über die wahrscheinliche Entwicklung abzuleiten.

Im Rahmen dieser Arbeit werden nur Mechanismen zur Informationsaufnahme berücksichtigt, die auf Anzeigesystemen beruhen. Informationsaufnahme durch Körperwahrnehmung wird nicht betrachtet (*non body impact* Aufgabenumgebungen).

4.2. Charakterisierung der Domänen

Baxter & Harrison (o.J.) fassen alle in Frage kommenden Domänen sicherheitskritischer dynamischer Mensch-Maschine-Systeme zu der einen Klasse *Control Room* zusammen. In Abgrenzung dazu ergeben sich bei der Charakterisie-

rung von Anwendungsdomänen und ihrer Aufgabenumgebungen anhand des vorgestellten Schemas in dieser Arbeit die drei Klassen:

- Command and Control
- Prozesskontrolle
- Kraftfahrzeugführung

Die Aufgabenumgebungen der Anwendungsdomänen innerhalb dieser Klassen sind sich hinsichtlich der Anforderungen an die Modellierung aufgrund ähnlicher Komplexität, Dynamik und Interaktionsanforderungen ähnlich.

4.2.1. Command and Control

Die Klasse *Command and Control* fasst die beiden folgenden Aufgabenumgebungen zusammen:

- Flugsicherung
- Schiffsführung

Diese Aufgabenumgebungen haben gemeinsam, dass es eine *variable Anzahl von Aufgabenobjekten* gibt. Der *Grad der Dynamik* ist verglichen mit den anderen beiden Klassen nicht besonders hoch. Die Aufgabenobjekte haben zudem eine *Eigendynamik*, die nicht festen Regeln unterliegt. Damit ergibt sich als wesentliche gemeinsame Anforderung das Handling einer Multitasking-Aufgabe, bei der alle Aufgabenobjekte klassifiziert und kontinuierlich überwacht werden müssen (Stanton & Baber 2006). Die Aufgaben im Bereich Command and Control sind deliberativ.

Flugsicherung

Aufgabenobjekte sind die Luftfahrzeuge und ihre Konstellationen untereinander. Displayobjekte sind die entsprechenden Darstellungen am Radarbildschirm. Fluglotsen in der Streckenflugkontrolle haben es mit ca. 40 Aufgabenobjekten zu tun.

Die Dynamik der Situation bezieht sich zum Einen auf die Anzahl der Aufgabenobjekte, denn Luftfahrzeuge fliegen in den zu kontrollierenden Sektor ein und bedeutungsvolle Konstellationen zwischen Flugzeugen entstehen neu und verschwinden später wieder. Andererseits ändert sich der Zustand der einzelnen Aufgabenobjekte.

Die Änderungen passieren z.T. durch Anweisung des Fluglotsen an den Piloten. Dieser muss dann die Anweisung innerhalb einer vorgegebenen Zeitspanne umsetzen. Änderungen von Kurs oder Höhe sind zwar von physikalischen Gesetzmäßigkeiten, den Flugeigenschaften des angesprochenen Flugzeuges abhängig, jedoch werden Zeitpunkt und Ausmaß der Änderung vollständig vom Piloten veranlasst. Zusätzlich gibt es als externen Faktor begrenzt vorhersehbare Wettereinflüsse. Der Prädiktionshorizont ist mithin auf wenige Minuten begrenzt. Die Verzögerung zwischen Änderung von Aufgabenobjekt und Displayobjekt ist durch das Radarsystem bedingt und liegt bei wenigen Sekunden.

Diese Aufgabenstruktur bedingt, dass alle Displayobjekte regelmäßig auf Abweichungen vom erwarteten Kurs hin kontrolliert werden müssen und potentielle Konflikte zwischen Luftfahrzeugen erkannt und klassifiziert werden müssen. Neben dieser kontinuierlichen Aufgabe gilt es, als Problemlösungsaufgaben erkannte Konflikte durch Eingriffe wirtschaftlich optimal zu lösen und den Verkehr dadurch zu steuern.

Für die Bearbeitung dieser Aufgabenstruktur ist eine Kontrollstruktur mit parallelen Zielen zur Überwachung und Klassifikation aller Aufgabenobjekte erforderlich. Das ist ein homeostatisches Ziel, das über die ganze Bearbeitungsdauer (Schicht) existiert. Zur Konfliktlösung muss es daneben erreichbare Kurzzeit-Ziele geben. Die Anforderungen an das Multitasking sind also, mehrere Hauptaufgaben (globale Überwachung/Klassifikation und einzelne Konfliktlösungen) und den Wechsel zwischen mehreren Einzelzielen (Monitoring aller Aufgabenobjekte) zu verwalten.

Die Flugsicherungsaufgabe ist deliberativ, denn es sind komplexe Problemlösungsschritte erforderlich, um die Bewegung von Flugzeugen zu antizipieren und Konflikte wirtschaftlich optimal zu lösen.

Schiffsführung

Die Aufgabenstruktur ist in der Schiffsführung ähnlich wie in der Flugsicherung. Die Aufgabe besteht in der Navigation des eigenen Schiffes in einer Umwelt, in der sich Hindernisse und andere Verkehrsteilnehmer befinden. Die Navigationsaufgabe wird durch Wettereinflüsse und Strömungen erschwert.

Aufgabenobjekte sind die Antriebselemente des eigenen Schiffes, Hindernisse und Schiffe des Umgebungsverkehrs. Es gibt weniger Aufgabenobjekte als in der Flugsicherung. Die zu den Aufgabenobjekten gehörenden Displayobjekte werden durch die Steuerungsgeräte der Brücke, elektronische Seekarten und Radar dargestellt. Sowohl die Struktur der Situation (Anzahl der Aufgabenobjekte) als auch ihre Elemente (Eigenschaften der Aufgabenobjekte) sind veränderlich. Änderungen der Eigenschaften der Aufgabenobjekte sind kontinuierlich.

Aufgrund der vergleichsweise geringen Schiffsgeschwindigkeiten und der hohen Totzeiten müssen die Entfernungen zwischen Schiffen relativ groß sein, so dass die Anzahl der Aufgabenobjekte des relevanten Umgebungsverkehrs geringer ist als in der Flugsicherung.

Wie in der Flugsicherung sind Zeitpunkt und Ausmaß von Änderungen im Umgebungsverkehr nicht vorhersehbar. Sie werden i.A. auch nicht zentral angewiesen, so dass eine sehr ähnliche Aufgabenstruktur zur Flugsicherung vorliegt: Die Elemente des Umgebungsverkehrs und Hindernisse in der Umgebung müssen ständig überwacht und ihr Gefährdungspotenzial klassifiziert werden. Parallel dazu muss die eigene Navigationsaufgabe bewältigt werden. Die Navigationsaufgabe ist

eine anspruchsvolle Aufgabe deliberativer Art, weil Wettereinflüsse, Strömungen und der Umgebungsverkehr bei der Kursberechnung berücksichtigt werden müssen.

4.2.2. Prozesskontrolle

Die Klasse *Prozesskontrolle in Leitwarten* fasst die folgenden Aufgabenumgebungen zusammen:

- Prozesskontrolle in verfahrenstechnischen Anlagen und Kraftwerken
- Flugführung

Diese Aufgabenumgebungen haben gemeinsam, dass es eine *konstante Anzahl von Aufgabenobjekten* gibt. Der *Grad der Dynamik* ist verglichen mit dem in Command and Control-Umgebungen hoch. Die Aufgabenobjekte haben eine *Eigendynamik*, die physikalischen Regeln unterliegt. Die Regeln sind dem Bediener jedoch nicht immer transparent. Die Bedienung in diesen Aufgabenumgebungen erfolgt anhand vorgegebener Prozeduren (SOP – *Standard Operation Procedures*). Damit ergibt sich als wesentliche gemeinsame Anforderung dieser Klasse von Aufgabenumgebungen das regelgeleitete Handling einer Aufgabe nach einer Prozedur, wobei der Zustand des kontrollierten Systems bei diagnostischen Teilaufgaben berücksichtigt werden muss. Oft ist hierfür wichtig, das dynamische Verhalten des Prozesses zu beobachten. Da Prozesse im Vergleich zur Aufgabenbearbeitung große Totzeiten haben können, müssen Teilaufgaben zur Diagnose zurückgestellt und nach zeitlichen Bedingungen wieder aktiviert werden können.

Prozesskontrolle in verfahrenstechnischen Anlagen und Kraftwerken

Leitwartenfahrer kontrollieren und steuern einen verfahrenstechnischen Prozess in chemischen Anlagen oder Kraftwerken. Aufgabenobjekte sind Prozessvariablen wie Temperaturen oder Drücke an Messstellen. Die Anzahl der Aufgabenobjekte kann sehr groß sein. Es gibt teils mehrere Displayobjekte pro Aufgabenobjekt und es kann nennenswerten Interaktionsaufwand zum Abruf des Displayobjektes geben. Aufgrund der Fragestellungen an die dynamische Entwicklung von Aufgabenobjekten gibt es Displayobjekte in der Form von Trendanzeigen mit Darstellungen der zeitlichen Entwicklung von Aufgabenobjektwerten.

Von speziellen Situationen bei der Anlagenwartung abgesehen, bleibt die Struktur des Systems immer stabil, d.h. es kommen keine neuen Aufgabenobjekte hinzu. Die Werte der Aufgabenobjekte ändern sich jedoch kontinuierlich nach physikalischen Gesetzmäßigkeiten ihres inneren Zusammenhangs.

Die Vorhersehbarkeit der Änderungen kann aber eingeschränkt sein, da es möglich ist, dass die Gesetzmäßigkeiten dem Wartenfahrer nicht transparent sind, und da es große Totzeiten geben kann, die die Kontrolle zusätzlich erschweren. Auch die Geschwindigkeit der Änderung hängt stark vom Prozess ab. Sie kann aber

deutlich größer sein als bei Command and Control Aufgabenumgebungen. Die Verzögerung der Änderung von Aufgabenobjekt bis zur Aktualisierung der dazugehörenden Displayobjekte ist gering.

Daraus ergeben sich folgende Kontrollstrukturen: Die Aufgaben werden weniger durch die Aufgabenobjekte strukturiert wie bei den Command and Control Systemen, sondern nach den vorgegebenen SOPs. Die Abarbeitung der SOP beinhaltet ein homeostatisches Ziel, den Prozess, der in der Anlage abläuft, in einem wirtschaftlichen und sicheren Arbeitsbereich zu halten. Dazu sind auch Überwachungsschritte erforderlich, die in die SOP (top-level Aufgabe) in Form erfüllbarer diagnostischer oder Klassifizierungsaufgaben eingebaut sind.

Viele Aufgabenumgebungen dieser Klasse sind deliberativ u.a. wegen der oft auftretenden Anforderung, Zeitverläufe und -dauern neben externen Ereignissen bei Entscheidungen und der Auswahl passender Aktionen zu berücksichtigen.

Flugführung

Aufgabenobjekte sind auch hier die in den Prozeduren vorgegebenen Prozessvariablen. Der Unterschied ist hier, dass der Prozess den Betrieb eines Flugzeugs mit seinen flugtechnischen Parametern darstellt. Die Anzahl der Aufgabenobjekte ist deutlich geringer als in vielen verfahrenstechnischen Prozessen.

Im Gegensatz zur verfahrenstechnischen Prozesskontrolle ist der Prozess transparent für Piloten, und die Dynamik der einzelnen Prozessvariablen weitgehend vorhersehbar (Faber 2002).

Für die Modellierung der Flugführungsaufgabe sind ähnliche Kontrollstrukturen wie bei der Prozesskontrolle in Leitwarten erforderlich.

4.2.3. Kraftfahrzeugführung

Die Kraftfahrzeugführung ist hochautomatisiert. Die Aufgabenumgebung ist deshalb reaktiv. Aufgabenobjekte sind andere Verkehrsteilnehmer in der unmittelbaren Umgebung, Verkehrsschilder und einige wesentliche Parameter der Kraftfahrzeugführung (Geschwindigkeit, Lenkwinkel). Die Aufgabe hat demnach eine vergleichsweise geringe Komplexität. Die meisten Aufgabenobjekte können direkt wahrgenommen werden und werden nicht über Displayobjekte vermittelt.

Da sich die direkte Umgebung während der Fahrt ständig ändert, ist die Aufgabenumgebung strukturell dynamisch. Das Maß, in dem sich die Eigenschaften der Aufgabenobjekte außer den eigenen Fahrparametern ändern, ist zwar prinzipiell nicht im Bereich des Fahrers, aber aufgrund der Erfahrung und der gemeinsamen Kenntnis der Verkehrsregeln antizipierbar.

Die Kraftfahrzeugführung hat aufgrund ihres reaktiven Charakters nicht dieselben Multitasking Anforderungen wie die anderen Aufgabendomänen. Erweitert man jedoch den Betrachtungsrahmen und nimmt zur Hauptaufgabe Kraftfahrzeugführung

eine untergeordnete Nebenaufgabe hinzu, entsteht eine Multitasking Aufgabe. Dabei ist die Nebenaufgabe aufgrund des sicherheitskritischen Charakters der Hauptaufgabe nur dann ausführbar, wenn gerade mentale Ressourcen nicht für die Hauptaufgabe benötigt werden.

Die Herausforderung für die Modellierung liegt also in der Modellierung dieses Schedulingverhaltens, das sich deutlich von den Anforderungen der anderen Aufgabenumgebungen unterscheidet.

4.3. Möglicher Anwendungszweck und Ziel der Bedienermodellierung

Modellierung und Simulation von Benutzerverhalten in dynamischen Mensch-Maschine-Systemen ist für unterschiedliche Anwendungszwecke relevant, die in allen Entwicklungs- bzw. Produktlebensphasen auftreten. Jedoch bestimmt der geplante Einsatz eines Bedienermodells Anforderungen an

- die schritthaltende Lauffähigkeit von Bedienermodellsimulatoren, d.h. ob die Simulation echtzeitlich mit anderen Simulationskomponenten interagieren muss,
- die Abbildungsgenauigkeit, insbesondere in Bezug auf die Behandlung von Dynamik und Informationsaufnahme in der Aufgabenumgebung,
- die Art der Kopplung von Aufgabenumgebung und Simulation des Bedienermodells,
- das erforderliche Repräsentationsformat für die Abbildung der dynamischen Situation,
- an den Umfang der modellierten Strategien und Prozeduren, die für die Steuerung der Aufgabenumgebung erforderlich sind.

Die folgenden Anwendungs- und Einsatzmöglichkeiten existieren für Bedienermodelle:

- Entwurfsunterstützung: simulationsgestützte Evaluation von virtuellen Prototypen (Bedienermodell zur Substitution von Versuchspersonen in Benutzungstests),
- Assistenzfunktionen und adaptive Mensch-Maschine-Schnittstellen (Bedienermodell zur Zustandserkennung),
- Ableiten von kognitiven Strategien für das Training (post-hoc Analyse des Bedienermodells),
- Ableiten einer Taxonomie für *Human Factors* Fehler zur Klassifikation von entsprechenden Ereignisberichten (post-hoc Analyse des Bedienermodells).

Lediglich bei den ersten beiden Einsatzzwecken hat die Dynamik von Mensch-Maschine-Systemen einen Einfluss auf die Bedienermodellierung, da in beiden Fällen eine Simulation des Bedienermodells benutzt werden muss, um echtzeitlich und quantifizierbar in einer Aufgabenumgebung zu interagieren. Dazu ist demzufolge auch eine Kopplung von Modellsimulation und Aufgabenumgebung erforderlich.

Für die beiden anderen Einsatzzwecke sind hingegen quantifizierte und ausführbare Modelle nicht erforderlich. In diesen Fällen sind konzeptionelle Bedienermodelle ausreichend. Die dafür erforderliche Art der Bedienermodellierung stellt keine besonderen Ansprüche an die Behandlung von Dynamik der Aufgabenumgebung (z.B. Formalisierung von Helikopterpilotenfehlern mit ICS, Busse & Johnson 1999).

4.4. Auswahl und Einordnung der Fallstudien

Aufgrund der Zielstellung dieser Arbeit wird auf den Anwendungszweck der Entwurfsunterstützung fokussiert. Die folgenden Modellierungsfallstudien wurden dafür im Rahmen dieser Arbeit durchgeführt:

- MoFL: Modellierung der kognitiven Leistungen von erfahrenen Fluglotsen in der Streckenflugkontrolle – Modellierungsschwerpunkt liegt auf Informationsaufnahme und Repräsentation der Verkehrssituation.
- DURESS: Modell der Aufgabenbearbeitung in der Mikrowelt DURESS (DUal REservoir System Simulation), einer prototypischen Prozesssimulation – Schwerpunkt ist die detaillierte Modellierung von Wahrnehmung der Informationen in der Aufgabensimulation.
- Rektifikationskolonne: Modell des Anfahrens und Regelns einer verfahrenstechnischen Anlage anhand einer Mikrowelt – Schwerpunkt ist Zeitwahrnehmung für die korrekte Aufgabenbewältigung.

Durch die Auswahl der mit diesen Fallstudien verknüpften Aufgabenumgebungen und Anwendungen wird eine breite Abdeckung des Problemraums erreicht. In Tabelle 4-2 werden diese Fallstudien in einer Matrix aus Aufgabencharakteristik und Anwendungsdomänen eingeordnet. Es ergibt sich eine breite Abdeckung des Problemraums.

Tabelle 4-2: Problemraum aus Anwendungszweck und Anwendungsdomänen zur Einordnung der Fallstudien

	reaktiv	**deliberativ**
Command and Control		MoFL
Prozesskontrolle	DURESS	Rektifikations-kolonne
Kraftfahrzeug-führung	Entwicklung MT-GOMS	

Insbesondere ist sichergestellt, dass es Fälle mit autonomen Aufgabenobjekten (Command and Control) und Aufgabenobjekten, deren Änderungen Regeln unterliegen (Prozesskontrolle), gibt. Es gibt reaktive (DURESS) und deliberative (MoFL, Rektifikationskolonne) Aufgaben und unterschiedliche Arten von Multitasking (Command and Control vs. Prozesskontrolle).

5. Darstellung der Fallstudien

In diesem Abschnitt werden die Fallstudien, auf denen diese Arbeit beruht, vorgestellt und modellierungsspezifische Aspekte berichtet. In Abschnitt 5.2 wird das Bedienermodell zur Aufgabenumgebung DURESS behandelt, in Abschnitt 5.3 wird das Modell zur Rektifikationskolonne vorgestellt. In Abschnitt 5.4 wird kurz die Einbettung der Arbeit zu MT-GOMS berichtet. Im nun folgenden Abschnitt 5.1 wird die Entwicklung des Fluglotsenmodells MoFL beschrieben.

5.1. Modell der Fluglotsenleistungen (MoFL)

Die Fallstudie Modell der Fluglotsenleistungen (MoFL) ist eine Formalisierung und Implementierung wesentlicher kognitiver Vorgänge bei der Bewältigung der Arbeitsaufgabe von Fluglotsen aus ingenieurpsychologischer Sicht. Es entstand 1996 bis 1999 in dem DFG-geförderten Vorhaben En-Route Controllers' Representation (EnCoRe) und in der DFG-Forschergruppe Mensch-Maschine-Interaktion in kooperativen Systemen der Flugsicherung und Flugführung. Im Rahmen der übergeordneten Zielsetzungen dieser Projekte wurden in dieser Fallstudie die Modellannahmen von MoFL formalisiert und eine ablauffähige Simulation des kognitiven Modells entwickelt (Leuchter et al. 1997, Niessen et al. 1998). Durch diese Fallstudie wurden Grundprobleme der Modellierung kognitiver Aktivitäten in dynamischen Mensch-Maschine-Systemen deutlich. Dadurch ließen sich Anforderungen an Werkzeuge zur Modellierung und Simulation von kognitiven Vorgängen bei „Real-World"-Aufgaben festlegen (Jürgensohn, Niessen & Leuchter 2000).

MoFL war das erste kognitive Modell der Bearbeitung der Flugsicherungsaufgabe (Hilburn 2004). Spätere Modelle, die teilweise auch in ACT-R implementiert wurden, haben einen geringeren Geltungsanspruch, insbesondere in Bezug auf *Human Factors*-Fragestellungen (z.B. Schäfer 2001, Taatgen 2001, Lebiere 2005). In MoFL wurde bereits ein großer Anteil der Aufgabe eines Radarlotsen im Streckenflugbereich umgesetzt. Der Fokus lag auf einer validen Simulation der mentalen Repräsentation der Situation, da die *Situation Awareness* des Fluglotsen eine bedeutende Fehlerursache ist und die Aufgaben- und Systemgestaltung für diesen Bereich optimiert werden muss.

5.1.1. Beschreibung der Mikrowelt

In Deutschland ist neben Eurocontrol und Skyguide hauptsächlich die Deutsche Flugsicherung GmbH (DFS) für die Kontrolle des Flugverkehrs verantwortlich. Die DFS überwacht von den Towern auf Flugplätzen aus den Verkehr auf den Rollbahnen. Von Zentralen in Bremen, Karlsruhe, Langen und München aus erfolgt die *approach*- und *en-route*-Kontrolle des Flugverkehrs auf der Basis von Flugplänen, computervermittelten Radarinformationen und Funkverkehr.

Die Flugsicherung verantwortet die sichere und ökonomische Steuerung des Luftverkehrs. Sie beruht auf einer Vorlaufplanung, die in Europa größtenteils an Flugrouten, die so genannten Luftstraßen, gebunden ist. Der Flugplan wird von den Flugzeugen auf ihrem Weg über Luftstraßen abgeflogen. Die Planung muss wegen äußerer Einflüsse (z.B. Wetter) geändert werden. Dadurch kann es trotz Planung zu einer gefährlichen Annhäherung zwischen zwei Luftfahrzeugen kommen. Es droht Kollisionsgefahr. Das Maß für die minimale Separation zwischen zwei Luftfahrzeugen hängt von der Messgenauigkeit im betroffenen Luftraum ab, die von der Radarabdeckung abhängig ist. In MoFL werden als Separationskriterien vereinfachend 100 Fuß vertikal und fünf nautischen Meilen horizontal benutzt.

Die Flugbewegungen werden von Fluglotsen am Radarschirm beobachtet. Sie müssen drohende Separationsunterschreitungen erkennen und durch Anweisungen an die Piloten der abwenden. Die Kontrolle eines Lotsen erstreckt sich auf seinen Sektor. Der Sektor ist ein Gebiet in dreidimensionaler Ausdehnung, das an mehrere andere Sektoren angrenzt. Beim Übergang von einem in den nächsten Sektor müssen sich Flugzeuge beim zuständigen Lotsen per Funk anmelden.

Es gibt *approach*-Sektoren in der Nähe von Flughäfen und *en-route*-Sektoren für den Streckenflug-Bereich, wo Flugzeuge bereits ihre Reisehöhe und -geschwindigkeit erreicht haben. In dieser Fallstudie wurde die Kontrolle im en-route-Sektor „West Radar 2“ (WR2) modelliert. WR2 lag zum Zeitpunkt der Fallstudie horizontal zwischen Magedburg, Leipzig und Fulda sowie vertikal zwischen etwa 5.000 bis 8.000 Metern über dem Meeresspiegel (s. Abbildung 5-1).

Bei normalem Verkehr ist jedem *en-route*-Sektor ein Team von zwei Fluglotsen zugeteilt. Der Planungslotse hat einen Zeithorizont von bis zu 20 Minuten. Er ist für die Koordination mit den Nachbarsektoren verantwortlich. Eine wichtige Informationsquelle des Planungslotsen sind neben dem Radarbild Flugstreifen, auf denen die vorgeplanten Routen und Überflugzeitpunkte vermerkt sind. Daneben sind auf den Flugstreifen Informationen zur gewünschten Route, Flughöhe und -geschwindigkeit, sowie eine Kategorisierung des Flugzeugtyps eingetragen. Der Planungslotse sortiert die Flugstreifen nach den erwarteten Überflugzeitpunkten.

Der andere Fluglotse heißt Radarlotse. Er übernimmt die lokale Kontrolle und kurzfristige Planung. Zur Vermeidung von Separationsunterschreitungen zwischen Luftfahrzeugen gibt er per Sprechfunk Anweisungen in Form Höhen-, Geschwindigkeits- oder Richtungsänderungen an die Piloten. Er muss die korrekte Ausführung seiner Anweisungen überwachen. Er trägt die Verantwortung für die Sicherheit im ihm zugeteilten Sektor. Die wichtigste Informationsquelle für diese Aufgabe ist das Radarbild.

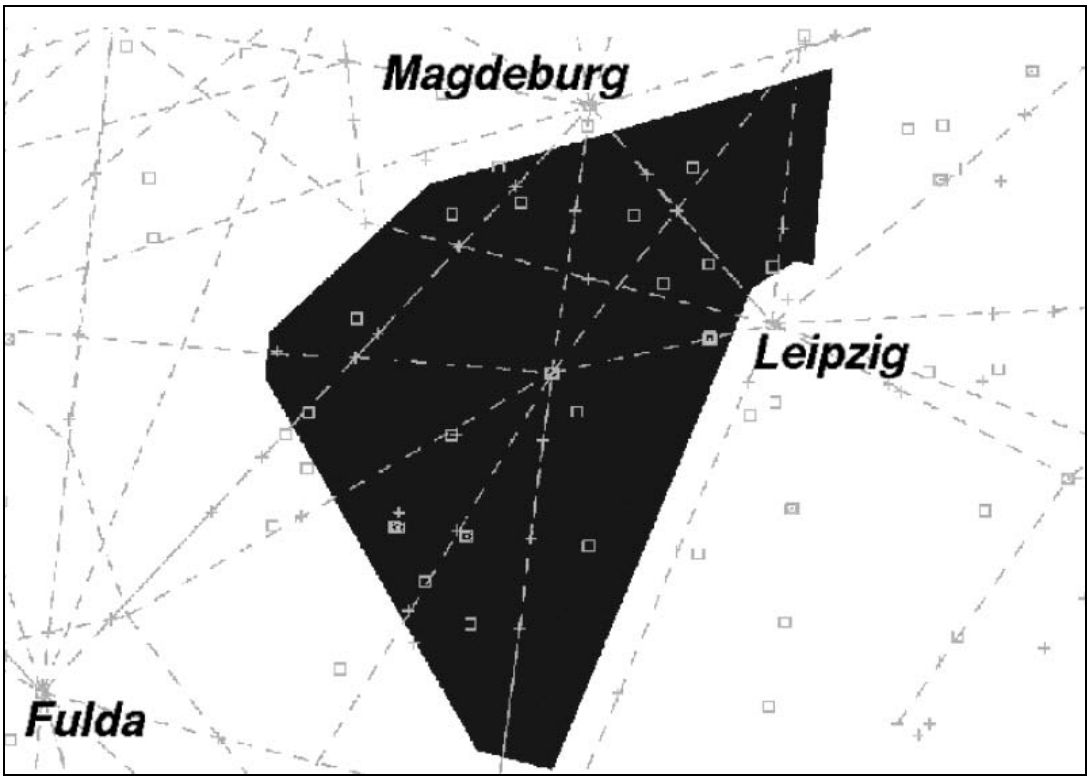


Abbildung 5-1: Horizontale Ausdehnung des Sektors West Radar 2 (Leuchter 1999)

Im Verlauf des Vorhabens EnCoRe wurde für experimentelle Erhebungen die Aufgabensimulation EnCoRe-PLUS (programmierte Luftraumsimulation) mit Borland C++ Version 3.1 unter MS-DOS entwickelt (Bierwagen 1996). EnCoRe-PLUS besteht aus einer einfachen Simulation des Flugverkehrs mit der Möglichkeit, interaktiv in den vorgeplanten Ablauf einzugreifen. Zusätzlich gibt es einen Arbeitsplatz, der den simulierten Verkehr als Radarbild und in Flugstreifen anzeigt. Die Darstellung und die Interaktionsmöglichkeiten sind analog zu dem Bediensystem, das zu dem Zeitpunkt in der Regionalkontrollstelle der DFS in Berlin-Tempelhof genutzt wurde (s. Abbildung 5-2). Die Kommunikation zwischen den Komponenten läuft über gemeinsame Dateien und Message-Pipes ab.

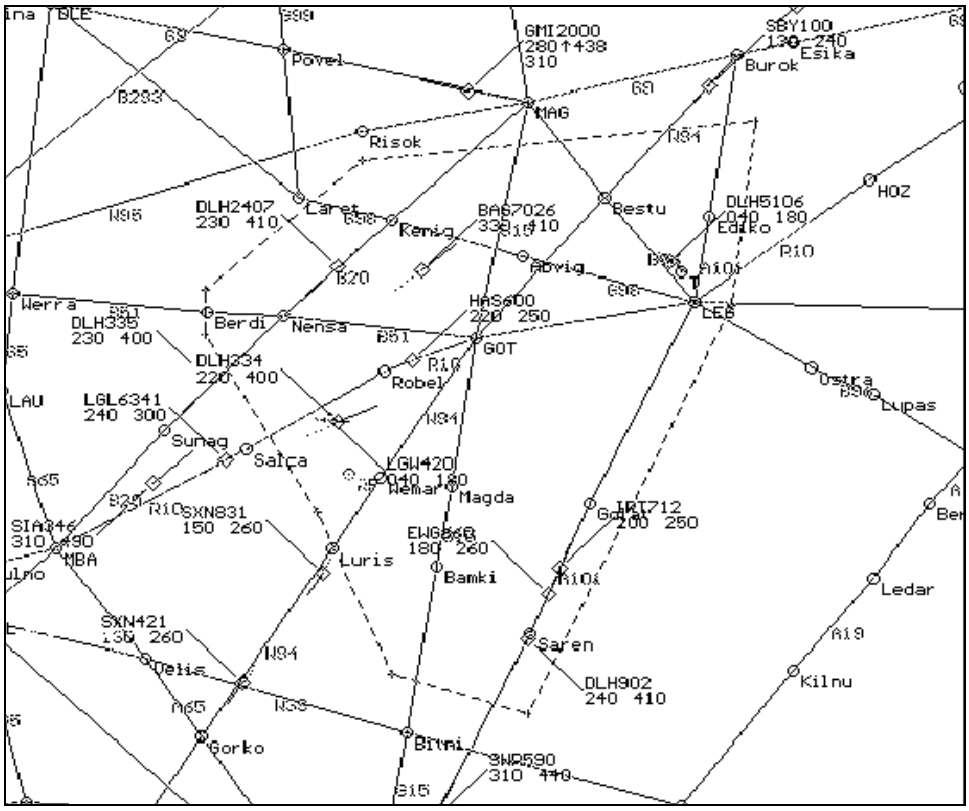


Abbildung 5-2: Detail Screenshot vom Radararbeitsplatz in EnCoRe-PLUS (Bierwagen 1996)

In der Forschergruppe Mensch-Maschine-Interaktion in kooperativen Systemen der Flugsicherung und Flugführung wurde eine andere deutlich komplexere Luftraumsimulation (gats: Generic Air Traffic Simulation Tools) auf der Basis von C++ unter AIX und verschiedene Lotsenarbeitsplätze (simco: *Sophisticated Interface for Monitoring, Controlling and Organizing*) mit dem auf C basierenden Framework ODS Toolbox[3] entwickelt (s. Abbildung 5-3). Die Benutzungsschnittstelle ist an den ODID IV Standard von Eurocontrol angelehnt. Die Kommunikation zwischen den Komponenten läuft über TCP/IP Sockets und über das standardisierte DIS-Protokoll (*distributed interactive simulation*, IEEE 1278), das auf UDP aufsetzt.

[3] ODS Toolbox ist ein Rapid Development Tool und ein Framework der Firma ISA Orthogon für die Entwicklung von ODS (*operational display system*).

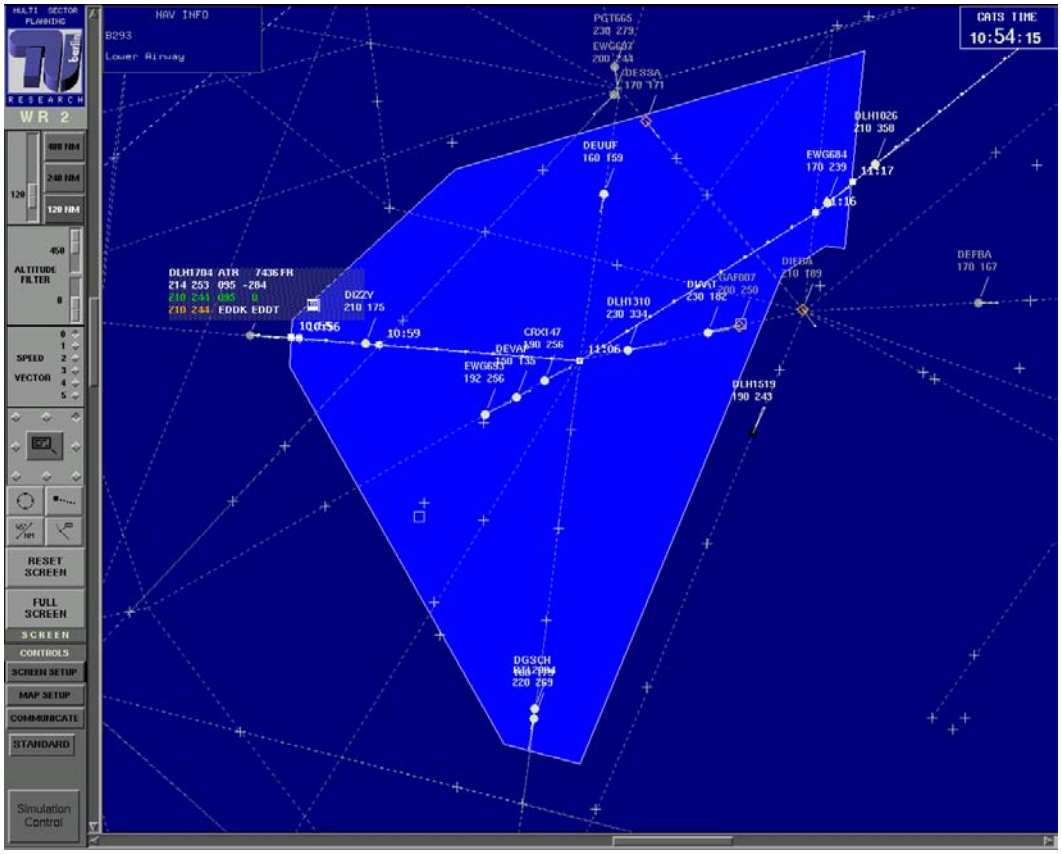


Abbildung 5-3: Screenshot des Fluglotsenarbeitsplatzes simco[4]

5.1.2. Erweiterungen an der Simulation

Das kognitive Modell muss Informationen über den Flugverkehr aus dem Radardisplay aufnehmen können und mit den simulierten Piloten über Sprechfunk interagieren können. Flugstreifen wurden als Informationsquelle vernachlässigt. Für die Kopplung des kognitiven Modells mit der jeweiligen Simulationsumgebung mussten EnCoRe-PLUS und gats/simco erweitert werden. Die Simulationsumgebungen waren auf mehrere unterschiedliche Rechner verteilt, die miteinander über ein LAN mit Übertragungsraten von 10 MBit/s (EnCoRe-PLUS) bzw. 100 MBit/s (simco/gats) vernetzt waren. Zur Interprozesskommunikation wurden TCP/IP-Sockets verwendet. Der Verbindungsaufbau über diese Sockets wird in Anhang C beschrieben.

Aufgabensimulation und kognitives Modell kommunizieren mit einem anwendungsspezifischen neu entwickelten Protokoll. Das Protokoll sollte einfach zu implementieren sein, denn auf der Seite des kognitiven Modells sind in erster Linie ACT-R-Mechanismen zu benutzen, mit denen sich Kommunikationsaspekte nur schwer abbilden lassen. Das Protokoll ist in Anhang D beschrieben. Es bezieht sich nicht auf die Gestaltung der Benutzungsoberfläche von Radarsystemen oder andere Faktoren der Aufgabenumgebung, sondern abstrahiert davon. Es ist lediglich ein

[4] http://www.ilr.tu-berlin.de/FF/grafiken/wr2_1.gif

psychologisch plausibilisierter Zugriff auf Informationen in der Luftraumsimulation möglich. Dabei wurden wesentliche Eigenschaften der Arbeitsumgebung von Fluglotsen berücksichtigt (Radarbildschirm, Organisation von Informationen, Sprechfunk mit Piloten).

In den folgenden beiden Abschnitten werden die spezifischen Architekturen und die erforderlichen Anpassungen der beteiligten Systeme dokumentiert.

EnCoRe-PLUS

EnCoRe-PLUS besteht aus mehreren Prozessen (Radarbildschirm inkl. Luftraumsimulation, Flugstreifendarstellung, Geisterpilotenarbeitsplatz), die auf mehreren Rechnern laufen. Das System ist in Borland C++ programmiert. Zur Interprozesskommunikation werden dateibasierte Pipes verwendet, die über ein Novell Netware Filesystem im lokalen Netzwerk verteilt werden.

Anpassungen waren bei der Interprozesskommunikation erforderlich. Dafür wurde eine neue Klasse für das Handling von Sockets eingeführt. EnCoRe-PLUS fungiert als Server, während sich das kognitive Modell als Client anmeldet. Aufgrund der beschränkten Multitasking-Fähigkeiten von MS-DOS und dem unverhältnismäßigen Aufwand, um eine eigene Prozessverwaltung mit Scheduling zum Handling mehrere Clients in der existierenden Systemarchitektur zu implementieren, kann der Server jeweils nur einen Client bedienen. Für den Zweck der angestrebten Simulationen war diese Vereinfachung sinnvoll, da diese Funktion nicht erforderlich war. Das Debugging der Schnittstellenimplementierung wurde dadurch jedoch erschwert.

Aufgrund der Systemarchitektur von EnCoRe-PLUS waren Änderungen in einigen Modulen bzw. Klassen erforderlich. Da das verwendete Kommunikationsprotokoll eine Chomsky Typ 2 Sprache repräsentiert, ist für das Parsing kein großer Aufwand erforderlich. Die für die Kommunikation benötigten Informationen waren zum großen Teil bereits in Datenstrukturen zugreifbar, lediglich für Anfragen nach `near` (s. Anhang D) war eine neue Funktion erforderlich, die die entsprechenden Informationen aus der Luftraumsimulation generiert.

Insgesamt waren an EnCoRe-PLUS Änderungen im Umfang von ca. 700 LOC C++ erforderlich, die sich komplett auf die Implementierung der Schnittstelle zur kognitiven Simulation MoFL und deren Kommunikationsprotokoll bezogen.

gats/simco

Eine grundlegende Designentscheidung war, das kognitive Modell an den Lotsenarbeitsplatz simco und nicht direkt an die Luftraumsimulation gats anzuschließen. Damit sollte erreicht werden, dass – zumindest potenziell – die Gestaltung der Benutzungsschnittstelle des Lotsenarbeitsplatzes einen Einfluss auf die Art der Informationsaufnahme durch das kognitive Modell hat und somit die

Benutzungsschnittstelle von simco mit Hilfe des kognitiven Modells bewertet werden kann.

Das Programm simco musste deshalb um eine Schnittstelle zur Kommunikation mit dem kognitiven Modell erweitert werden, mit der das anwendungsspezifische Protokoll (s. Anhang D) verwendet werden kann. Die Struktur von simco resultiert aus der zugrunde liegenden ODS-Toolbox. Die ODS-Toolbox besteht nach Flicker (2001) aus den Teilen *Controller Working Position Kernel System* (CWP), *Interface Editor System* (IES) und weiteren Servicekomponenten. Als Betriebssystemumgebung sind Windows NT und UNIX vorgesehen. Die Visualisierung nutzt OSF/Motif unter X11 (auch unter Windows). Die Implementierung von simco geschah unter Linux.

Das CWP ist die Laufzeitumgebung, die ein User Interface Management System (UIMS) und einen Objekt-Server beinhaltet. Das Objekt-System der ODS-Toolbox geht von einem *Facade* Design Pattern aus (Gamma et al. 1995): Zu jedem Datenobjekt (*Conceptual Object*) existieren ein oder mehrere *Presentation Objects*, die für die Anzeige und Manipulation durch das UIMS benutzt werden. Der CWP Kernel kann über verschiedene eingebaute Schnittstellen angesprochen werden: Es unterstützt die Standards RPC (*Remote Procedure Calls*) und CORBA (*Common Object Request Broker Architecture*) auf Schnittstellenebene, ASTERIX (*All Purpose Structured Eurocontrol Radar Information Exchange*) und ASN.1 (*Abstract Syntax Notation.1*) auf Protokollebene. Außerdem hat der CWP Kernel ein API, das mit der Programmiersprache C benutzt werden kann. Das Interface Editor System ist der GUI-Builder der ODS-Toolbox. Es besteht aus den Komponenten *Layout Editor*, *Rule Editor*, *Widget Interface*, *Map Editor*, *Communication Editor*. Für die Anbindung von MoFL ist potenziell der *Communication Editor* relevant. Er wird benutzt, um Datenformate von externen Kommunikationsprotokollen zu definieren. Die Service-Komponenten der ODS-Toolbox beziehen sich auf die Gegebenheiten der Ablaufumgebung X11: *ODS Window Manager*, *Color Server* und *Distributed Presentation Facility*. Sie beinhalten auch eine Recording und Replay-Funktion.

Externe Vorgabe war, das Kommunikationsprotokoll von MoFL nicht an die neue Aufgabensimulation anzupassen. Eine Integration über die bereits vorhandenen Schnittstellen war deshalb trotz des Tools *Communication Editor* und der eingebauten Schnittstellen der ODS Toolbox nicht möglich. Eine neue Schnittstelle musste mit dem C API des CWP Kernels realisiert werden. Dazu wurde die Komponente `datafeeder`, die auch zur Verbindung mit der Luftraumsimulation gats dient, von Mitarbeitern des Instituts für Luft- und Raumfahrt der TU Berlin erweitert.

Es waren sehr große Eingriffe in den Code erforderlich. Etwa 900 LOC in ANSI C bilden das Modul zur Abarbeitung des MoFL-Protokolls. Auch einige neue Funktionen mussten entwickelt werden, um die möglichen Anfragen von MoFL zu beantworten. Die Anbindung an einen Socket erwies sich dagegen als sehr leicht.

Simco agiert als ein Server, der so lange wartet („busy wait“), bis sich ein MoFL-Client mit ihm verbindet. Mehr als eine Client-Instanz wird nicht berücksichtigt, da in der Programmstruktur mit dem CWP Kernel System keine Kindprozesse erzeugt werden können.

5.1.3. Kognitive Modellierung

Das kognitive Modell MoFL (Modell der Fluglotsenleistungen) zielt insbesondere auf eine valide Repräsentation der aktuellen Verkehrssituation ab (von Fluglotsen „picture“ genannt). Die simulierte mentale Repräsentation besteht aus einer verkürzten Repräsentation von Objekten und Relationen zwischen Objekten im kontrollierten Sektor. Sie muss dynamisch mit den Änderungen im beobachteten Luftraum fortgeschrieben werden. Mit den Chunks im „picture“ kann das Multitasking zwischen den parallel aktiven Teilaufgaben gesteuert werden. Durch die mentale Simulation und Fortschreibung der Chunks im „picture“ wird auf die zukünftige Entwicklung des Luftraums geschlossen. Das „picture“ ist als Modell des Arbeitsgedächtnisses eines erfahrenen Fluglotsen fehlerhaft, da es z.B. teilweise veraltet oder sogar unvollständig ist.

Die Informationsverarbeitung in MoFL ist in folgende Module gegliedert: „*picture*“, *Aufgabensimulation*, *Datenselektion*, *Antizipation*, *Konfliktlösung*, *Update* und *Kontrollprozeduren*. Die Prozesse dieser Module bauen die Repräsentation der aktuellen Situation (das „picture“) auf und greifen während diagnostischer Prozesse darauf zu. Abbildung 5-4 illustriert das Modell mit seinen Bestandteilen. Das Modell besteht aus den Prozessen *Monitoringzyklus*, *Antizipationszyklus*, *Problemlösungszyklus* und *Kontrolle*. *Datenselektion* und *Antizipation* beinhalten diagnostische Verarbeitungsschritte. Mit Prozessen aus dem Modul *Konfliktlösung* werden Eingriffe in die simulierte Aufgabenumgebung (EnCoRe-PLUS bzw. simco/gats) bereitgestellt. Mit *Update* wird das „*picture*“ aktualisiert. Das „*picture*“ stellt die internen Daten- bzw. Gedächtnisstrukturen dar, und die *Aufgabensimulation* repräsentiert die externe Aufgabenumgebung. Die kontextabhängige Sequenzierung der Verarbeitungsschritte wird durch die *exekutiven Kontrollprozeduren* vorgenommen.

Das kognitive Modell ist in der kognitiven Architektur ACT-R 3.0 implementiert.

„picture“

Im „picture“, dem simulierten Arbeitsgedächtnis des Fluglotsen, sind alle situationsrelevanten Objekte, Ereignisse und Relationen zwischen Objekten repräsentiert. Auch die räumliche Struktur des Sektors mit den darin vorhandenen Luftstraßen und ihren Kreuzungspunkten ist im „picture“ enthalten. Bei der Repräsentation wird zwischen Objekten mit Signalcharakter und Objekten ohne Signalcharakter unterschieden. Diejenigen mit Signalcharakter werden „fokal“, die

ohne „extrafokal“ repräsentiert. Zusätzlich gibt es „Ereignisse“, also handlungsrelevante Luftfahrzeugrelationen, die immer fokal repräsentiert werden.

Das „picture“ wird in ACT-R durch ein semantisches Netz aus Chunks, die die Luftraumobjekte repräsentieren, und Relationen zwischen ihnen modelliert. In der kognitiven Architektur ACT-R 3.0 ist es nicht möglich, eine analoge, bildhafte Repräsentation im Arbeitsgedächtnis nutzen. Einige der in MoFLs „picture“ verwendeten Objekte haben jedoch räumliche Merkmale, über die sie zugegriffen werden müssten. Deshalb müssen in MoFL Operationen zum analogen Abruf solcher Objekte nachgebildet werden. Ein Beispiel ist der Zugriff über die Entfernung zu anderen Luftraumobjekten.

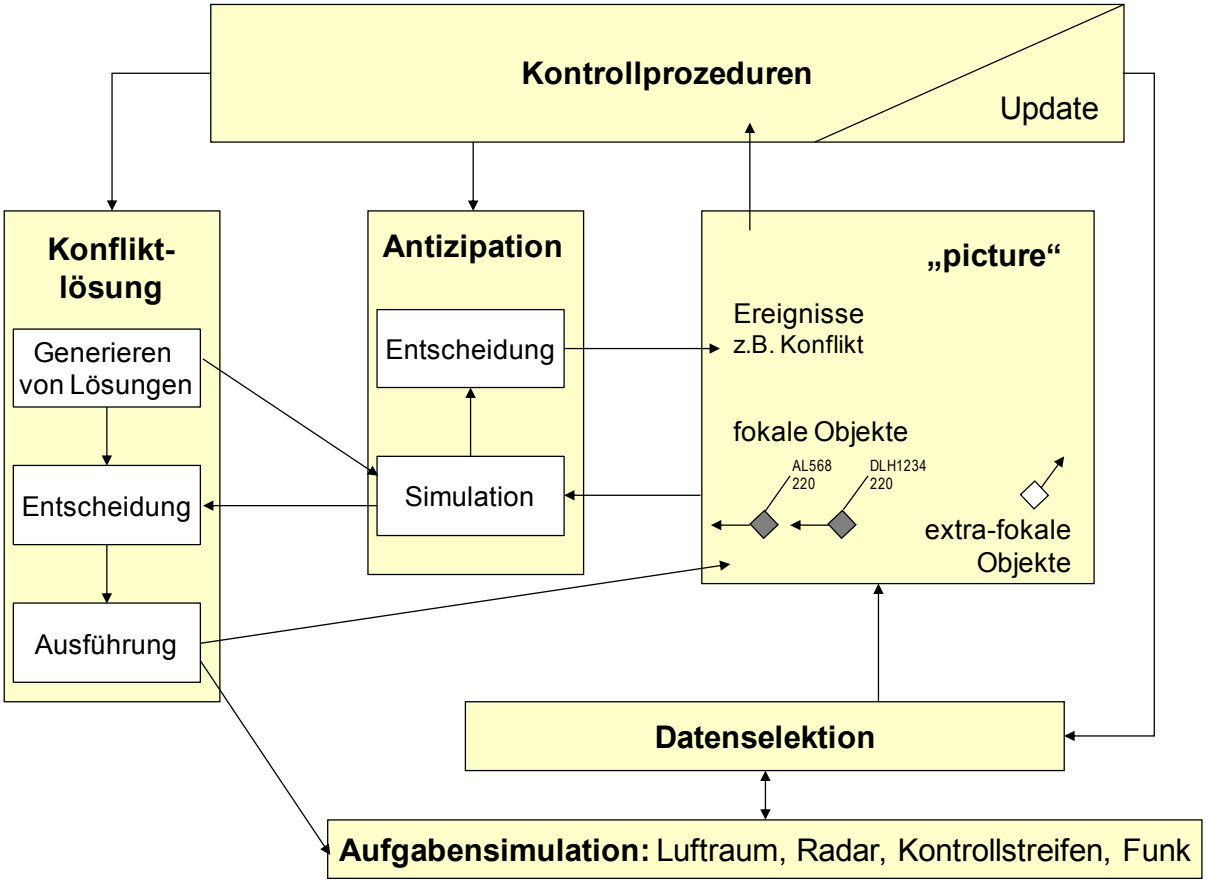


Abbildung 5-4: Die wesentlichen Komponenten des Modells der Fluglotsenleistungen (nach Leuchter et al. 1997)

Die Fokalität wird konzeptionell durch die Aktivierung der entsprechenden Chunks nachgebildet. Chunks, die fokale Objekte repräsentieren, haben demnach eine hohe Aktivierung, Chunks extrafokaler Objekte sind niedrig aktiviert. Aufgrund der Annahmen der ACT-R-Architektur ist die Nutzung der architekturimmanenten Berechnungsvorschriften für die subsymbolischen Parameter nicht möglich. Die Aktivierung wäre demnach nämlich abhängig von der Häufigkeit des Zugriffs auf einzelne Chunks. Aufgrund der Dynamik der Situation korreliert die Wichtigkeit einzelner Chunks aber nicht mit deren Zugriff. Stattdessen müssen explizite

Funktionsaufrufe im Kontrollfluss der Anwendung benutzt werden, um die Aktivierung zu modifizieren. Die Aufrufe müssen durch äußere Reize oder vom Modell gefolgerte Diagnosen initiiert werden.

In Abbildung 5-5 ist die verwendete Klassenhierarchie der Chunks dargestellt. *Luftraum-Objekte* zeichnen sich durch räumliche Eigenschaften, etwa die Position auf dem Radarschirm, aus. *Ereignisse* werden benutzt, um inferierte Relationen zwischen Luftraum-Objekten zu repräsentieren. Chunks vom Typ *Kontrollwissen* werden intern verwendet, um die einzelnen geplanten Verarbeitungsschritte zu repräsentieren und so die Multi-Tasking-Verarbeitung zu steuern.

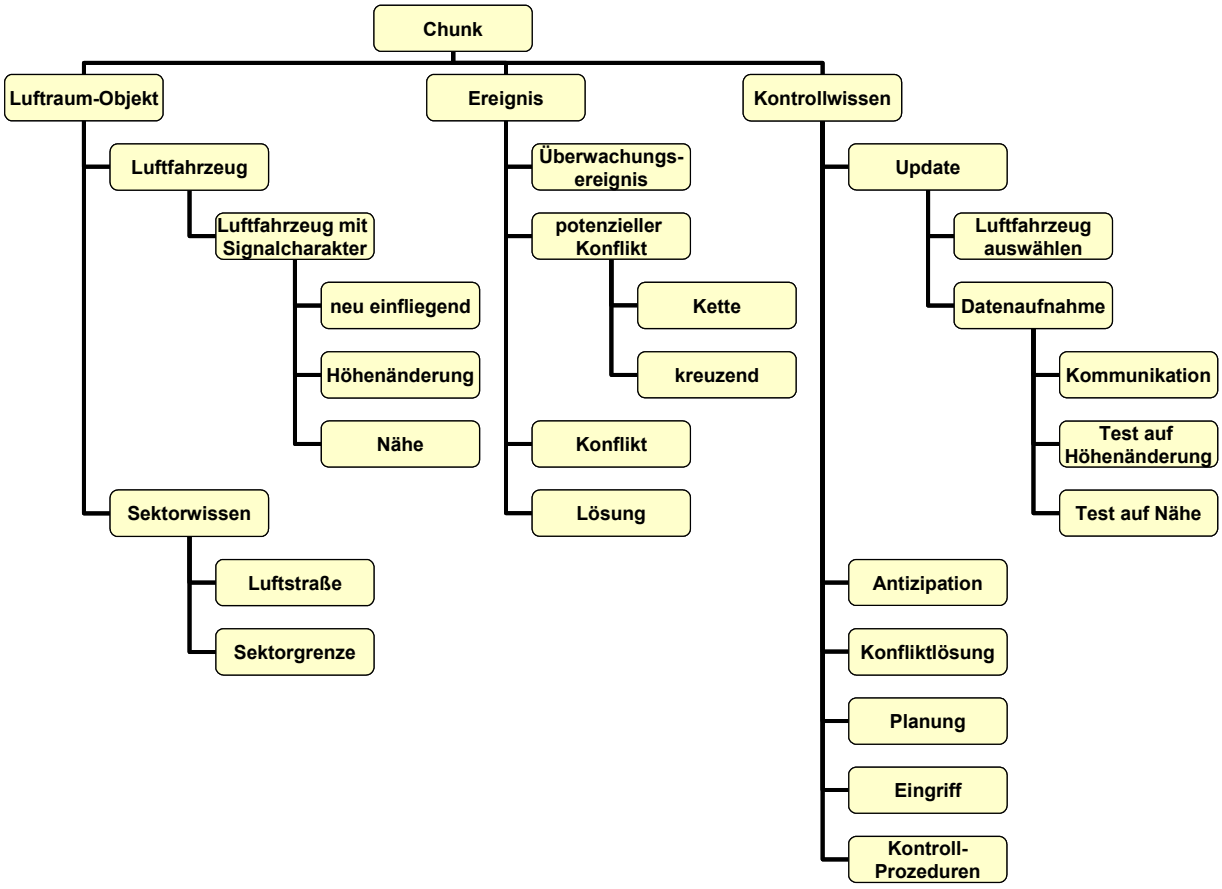


Abbildung 5-5: Ausschnitt aus der Vererbungshierarchie der Chunk-Typen in MoFL

Der Monitoringzyklus: Datenselektion und Update

Im Monitoringzyklus nimmt das kognitive Modell Informationen aus der Aufgabenumgebung auf und integriert sie in das „picture“. Dazu gehört die Auswahl der Objekte, für die aktuelle Informationen gesucht werden. Bei der Informationsaufnahme werden nicht alle zur Verfügung stehenden Informationen in das „picture“ integriert, sondern es wird eine plausibilisierte Auswahl getroffen. Dadurch ist die Repräsentation der Luftfahrzeuge verkürzt. Die wichtigsten Informationen eines Luftfahrzeuges sind das „Callsign“, also die Kennung des Luftfahrzeuges, die Flughöhe und Flugrichtung. Diese Informationen werden als

Basisdatensatz aufgenommen. Von Fall zu Fall werden spezifischere Informationen selektiert, die für die Verarbeitung erforderlich sind.

Der Datenselektionsteil im Monitoringzyklus basiert auf dem Abscannen des Radarbildschirms nach besonderen Merkmalen von Luftfahrzeugen und Sektorgegebenheiten, die auf ein mögliches Konfliktpotenzial hinweisen und eine weitere Überwachung eines Luftfahrzeuges erforderlich machen. Beispiele dafür sind Vertikalbewegungen, laterale Nähe zwischen Luftfahrzeugen und Kreuzungspunkte von Luftstraßen, an denen oft Konflikte auftreten. Flugzeuge, die diesen Merkmalen entsprechen, werden als *fokal* markiert. Auch neu einfliegende Luftfahrzeuge, die neu im „picture“ repräsentiert werden, sind zunächst *fokal*. Flugzeuge ohne diese Merkmale sind *extrafokal* repräsentiert. Sie haben eine geringere Bedeutung in der aktuellen Situation. Solche Luftfahrzeuge werden nur mit den allernötigsten Informationen (laterale Position, rudimentärer Richtungshinweis) im „picture“ repräsentiert. Das kontinuierliche Update aller repräsentierten Objekte ist wegen ihrer Eigenbewegung erforderlich. Die Updatefrequenz ist für fokale Objekte höher als für extrafokale.

Zur Simulation der Informationsaufnahme des Fluglotsen am Radarschirm wurde die oben beschriebene physikalische Schnittstelle zwischen kognitivem Modell und externer Aufgabenumgebung verwendet. Das Kommunikationsprotokoll zwischen kognitivem Modell und externer Aufgabenumgebung (s. Anhang D) wird von einer eigenen Softwareschicht abgearbeitet (ACT-COM, s. 6.2). Sie hat die Form einer Erweiterung von ACT-R 3.0 und ist in ein eigenständiges LISP-Modul ausgelagert. Somit kann die Softwareschicht in beliebigen ACT-R-Modellen verwendet werden. Das ACT-COM-Modul erhält eine Spezifikation des Protokolls als Parametrierung. Optional kommen Namen von Call-Back-Funktionen hinzu, die beim Empfang einzelner Protokollelemente ausgeführt werden. Bei der Spezifikation der Protokollelemente bietet ACT-COM zwei Modi für synchrone und asynchrone Kommunikation. Im Kontext von MoFL bedeuten die Modi:

- *Asynchrone Kommunikation:* Von der externen Aufgabenumgebung (EnCoRe-PLUS oder simco/gats) wird das Auftreten von Ereignissen in der Luftraumsimulation gemeldet (z.B. vom Luftfahrzeug initiierte Sprechfunkkommunikation). Nach der Anwendung einer jeden ACT-R Produktionsregel wird eine ACT-COM eigene LISP-Funktion aufgerufen, die prüft, ob in der Zeit der Abarbeitung der letzten Produktionsregel neue Meldungen aus der asynchronen Kommunikation eingegangen sind. In diesem Fall wird dann eine der Meldung entsprechende LISP Call-Back-Funktion ausgeführt, die standardmäßig einen neuen Chunk im „picture“ erzeugt, der das asynchron gemeldete Ereignis repräsentiert.
- *Synchrone Kommunikation:* Das kognitive Modell kann über ACT-COM aktiv Informationen in der Luftraumsimulation abfragen, um bestimmte Luftfahrzeug-

daten zu aktualisieren oder andere Informationen gezielt aus der Aufgabenumgebung auszulesen. Die als Antwort auf die synchrone Anfrage empfangenen Daten werden durch entsprechende Call-Back-Funktionen in Form von Chunks im „picture“ repräsentiert.

Die Datenselektionsprozeduren sind anwendungsspezifisch. Sie sind so implementiert, dass sie durch eine Reihe von zueinander passenden Zielen auf dem Zielstapel ausgelöst werden. Werden diese Ziele abgearbeitet, ergibt sich die folgende Sequenz von Prozessschritten:

1. Auswahl eines Flugzeug: Gemäß der Fokalität von repräsentierten Luftfahrzeugen und des Zustandes des „pictures“ wird ausgewählt, welches Flugzeug als nächstes aktualisiert wird.
2. Abruf von Informationen: Abhängig vom Zustand des zu aktualisierenden Objektes wird ausgewählt, welche Informationen in der externen Aufgabenumgebung abgefragt werden müssen. Entsprechende LISP-Funktionen in ACT-COM werden aufgerufen. Die Antwort wird an eine Call-Back-Funktion übergeben, die ein entsprechendes Ziel als Chunk generiert.
3. Übernahme abgerufener Informationen in das „picture“: Dieses Ziel triggert eine Produktion, die den Chunk modifiziert, der die abgerufenen Informationen repräsentiert.
4. Überprüfung von Signalbedingungen: Der gerade aktualisierte Chunk im „picture“ wird auf Änderungen seines Signalcharakters geprüft (z.B. Vertikalbewegungen oder Annäherung anderer Luftfahrzeuge).

In ACT-R besitzt jeder Chunk eine Aktivierung. Aufgrund des Konfliktresolutions-algorithmus‘ von ACT-R beeinflusst die Aktivierung, welche Produktionsinstanz feuert. Höher aktivierte Chunks, die im Bedingungsteil einer Produktion referenziert werden, werden geringer aktivierten vorgezogen. Für die hier verwendete Folge von Prozessschritten bedeutet dies, dass das Luftfahrzeug, für das als nächstes Informationen abzurufen sind, das mit der höchsten Aktivierung sein muss.

Wegen der Dynamik der Aufgabensituation sind die ACT-R eigenen subsymbolischen Mechanismen zur Auswahl der relevanten Chunks nicht geeignet (s. Abschnitt „picture“). Deshalb wird in MoFL ein anwendungsspezifischer Algorithmus zur Auswahl verwendet. Dazu wird in MoFL zuerst der Luftfahrzeug-Chunk ausgewählt, dessen Update am längsten zurückliegt. Dabei wird auch die Fokalität berücksichtigt, indem zwei unterschiedliche Schwellen bei fokalen und extrafokalen Objekten für die maximal zulässige Zeit als Parameter definiert sind, in der ein Luftfahrzeug nicht aktualisiert werden braucht.

Der Zeitpunkt des letzten Updates muss als zusätzliches technisch erforderliches Attribut am Luftfahrzeug-Chunk repräsentiert werden. Eine Referenz auf den so ausgewählten Chunk wird dann im aktuellen Ziel vermerkt. Die Aktivierung des ausgewählten Chunks wird dadurch erhöht, denn das aktuelle Ziel ist eine Quelle von

zusätzlicher Aktivierung für Chunks, die von ihm referenziert werden (s. Abbildung 3-3). Dieses Auswahlschema wird in analoger Form für die Selektion des aktuell wichtigsten Objekts zur Antizipation, zur Konfliktlösung und zum Eingriff in den Verkehr verwendet.

Antizipationszyklus: Vorhersage

Für die Diagnose zukünftiger Verkehrskonstellationen ist die Vorhersage der zukünftigen Positionen von Luftfahrzeugen erforderlich. Die mentale Simulation der Flugbewegungen ist nicht mathematisch korrekt, sondern basiert auf Heuristiken. In die Antizipation werden nur fokal repräsentierte Objekte einbezogen. Resultat ist die Entscheidung, ob Luftfahrzeuge zukünftig Separationskriterien verletzen werden, konfliktfrei oder potenziell gefährdet, also weiter überwachungsrelevant sind. Auf der Ebene der Chunks im Arbeitsgedächtnis bedeutet dies, dass fokal repräsentierte Objekte entweder als Ereignis repräsentiert werden, das zukünftig verarbeitet wird, oder zum extrafokalen Objekt zurückgestuft werden. Folgende Ereignisse können erzeugt werden:

- Wenn ein Konflikt erkannt wird, wird das Ereignis „Konflikt“ erzeugt. In ihm wird u.a. repräsentiert bis wann die Konfliktlösung spätestens erfolgt sein muss.
- Ist die Unsicherheit bei der Konflikterkennung zu groß, weil die potenzielle Separationsunterschreitung weit in der Zukunft liegt, wird ein Überwachungsereignis erzeugt, das bewirkt, dass der potenzielle Konflikt weiter überwacht wird, bis eine hinreichend sichere Entscheidung getroffen werden kann.

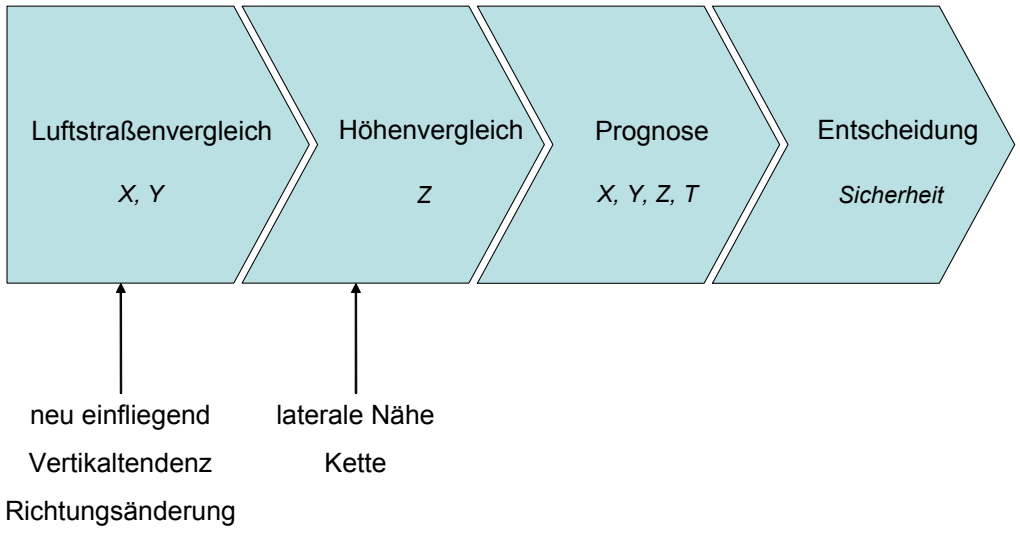


Abbildung 5-6: Sequenz der Produktionen im Antizipationsprozess in MoFL

Der Antizipationszyklus besteht aus einer Folge von Tests, die mit einem jeweils eigenen Satz von Produktionsregeln implementiert sind (s. Abbildung 5-6):

1. Befinden sich die Luftfahrzeuge auf einer gemeinsamen oder auf kreuzenden Luftstraßen?

2. Sind beide Flugzeuge auf gleicher Flughöhe oder ändert eines seine Höhe (steigen oder sinken)?
3. Kommt es bei gleichbleibender Bewegung der beteiligten Flugzeuge zu einer Unterschreitung der Separationskriterien? Die Simulation der Bewegung beruht auf der Fortschreibung des Richtungsvektors (*velocity leader*), der am Radarschirm visualisiert wird.
4. Wie sicher ist die Antizipation der Bewegung? Dazu wird die vorhergesagte Zeit bis zur Unterschreitung des Separationskriteriums benutzt. Daraus wird auch der letztmögliche Zeitpunkt zum Eingriff in den Verkehr abgeleitet und gespeichert.

In dieser Sequenz werden existierende Luftfahrzeug-Chunks geändert (z.B. durch Setzen der Fokalität) und neue Ereignis-Chunks erzeugt.

Problemlösungszyklus: Konfliktlösung und Eingriff

Die im Antizipationszyklus erkannten potenziellen Konflikte werden im *Problemlösungszyklus* durch Eingriffe in den Verkehr gelöst. An jedem Chunk des „pictures“, der einen potenziellen Konflikt repräsentiert, ist der vorhergesagte Eintrittszeitpunkt als Attribut gespeichert. Dadurch ist es im Problemlösungszyklus möglich, alle potenziellen Konflikte nach Dringlichkeit zu sortieren. In jedem einzelnen Durchlauf des Problemlösungszyklus‘ wird der jeweils dringlichste Konflikt bearbeitet. Für die Lösung der einzelnen Konflikte wird auf einige wenige stereotype Lösungsmuster zurückgegriffen, die an die aktuelle Situation angepasst werden. Dazu werden Konflikte klassifiziert. Jeder Konfliktklasse sind mehrere Standardlösungstypen zugeordnet, die eine Präferenzreihenfolge haben.

Eine so generierte Lösung wird innerhalb des Problemlösungszyklus‘ mit Prozeduren des Antizipationsmoduls auf ihre Auswirkungen auf den restlichen Verkehr hin getestet. Wenn keine Folgekonflikte auftreten, wird die geprüfte Lösung durch einen Eingriff in den Verkehr ausgeführt. Lösungen haben die Form eines Plans, also einer Sequenz von Handlungen, durch deren Ausführung ein potenzieller Konflikt vermieden wird. Chunks, die solche Pläne im „picture“ repräsentieren, haben als Attribut den Zeitpunkt, bis zu dem ihre Ausführung spätestens begonnen haben muss, damit der jeweilige Konflikt noch vermieden wird. Um eine Eingriffssequenz auszuführen, wird eine LISP-Funktion aus der ACT-COM-Softwareschicht zur Interaktion mit der Aufgabenumgebung benutzt.

Kontrollprozeduren

Die Dynamik der Situation erfordert das Vorhandensein einer Möglichkeit zur flexiblen Sequenzierung der Verarbeitungsprozesse. Außerdem muss es möglich sein, auf neue Ereignisse zu reagieren und die Sequenz der Verarbeitungsschritte daran anzupassen. Die Koordination der einzelnen Verarbeitungsschritte in den Modulen Datenselektion und Update, Antizipation und Konfliktlösung wird durch die *Kontrollprozeduren* ermöglicht.

In ACT-R 3.0 gibt es keine architekturimmanente Schedulingkomponente. Für MoFL wurde stattdessen ein anwendungsspezifisches Framework zum Wechsel zwischen mehreren parallel anstehenden Verarbeitungsschritten entwickelt. Die Aufgabenstruktur besteht aus einer Reihe parallel gültiger Ziele. So ist z.B. dem Update jedes einzelnen Flugzeugs ein Ziel zugeordnet. Aufgrund der Dynamik gibt es keine statische Aufgabenstruktur. Stattdessen kommen Ziele z.B. bei neu einfliegenden Luftfahrzeugen oder anderen Ereignissen hinzu, ändern sich oder werden obsolet. Jedes der Module Datenselektion, Antizipation oder Konfliktlösung repräsentiert einen Aufgabentyp. Die übergeordnete Aufgabe des Scheduling wird vom Modul „Kontrollprozeduren“ übernommen.

Das Scheduling in MoFL verwendet folgende Granularität für Tasks: Ein Task ist die Verarbeitung eines Objektes mit einem der Prozesse Update, Antizipation, Konfliktlösung. Tasks sind abgesehen von der Abarbeitung asynchroner Kommunikation in MoFL nicht unterbrechbar. Der Scheduler wählt den aktuell dringendsten Task und führt ihn aus. Im Fall einer Unterbrechung durch asynchrone Kommunikation wird ein hochaktivierter Chunk als Merker für den fortzusetzenden Task erzeugt. Nach der Abarbeitung eines Tasks wird die Dringlichkeit aller anstehenden Tasks neu geprüft.

Um die Tasks zu triggern und ununterbrechbar zu machen, wurde eine neue Klasse von Chunks eingeführt (s. Abbildung 5-5). Das Framework zum Multitasking in MoFL verlangt, dass ausschließlich solche Kontroll-Chunks auf dem Zielstapel verwendet werden. Wenn der so getriggerte Task abgearbeitet wird, kann der Goalstack mit weiteren Task-spezifischen Kontroll-Chunks verwendet werden. Der Kontroll-Chunk, durch den die aktuelle Aufgabe getriggert wurde, muss am Ende des Tasks vom Zielstapel entfernt werden. Als unterster Kontroll-Chunk bleibt während der ganzen Laufzeit das „Kontrollfluss-Ziel“, das die Kontrollprozeduren und damit das Scheduling des nächsten Tasks auslöst. In MoFL wird folgende Heuristik zur Auswahl des dringendsten Tasks benutzt:

1) Zuerst wird geprüft, ob eine bereits erstellte Lösung durch einen Eingriff in den Verkehr ausgeführt werden muss. Dabei wird die dort repräsentierte späteste Eingriffszeit berücksichtigt.
2) Dann werden anstehende Konfliktereignisse an die Konfliktlösung übergeben. Da die Konflikt-Chunks bei ihrer Erzeugung durch die Antizipation einen spätest-möglichen Lösungszeitpunkt erhalten, kann diese Information genutzt werden, um die Dringlichkeit jedes einzelnen Konfliktereignisses festzustellen.
3) Danach werden Monitoringereignisse und Flugzeug-Chunks mit Signalcharakter (neu einfliegend, Höhe wechselnd, nah zu anderen Luftfahrzeugen) geprüft. Für sie werden Update und Antizipation angestoßen.

4) Liegen keine anderen Aufgaben vor, die ausgeführt werden müssen, wird für das am längsten nicht aktualisierte extrafokale Luftfahrzeug Update und Antizipation angestoßen.

5.1.4. Validierung

Die Modellierung des „pictures“ und der Verarbeitungsprozesse stützen sich auf die Ergebnisse mehrerer Erhebungen. In den folgenden Abschnitten werden die Annahmen und der Entwurf des kognitiven Modells anhand der empirischen Belege plausibilisiert.

Es gab zwei Experimente zur *Datenselektion*. Sie zeigten, dass Konstruktion und Aufrechterhaltung des „pictures“ auf einer sehr reduzierten Datenbasis stattfinden. Zur Repräsentation werden hauptsächlich das „Callsign“, also die Kennung, des Luftfahrzeuges, dessen Flughöhe sowie die jeweilige Flugrichtung aufgenommen (Bierwagen et al. 1995, 1997). Darüber hinaus ergaben Interviews, ein Experiment und vergleichbare Studien aus der Literatur, dass Lotsen während des Abscannes des Radarbildschirms nach spezifischen Merkmalen von Luftfahrzeugen und Sektorgegebenheiten suchen, die auf eine weitere Überwachung oder auf ein Konfliktpotential des Luftfahrzeuges hinweisen.

Die unterschiedlichen *Update-Raten für fokale und extrafokale Objekte* werden durch Ergebnisse aus einem Gedächtnistest gestützt: Positionen von extrafokalen (irrelevanten) Flugzeugen wurden bei der Reproduktion zeitlich rückversetzt, wohingegen Positionen von aufmerksamkeitserfordernden (fokalen) Objekten (z.B. konfligierende und im Steig- oder Sinkflug befindliche) korrekt reproduziert wurden (Niessen & Eyferth 2001).

Die *Klassifikation von Verkehrskonstellationen* in „konflikthaft“, „sicher separiert“ und „weiter zu überwachen“ wurde durch Befunde aus einer Kategorisierung von Konstellationen durch Lotsen unterstüzt (Niessen et al. 1998).

Aufgrund der hohen Komplexität des Modells und des umfassenden Charakters der Simulation der kognitiven Vorgänge ist eine Validierung des Gesamtverhaltens nicht möglich. Eine Validierung des Timings des Modellablaufs und der Zeitparameter zur Konflikterkennung wurde deshalb nicht durchgeführt.

5.1.5. Ergebnisse

MoFL ist das erste implementierte kognitive Modell von mentalen Vorgängen bei Fluglotsen (Hilburn 2004). Die Modellierung basiert auf umfangreicher experimentalpsychologischer Empirie (Niessen & Eyferth 2001). MoFL war zur Zeit der Entwicklung einer der ersten Versuche, ACT-R für Problemlösen in Aufgabenumgebungen mit sich dynamisch ändernder Situation einzusetzen und einen Teil der

Interaktion des Bedieners mit einer komplexen simulierten Aufgabenumgebung architekturnah zu implementieren.

Als Fallstudie in dieser Arbeit ist der Zweck von MoFL, die Anwendbarkeit der ausgewählten kognitiven Architektur ACT-R für die Bedienermodellierung in dynamischen Mensch-Maschine-Systemen zu bestätigen, das neu entwickelte Werkzeug ACT-R Shell und das neu entwickelte Framework ACT-COM zu testen sowie etwaige offene Probleme bei der Verwendung von ACT-R, ACT-R Shell und ACT-COM zu identifizieren.

Bei der Modellbildung und der Implementierung von MoFL wurden die wesentlichen Lücken der Theorie von ACT-R identifiziert, die eine Anwendung in dynamischen Aufgabenumgebungen behindern. Anforderungen, die nicht mit den architektur-immanenten Mechanismen von ACT-R umgesetzt werden konnten, betreffen Multitasking, Kopplung mit der Aufgabensimulation, und das Update der mentalen Repräsentation der Situation als Folgerungen aus der dynamischen Situation.

Deshalb wurden ACT-COM und das Framework zur Realisierung von Multitasking konzipiert und entwickelt. Der Einsatz dieser Frameworks in MoFL hat gezeigt, dass das Scheduling-Framework auf die konkrete Anwendung Flugsicherung mit relativ großen Zeitkonstanten und einer überschaubaren Menge paralleler Tasks begrenzt ist, es wird deshalb nicht als eigenständiges und übertragbares Ergebnis in Abschnitt 6 dargestellt.

ACT-COM ist hingegen ein verallgemeinerbares Konzept zur protokollgestützten Kommunikation mit einer externen Aufgabenumgebung, das mit wenig Aufwand an existierende Simulatoren angepasst werden kann. ACT-COM beinhaltet jedoch nur wenige wahrnehmungsbezogene Einschränkungen. Da mit den PM-Modulen in neueren ACT-R Versionen eine Architektur zur Modellierung und Simulation von Wahrnehmung und Motorik zur Verfügung steht, die für sich genommen jedoch aus softwaretechnischer Sicht noch nicht ausreichend gut zu verwenden ist (s. Fallstudie DURESS im nächsten Abschnitt), wird mit Agimap noch ein weiteres Framework für die Kopplung neu entwickelt. Da es erst später entstanden ist, konnte Agimap in der Fallstudie MoFL noch nicht eingesetzt werden.

Der hohe Aufwand bei der Benutzung von ACT-R als Modellierungs- und Simulationswerkzeug und die damit einhergehende hohe Fehlerwahrscheinlichkeit bei der Verwendung haben deutlich gemacht, dass eine zusätzliche Werkzeugunterstützung für die anwendungsorientierte Verwendung erforderlich ist. ACT-R Shell ist ein erstes Werkzeug, das bei Modellimplementierung, Simulation und Interpretation der Simulationsergebnisse unterstützt. Ohne dieses Werkzeug wäre es praktisch nicht möglich gewesen, eine lauffähige Simulation von MoFL zu realisieren.

Neben den Lücken von ACT-R wurde jedoch auch das Potenzial deutlich, das eine Modellierung mit einer *unified theory of cognition* (s. Abschnitt 3.2) beinhaltet.

So können bspw. die vorgegebenen Begrenzungen des Arbeitsgedächtnisses direkt übernommen werden, um die eingeschränkte Verfügbarkeit von Chunks in Entscheidungsprozessen zu simulieren, und die eingebauten und erprobten Lernmechanismen von ACT-R können ohne zusätzlichen Modellierungsaufwand benutzt werden, um die Eignung neuer Sektoren simulativ zu prüfen.

Im nächsten Abschnitt wird die zweite Fallstudie DURESS beschrieben.

5.2. DURESS

Diese Fallstudie wurde während eines Gastaufenthaltes am Cognitive Systems Engineering Lab (CSE) der University of Tokyo am Lehrstuhl von Prof. Furuta mit Unterstützung von DAAD und JSPS[5] durchgeführt. Sie hat zum Ziel, die beiden neu entwickelten Erweiterungen Agimap und Planex anzuwenden und eine Bewertung der dazugehörenden Werkzeuge zu ermöglichen. Die Fallstudie basiert auf einer Experimentalumgebung vom CSE und auf vom CSE erhobenen Daten. In dieser Fallstudie stehen die wahrnehmungsnahen und automatisiert ablaufenden kognitiven Programme im Vordergrund, wie sie in einfachen Kontrollaufgaben benötigt werden. Im Step Ladder Framework von Rasmussen (1983) ist ein Großteil der Aufgabe auf der Skill-Ebene anzusiedeln.

5.2.1. Beschreibung der Mikrowelt

DURESS ist eine Mikrowelt, die am Cognitive Engineering Laboratory der Universität Toronto entwickelt wurde, um ökologisches Interface Design zu untersuchen (Vicente 1995, Vicente et al. 1995). DURESS steht für *dual reservoir system simulation*. Es ist ein Modell einer Prozesskontrollsituation eines Speisewasserzuflusses mit sechs Ventilen, zwei Pumpen und zwei Heizelementen (s. Abbildung 5-7). Es gibt elf Stellgrößen, die Pumpenleistung, Ventildurchlass und Heizleistung bestimmen. Als Zielgrößen werden Temperatur- und Durchfluss für zwei Auslassstellen vorgegeben.

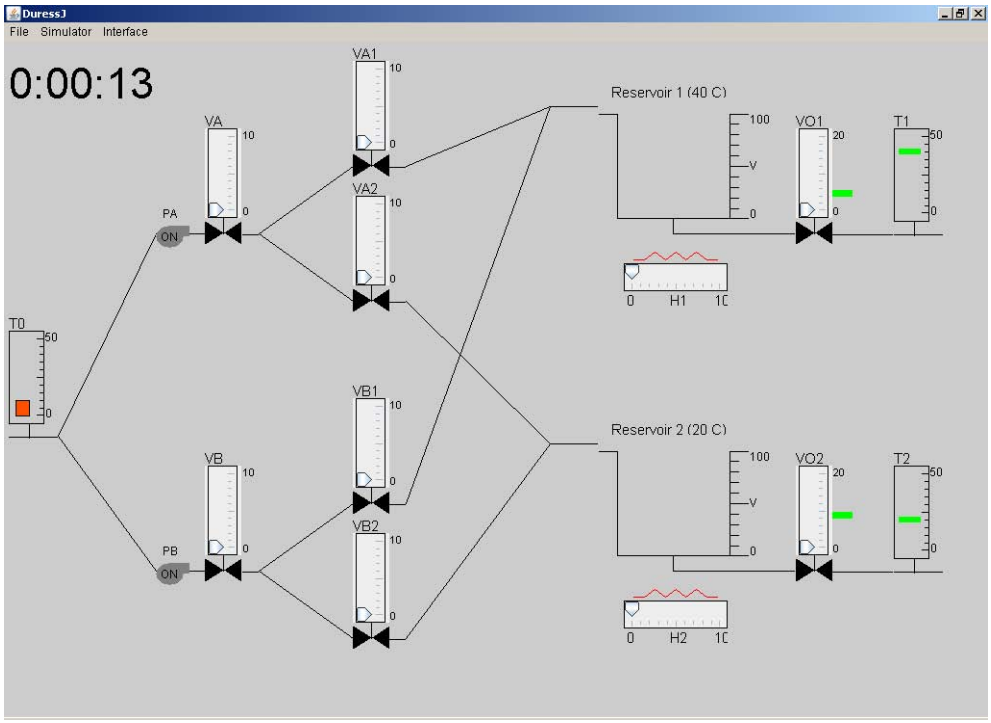


Abbildung 5-7: Screenshot von DuressJ

[5] Deutscher Akademische Austauschdienst, Japan Society for Promoting the Sciences im Programm „Summer Fellowship“

Die Leistung der zwei Pumpen und die Durchlassflüsse aller Ventile müssen geregelt werden, um einen extern vorgegebenen Füllstand der beiden Tanks zu erreichen. Die Tanks werden beheizt, hier ist die Heizleistung zu regeln, damit der Auslass aus beiden Tanks mit einem extern vorgegebenen Auslass an Durchsatz und Temperatur erreicht wird. Es gibt an den Heizelementen eine Automatik, die die Heizung bei leerem Tank abschaltet. Die Prozessvariablen werden im Strukturbild der simulierten Anlage als numerische Werte und mit einer analogen Repräsentation als vertikale oder horizontale Skala angezeigt.

Furuta verwendet die funktional äquivalente und in der Gestaltung an einen eigenen Widget-Satz angepasste Version J-DURESS[6] (s. Abbildung 5-8) (Kanno et al. 2003). Die Werte der Prozessvariablen werden am selben Widget gesteuert und dargestellt. Es gibt nur an den Tanks eine visuelle analoge Repräsentation anhand des Füllstandes. An den Bedienelementen kann der Wert nur in einzelnen Schritten erhöht oder verringert werden. Wo es keine Bedienelemente gibt, werden Werte digital als Schrift angezeigt. Zusätzlich gibt es am Kopf des Fensters eine Doppelreihe aller möglichen Alarme. Tritt eine Verletzung einer Bedingung auf, wird das entsprechende Feld der Kopfleiste rot hinterlegt. Wenn die Bedingung nicht mehr besteht, ändert sich die Hintergrundfarbe wieder zu hellgrau. Voralarme gibt es in dieser Mikrowelt nicht.

Zur Analyse des Operator Cognitive Crew System (OCCS, Furuta & Kondo 1993, Furuta et al. 1999) wurde die funktionale Struktur der Steuerungsaufgabe in DURESS als Plan Library formalisiert abgelegt. Die 769 repräsentierten Pläne bestehen im Wesentlichen aus einem Ziel und einer dazugehörigen Handlungssequenz. Zu einem Ziel gibt es mehrere alternative Pläne. Die Qualität der einzelnen Pläne in Bezug auf Sicherheit, Verlässlichkeit, Effizienz und Effektivität ist in Furutas Plan Library nicht repräsentiert. Die Ziele der Pläne beziehen sich auf Prozessvariablen an den Tanks (s. Tabelle 5-1) und an den Auslassen (s. Tabelle 5-2).

[6] Dies ist nicht die Version DuressJ von B. Cosentino und A. Ross.

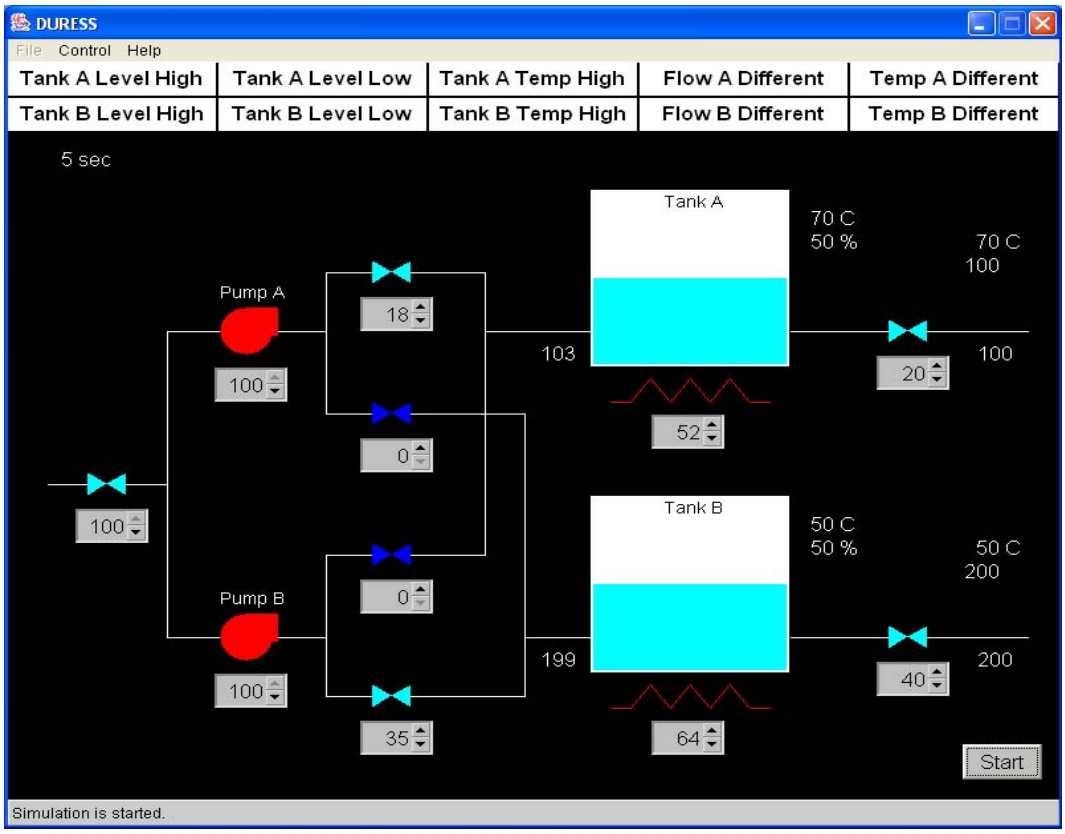


Abbildung 5-8: Screenshot von J-DURESS

Tabelle 5-1: Pläne mit Prozessgrößen zu den Tanks

	Zielbeschreibung	
Plan	**Tank**	**Richtung**
1	A	Mehr
2	B	Weniger
190	…	

Tabelle 5-2: Pläne mit Prozessgrößen zu den Auslassen

	Zielbeschreibung		
Plan	**Auslassventil**	**Richtung**	**Variable**
191	Oben	Höher	Temperatur
192	Oben	Mehr	Auslass
193	Unten	Niedriger	Temperatur
194	Unten	Weniger	Auslass
769		…	

5.2.2. Erweiterungen an der Simulation

Die DURESS-Mikrowelt in der Ausprägung J-DURESS wurde herangezogen, um die Anwendbarkeit der neuen ACT-R/PM-Erweiterungen Agimap und Planex zu

erproben. Ein planbasiertes kognitives Modell auf der Ebene der Blickbewegungen wurde dafür zur Online-Steuerung der Mikrowelt J-DURESS entwickelt und integriert.

J-DURESS ist in Java mit proprietären JBuilder-Bibliotheken für die Widgets programmiert. Die Programmierung von J-DURESS folgt einem eventgesteuerten Architekturansatz und nicht dem Publisher-Subscriber Design Pattern der Model View Controller Architektur (MVC), für das Agimap ursprünglich entwickelt wurde. Trotzdem war es sehr einfach, J-DURESS mit einem externen ACT-R/PM-Prozess per Interprozesskommunikation zu verbinden und das Agimap Kommunikationsprotokoll zu implementieren.

Der Aufwand für die Analyse der undokumentierten und unkommentierten Legacy-Software von ca. 3500 Java LOC, wovon sich etwa 2200 Zeilen auf die Simulation des Prozesses und der Rest auf das GUI beziehen, lag im Rahmen von etwa zwei Personentagen.

Für die Anpassung der Eventstruktur an das Publisher-Subscriber Schema und die Anbindung eines externen Prozesses über Sockets wurde die generische Klasse `RemoteControl` als Java-Klasse entwickelt (s. Abbildung 5-9). `RemoteControl` ist ein Singleton, d.h. es gibt höchstens eine Instanz dieser Klasse zur Laufzeit im Objektsystem des Prozesses. Auf die Instanz kann mit der Klassenmethode `getInstance()` zugegriffen werden. Falls noch keine Instanz existiert (in dem Klassenattribut `instance` gespeichert), wird ein Objekt instanziiert.

Das Object `RemoteControl` ist ein `Thread`. Das bedeutet, dass es nebenläufig zum Rest der Anwendungslogik abläuft. In diesem nebenläufigen Prozess wird über TCP/IP ein Socketserver auf einem zu definierenden Port (`RemoteControl.PORT`) bereitgestellt. Versucht ein Client sich mit diesem Server zu verbinden, wird ein weiterer nebenläufiger Prozess abgespalten, der ab diesem Zeitpunkt exklusiv für die Kommunikation mit genau diesem Client zuständig ist. Der Server-Thread bleibt so für zukünftige Verbindungsversuche anderer Clients verfügbar. Die Threads, die für jeden Client instanziiert werden, sind Instanzen der Klasse `ClientHandler`, die eine innere Klasse von `RemoteControl` ist, damit ihre Instanzen auf denselben Datenraum wie `RemoteControl` zugreifen können.

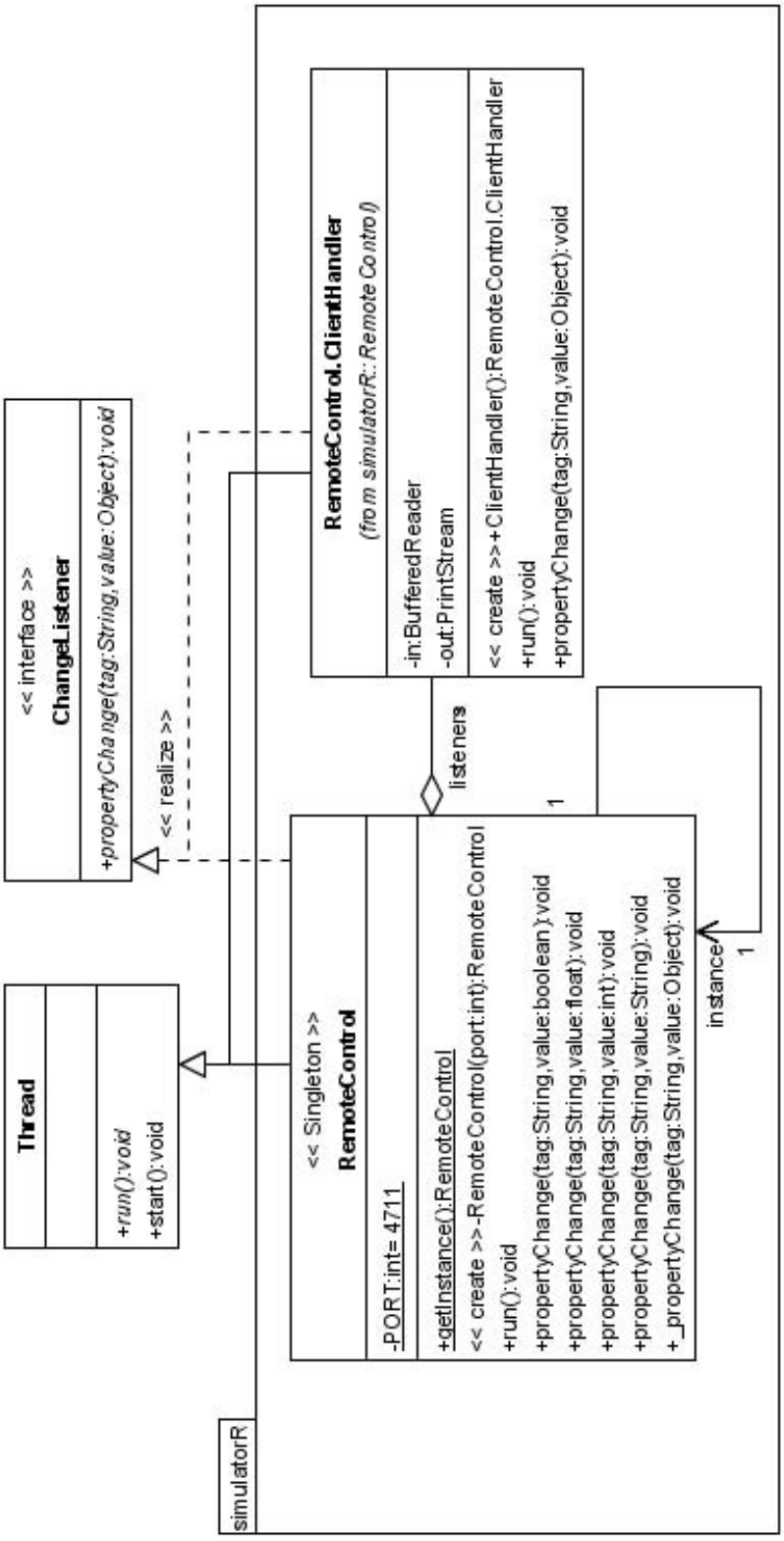


Abbildung 5-9: Klassendiagramm Remote Control

Alle abgespaltenen `ClientHandler` Threads werden in der privaten Datenstruktur `listener` von `RemoteControl` als `ChangeListener` referenziert. Dadurch kann

das Publisher-Subscriber Design Pattern (Gamma et al. 1995) implementiert werden. Alle verwendeten Widget-Klassen wurden zu JavaBean-artigen Klassen umgebaut. Die Setter-Methoden dieser Beans benachrichtigen die Instanz von `RemoteControl` bei Änderungen des im Widget angezeigten Wertes über die `propertyChange(String,_)` Methode. Der erste Parameter gibt den Namen der geänderten Prozessvariablen an, der zweite den neuen Wert. Da das Modell des jeweiligen Kontrollwidgets vielfältig sein kann, hat diese Methode eine Vielzahl von Signaturen für unterschiedliche Wertetypen. Wegen des schlechten Designs der Legacy Anwendung konnten diese Aufrufe von propertyChange`(String,_)` nicht im Modell angebracht werden, sondern mussten in allen Controllern, die das Modell ändern, verankert werden. Dadurch war das Einbringen von ca. 35 Aufrufen von `RemoteControl.getInstance().propertyChange(tag,value)` erforderlich. Bei einer Wertänderung wird der entsprechende Aufruf an `RemoteControl` über die `ClientHandler` in `listener` an alle über TCP/IP verbundenen Clients weitergeleitet.

Die Implementierung von `propertyChange` folgt dem Kommunikationsprotokoll von Agimap. Die Klasse `ClientHandler` implementiert in der `run()` Methode die Kommandoverarbeitungsschleife des Agimap-Protokolls über die Streams `in` und `out`. In der Implementierung werden die folgenden Kommandos unterstützt: `GET`, `SET`, `INC` und `DEC`.

Wird ein Wert geändert, werden nicht nur alle `listener` benachrichtigt, sondern der neue Wert wird auch anhand des `tag`-Strings in einer `Hashmap` in der Instanz von `RemoteControl` gespeichert. Dadurch ist es beim Aufbau der Verbindung eines Clients über einen neu instanziierten `ClientHandler` einfach möglich, die bisher passierten Wertänderungen „in einem Rutsch“ durchzureichen.

Der Aufwand für die Anpassung und Integration dieser generischen Softwarekomponente zur Einpassung in die bereits existierende anders geartete Architektur der Legacy-Anwendung war gering und lag im Bereich von mehreren Personenstunden.

5.2.3. Kognitive Modellierung

Die Anbindung auf der ACT-R/PM-Seite wurde mit Hilfe des Agimap-Editors erstellt. Dazu wurden die Widget-Typen Button und Text und die Mapper MapText2Color, MapText und MapNumber2Pos verwendet. Aus einem Screenshot der Mikrowelt wurden die entsprechenden Anzeige-Stellen mit einem Grafikprogramm ausgeschnitten (s. Abbildung 5-10). Das resultierende Hintergrundbild für Agimap wurde in den Editor geladen und die Widgets und Mapper an die entsprechenden Stellen gesetzt.

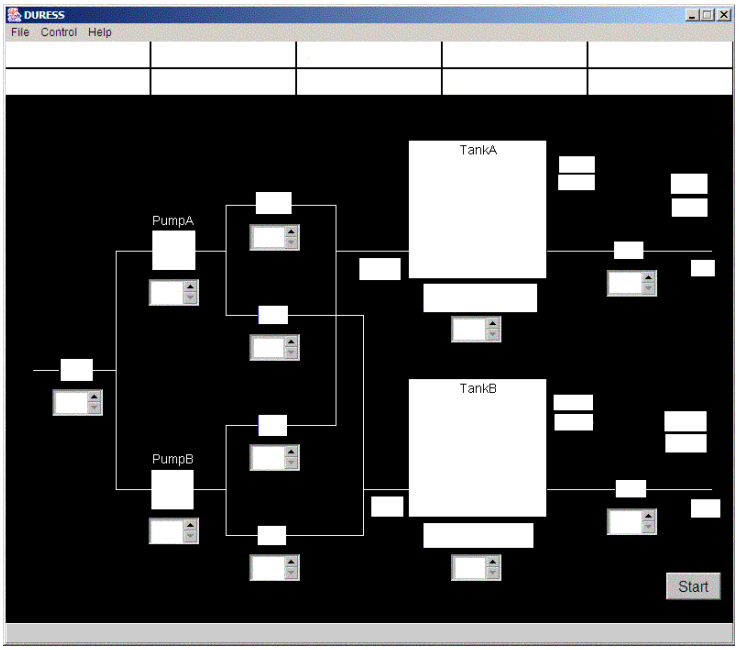


Abbildung 5-10: Hintergrundbild J-DURESS für Agimap

Die interaktiven Elemente sind kleine Buttons, die mit entsprechenden Befehlen für die Simulation gekoppelt sind. In Quelltext 5-1 ist ein Teil des Widgets der oberen Pumpe wiedergegeben. Der korrespondierende tag der oberen Pumpe in der Simulation ist `PumpA.power`. Entsprechend wird das Kommando `inc PumpA.power` an den `ClientHandler` geschickt.

```
<Button
        width="20"
        x="200.0"
        height="20"
        text="+"
        y="290.0"
        actid="incpumpa"
        action="inc PumpA.power" />
```

Quelltext 5-1: XML Codefragment der Agimap Repräsentation von J-DURESS

Die gesamte Definition der Bedienungsoberfläche der Mikrowelt besteht für Agimap aus 74 Elementen. Die XML-Repräsentation wird von der Agimap-Toolchain automatisch in ACT-R/PM Code übersetzt. Das resultierende LISP File ist 761 LOC groß und kann zusammen mit den Agimap LISP-Modulen direkt in das ACT-R/PM-Programm eingebunden werden. Der Aufwand für den Entwurf am Agimap-Editor und die Integration betrug wenige Personenstunden.

Die Ablaufkontrolle des kognitiven Modells basiert auf der Plan Library von Furuta. Die 769 Pläne lagen in einer Repräsentation als Prolog Quelltext vor. Mit

einem einfachen Perl-Programm konnten daraus ohne großen Aufwand direkt ACT-R/PM Statements für die Verwendung mit Planex generiert werden (s. Quelltext 5-2).

```
(add-dm (method_0674
            isa       method
            goal      increase_outlet_temp_of_valve6
            step1     valve1_decrease
            step2     pumpB_decrease
            step3     valve4_decrease
            step4     valve2_decrease
            step5     heaterA_increase))
```

Quelltext 5-2: ACT-R/PM Repräsentation einer Methode für Planex

Für die Verwendung mit Planex wurden die Ziele und Operatoren anhand der Definitionen aus der Plan Library abgeleitet (s. Tabelle 5-3). Die Ziele haben symbolische Namen, die in der Definition der Chunks für Methoden für den Slot `goal` verwendet werden. Die Operatoren werden als symbolische Namen für die Schritte verwendet.

Tabelle 5-3: Abgeleitete Ziele und Operatoren von DURESS für Planex analog Tabelle 5-1 und Tabelle 5-2

Ziele	Operatoren	Operatoren forts.
decrease_level_of_tankA	pumpA_increase	valve2_decrease
decrease_level_of_tankB	pumpA_decrease	valve3_increase
increase_level_of_tankA	pumpB_increase	valve3_decrease
increase_level_of_tankB	pumpB_decrease	valve4_increase
decrease_outlet_flow_of_valve6	heaterA_increase	valve4_decrease
decrease_outlet_flow_of_valve7	heaterA_decrease	valve5_increase
decrease_outlet_temp_of_valve6	heaterB_increase	valve5_decrease
decrease_outlet_temp_of_valve7	heaterB_decrease	valve6_increase
increase_outlet_flow_of_valve6	valve1_increase	valve6_decrease
increase_outlet_flow_of_valve7	valve1_decrease	valve7_increase
increase_outlet_temp_of_valve6	valve2_increase	valve7_decrease
increase_outlet_temp_of_valve7		

Für jeden Operator gibt es eine zugeordnete Produktion `initiate_action_<<operatorname>>`, die feuert, wenn der Operator zur Ausführung kommen soll. Die Produktionen, die für die DURESS Mikrowelt verwendet werden, haben alle dieselbe Struktur. Quelltext 5-3 zeigt die Ausführung des Operators `valve3_increase`. Wenn die Planex Umgebung den Operator `valve3_increase` zur Ausführung auswählt, kann diese Produktion feuern, dadurch den Zustand im Planex Framework ändern und erzeugt ein Unterziel zur Umsetzung des Operators mit Agimap. In diesem Fall wird der Button `incvalve3` fünf Mal geklickt werden. Dazu werden alle Agimap Mechanismen zum Suchen des Ziels, zum Bewegen der Maus auf das Ziel und zum wiederholten Klicken verwendet werden. Wenn die Agimap-Verarbeitung des Operators beendet ist, wird das Framework zum ursprünglichen Ziel `goal` mit `operator valve3_increase` zurückzukehren versuchen.

```
(P initiate_action_valve3_increase
   =goal>
     isa goal
     state operator_selected
     operator valve3_increase
  ==>
   =goal>
     state operator_engaged
   +goal>
     isa Agimap-clickButton
     state retrieve
     target incvalve3
     numberofclicks 5
     topgoal =goal
)
```

Quelltext 5-3: ACT-R/PM Produktion zur Ausführung eines Operators

Der Rest des kognitiven Modells in ACT-R/PM ist für die Einbindung der planbasierten Verarbeitung zuständig. Der stuffing-Mechanismus des Vision-Moduls von ACT-R/PM wird verwendet, um Alarmzustände in der Aufgabenumgebung zu entdecken. Dazu sind elf Produktionen erforderlich. Dann wird die Situation von einer aus 42 Produktionen analysiert und der entsprechende Plan aus dem Arbeitsgedächtnis abgerufen. Der ausgewählte Plan wird danach mit dem Planex Framework unter Zuhilfenahme von Agimap ausgeführt.

5.2.4. Validierung

Das resultierende Modell ist stark auf Perzeption fokussiert. Die Auswahl eines korrekten Plans erfordert zwar einige logische Schlüsse, die Aufgabe ist jedoch einfach und auf Skill-Ebene lösbar. Es liegen einige empirische Daten zur Evaluation des kognitiven Modells vor (Kanno et al. 2003). Sie beziehen sich auf Teamarbeit an

der J-DURESS Mikrowelt (s. Abbildung 5-11). Jedes interaktive Element der Aufgabenumgebung wird einem von zwei zusammenarbeitenden Operateuren zugeordnet. Die Zuordnung ist beiden transparent. Beide können aber den gesamten Zustand der Anlage sehen. Eine Reihe von Testszenarien wurde anhand dieses Experimentalaufbaus durchgespielt. Im Verlauf der Durchläufe wurden zu definierten Zeitpunkten Störungen bzw. Zieländerungen in die Simulation eingespielt.

Es hat sich jedoch gezeigt, dass die dabei erhobenen Daten für eine Evaluierung des kognitiven Modells nicht ausreichen. Die Daten von Kanno et al. (2003) lassen nicht auf zu bevorzugende Pläne schließen. Die Daten können aber dazu verwendet werden, das Timing in der Interaktionsverarbeitung von Agimap zu kalibrieren.

Die Team-Charakteristik wurde durch eine einfache Modifikation und Duplizierung des Modells nachvollzogen. Der Java-Prozess der Simulation ist durch zwei parallele Clients (also insgesamt vier parallel laufende Threads) auf einem Notebook der Pentium M Klasse nicht merklich verlangsamt worden und konnte weiterhin in Echtzeit betrieben werden. Ein Abgleich mit den Testszenarien wurde durchgeführt. Wenn auch keine formal gültige Evaluierung des Modells durchgeführt werden konnte, ergaben sich doch in allen Situationen sinnvolle Bediensequenzen mit plausiblem Timing.

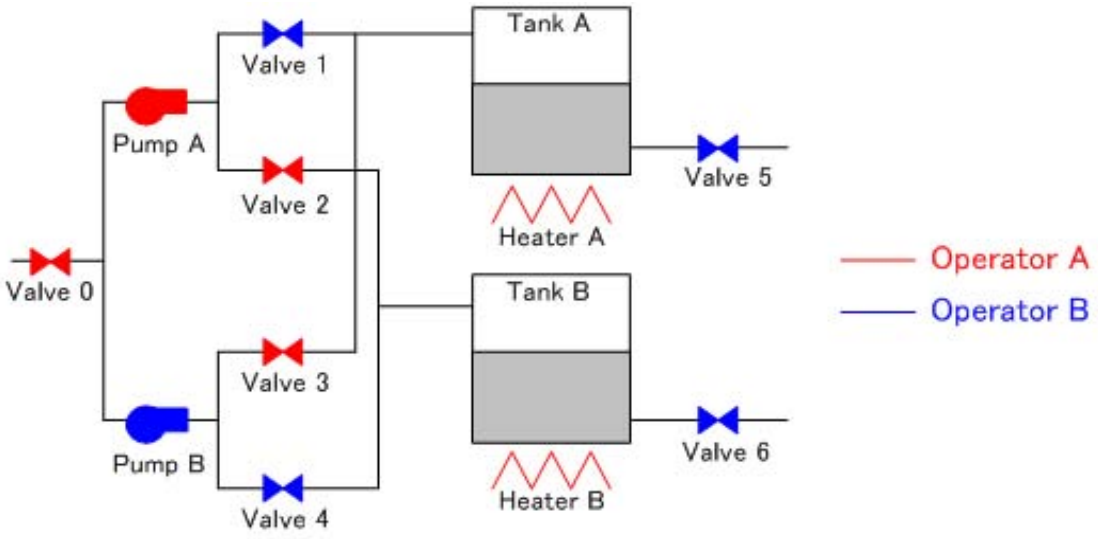


Abbildung 5-11: Teamaufgabe J-DURESS (Kanno et al. 2003)

5.2.5. Ergebnisse

Die Entwicklung des DURESS-Modells dient in dieser Arbeit als Fallstudie, mit der die Anwendbarkeit der ausgewählten kognitiven Architektur ACT-R und der Aufgabenmodellierung mit KLM im Zusammenspiel mit dem neu entwickelten Werkzeug und Framework Agimap bzw. mit dem Framework Planex für die Bedienermodellierung in dynamischen Mensch-Maschine-Systemen überprüft wurde.

Als erstes Ergebnis ist festzuhalten, dass die KLM-artige Repräsentation der Plan-Library ein einfach zu verwendender Formalismus ist, mit dem das prozedurale

Wissen für die Bedienermodellierung gespeichert werden konnte. ACT-R konnte zudem erfolgreich als kognitive Architektur für die Modellierung der DURESS-Bedienung verwendet werden. Für die Verbindung von KLM und ACT-R ist jedoch die Neuentwicklung des Frameworks Planex erforderlich gewesen, mit der die KLM-Prozeduren in ACT-R integriert werden konnten.

Aus der Fallstudie MoFL war bereits bekannt, dass die einfache Kopplung mit externen Aufgabenumgebungen über ACT-COM möglich ist. Um aber auch die wahrnehmungsnahen Eigenschaften der kognitiven Architektur, die mit der PM-Erweiterung von ACT-R verfügbar wurden, nutzen zu können, ist erheblicher Modellierungsaufwand erforderlich. Das Framework und Werkzeug Agimap, das deshalb neu entwickelt wurde, konnte Effektivität und Effizienz der Modellierung und Realisierung der Kopplung erheblich verbessern.

Der Entwurf der Software-Architekturen für Agimap und Planex war für den Anwendungszweck geeignet. Auf der Seite des kognitiven Modells sind die Leistung und der Integrationsaufwand für die Verwendung von Planex und Agimap zu bewerten. Auf der Ebene von Planex (Ziele, Pläne, Methoden und Operatoren) und auf der Ebene von Agimap (Operatoren, Ereignisdetektion) arbeiten die Frameworks in sich geschlossen. Die Schnittstellen zwischen diesen Systemen sind klar und erfordern nur geringen Integrationsaufwand.

Die Kommunikation mit der externen Simulation arbeitete sicher und war mit sehr geringem Aufwand einzurichten. Der Aufwand für die Anpassung der externen Simulation war gering, obwohl eine andere Software-Architektur beim Legacy-System vorliegt. Jedoch war von großem Vorteil, dass das Legacy-System in Java programmiert war. Wäre eine andere Programmiersprache verwendet worden oder läge kein Quellcode vor, hätte eine neue Komponente z.B. nach dem Mediator Design Pattern (Gamma et al. 1995) – dabei wird die Kommunikation zwischen den Komponenten nicht direkt, sondern durch zusätzlich eingeschaltete Komponenten (Mediatoren) abgewickelt – entwickelt werden müssen. Deren Wiederverwendbarkeit wäre mit hoher Wahrscheinlichkeit nicht gegeben.

Da bei der Integration mehrere Toolchains zur Unterstützung bereitstehen und Quellcode-Transformationen (z.B. Plan Library nach ACT-R/PM Planex method-Chunks) nicht von Hand, sondern mit Perl-Skripten durchgeführt wurden, traten bei der Integration nur wenige und dann sofort sichtbare Fehler auf.

Im nächsten Abschnitt wird die dritte Fallstudie zur Bedienermodellierung mit ACT-R vorgestellt. Dort geht es um ein Bedienermodell zum Anfahren einer Rektifikationskolonne. Ähnlich wie bei DURESS ist die Aufgabenumgebung ein computergestütztes Prozessleitsystem, das auch die Simulation des Prozesses beinhaltet.

5.3. Rektifikationskolonne

Diese Fallstudie wurde in der Nachwuchsforschergruppe *Modeling of User Behaviour in Dynamic Systems* (MoDyS) an der TU Berlin durchgeführt. Es war das Ziel, die kognitive Modellierung im Anwendungsbereich der Kontrolle chemischer Prozesse einzusetzen und dabei die Methodik mit den neu entwickelten Werkzeugen Agimap und Timer zu testen. Gleichzeitig wurden in dieser Fallstudie die Begrenzungen der Standardmethoden zur hierarchischen Aufgabenanalyse mit GOMS-Methoden aufgezeigt.

5.3.1. Beschreibung der Mikrowelt

Die Rektifikationskolonne ist eine Mikrowelt, mit der mehrere Versuchsreihen zu Fragen der Zeitwahrnehmung und -verarbeitung durchgeführt wurden. Es handelt sich um die Simulation eines Prozesses, der auch in einer realen Versuchsanlage am Fachgebiet Dynamik und Betrieb technischer Anlagen der TU Berlin abläuft. Die Simulation und die Visualisierung der Benutzungsschnittstelle wurden mit dem System ProperEduct (Urbas 1999) realisiert. ProperEduct stellt eine netzwerkfähige komponentenorientierte Middleware für Prozesssimulation und für die Visualisierung der Prozesskontrollschnittstelle auf Basis von Java 2 SE zur Verfügung.

Die Aufgabe bei der Steuerung der Rektifikationskolonne ist einer Situation beim Anfahren des Prozesses entnommen. Im Tank A wird ein flüssiges Stoffgemisch erhitzt (Abbildung 5-12); Dampf steigt auf und wird in der Kolonne B abgekühlt, im Kopf der Kolonne fließen die getrennten Stoffe als Kondensat (C). Es gibt vielfältige Fehlersituationen in der Anlage. Beispielsweise können Ventile oder Leitungen undicht sein. Einige Fehlersituationen können dadurch entdeckt werden, dass auch nach längerer Zeit in C kein Stofffluss beginnt. Welche Zeit vergehen muss, bis der Stofffluss beobachtet werden kann, hängt von den Stoffen und anderen Parametern des Prozesses ab. Bedingung ist jedoch, dass die Heizung im Bereich A arbeitet, damit die Zeitspanne beginnen kann.

In den experimentellen Untersuchungen wurde als Ziel für die Versuchspersonen vorgegeben, dass nach einer bestimmten Zeit im Bereich von einer bis wenigen Minuten der Fluss nach Tank C beginnen muss (Schulze-Kissing et al. 2005). Wenn diese Bedingung nicht eintritt, liegt eine Störung in der Anlage vor. In diesem Fall muss die Anlage mit einem Not-Aus abgeschaltet werden (Abbrechen-Button unten rechts). Bildet sich Flüssigkeit in Tank C, ist die Aufgabe die Anlage anzufahren erfüllt und der OK-Button muss geklickt werden. Als experimentelle Bedingungen kamen unterschiedliche Belastungen bei der Führung des Prozesses im Bereich A durch Wahl der Prozessparameter vor. Die Instruktion war so gewählt, dass sowohl retrospektive, als auch prospektive Zeitschätzung induziert wurde.

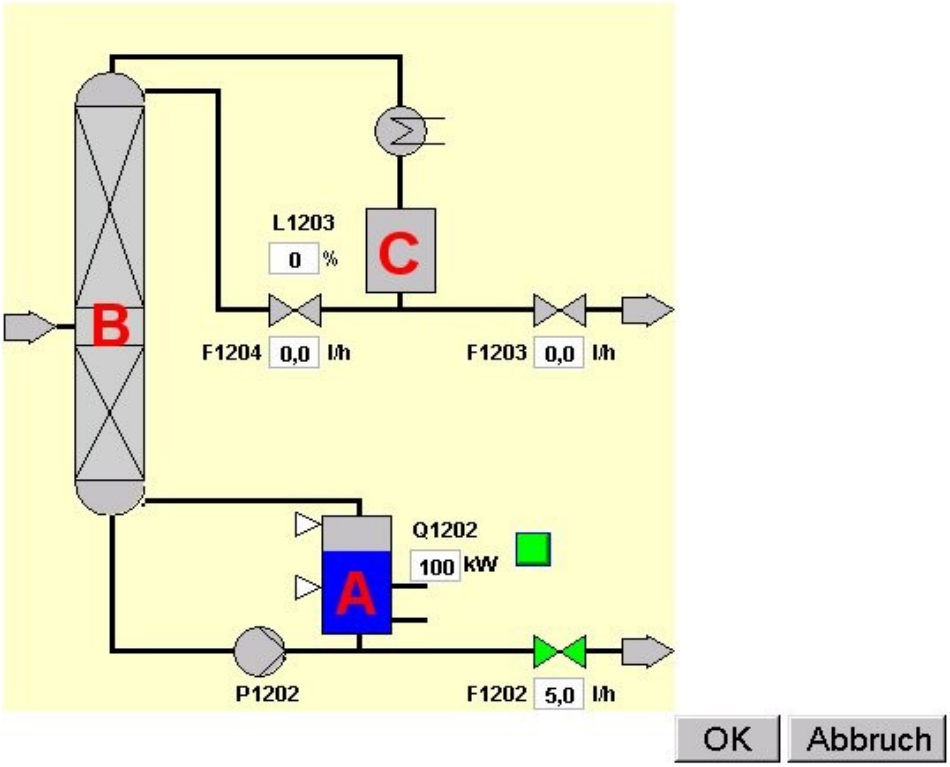


Abbildung 5-12: Screenshot der Mikrowelt Rektifikationskolonne

5.3.2. Erweiterungen an der Simulation

ProperEduct basiert vollständig auf dem Model View Controller Design Pattern (Gamma et al. 1995), das über das JavaBeans Komponentenframework umgesetzt wurde. Das Framework hat auch ein Kommunikationsprotokoll, mit dem der Prozess von außen gesteuert werden kann. Alle Aktionen, die mit einer ProperEduct eigenen Benutzungsoberfläche durchgeführt werden können (Werte auslesen, Werte manipulieren), können so auch von außen aktiviert werden. Das Protokoll definiert eine einfache Sprache, mit Prädikat-Objekt Syntax. Jede Größe in einem Prozessmodell in ProperEduct hat einen eindeutigen Bezeichner, der als Objekt benutzt werden kann. Die Prädikate repräsentieren die möglichen Aktionen.

ProperEduct wurde um ein Kommunikationsmodul erweitert, mit dem die Mikrowelt Rektifikationskolonne an ein kognitives Modell angeschlossen wurde. Das Modul wurde in derselben Form noch in anderen Mikrowelten (z.B. Müllverbrennungsanlage von Schoppek) zur Fernsteuerung durch externe kognitive Modelle angewendet.

Die Klasse `DCSModServer` kann als zentraler Einstiegspunkt für ProperEduct-Programme benutzt werden, wenn es als Simulationsserver für ein kognitives Modell benutzt werden soll. Sie enthält genau zwei Klassenmethoden, d.h. `DCSModServer` wird nicht instanziiert, sondern dient nur als Modul, um die Simulationsserverfunktionalitäten zu kapseln. Die Methode `startServer(int)` öffnet einen `ServerSocket` und startet damit einen `ListeningThread`, der

nebenläufig auf ankommende Verbindungswünsche von kognitiven Modellen wartet. Über die Methode `main(String[])` in `DCSModServer` wird das Programm gestartet. Als Kommandozeilenparameter wird der Pfad zu einer Konfigurationsdatei angegeben, die die Anlagenkonfiguration enthält. Mit diesem File wird das ProperEduct-Framework parametriert und die Prozesssimulation gestartet.

Verbindet sich ein Client mit dem `ListeningThread`, der nebenläufig zur Prozesssimulation auf einem vorgegebenen Port auf TCP/IP Verbindungswünsche wartet, wird ein weiterer Thread von der Klasse `ServerThread` gestartet, der die Verbindung mit dem neuen Client bedient. Er leitet Anfragen in der ProperEduct-Protokollsprache an die entsprechende Komponente im Framework weiter und benachrichtigt den Client über Änderungen der Prozess- und Stellgrößen. Dazu bedient es sich der Property-Change-Infrastruktur von JavaBeans: Das ProperEduct-Framework basiert auf JavaBeans-Komponenten. Jedes Stellteil und jede explizit repräsentierte Prozessgröße ist als ein solches zustandsbehaftetes Bean implementiert. Der Zustand wird durch *Properties*, die Werte annehmen können, repräsentiert. ProperEduct implementiert an zentraler Stelle das JavaBeans Interface `PropertyEditor`. Hier sind alle JavaBeans, die Stellteile und relevante Prozessgrößen repräsentieren, bekannt. An diesem Objekt können sich sogenannte *Listener*, also Objekte, die das `PropertyChangeListener` Interface implementieren, anmelden. Wenn der `PropertyEditor` eine Änderung des Zustandes des Beans registriert, wird allen angemeldeten *Listenern* ein `PropertyChangeEvent` Objekt geschickt, in dem das betroffene *Property* und dessen neuer Wert repräsentiert sind.

Diese Software-Architektur macht sich die `DCSModServer`-Komponente zu Nutze (s. Abbildung 5-13): Wenn ein neuer `ServerThread` für einen Client gestartet wird, wird für alle verwendeten *Properties* jeweils ein neuer `Listener erzeugt`, der beim zentralen `PropertyEditor` angemeldet wird. Dieser Umweg über Instanzen der Klasse `Listener` ist notwendig, weil die `PropertyChangeEvent`s `evt` kein korrektes Ergebnis für `evt.getPropertyName()` liefern. Deshalb wird der *Property*-Name im Attribut `property` der `Listener`-Objekte repräsentiert, so dass die Methode `propertyChange(evt)` über `this.property` auf den *Property*-Namen zugreifen kann.

Wird nun ein *Property* geändert, bekommen alle angeschlossenen Clients über ihren `ServerThread` eine Nachricht mit dem entsprechenden `PropertyChangeEvent`. Die Methode wandelt Property-Namen (ProperEduct-intern *tag*) und Wert in die dem Agimap-Protokoll entsprechende Repräsentation und liefert sie über die Socketverbindung aus.

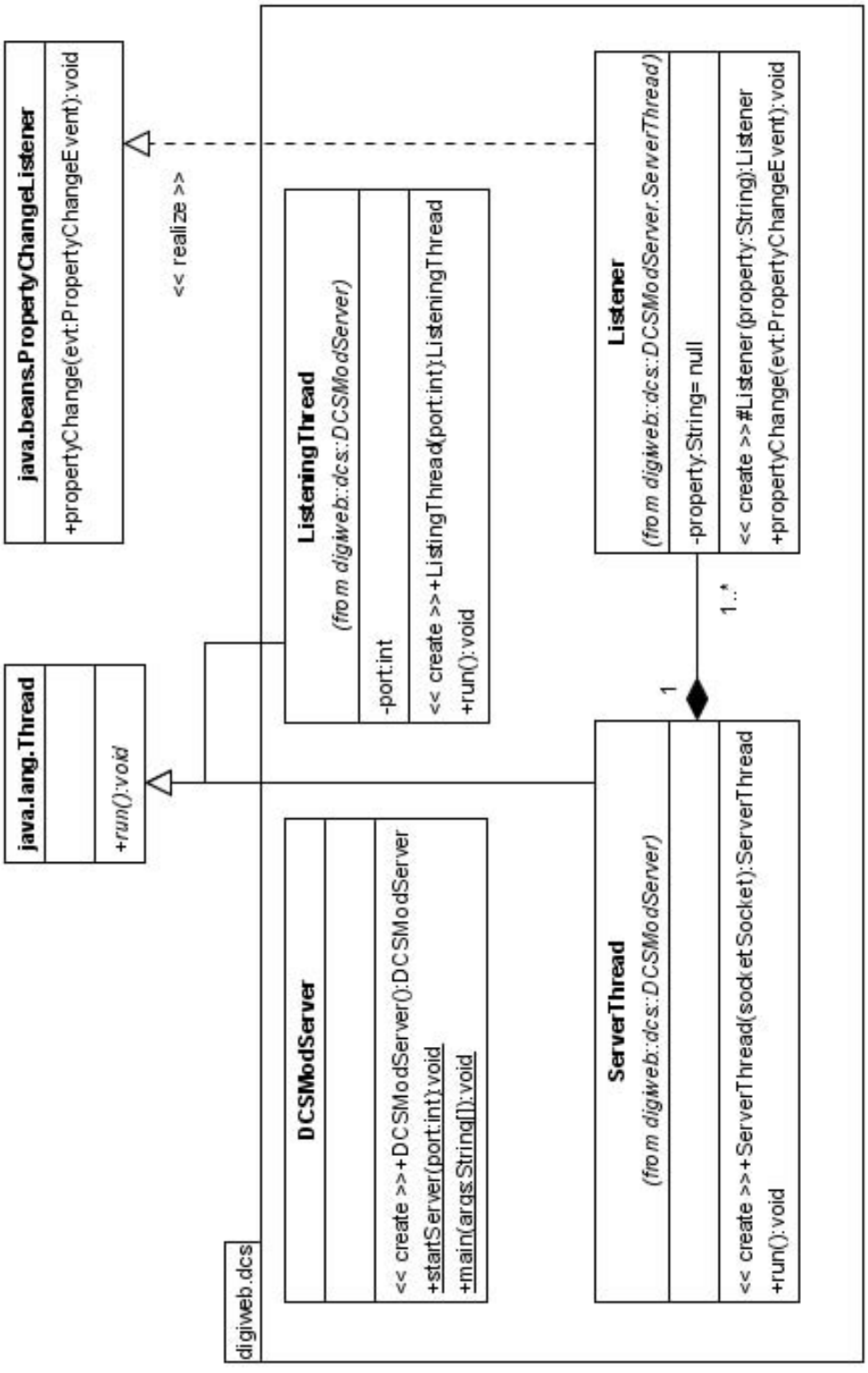


Abbildung 5-13: UML-Klassendiagramm zur ProperEduct-Erweiterung DCSModServer

5.3.3. Kognitive Modellierung

Das kognitive Modell ist in ACT-R/PM implementiert. Es nutzt die Erweiterungen Agimap und Timer.

Die Verbindung zwischen Simulationsumgebung und kognitivem Modell wird auf der Simulationsseite mit der Komponente `DCSModServer` und auf der Seite des kognitiven Modells mit Agimap realisiert. In Agimap werden zwei Buttons, ein Text-Mapper, fünf Text2Color-Mapper und ein Number2Pos-Mapper definiert (s. Abbildung 5-14).

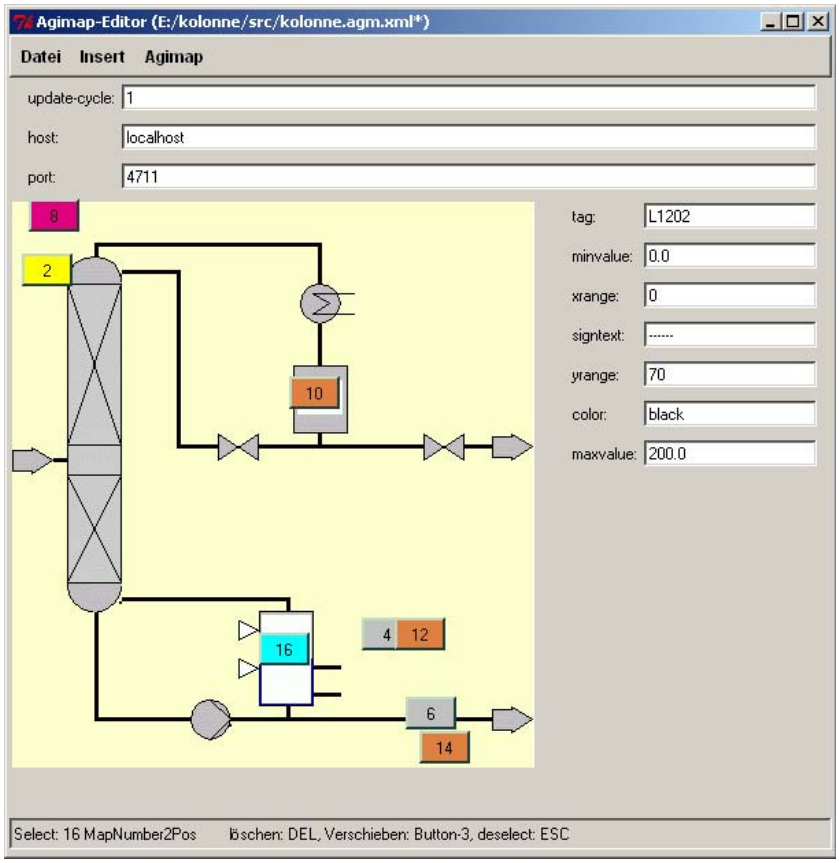


Abbildung 5-14: Screenshot des Agimap-Editors bei der Bearbeitung des Interaktionsmodells der Rektifikationskolonne

Abbildung 5-15 zeigt exemplarisch, wie diese Elemente eingesetzt werden. Der Tank A (s. Abbildung 5-12) hat einen Füllstand und zwei Indikatoren, die die Grenzen für die Abschaltautomatik der Heizung angeben. Der Füllstand wird auf eine Linie von „----“ abgebildet (Number2Pos-Mapper). Die Position dieser Linie auf dem Hintergrundbild (rechter Screenshot) ist analog zum Füllstand des Tanks. Die Indikatoren ändern ihre Farbe: Ist der Level zu niedrig, wird der untere Indikator rot. Ist der Füllstand zu hoch, wird der obere Indikator rot. Dieses Verhalten wird durch zwei Text2Color-Mapper nachgebildet.

Die Interaktionselemente Heizung und Ventil werden durch je einen Text2Color-Mapper (Anzeige des Zustandes an/aus, offen/geschlossen) und je ein Button Element zur Interaktion) abgebildet.

Die Arbeitsweise ist so, dass der Systemzustand, also die Werte *valve*, *heater*, *level* und *trend* in Slots eines Zieles vom Typ `systemstate` repräsentiert werden. Zum Aufbau dieser Repräsentation wird Agimap benutzt.

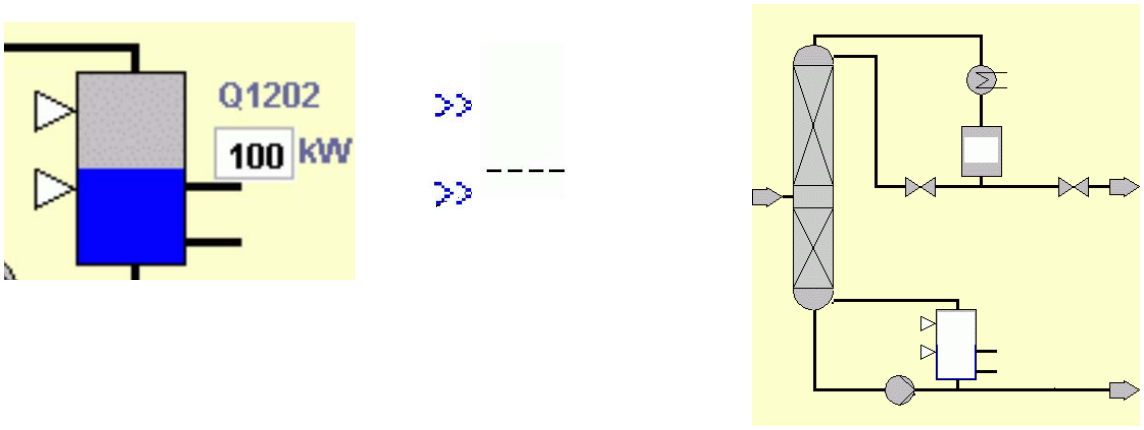


Abbildung 5-15: Verwendung der Agimap-Elemente. links: Detail der Originaloberfläche, Mitte: Nachbildung in Agimap, rechts: leeres Hintergrundbild, in das die Agimap-Elemente eingeblendet werden

Die Kontrolle des Prozesses läuft nach einem einfachen Schema ab. Abbildung 5-16 zeigt die Struktur der Aufgabe: Um die Anlage anzufahren, muss der Tank (A in Abbildung 5-12) geheizt werden. Dazu muss sich der Pegel zwischen den Markierungen befinden, sonst schaltet eine Sicherheitsautomatik das Heizelement ab. Entweder der Pegel ist noch nicht im erlaubten Bereich, sondern darüber oder darunter. Dann muss der korrekte Pegel erst durch Betätigung des Abflussventils (öffnen oder schließen), bzw. Warten erreicht werden. Wenn der Pegelstand schon im erlaubten Bereich ist, dann muss der Pegel natürlich gehalten werden. Dazu muss auch das Ventil betätigt und gewartet werden. Zusätzlich muss die Heizung aktiviert werden.

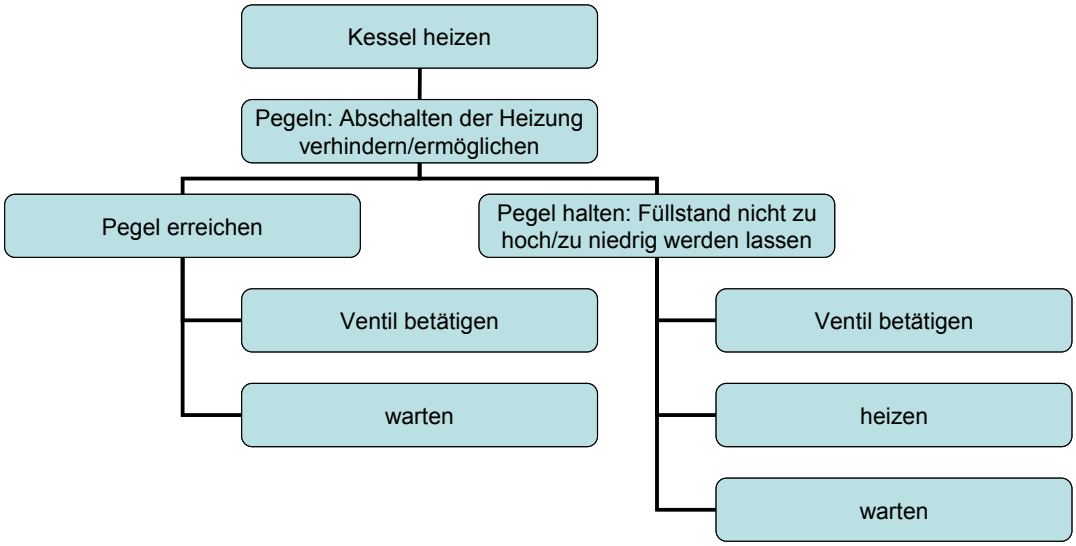


Abbildung 5-16: Zielhierarchie für die Kontrolle des Rektifikationskolonnenprozesses

Tabelle 5-4: Abbildung Zustandsraum auf Aktionen und Regeln

Transformation Zustandsraum (Regeln)						
	Zustand				Aktion	
Regel	level	trend	heater	valve	heater	level
unmöglich	+	+	on	open		
5	+	+	on	close		open
-	+	+	off	open	warten	
5	+	+	off	close		open
unmöglich	+	0	on	open		
5	+	0	on	close		open
-	+	0	off	open	warten	
5	+	0	off	close		open
unmöglich	+	-	on	open		
5	+	-	on	close		open
-	+	-	off	open	warten	
5	+	-	off	close		open
-	0	+	on	open		
2	0	+	on	close		open
1	0	+	off	open	on	
1	0	+	off	close	on	
-	0	0	on	open		
-	0	0	on	close		
1	0	0	off	open	on	
1	0	0	off	close	on	
3	0	-	on	open		close
-	0	-	on	close		
1	0	-	off	open	on	
1	0	-	off	close	on	
4	-	+	on	open		close

Transformation Zustandsraum (Regeln)						
	Zustand				Aktion	
Regel	level	trend	heater	valve	heater	level
unmöglich	-	+	on	close		
4	-	+	off	open		close
-	-	+	off	close	warten	
4	-	0	on	open		close
unmöglich	-	0	on	close		
4	-	0	off	open		close
-	-	0	off	close	warten	
4	-	-	on	open		close
unmöglich	-	-	on	close		
4	-	-	off	open		close
-	-	-	off	close	warten	

Diese Struktur kann man in einer einfachen Tabelle zusammenfassen. Tabelle 5-4 zeigt den kompletten Zustandsraum. Er wird aufgespannt durch die drei relevanten Systemgrößen *level* (Füllstand im Tank, Werte: +/zu hoch, 0/im erlaubten Bereich, -/zu niedrig), *heater* (Heizelement an oder aus) und *valve* (Abflussventil, geöffnet oder geschlossen). Als vierte Größe kommt der *trend* dazu. Er gibt an, ob der Füllstand steigt, konstant ist oder fällt. Diese Größe ist in der Oberfläche nicht direkt ablesbar. Er wird durch den Vergleich der aktuellen Füllhöhe mit einem aus dem Arbeitsgedächtnis abgerufenen früher aufgenommenen Wert ermittelt. Dazu werden Agimap-Mechanismen genutzt.

Diese vier Systemgrößen bestimmen den Zustand des Systems. Aus ihnen wird die passende Aktion abgeleitet. In Tabelle 5-4 sind alle denkbaren Kombinationen aufgeführt. Einige dieser Zustände sind vom System nicht erreichbar. Sie sind mit rot markiert. Wenn der Pegelstand zu hoch oder zu niedrig ist, kann die Heizung nicht an sein, denn die Automatik schaltet sie ab.

Zwei Aktionen sind möglich. Das Ventil kann geöffnet oder geschlossen werden, um den Füllstand zu regulieren. Die Heizung kann ein- und ausgeschaltet werden. Auch die Heizung hat einen Einfluss auf den Füllstand, da sie Flüssigkeit verdampft.

In einigen Fällen braucht keine Aktion initiiert werden. Beispielsweise kann der Füllstand von einem zu hohen Level kommend (level +) gerade fallen (trend -). Die Heizung ist ausgeschaltet, da der Level zu hoch ist (heater off). Das Ventil ist

geöffnet (valve open). Keine Aktion ist erforderlich, da der Level in den erwünschten Bereich fällt. Erst wenn dieser Bereich erreicht ist, kann die Heizung eingeschaltet werden.

Tabelle 5-5: Regeln zu Kontrolle von Pegel und Heizung

R1	level ok		&& heater off	**==>**		toggle heater
R2	level ok	&& valve close	&& trend up	**==>**	toggle valve	
R3	level ok	&& valve open	&& trend down	**==>**	toggle valve	
R4	level -	&& valve open		**==>**	toggle valve	
R5	level +	&& valve close		**==>**	toggle valve	

Dieser Zustandsraum wird auf fünf Regeln reduziert (s. Tabelle 5-5), die direkt als ACT-R/PM Produktionen ausgedrückt werden können. Die Regeln werden direkt aus der Tabelle abgeleitet. Sie stellen eine Normalisierung der Tabelle dar. Die Aktion, die an Ventil und Heizung möglich ist, ist eine Äderung des Zustands: toggle. D.h. wenn das Ventil bereits offen ist, kann es nicht geöffnet werden, sondern nur in den gegenteiligen Zustand „geschlossen“ *getoggelt* werden. Agimap wird dazu benutzt, den Systemzustand im *current goal* zu repräsentieren. Die ACT-R/PM-Produktionen, die diese fünf Regeln abbilden, testen das *current goal* auf entsprechend belegte *Slots* und initiieren ggf. das entsprechende Agimap-Kommando, um einen Eingriff in der Simulation vorzunehmen (s. Quelltext 5-4).

```
(P R1
=goal>
  isa systemstate
  level ok
  heater off
==>
+goal>
  isa Agimap-clickButton
  state retrieve
  target buttonHeater
)
```

Quelltext 5-4: ACT-R/PM Produktion für Regel R1

Das Anfahren der Rektifikationskolonne ist eine Multitaskingaufgabe. Die Gesamtaufgabe besteht nicht nur aus der Kontrolle der Heizung im unteren Bereich des Prozessschaubildes. Hier geht es darum, den Kessel zu heizen und dazu den Pegel des Tanks in einem erlaubten Bereich zu halten. Die übergeordnete Aufgabe bezieht sich auf die Diagnose einer Fehlersituation in der Anlage. Wenn nach einer Minute heizen von Tank A (s. Abbildung 5-12) noch keine Flüssigkeit in Tank C

vorliegt, liegt ein Fehler in der Anlage vor, und das Anfahren muss abgebrochen werden. Quelltext 5-5 zeigt den Versuch einer Formalisierung der übergeordneten Aufgabe in NGOMSL.

```
Method for goal: Anlage anfahren
            Step 1. Decide: If <<Flüssigkeit in C>>,
                then accomplish: Anfahren korrekt signalisieren
            Step 2. Decide: If <<zu lange keine Flüssigkeit in C>>,
                then accomplish: Fehlfunktion signalisieren
            Step 3. Accomplish goal: Kessel heizen
            Step 4. Goto 1

Method for goal: Fehlfunktion signalisieren
            Step 1. BB Cancel
            Step 2. Return with goal accomplished

Method for goal: Anfahren korrekt signalisieren
            Step 1. BB OK
            Step 2. Return with goal accomplished
```

Quelltext 5-5: NGOMSL Notation der übergeordneten Aufgabe

Der Versuch einer Formalisierung mit NGOMSL schlägt fehl, weil die Multitaskinganforderung nicht formalisiert werden kann. Schritt drei der ersten Methode beinhaltet die Erfüllung des Zieles den Kessel zu heizen. Diese Kontrollaufgabe ist ein andauernder Prozess, der mit den Regeln eins bis fünf (s. Tabelle 5-5) modelliert werden kann. Von Zeit zu Zeit muss aber zur übergeordneten Aufgabe gewechselt und geprüft werden, ob eine Fehlersituation in der Anlage vorliegt, wenn zu lange keine Flüssigkeit in C beobachtet werden kann. Mit GOMS kann immer nur ein linearer Ablauf modelliert werden.

In der Umsetzung des kognitiven Modells mit ACT-R/PM könnten die Situationen, in denen bei der Kontrolle der Heizaufgabe gewartet werden muss – also keine der Aktionen `toggle valve` und `toggle heater` initiiert werden – genutzt werden, um die übergeordnete Aufgabe zu bearbeiten. Dieser Fall führt aber je nach Konfiguration der Prozessdynamik dazu, dass die übergeordnete Aufgabe zu selten und zu künstlichen Zeitpunkten Aufmerksamkeit erhält.

Um Multitasking umzusetzen, muss zuerst geklärt werden, an welchen Punkten die Aufgabenabarbeitung unterbrochen werden könnte, um zu einer anderen Aufgabe zu springen. Für die Kontrolle der Rektifikationskolonne gibt es zwei nebenläufige Anforderungen: Die übergeordnete Aufgabe und die Kontrollaufgabe des Heizkessels. Es lagen keine empirischen Anhaltspunkte vor, die bei der Wahl der Unterbrechungspunkte hätten helfen können. Deshalb wurde ein *best guess* Ansatz gewählt und post hoc geprüft.

Im Fall der Kontrolle des Heizkessels gibt es nur eine flache Hierarchie von Produktionen. Die Vermutung zu den Unterbrechungspunkten ist deshalb, dass die

Aufgabe an den „Top-Punkten“, also beim jeweiligen Einstieg in die weitere Verarbeitung, unterbrechbar ist. Wenn hingegen den Unterzielen gefolgt wird, sollte die Abarbeitung, die sehr schnell beendet wird, als ununterbrechbares Programm angesehen werden und deren Ende abgewartet werden.

Die Verarbeitungssequenz im Einzelnen: Wenn im Ziel vom Typ `systemstate` bereits der aktuelle Zustand des Systems repräsentiert ist, wird eine der fünf Produktionen zur Kontrolle des Pegels und der Heizung (s. Tabelle 5-5) feuern. Wenn es nicht zum Warten kommt, weil keine der fünf Regeln anwendbar ist, wird von hier aus ein Unterziel zum Initiieren einer Aktion gebildet. Diese Anforderung wird in Agimap abgearbeitet. Danach wird der Systemzustand aktualisiert. Dazu werden drei Unterziele gebildet, die sich auf die drei direkt beobachtbaren Werte `heater`, `valve` und `level` beziehen. Der Trend wird bei der Inspektion des Füllstandes von den Agimap-Routinen automatisch erfasst.

Es ergibt sich die folgende Sequenz (Einrückungen: Unterzielbildung, „|“: Alternativen):

[Unterbrechungspunkt] systemstate liegt vor

Auswahl: R1 | R2 | R3 | R4 | R5 | warten

Agimap: toggle heater | toggle valve

[Unterbrechungspunkt] systemstate aktualisieren

Agimap: heater | valve | level

An den Unterbrechungspunkten wird zur übergeordneten Aufgabe gesprungen. Hier gibt es genau eine Sequenz:

Timer: vergangene Zeitdauer prüfen

Agimap: Bereich C auf Flüssigkeit prüfen | Ende

Diese Sequenz ist nur kurz. Zu Beginn des Simulationsszenarios gibt es jeweils nur einen kurzen Abruf der Zeitdauer mit Timer. Später wird diese Zeitbedingung erfüllt und es wird mit Agimap geprüft, ob bereits Flüssigkeit im Bereich C (s. Abbildung 5-12) vorhanden ist. Wenn die Zeitdauer zu groß ist oder Flüssigkeit in C vorhanden ist, wird das kognitive Modell mit entsprechender Ausgabe abgebrochen, denn die übergeordnete Diagnoseaufgabe ist erfüllt.

Zeitbedingungen können mittels Timer ausgedrückt werden. Im Fall der Rektifikationskolonne wird der Modus prospektiver Zeitdauerschätzung verwendet. Als Referenzpunkt wird der Beginn des Heizens genommen. Falls das Heizen im Laufe des Modells unterbrochen wird, läuft die beobachtete Zeitdauer trotzdem weiter.

Um Zeitbedingungen umzusetzen, muss mittels Timer ein Referenzzeitpunkt gesetzt werden. Später kann eine Anfrage zur subjektiv vergangenen Zeit seit diesem Referenzzeitpunkt an den Timer-Buffer gestellt werden. Das Ergebnis kann im nächsten Zyklus *retrieved* werden und steht danach als *Slot* des aktuellen Ziels

zur Verfügung. Hier kann mit numerischen Operationen das Ergebnis der Timer-Abfrage mit einem Zeitdauerreferenzwert verglichen werden.

Die Validierung der Modellannahmen ergibt sich aus dem Vergleich der Interaktion von Modell und Bediener mit der Aufgabenumgebung. Die beobachtbare Interaktion ist dabei anhand der Kontrollaufgabe (Regelung von Pegel und Temperatur) vorgegeben.

5.3.4. Ergebnisse

Mit dieser Fallstudie wurde die Verwendung von NGOMSL zur Aufgaben-modellierung und von ACT-R als kognitive Architektur bei einer Multitasking-Aufgabe, bei der die Zeitwahrnehmung eine besondere Rolle spielt, getestet.

Es hat sich gezeigt, dass NGOMSL für die Aufgabenmodellierung nicht geeignet ist, denn Aufgabenwechsel können nicht adäquat modelliert werden. Deshalb wurde mit Multitasking GOMS eine Erweiterung zum Umgang mit Multitasking entworfen, die jedoch in dieser Fallstudie nicht zum Einsatz gekommen ist. Stattdessen wurde ein Produktionssatz aus Kontrollregeln durch Normalisierung aus einer Analyse des Zustandsraumes der Systemgrößen abgeleitet. Für die Aufgabenmodellierung wurde also in diesem Fall auch ACT-R verwendet.

ACT-R kann prinzipiell mit der neu entwickelten Erweiterung Agimap zur Bedienermodellierung bei dieser Aufgabenumgebung verwendet werden. Der Einsatz von Agimap bei DURESS und dieser Fallstudie belegt die Generizität von Agimap.

Für die Realisierung der Zeitwahrnehmung musste die ACT-R Erweiterung Timer neu entwickelt werden. Die Timer-Erweiterung ist in Form eines neuen ACT-R Buffers realisiert. Dadurch ist eine hohe Wiederverwendbarkeit gegeben, denn im Gegensatz zu Frameworks, die möglicherweise die Verwendung von anderen Frameworks behindern können, gibt es bei Buffern keine Interaktion zwischen explizit repräsentierten Modellteilen. Die Erweiterung ist somit ein Bestandteil der kognitiven Architektur und damit generisch anwendbar.

Das kognitive Modell für die Bedienung der Rektifikationskolonne ist in der Lage, mit einem einfachen Schema Multitasking zwischen einer übergeordneten zeitbehafteten Aufgabe und der eigentlichen Kontrollaufgabe durchzuführen. Dazu mussten jedoch als Teil der Modellbildung Annahmen über Unterbrechbarkeit und Unterbrechungspunkte als getroffen werden.

5.4. Entwicklung Multitasking GOMS

Die Entwicklung von Multitasking GOMS wurde innerhalb des größeren Drittmittelprojekts „Vernetztes Fahren“ durchgeführt. Im Rahmen dieser Arbeit wurde aber nur die Konzeption und Spezifikation von Multitasking GOMS bearbeitet, so dass die Inhalte von „Vernetztes Fahren“ als übergeordnetem Projekt hier nicht berichtet werden. Multitasking GOMS wird im Ergebnis-Teil dieser Arbeit vorgestellt.

5.5. Zusammenfassung

In diesem Kapitel wurden die drei durchgeführten Fallstudien MoFL, DURESS und Rektifikationskolonne beschrieben. In diesen Fallstudien wurden ablauffähige Bedienermodelle zur Simulation von Human Factors relevanten Aspekten der Interaktion in dynamischen Mensch-Maschine-Systemen erstellt. Dafür wurden die existierenden Systeme ACT-R, KLM und (N)GOMSL verwendet.

Die Fallstudien wurden durchgeführt, um festzustellen inwieweit die existierenden Methoden, Systeme und Werkzeuge zur Bedienermodellierung in dynamischen Mensch-Maschine-Systemen verwendbar sind. Die dabei identifizierten Probleme wurden durch neu entwickelte Erweiterungen und Werkzeuge gelöst. Die Nützlichkeit der Neuentwicklungen wurde im Einsatz nachgewiesen.

Die Fallstudie MoFL hat am Beispiel der Flugsicherungsaufgabe gezeigt, dass ACT-R für den Anwendungszweck im Prinzip geeignet ist und der Ansatz der Modellierung mit einer kognitiven Architektur, deren Konzeption mit dem Ziel einer *unified theory of cognition* vorangetrieben wird, trägt. Die Werkzeugunterstützung ist jedoch mangelhaft. Um den Modellbildungs- und Entwicklungsprozess zu unterstützen, wurde deshalb das neue Werkzeug ACT-R Shell entwickelt, das sich in der Fallstudie als nützlich erwiesen hat. Aufgrund der dynamischen externen Aufgabenumgebung war es zudem erforderlich, Erweiterungen für das Multitasking und zur Kopplung mit einer externen Aufgabenumgebung (ACT-COM) zu entwickeln. Die Erweiterung zum Multitasking ist nicht über die spezifische Charakteristik der Flugsicherungsaufgabe hinaus verallgemeinerbar. ACT-COM hingegen hat einen generischen Ansatz und kann in anderen Modellen weiterverwendet werden.

Die neueren Versionen von ACT-R enthalten jedoch das PM-Framework, mit dem auch die Modellierung auf der Wahrnehmungsebene möglich ist. Um diese Fähigkeit bei der Kopplung mit einer externen Aufgabenumgebung zu nutzen, wurde das Werkzeug und die ACT-R Erweiterung Agimap entwickelt, die in der DURESS-Fallstudie und beim Modell zur Rektifikationskolonne im Bereich der Bedienung verfahrenstechnischer Anlagen erfolgreich getestet wurden.

Die Analyseebene der Aufgabenmodellierung wurde bei DURESS und der Rektifikationskolonne zusätzlich zur kognitiven Ebene berücksichtigt. In DURESS wurde KLM zur Repräsentation der *Standard Operation Procedures* verwendet. Die KLM-artige Repräsentation musste jedoch durch die Neuentwicklung Planex in ACT-R

integriert werden. Diese Erweiterung in Form eines Frameworks ist generisch und in anderen ACT-R Modellen wiederverwendbar.

In der Fallstudie Rektifikationskolonne wurden zusätzliche Anforderungen an die Zeitwahrnehmung gestellt, die mit ACT-R nicht erfüllt werden können. Die Entwicklung der Erweiterung Timer zur psychologisch plausiblen Zeitwahrnehmung in Form eines ACT-R Buffers ermöglicht es, die in der Fallstudie gestellten Anforderungen an die Multitasking-Fähigkeiten mit ACT-R zu erfüllen.

In dieser Fallstudie wurde auch NGOMSL verwendet, um die Aufgabenbearbeitung zu modellieren. Der Aufgabenwechsel kann jedoch so nicht modelliert werden. Die Folgerung aus der Fallstudie Rektifikationskolonne für NGOSL ist daher die Erkenntnis, dass NGOMSL um Multitasking-Fähigkeiten erweitert werden muss. Multitasking GOMS ist eine entsprechende Erweiterung von GOMSL. Dieses neue System zur Aufgabenmodellierung wurde im Rahmen dieser Arbeit nicht erprobt. Es wurde aber im Drittmittelprojekt „Vernetztes Fahren“ erfolgreich verwendet.

Die Fallstudien haben gezeigt, dass ACT-R Modelle eine hohe Komplexität haben, die nur schwer zu handhaben ist. Um die Entwicklungsarbeit effizienter zu gestalten, potenzielle Fehlerquellen zu vermeiden und die Kommunikation über Modelle zu verbessern, wurde ein Werkzeug zur Visualisierung von ACT-R Modellen entworfen und in einer Diplomarbeit als Plugin für die Entwicklungsumgebung Eclipse implementiert. Da das Programm nur erlaubt, existierende Modelle zu visualisieren und die grafische Modellierung damit noch nicht möglich ist, wurde es nicht in Fallstudien eingesetzt, steht jedoch als Ergebnis aus den Erfahrungen der Fallstudien zur Verfügung.

Die neu entwickelten Werkzeuge, Frameworks und Erweiterungen von ACT-R sowie die GOMSL-Erweiterung Multitasking GOMS sind die Hauptergebnisse dieser Arbeit. Im nächsten Kapitel werden diese Ergebnisse in ihrer Konzeption, Abgrenzung zu existierenden Ansätzen, Entwurf, ggf. Parametrierung und Implementierung dargestellt. Die Bewertung der einzelnen Erweiterungen in Bezug auf Effektivität und Effizienz der Bedienermodellierung erfolgt aus einer Software Engineering Perspektive. Am Ende des folgenden Kapitels werden in Abschnitt 6.9 diese Einzelergebnisse in einer Gesamtbewertung zusammengefasst.

6. Darstellung der Ergebnisse

In den Fallstudien wurden Hemmnisse identifiziert, die die anwendungsorientierte Entwicklung von Bedienermodellen erschweren. Sie betreffen zum einen die Effizienz der Entwicklung, im Sinne einer fehlenden Unterstützung von Aufgaben im Entwicklungsprozess von Bedienermodellen, zum anderen die Fähigkeiten der Plattformen und Theorien in Bezug auf die Umsetzung von Anforderungen der Aufgabenumgebungen in dynamischen Mensch-Maschine-Systemen.

Für den Bereich der Entwicklungseffizienz kann die Bedienermodellierung mit der Programmierung verglichen werden. Teilweise wird versucht, bewährte Werkzeuge aus der Softwareentwicklung für die Modellierung einzusetzen. Beispiele sind Soar- und ACT-R-Modes für den Emacs-Editor. Es zeigt sich aber, dass nur wenige existierende Werkzeuge für die Bedienermodellierung anwendbar oder zumindest anpassbar sind. Deshalb ist es erforderlich neue Werkzeuge zu entwickeln und im Einsatz zu testen. Die zu unterstützenden Einsatzbereiche ergeben sich aus dem Entwicklungsprozess (s. 2.6). Aus den Erfahrungen der Softwareentwicklung scheinen insbesondere die Bereiche Modellierung (Editoren), Simulation (Ablaufumgebung), Auswertung (Logging), Debugging und Wartung (Visualisierung von fremden Modellen) relevant.

Der Einsatz für die Bedienermodellierung bei Aufgabenumgebungen von dynamischen Mensch-Maschine-Systemen muss hingegen durch das Bereitstellen von Frameworks und Erweiterungen der kognitiven Architekturen bzw. Theorien angegangen werden. Frameworks unterscheiden sich von Erweiterungen dadurch, dass sie eine Reihe von existierenden Theorie- bzw. Architekturelementen in einer bestimmten Zusammenstellung kombinieren. Diese Zusammenstellung soll dann in unterschiedlichen Modellierungsprojekten wieder verwendet werden. Die Zusammenstellung der Elemente ermöglicht die Anpassung an die konkrete Aufgabe durch Erweiterungspunkte (Hot Spots, s. Pree 1997). Eine Erweiterung fügt hingegen neue Theorie- bzw. Architekturelemente hinzu. Auch Erweiterungen sollen wiederverwendbar sein.

Die Dynamik der hier interessierenden Aufgabenumgebungen erfordert zusätzlich zum Leistungsumfang der bestehenden Umgebungen zur Bedienermodellierung die Fähigkeit zum Multitasking. Außerdem muss durch neu zu entwickelnde Erweiterungen oder Frameworks die Analyse von Bedienfehlern, die mit einem falschen Zeitempfinden zu tun haben, möglich sein. Die Kopplung von Modell und dynamischer Aufgabenumgebung ist erforderlich, um die Interaktion von Bedienermodell und technischem System adäquat abzubilden. Zusätzlich zum Problemkreis der Dynamik ist die Integration deklarativ repräsentierter SOPs bzw. von anderweitig vorliegenden Aufgabenmodellen eine wichtige Anforderung zur praktischen Anwendung.

Die praktische Anwendung muss zusätzlich durch eine neu zu erstellende Guideline unterstützt werden. Sie muss eine Hilfestellung zur Auswahl der Analyseebene, der Modellierungsformalismen und der prinzipiellen Struktur der zu entwickelnden Bedienermodelle enthalten. Insgesamt ist die Trennung nach den Analyseebenen Kognition und Aufgabe erforderlich, wobei die Compilationsansätze, die eine Integration dieser beiden Analyseebenen beinhalten, berücksichtigt werden müssen. Tabelle 6-1 zeigt eine Zuordnung der neuen Entwicklungen zu den hier beschriebenen Bereichen.

Tabelle 6-1: Entwicklungsbereiche Bedienermodellierung

	Guideline	**Modellierungs-werkzeug**	**Framework**	**Erweiterung**
Analyseebene Kognition	Modellierungs-rationale ACT-R/PM KLM NGOMSL	Visualisierung, Agimap, ACT-R Shell	ACT-Com, Agimap	Timer
Analyseebene Aufgabe		Agimap, MT-GOMS	Planex	MT-GOMS

In den folgenden Abschnitten werden diese Entwicklungen vorgestellt.

6.1. Guideline zur Bedienermodellierung

Im folgenden Abschnitt wird die Auswahl der Modellierungsumgebungen und *best practice* der Bedienermodellierung beschrieben. Auf der Analyseebene der Kognition sind Ansätze mit kognitiven Architekturen denen für die ad hoc Modellierung vorzuziehen, weil sie durch ihre erprobte und verallgemeinernde Struktur die Wiederverwendung von Modellierungswissen und in den Architekturen vorhandenen Elementen ermöglichen.

ACT-R/PM ist anderen kognitiven Architekturen wegen des großen Umfangs von modellierbaren Phänomenen aus dem Bereich *Human Factors* vorzuziehen. Alle Bereiche von Wahrnehmung über Gedächtnis und Lernen bis Problemlösen sind abgedeckt.

In ACT-R/PM können komplexe Funktionen durch den Aufruf von LISP-Funktionen im Wirtssystem aus Produktionen heraus implementiert werden. LISP-Funktionen sollten so wenig wie möglich benutzt werden, da sie die kognitive Verarbeitung nicht adäquat abbilden, denn der Verarbeitungsaufwand ist nicht analog zur Implementierung mit Produktionen und außerdem stehen den

subsymbolischen Verarbeitungsprozessen die Informationen über die Anwendung nicht zur Verfügung. In einigen Fällen ist es jedoch erforderlich LISP-Funktionen während der Verarbeitung aufzurufen. Dann sollten sie jedoch nicht aus Produktionen heraus angewendet werden, sondern durch Erweiterung von Hot Spots der Architektur (z.B. Hooks).

Bei der Modellierung mit ACT-R/PM hat sich das zu implementierende Kontrollschema als kritisch erwiesen. Auf die Benutzung eines Goalstacks sollte verzichtet werden. Stattdessen muss das Zielhandling nach einem selbst gewählten Schema implementiert werden. Als sehr nützlich hat sich erwiesen, nachfolgende Schritte in einer im Voraus geplanten Kette von Produktionen durch Setzen und Prüfen eines Slots `step` bzw. `state` zu implementieren. Für das Management von mehreren Zielen (Unterziele, parallele Ziele) hingegen sollte das vorherige Ziel in einem Slot `goal` gespeichert werden, um von einem Unterziel aus zurückspringen zu können oder Verwaltungsinformationen für Multitasking zu speichern.

Bei der Bedienermodellierung mit ACT-R/PM ergeben sich Handhabungsprobleme bei komplexeren Modellen. Da ACT-R/PM keine Modularisierungskonzepte vorsieht, sollte durch die Einführung von Konventionen eine erhöhte Übersichtlichkeit im Modell geschaffen werden. Dazu hat sich eine Konvention zur Benennung von Produktionen bewährt. Produktionsnamen bekommen einen Präfix mit dem Modulnamen, dann eine Bezeichnung der mit ihr implementierten Funktion (oft der Name oder Typ des Ziels), dann den Verarbeitungsschritt (oft der Name des Slots `step`) und schließlich eine Nummer, um mehrere Produktionen zum selben Schritt voneinander abzugrenzen.

Bedienermodellierung enthält immer einen Anteil Aufgabenmodellierung, denn die Ergebnisse einer Aufgabenanalyse können entweder mit deutlich geringerem Aufwand als bei der kognitiven Modellierung analysiert werden oder bilden zumindest die Grundlage zum Aufbau der Produktionen. Aus den bekannten Methoden zur Aufgabenmodellierung sollte KLM wegen der konzeptionellen Einfachheit und NGOMSL wegen der Analogie zu ACT-R/PM bei der Zielverarbeitung gewählt werden.

Wo möglich sollte der schnell erlernbare und leicht einsetzbare Formalismus von KLM verwendet werden. KLM ist jedoch auf sehr einfache Aufgaben beschränkt. Mit NGOMSL können komplexere Aufgaben modelliert werden als mit den anderen Ansätzen zu Aufgabenmodellierung. Auch Aufgabennetzwerke und *Human Reliability Analysis* (HRA) sind weniger nützlich, da mit ihnen wissensbasierte Aufgaben nur schlecht modelliert werden können. Im Fall von HRA ist zusätzlich die Parallelverwendung von Modellen für Designunterstützung und als Unterstützungssystem nicht gegeben.

Bei der Aufgabenmodellierung mit NGOMSL sollten folgende Kriterien für einen guten Schnitt von Methoden berücksichtigt werden:

- Methoden sollten mehrfach verwendet werden.
- Methoden sollten überschneidungsarm sein.
- Methoden sollten kurz sein; lange Methoden sollten durch Einführung eines Zwischenziels aufgespalten werden.
- Die Verarbeitung sollte nicht mehr als sieben Chunks im Arbeitsgedächtnis bedingen, sonst müssen Zwischenschritte eingeführt werden.
- Methoden sollten so gestaltet werden, dass das jeweilige Ziel als letzter Schritt erreicht wird.

6.2. ACT-COM

ACT-COM ist eine neue Erweiterung von ACT-R, die als Middleware die Kopplung und Interaktion zwischen Aufgabensimulation und kognitivem Modell realisiert. Zielstellung bei der Entwicklung war, ein Kommunikationsmodul bereitzustellen, das die bidirektionale Kopplung über ein Netzwerk erlaubt und ein auswechselbares Interaktionsprotokoll beinhaltet, mit dem wesentliche Aspekte der menschlichen Informationsaufnahme in *Command and Control* Systemen vereinfachend nachgebildet werden können. Das kognitive Modell darf dabei nicht perfekte Information (korrekt und verzugsfrei) geliefert bekommen, was technisch ohne Probleme realisierbar ist, sondern es sollen Beschränkungen bei der Informationsaufnahme abgebildet werden können. Jedoch sollen dabei nicht alle ergonomischen Aspekte der Informationsaufnahme in unterschiedlich gestalteten GUI-Varianten erfasst werden (z.B. unterschiedliche Farb- oder Anordnungsvarianten).

6.2.1. Einsatz in Fallstudien

ACT-COM wurde im kognitiven Modell MoFL (s. 5.1) eingesetzt. Daher liegen Erfahrungen zur Kopplung eines kognitiven Modells mit mehreren unterschiedlichen Aufgabensimulationen vor (EnCoRe-PLUS, simco und einer perl-Luftraumsimulation, Leuchter & Jürgensohn 2000). Die einzige Informationsquelle, die dem Fluglotsenmodell für die Informationsaufnahme zur Verfügung steht, ist das simulierte Radarbild der unterschiedlichen Aufgabensimulationen. Zusätzlich wird die Sprechfunkinteraktion von Fluglotsen und Piloten über diese Schnittstelle simuliert. Das verwendete Protokoll ist in Anhang D spezifiziert.

6.2.2. Entwurf

ACT-COM ist eine Softwareschicht, die externe Aufgabenumgebung und kognitives Modell miteinander koppelt. Die Verbindung wird über TCP/IP-Sockets realisiert. Daher können Aufgabenumgebung und Simulation des kognitiven Modells auf unterschiedlichen Rechnern laufen, die über ein Netzwerk verbunden sind (s. Abbildung 6-1).

Der Kopplung liegt ein Protokoll zugrunde, das die Kommunikation zwischen Modell und Aufgabenumgebung beschreibt. ACT-COM besteht aus einem Parser, der das Kommunikationsprotokoll verarbeitet. Der Parser ist generisch, denn er kann zur Verarbeitung unterschiedlicher Protokolle parametriert werden. Die parsbaren Protokolle werden durch eine Chomsky Typ-2 Grammatik (kontextfrei) beschrieben.

Das Protokoll ist konzeptuell aus einer Reihe von Sprechakten aufgebaut. Das Protokoll legt Dialogsequenzen von Sprechakten fest. Jedem Sprechakt ist eine Call-Back-Funktion zugeordnet, die in CommonLISP implementiert werden muss.

Die Kommunikationsergebnisse werden über Chunks im Arbeitsgedächtnis repräsentiert, so dass mit ACT-R-eigenen Mechanismen darauf zugegriffen werden

kann. Dazu werden in den entsprechenden Call-Back-Funktionen Chunks mit ACT-R-Makros erzeugt - Fall (1) in Abbildung 6-1. Die Kommunikationsverbindung ist jedoch bidirektional. Fall (2) in Abbildung 6-1 zeigt die Kommunikation in die andere Richtung: Das ACT-R-Modell ruft über den Bedingungsteil einer Produktion eine entsprechende Funktion in ACT-COM auf, die einen Sprechakt in Form einer protokollkonformen Zeichenkette ausführt und über das Netzwerk zur externen Aufgabenumgebung sendet.

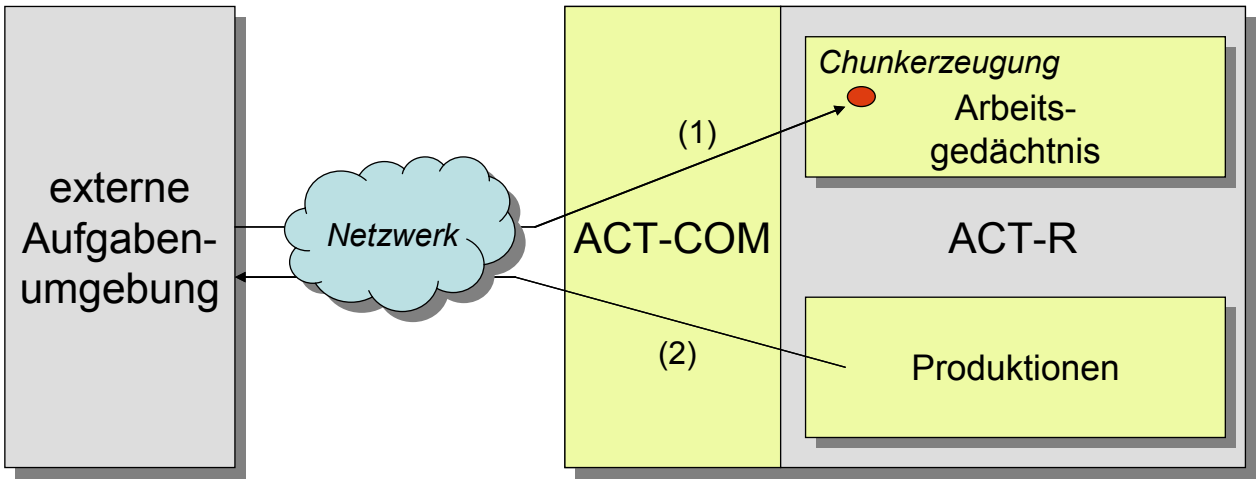


Abbildung 6-1: Schematischer Aufbau der Kommunikation mit ACT-COM, (1) Meldungen erzeugen Chunks, (2) Anfragen und Aktionen werden mit Produktionen ausgelöst

In beiden Fällen muss der Sprechakt von ACT-COM protokolliert werden, um die Korrektheit des weiteren Dialogablaufes zu prüfen, der durch das aktive Protokoll spezifiziert wird.

Es gibt zwei unterschiedliche Kommunikationsmodi, die mit diesem Schema realisiert werden können:

- *Asynchrone Kommunikation:* Die Aufgabenumgebung meldet dem kognitiven Modell Ereignisse. Repräsentationen der gemeldeten Ereignisse werden von den getriggerten Call-Back-Funktionen automatisch als neue Chunks im Arbeitsgedächtnis gespeichert.
- *Synchrone Kommunikation:* Das kognitive Modell fragt aktiv spezifische Informationen in der externen Aufgabenumgebung ab. Die Aufgabenumgebung antwortet auf die Anfrage. Die empfangenen Daten werden in das Arbeitsgedächtnis durch die passenden Call-Back-Funktionen integriert.

Asynchrone Kommunikationsereignisse können zu jedem Zeitpunkt auftreten und z.B. einen Dialog der synchronen Kommunikation unterbrechen. Das Kommunikationsprotokoll muss entsprechend gestaltet sein.

6.2.3. Berücksichtigung von Timing

Da der Anwendungsbereich des Parsers die Abbildung von Interaktion bei dynamischen Mensch-Maschine-Systemen ist, berücksichtigt er neben dem Protokoll auch das gewünschte Timing der Kommunikation. Es gibt dafür zwei Parameter, die die Verarbeitung jedes Dialogblocks (Sequenz von Sprechakten) steuern.

Der erste Parameter gibt die für die Abarbeitung des Dialogblocks zur Verfügung stehende Zeit an. Der Wert dieses Parameters ist entweder eine natürliche Zahl oder das Symbol „*“. Wenn der Wert eine Zahl größer als 0 ist, dann wird er als die für die Ausführung und Abarbeitung des Dialogblocks zur Verfügung stehende Zeit in Sekunden interpretiert. Zu den Werten „*“ und 0 s.u.

Der zweite Parameter gibt an, ob der Empfang von Daten blockierend oder nicht blockierend sein soll. Es gibt zwei Ausprägungen „+“ und „-“.

Es ergeben sich vier sinnvolle Kombinationen von Werten der beiden Parameter:

- (n +): Wenn die Zeit n Sekunden zum kompletten Empfang und Abarbeitung des Dialogblocks überschritten ist, wird die weitere Verarbeitung dieses Blocks terminiert. Ist die Zeit noch nicht verstrichen, blockiert die Empfangsroutine – alle anderen Verarbeitungsanforderungen werden gestoppt, bis der Block abgearbeitet ist, längstens aber bis n Sekunden verstrichen sind.
- (n -): Wenn die Zeit n Sekunden zum kompletten Empfang und Abarbeitung des Dialogblocks überschritten ist, wird die weitere Verarbeitung dieses Blocks terminiert. Ist die Zeit noch nicht verstrichen, blockiert die Empfangsroutine jedoch nicht, bis der Block abgearbeitet ist.
- (* -): Nicht-blockierendes Empfangen des Dialogblocks ohne Zeitbeschränkung
- (0 -): Test, ob neue Dialogelemente eingetroffen sind, wenn ja, weiter wie bei (* -), wenn nein, wird die Abarbeitung dieses Dialogelements abgebrochen.

Für die Zeittests wird die reale Zeit innerhalb des ACT-R Wirtssystems CommonLISP verwendet, nicht die wahrgenommene Zeit (s. nächster Abschnitt 6.3).

6.2.4. Abgrenzung

Die Bereitstellung wahrgenommener Informationen als Chunks wird in mehreren Systemen zur kognitiven Modellierung umgesetzt: In den Wahrnehmungspufffern von ACT-R/PM (Byrne & Anderson 1998) und über I/O-Link Objekte im Arbeitsgedächtnis bei Soar (Lonsdale & Ritter 2000).

Die Spezifikation der Kommunikation über ein formales Protokoll, die Bereitstellung von Call-Back-Funktionen zur Interpretation und Chunk-basierten Integration der Nachrichten und die Berücksichtigung von Timing wird in keinem anderen System unterstützt. Dieser Ansatz macht es leicht, existierende Aufgabenumgebungen mit bereits vorhandenen Kommunikationsschnittstellen anzubinden.

6.2.5. Implementierung

Die Kopplung zwischen kognitivem Modell und Aufgabensimulation geschieht über TCP/IP-Sockets. Da die Socketkommunikation kein Element in Standard CommonLISP, dem Wirtssystem von ACT-R, ist, muss die Socketkommunikation auf anderem Weg realisiert werden. Anhang C beschreibt die softwaretechnische Realisierung der Socketkommunikation unter Standard CommonLISP.

Der Parser für das Kommunikationsprotokoll ist generisch. Mit ihm können beliebige kontextfreie Sprachen verarbeitet werden. Die Sprache wird anhand der möglichen Sequenzen der Sprechakte in einer Chomsky Typ-2 Grammatik repräsentiert (Beispiel s. Quelltext 6-1). Konzeptuell wird dadurch ein Zustandsübergangsgraph beschrieben (s. Beispiel Anhang D). Der syntaktische Aufbau der Sprechakte selber wird davon getrennt in einer regulären Grammatik (Typ-3, Beispiel s. Quelltext 6-2) spezifiziert. Durch die Trennung dieser beiden Anteile kann die Sprache effizienter beschrieben werden. Außerdem ist es leichter möglich, Protokolländerungen vorzunehmen.

```
(defconstant *protocol-flow*
  `(( register (,*mofl-register-t* ,*mofl-register-b*)
                   (s -> register)
                   (s -> tell s)
                   (s -> tell-2 s)
                   (s -> goodby s)
                   (s -> goodby-2 s)
                   (s -> empty s))
    …
   ( check (,*mofl-check-t* ,*mofl-check-b*)
               (s -> tell)
               (s -> tell-2)
               (s -> goodby)
               (s -> goodby-2 s)
               (s -> empty))))
```

Quelltext 6-1: Ausschnitt aus der Protokollspezifikation von MoFL in der Notation von ACT-COM (`*mofl-register-t*` etc. sind Parameter, die das Timing der Protokolleinheiten beschreiben)

Jedem Sprechakt wird mit einer Tabelle eine CommonLISP-Funktion zugeordnet, die ausgeführt wird, wenn eine entsprechende syntaktische Einheit empfangen wird. Die Signatur der Call-Back-Funktionen beinhaltet einen Parameter, der den empfangenen Sprechakt enthält. Ergebnis der Funktion ist ein Wahrheitswert, der anzeigt, ob der Sprechakt syntaktisch und semantisch korrekt war. In einer solchen Call-Back-Funktion werden dann ACT-R-spezifische LISP-Makros ausgeführt, die das Arbeitsgedächtnis entsprechend ändern, indem Chunks neu erzeugt, gelöscht oder deren Attribute geändert werden.

Um auch die Verarbeitung asynchroner Kommunikation zu ermöglichen, wird zu definierten Zeitpunkten von dem generischen Kommunikationsmodul an den Eingabekanälen überprüft, ob eine neue vollständige syntaktische Einheit empfangen wurde und ggf. die zugeordnete Call-Back-Funktion ausgeführt. Das geschieht außerhalb der eigentlichen ACT-R Kontrolle, indem der ACT-R-spezifische `Cycle-Hook` benutzt wird, dessen Wert – als Funktion interpretiert – regelmäßig zwischen der Anwendung zweier Produktionen ausgeführt wird.

```
(defconstant *protocol-ident*
  '(    (register "^register$")
        (reply-1  "^reply-1\(.*(,.*,.*)*\)$")
        (reply-4  "^reply-4\(.*,ok\)$")
        (tell     "^tell\(.*,.*,.*,.*,.*\)$")
        (tell-2   "^tell-2\(.*\)$")
        (goodby   "^goodby\(.*\)$")
        (goodby-2 "^goodby-2\(.*\)$")
        (empty    "^reply\(\)$")    ))
```

Quelltext 6-2: Spezifikation der Syntax der Sprechakte von MoFL in der Notation von ACT-COM

6.2.6. Bewertung

ACT-COM wurde für die Anbindung des Fluglotsenmodells MoFL an drei unterschiedliche externe Aufgabenumgebungen verwendet. Die Kommunikationsprotokolle waren strukturell ähnlich, im Detail aber unterschiedlich. Die Anwendbarkeit und Anpassbarkeit des Protokollmechanismus' von ACT-COM wird anhand dieser Fallstudien bewertet.

ACT-COM bietet die Möglichkeit mit geringem Aufwand existierende externe Aufgabenumgebungen an ACT-R anzuschließen, so dass Informationen aus der Aufgabenumgebung aufgenommen werden können und Eingriffe vorgenommen werden können. Die Kapselung über ein Protokoll und Call-Back-Funktionen erlaubt eine hohe programmatische Flexibilität.

Die Definition und Repräsentation des Protokolls erfordert jedoch Erfahrung und ist aufgrund der unübersichtlichen CommonLISP-Syntax fehleranfällig. Für jeden Sprechakt muss eine eigene Call-Back-Routine programmiert werden.

Der Protokollmechanismus mit kontextfreien Sprachen ist für die in den Fallstudien behandelten externen Aufgabenumgebungen übermächtig – ein einfacherer Mechanismus wie bei Agimap (s. 6.6) ist ausreichend.

Die Fähigkeit das Timing der Protokollabarbeitung zu berücksichtigen war in der Entwicklung zum Umgang mit Verklemmungen hilfreich. Für anwendungsspezifische Zwecke wurde diese Fähigkeit in den Fallstudien nicht benötigt.

Die Erfahrungen sind in das Design von Agimap (s. 6.6) eingeflossen.

ACT-COM ist als wiederverwendbares Modul ausgelegt. Die Wiederverwendbarkeit ist aufgrund der Generizität der Lösung gegeben, da das konkret zu ver-

wendende Protokoll und die Call Back Funktionen ausgetauscht werden können. Der grammatikalische Formalismus der kontextfreien Sprachen hat sich als ausreichend allgemein erwiesen. Die Schnittstellen des Moduls zum ACT-R Kern sind die zu spezifizierenden Funktionen, die beim Eintreffen von Sprechakten als Call Backs ausgeführt werden. Zur Nutzung des Moduls muss der ACT-R eigene `Cycle-Hook` auf eine bestimmte Funktion des Moduls gesetzt und das Protokoll mit den beiden Konstanten `*protocol-ident*` und `*protocol-flow*` definiert werden. Durch die Kapselung der generalisierbaren Modellelemente innerhalb des Moduls und die explizite Auflistung der Protokoll-spezifischen Anteile in den Variablen `*protocol-ident*` und `*protocol-flow*` wird die Übersichtlichkeit der kognitiven Modelle erhöht, die ACT-COM nutzen. Das Kommunikationsprotokoll ist über diese Spezifikation gut dokumentiert und kann später auch leicht an Änderungen der Schnittstelle der Aufgabenumgebung angepasst werden.

6.3. Timer

Timer ist eine neue Erweiterung von ACT-R, mit der die menschliche Zeitwahrnehmung modelliert werden kann. Solch eine Erweiterung ist erforderlich, denn die subjektiv wahrgenommene Zeit ist gegenüber der real verstrichenen Zeit verzerrt. Die Verzerrung hängt von der gegenwärtigen Arbeitsbelastung ab. Es kann unter bestimmten Voraussetzungen sowohl zu einer Unterschätzung oder Überschätzung kommen. Die ACT-R-Erweiterung Timer berechnet das Ausmaß der Verzerrung der wahrgenommenen Zeit mit einem psychologisch plausibilisierten Modell.

Zeitwahrnehmung ist wesentlich bei der kognitiven Modellierung von dynamischen Aufgabenumgebungen. Zum Einen werden Zeitinformationen für das Scheduling von parallel anstehenden Aufgaben bei Multitasking-Bedingungen benötigt, zum Anderen treten in Diagnoseaufgaben zeitliche Bedingungen auf, bei denen Zeitdauerüberschreitungen auf Fehlersituationen schließen lassen.

Eine Erweiterung von ACT-R ist erforderlich, da diese kognitive Architektur bislang keine explizite Verarbeitung von Zeitdauern ermöglicht. Lediglich der Zugriff auf die Systemzeit des Wirtssystems CommonLISP ist über Umwege möglich. Zeitdauern können dann durch Subtraktion von Zeitpunkten gewonnen werden. Alle Zeitverarbeitung muss aber mit Hilfe von ACT-R-externen LISP-Funktionen geschehen. Eine psychologisch plausible Verzerrung von Zeitdauern ist so nicht möglich.

In der Literatur wird die Zeitwahrnehmung auf die Aufgabe der Schätzung der Zeitdauer eines Intervalls zurückgeführt. Dabei werden zwei unterschiedliche Modi unterschieden: retrospektive und prospektive Zeitschätzung (z.B. Zakay & Block 1997). Bei der retrospektiven Zeitschätzung ist zu Beginn des zu schätzenden Intervalls die Aufgabe noch nicht bekannt. Der Beginn des Intervalls wird dafür anhand eines kommunizierbaren Ereignisses markiert. Zum Zeitpunkt der Dauerschätzung muss dann die Dauer seit dem betreffenden Ereignis geschätzt werden. Bei der prospektiven Zeitschätzung ist zum Beginn des Intervalls die Aufgabe bekannt, so dass eine mehr bewusste Zeitdauerwahrnehmung an die Stelle der retrospektiven Zeitdauerschätzung tritt.

Der Vergleich beider Modi ergibt, dass bei retrospektiver Zeitschätzung kürzere Dauern geschätzt werden als bei der prospektiven. Die prospektiven Urteile liegen dichter an der real vergangenen Zeitdauer. Unter hoher Arbeitsbelastung ergibt sich eine Überschätzung der vergangenen Zeit. Die Auswirkung bei dynamischen Aufgabenumgebungen mit Diagnoseaufgaben, die eine Zeitdauerschätzung beinhalten, ist, dass Operateure eine längere Dauer abwarten, bis sie zu einer Diagnose kommen, wenn sie parallel eine andere Aufgabe bearbeiten.

6.3.1. Entwurf retrospektiver Zeitdauerschätzung

Für die Umsetzung in ACT-R wird der Workload über die Anzahl feuernder Produktionen pro Zeiteinheit modelliert. Aufgrund der Erfahrung, dass auch für technisch bedingte und interne bzw. unbewusste Vorgänge Produktionen benutzt werden müssen, wird eine Klasse von Produktionen gebildet, die diese kognitiv wenig belastenden Verarbeitungsschritte beinhaltet. Produktionen dieser Klasse wirken sich nicht auf den intern repräsentierten Workload aus, der die Zeitdauerschätzung beeinflusst. Die anderen Produktionen werden als mentale Kontextwechsel interpretiert und tragen zur intern gemessenen Arbeitsbelastung bei. Sie wirken sich damit verzerrend auf die Zeitdauerwahrnehmung aus.

Für den Modus der retrospektiven Zeitdauerschätzung müssen Ereignisse, die den Beginn von Intervallen kennzeichnen, gespeichert werden. Dazu werden Chunks mit einer eindeutigen ID benutzt, um solche Ereignis zu kennzeichnen. Da diese Chunks eine andere Funktion haben und anderen Verarbeitungsmechanismen unterliegen als andere Chunks, wird ein eigenständiger Buffer mit speziellen Eigenschaften angenommen. Dieser Speicher bzw. die Anzahl von parallel existierenden Chunks dieses Typs muss begrenzt sein. Die ACT-R eigenen Mechanismen zur Begrenzung des Arbeitsgedächtnisses werden bewusst nicht verwendet.

Zum Abruf der wahrgenommenen Zeitdauer wird ein entsprechendes Kommando an diesen Buffer gesendet (s. Abbildung 6-2). Dabei muss die ID eines Intervall-kennzeichnenden Ereignisses mit angegeben werden. Die ID muss also zusätzlich als normaler Chunk aus dem Retrieval-Buffer abgerufen werden, da es sich um den retrospektiven Modus handelt, bei dem die Aufgabe Zeitdauerschätzung bei Beginn des Intervalls noch nicht bekannt war. Das Ergebnis steht danach als Ergebnis-Chunk zur Verfügung und die subjektiv verstrichene Zeit kann ausgelesen werden.

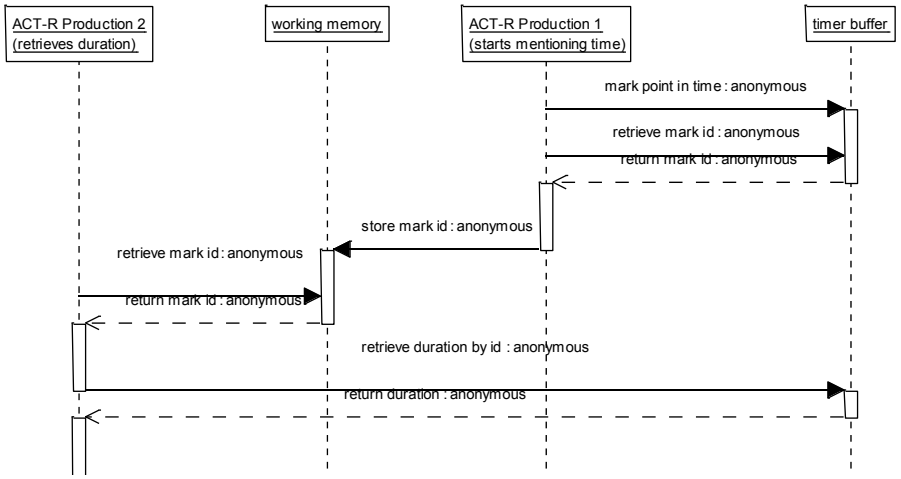


Abbildung 6-2: Sequenzdiagramm für die Verwendung des Timer-Buffers (retrospektive Zeitdauerschätzung)

6.3.2. Parametrierung und Implementierung

Für die retrospektive Zeitdauerschätzung wird implementierungstechnisch eine Menge von gleichzeitig laufenden Countern angenommen, die in jedem Ausführungszyklus in Abhängigkeit der Zugehörigkeit zur Klasse der gefeuerten Produktion erhöht werden. Dafür wird eine Hashmap mit Ereignis-ID als Schlüssel verwaltet.

Im Normalfall wird das Feuern einer Produktion mit einen Kontextwechsel gleichgesetzt, wenig belastende Produktionen ohne Kontextwechsel werden explizit durch ein parametrierbares Präfix im Namen ausgezeichnet („Idle-Produktionen“). Durch das Kommando zur Abfrage des Buffers wird der aktuelle Wert des internen Counters an eine externe Stelle übertragen: Die externe Repräsentation wird durch einen Slot an einem Chunk (normalerweise das aktuelle Ziel) bezeichnet.

Zur Berechung dieser Zeitdauer wird ein Zusammenhang zwischen den zwei Arten von Kontextwechseln und dem Grad der Verzerrung der Zeitdauerschätzung angenommen. Freie Parameter dieses Modells sind

- die Größe des zur Speicherung von Referenzen zur Verfügung stehenden Speichers (`*timer-capacity*`), also die Größe der Hashmap,
- die wahrgenommene Zeitdauer eines Kontextwechsels durch das Feuern einer Idle-Produktion (`*timer-factor-idle*`) und
- die wahrgenommene Zeitdauer eines Kontextwechsels durch das Feuern einer normalen Produktion (`*timer-factor-nonidle*`).

Der erste freie Parameter `*timer-capacity*` wird – wie auch an anderer Stelle in dieser Arbeit (s. 6.7) – mit einem Defaultwert von sieben angenommen. Für die beiden anderen Parameter wurden sinnvolle Werte aus der Literatur abgeleitet. Dazu wurden die experimentell ermittelten Resultate von McClain (1983) und Block et al. (1980) herangezogen und unter der Annahme eines linearen Zusammenhangs zwischen Beanspruchung und Verzerrung der Zeitdauerschätzung verrechnet. Die beiden anderen Parameter sind Faktoren, mit denen die tatsächlich vergangene Zeit multipliziert wird. Die tatsächlich vergangene Zeit einer Produktionsanwendung, also der Instanziierung, Auswahl und zum Feuern, kann in ACT-R 6.0 aus dem Systemparameter `*default-action-time*` in Sekunden ausgelesen werden. Für den Faktor `*timer-factor-idle*` ergibt sich mit 21/27 ein Wert kleiner als eins und für `*timer-factor-nonidle*` mit 8/3 ein Wert größer als eins. Das bedeutet, dass nach diesem Modell der Verzerrung die Dauer unter kognitiver Last über- und ohne Beanspruchung unterschätzt wird. Dem Ergebnis wird zusätzlich weißes Gauß’sches Rauschen hinzugefügt.

Quelltext 6-3 zeigt, wie Timer als neuer Buffer in ACT-R 6.0 verankert wird. Dabei werden die Funktionen `get-timer`, `set-timer` und `del-timer` als Schnittstellen zur Interaktion mit dem Buffer vereinbart. Sie werden ausgeführt, wenn im Bedingungsteil (lhs = *left hand side*) bzw. im Aktionsteil (rhs = *right hand side*) einer Produktion

per `=timer>` (`:equal-lhs` = `get-timer`), `+timer>` (`:plus-rhs` = `set-timer`) oder `-timer>` (`:minus-rhs` = `del-timer`) zugegriffen wird. Zum Transfer der Dauerergebnisse wird der Chunk `timerduration` erzeugt und im Buffer abgelegt. Fehlersituationen beim Umgang mit dem Buffer werden über den Chunk-Typ `timer-failure` als Ergebnis signalisiert.

```
(defvar *timer*)
(define-buffer timer *timer*
  :equal-lhs    get-timer
  :plus-rhs     set-timer
  :minus-rhs    del-timer)
(chunk-type timer-duration
            Id
            duration)
(add-dm (timerduration isa timer-duration
                       id nil
                       duration nil))
(setf *timer* (get-wme 'timerduration))
(chunk-type timer-failure)
```

Quelltext 6-3: Integration von Timer als Buffer in ACT-R

Die Referenzereignisse, die den Beginn eines Schätzintervalls kennzeichnen, werden in einem Hashmap als Chunks vom Typ `timer-reference` abgelegt (s. Quelltext 6-4).

```
(defvar *timer-references* (make-hash-table))
(chunk-type timer-reference
            id
            timestamp
            duration)
```

Quelltext 6-4: Chunk-Typ für die Repräsentation von Ereignissen zur Definition des Beginns von Schätzintervallen

Mit dem neuen Buffer `timer` kann über das API in Tabelle 6-2 interagiert werden. Es werden drei unterschiedliche Funktionen unterstützt:

(A) Das Setzen eines neuen Referenzzeitpunkts (A) über ein Kommando im Aktionsteil einer Produktion (RHS = *right hand side*).

(B) Die retrospektive Abfrage einer Zeitdauer seit einem Ereignis `=id` über ein Kommando im Aktionsteil einer Produktion. Das Ergebnis steht danach im Timer-Buffer zur Verfügung und kann über eine entsprechende Match-Spezifikation im Bedingungsteil einer weiteren Produktion (LHS = *left hand side*) ausgelesen werden. Das Ergebnis steht danach in `=duration` zur Verfügung.

(C) Aus implementierungstechnischen Gründen kann auch das Löschen eines Referenz-Chunks erforderlich sein. Dazu kann ebenfalls ein entsprechendes Kommando an den Timer-Buffer im Aktionsteil einer Produktion gesendet werden.

Tabelle 6-2: API zum Timer-Buffer für retrospektive Zeitdauerschätzung

(A)	RHS:
	`+timer>`
	`isa timer-reference`
	`mode retrospective`
	`id =id`
		neue Referenz setzen
		(max. Anzahl `*timer-capacity*`)

(B)	Schritt 1 (RHS):
	`+timer>`
	`isa timer-duration`
	`id =id`
		Kommando zum Abfragen der geschätzten Zeitdauer an Timer-Buffer schicken.

	Schritt 2 (LHS):
	`=timer>`
	`isa timer-duration`
	`id =id`
	`duration =duration`
	oder
	`=timer>`
	`isa timer-failure`
		Abfragen der geschätzten Zeitdauer oder Test auf einen Fehler (z.B. ID unbekannt oder vergessen)

(C)	RHS:
	`-timer>`
	`isa timer-reference`
	`id =id`
		Timer-Referenz löschen (nur für technische Erfordernisse)

An jedem gespeicherten Chunk, der einen Referenz-Zeitpunkt markiert, wird nach jedem Produktionsschritt die aktuelle subjektiv verstrichene Zeit im Slot `duration` repräsentiert. Um das zu erreichen, wird nach jedem Feuern einer Produktion eine entsprechende Update-Funktion ausgeführt. Dazu wird eine anonyme Funktion („Lambda-Ausdruck“) an einen entsprechenden Hook gebunden, der in der ACT-R Architektur dafür definiert ist.

Quelltext 6-5 zeigt eine vereinfachte Fassung der Funktion. Dem Lambda-Ausdruck wird die aktuelle Produktionsinstanz als Parameter übergeben. Daher kann geprüft werden, ob der Produktionsname mit dem Präfix beginnt, der Idle-

Produktionen kennzeichnet (`*timer-idle-prefix*`). Es wird der entsprechende Wert `*timer-factor-idle*` oder `*timer-factor-nonidle*` an die lokale Variable `factor` gebunden. Als globale Variable wird `*timer-lasttimestamp*` an den aktuellen Zeitpunkt gebunden. Dadurch kann in `t-real` berechnet werden, wie viel Realzeit seit der letzten Produktionsanwendung vergangen ist. Mit `maphash` werden dann alle Chunks, die in `*timer-references*` Referenzzeitpunkte repräsentieren, aktualisiert. Deren Slot `duration` wird auf den neuen Wert gesetzt. Der neue Wert wird berechnet, indem der alte Wert um `factor * t-real` erhöht wird. Das ist die um `factor` verzerrte Zeit, die seit dem letzten Update vergangen ist.

```
(setf *firing-hook-fn*
      (lambda (instantiation)
          (let* ((p      (mk (instantiation-production instantiation))
                 (now    (get-time))
                 (t-real (- now *timer-lasttimestamp*))
                 (prel   (length *timer-idle-prefix*))
                 (factor (if (and (> (length p) prel)
                                  (string= *timer-idle-prefix* p
                                           :start1 0 :end1 prel
                                           :start2 0 :end2 prel))
                             *timer-factor-idle*
                            *timer-factor-nonidle*)))
            (setf *timer-lasttimestamp* now)
            (maphash #'(lambda (key wme)
                 (let ((duration (get-slot-value wme 'duration)))
                     (mkeval (modwme ~a duration ~a)
                             wme
                            (+ duration (* factor t-real))))
              *timer-references*))))
```

Quelltext 6-5: Neuberechnung aller Counter nach dem Feuern einer Produktion. Die Produktionsinstanz wird der lambda-Funktion übergeben als `instantiation` (vereinfacht).

6.3.3. Prospektive Zeitdauerschätzung

Für die prospektive Zeitschätzung wird ein anderes Modell verwendet. Es basiert strukturell auf dem *Attentional Gate Modell* (Zakay & Block 1997). Es bildet die bewusste Zuwendung auf die Zeitdauerschätzung ab. Auch hier kommt es zu einer Verzerrung. Sie resultiert aus der Beschränktheit eines internen Counters, der zur Verfolgung der verstrichenen Zeit verwendet wird. Unter der Annahme, dass nur eine Zeitdauer bewusst verfolgt werden kann, wurde genau ein interner Counter definiert, der bei jedem Kontextwechsel (Feuern von Produktionen, die nicht zur Idle-Klasse gehören) um `*timer-counter-inc*` erhöht wird. Die Kapazität dieses internen Counters ist begrenzt auf `*timer-counter-size*`; größer kann der Wert nicht werden.

Der interne Counter kann bewusst durch ein Kommando (`timer-reset`) an den Timer-Buffer zurückgesetzt werden, dann wird wieder bei 0 angefangen aufzuaddieren.

Durch das `timer-updatecounter`-Kommando an den Timer-Buffer wird der aktuelle Wert dieses internen Counters an eine externe Stelle übertragen: Der gewünschte Ort der externen Repräsentation wird durch einen Slot an einem Chunk (normalerweise das aktuelle Ziel) bezeichnet. Der dort repräsentierte Wert wird automatisch um den im internen Counter repräsentierten Wert erhöht und der interne Counter zurückgesetzt.

Diese Funktionen sind analog zu denen bei der retrospektiven Zeitdauerschätzung in ACT-R integriert worden (s. Quelltext 6-6). Die Wahl der Werte für die beiden freien Parameter `*timer-counter-size*` und `*timer-counter-inc*` resultiert aus einer ersten Analyse der Annahmen der zugrunde liegenden Theorie.

```
(chunk-type timer-reset)
(chunk-type timer-updatecounter
            chunk
            slot)
(chunk-type timer-failure)
(defparameter *timer-counter-size*   3)
(defparameter *timer-counter-inc*    *default-action-time*)
```

Quelltext 6-6: Chunk-Definitionen und freie Parameter für die Implementierung des prospektiven Modus

Die Programmierschnittstelle für die Nutzung der prospektiven Zeitschätzung mit dem Timer-Buffer ist in Tabelle 6-3 dargestellt. Es liefert genau zwei Kommandos für den Aktionsteil von Produktionen zum Zurücksetzen des Counters und zur Übertragung des aktuellen Schätzwertes.

Tabelle 6-3: API zum Timer-Buffer für prospektive Zeitdauerschätzung

(A)	RHS:
	`+timer>`
	`isa timer-reset`
		internen Counter zurücksetzen (auf 0 setzen)

(B)	RHS:
	`+timer>`
	`isa timer-updatecounter`
	`chunk =chunk`
	`slot =slot`
		den Wert des internen Counters an den externen übertragen und internen Counter zurücksetzen.

Quelltext 6-7 zeigt die Integration der neuen Kommandos in die Interface-Funktion des Timer-Buffers. In den beiden relevanten Zweigen der Fallunterscheidung wird `*timer-counter*` zurückgesetzt bzw. dessen Wert an einen anderen Chunk-Slot übertragen.

```
(defun set-timer (arguments)
 (let* ((name    (pop arguments))
        (buffer  (pop arguments))
        (specs   (funcall (pop arguments)))
        (isapos  (position 'isa specs))
        (isa     (nth (+ 1 isapos) specs))
        (cmd     (string-downcase (format nil "~a" isa))))
   (cond
    …
    ;;; reset counter
    ((string= cmd "timer-reset")
     (setf *timer-counter* 0.0))
    …
    ;;; request current counter
    ((string= cmd "timer-updatecounter")
     (let* ((chunkpos   (position 'chunk specs))
            (chunk      (nth (+ 1 chunkpos) specs))
            (slotpos    (position 'slot specs))
            (slot       (nth (+ 1 slotpos) specs))
            (newval     (mkeval (+ ~a ~a)
                                (get-slot-value (get-wme chunk) slot)
                                *timer-counter*)))))
         (mkeval (modwme ~a ~a ~a) chunk slot newval)
         (setf *timer-counter* 0.0)))
    …
)))
```

Quelltext 6-7: Ausschnitt der für die prospektive Schätzung relevanten Teile aus der Funktion, die das Timer-Buffer Kommando implementiert

Wie bei der retrospektiven Zeitdauerschätzung muss nach jedem Produktionsschritt der Counter aktualisiert werden. Zusätzliche Logik ist zum Beschränken der Counterkapazität erforderlich (s. Quelltext 6-8).

```
(defvar *timer-counter* 0.0)
(setf *firing-hook-fn*
  (lambda (instantiation)
    (unless idle (prong
       (setf *timer-counter* (+ *timer-counter* *timer-counter-inc*))
       (if (> *timer-counter* *timer-counter-size*)
           (setf *timer-counter* *timer-counter-size*))))
```

Quelltext 6-8: Timer-Buffer für prospektive Zeitdauerschätzung

Die adäquate Anwendung dieses Schemas erfordert im Modell eine anwendungsspezifische Hinwendung auf die Zeitdauerschätzung. Insbesondere muss der Counter regelmäßig durch expliziten Kommando-Aufruf aus einer Produktion an einen Chunk-Slot übertragen werden. Gelingt das nicht oft genug, weil Ressourcen dazu fehlen, kommt es zu einer Unterschätzung der verstrichenen Zeitdauer.

6.3.4. Abgrenzung

Bis jetzt bietet keine kognitive Architektur standardmäßig Zugriff auf eine psychologisch plausible Zeitwahrnehmung. Es gibt jedoch für ACT-R eine experimentelle Erweiterung, die auch in Form eines neuen Buffers implementiert wurde (Taatgen et al. 2004, 2005). Hier wird aber ein völlig anderes Modell der Zeitverzerrung benutzt als im vorgestellten Ansatz. Prospektive Zeitwahrnehmung wird nicht unterstützt. Bei der retrospektiven Zeitschätzung wird auf eine Registrierung von Kontextwechseln durch Zählen von Produktionsanwendungen verzichtet. Stattdessen wird der Rückgriff auf vergangene Zeitdauer mit einer quadratischen Funktion verzerrt, die unabhängig von der mentalen Beanspruchung ist.

6.3.5. Anwendung in Fallstudie und Bewertung

Der Timer-Buffer wurde in der Fallstudie zur Rektifikationskolonne eingesetzt. Dabei wurden beide Modi benutzt. Die Erfahrungen aus der Anwendung zeigen, dass der retrospektive Modus bei der Modellierung leicht anzuwenden ist (Pape et al. 2005). Der prospektive Modus, der bei einigen Aufgabenumgebungen jedoch zwingend eingesetzt werden muss, erfordert eine explizite Modellierung der Aufmerksamkeitshinwendung auf die Zeitdauerschätzaufgabe, was einen hohen Modellierungsaufwand bedeutet, da dies die Architektur im Gegensatz zum anderen Modus nicht automatisch übernimmt.

Der in dieser Arbeit vorgestellte Ansatz der retrospektiven Zeitdauerschätzung wurde von Dzaack et al. (2006) weitergenutzt und auf der Basis umfangreicher

empirischer Untersuchungen neu parametriert und im Kontext einer synthetischen Aufgabe zum Test der Aufmerksamkeit und Konzentration erfolgreich eingesetzt.

Die Timer Erweiterung von ACT-R ist in Form eines Buffers als Modul gekapselt und somit in hohem Maß wiederverwendbar. Zum Ansprechen der Schnittstelle des Moduls werden Kommandos und Strukturen verwendet, die analog zu anderen Modulen gestaltet sind. Dadurch fügt sich diese Schnittstelle gut in ACT-R ein. Das Timer Modul könnte damit auch als Teil der kognitiven Architektur von der ACT-R *community* akzeptiert werden.

Die Verarbeitungsprozesse und Strukturen, die innerhalb des Buffer-Moduls verwendet werden, um die Timer Funktionalität zu realisieren, sind für den Modellierer nicht sichtbar. Damit wird das Prinzip des *information hiding* aus der Softwaretechnik aufgegriffen. Dadurch dass ausschließlich die Schnittstelle des Moduls nach außen hin sichtbar ist, wird gewährleistet, dass die inneren Funktionsprinzipien des Moduls bei Beibehaltung der Schnittstellenspezifikation ausgetauscht werden können, ohne dass Programme bzw. kognitive Modelle, die das Modul verwenden, geändert werden müssten. Dadurch ist es z.B. im Fall des Timer Moduls möglich, ein abweichendes Verhalten bei der Zeitwahrnehmung zu implementieren, wenn die kognitive Architektur aufgrund neuer Beobachtungen geändert werden muss. Das neue Timer Modul ist bei Beibehaltung der Schnittstelle rückwärtskompatibel und kognitive Modelle können den neuen Buffer weiterverwenden.

6.4. Visualisierung von Produktionssystemen

Visualisierungstechniken spielen eine wesentliche Rolle in der Modellierung. Grafische Repräsentationen von Modellen helfen bei der Kommunikation und Ausbildung. Ebenso erleichtern sie das Verständnis fremder Modelle. Im Bereich der kognitiven Modelle fehlen bisher Visualisierungstechniken. Das hier vorgestellte System ist eine neue Entwicklung zur grafischen Repräsentation wesentlicher ACT-R-Modellelemente in Form einer grafischen Sprache und eines grafischen Editors, mit dem anhand dieser grafischen Sprache modelliert werden kann.

Es ist somit ein System zur Softwarevisualisierung. Zur Einordnung des Typs der Visualisierung wird eine Taxonomie verwendet (Price et al. 1993). Diese Taxonomie unterscheidet die Eigenschaften Ausrichtung (*scope*), Inhalt (*content*), Form (*form*), Methode (*method*), Interaktion (*interaction*) und Effektivität (*effectiveness*). Im Rahmen dieser Arbeit ist nur der Inhaltsaspekt relevant. Abbildung 6-3 zeigt die Ausprägungen der möglichen Inhaltsaspekte von Softwarevisualisierung.

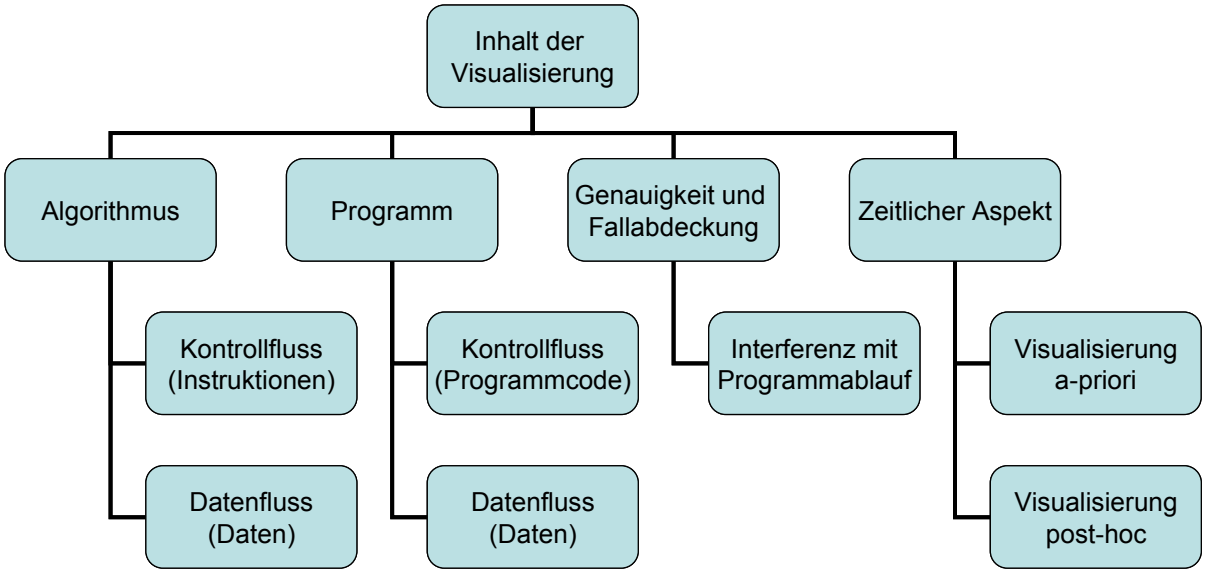


Abbildung 6-3: Taxonomie der möglichen Inhaltsaspekte von Softwarevisualisierung (nach Price et al. 1993)

Algorithmus/Programm: Die Visualisierung kann sich auf die abstrakte Idee des Algorithmus beziehen oder die konkrete Realisierung im Programmcode darstellen. In beiden Fällen kann der Kontrollfluss, also der prozedurale Ablauf der Funktionen, oder der Datenfluss, also die Nutzung von Daten und ihre Übertragung zwischen Funktionsbereichen, visualisiert werden.

Genauigkeit und Fallabdeckung: Die grafische Darstellung muss sich nicht auf das gesamte Programm beziehen, sondern es kann ein interessierender Aspekt ausgewählt werden. Wenn die Visualisierung parallel zum Programm erzeugt und angezeigt wird, kann es zu einer Interferenz mit dem Programmablauf kommen.

Zeitlicher Aspekt: Die Visualisierung basiert auf Daten. Die Daten können aus dem Programmablauf in Form von Logfiles gewonnen werden. Dann kann die Visualisierung nur post-hoc geschehen. Das Visualisierungssystem kann jedoch auch das zugrunde liegende Programm oder den Algorithmus analysieren und eine a-priori Visualisierung daraus generieren.

In dieser Arbeit wird ein System zur Visualisierung *des Algorithmus*, der einem Programm (bzw. ACT-R-Modell) zugrunde liegt, beschrieben. Die *Abdeckung ist komplett*, es gibt *keine Interferenz* mit dem ACT-R-Modell. Das System visualisiert den *Kontrollfluss* des Algorithmus sowie einen fokussierten Ausschnitt der *Daten*, die verarbeitet werden, *a-priori*.

6.4.1. Entwurf der grafischen Modellierungssprache

Anwendungsorientierte ACT-R/PM-Modelle sind groß und komplex. Sie bestehen oft aus mehreren hundert Produktionen. Deshalb ist es sehr schwer, vorliegende ACT-R-Modelle von anderen Modellierern zu verstehen und auch in eigenen Modellen den Überblick zu behalten. Insbesondere im Modellentwurf und bei der Fehlersuche fehlt oft das Verständnis für den intendierten Ablauf und Seiteneffekte von Produktionen. Die Visualisierung mit einer grafischen Sprache soll insbesondere helfen, den Kontrollfluss und die möglichen Produktionssequenzen zu verstehen.

Im Gegensatz zu imperativen Programmen ist der Kontrollfluss in Produktionssystemen wie ACT-R nicht im Quelltext explizit spezifiziert. Stattdessen wird die Verkettung von Produktionen erst zur Laufzeit durch die Konfliktresolution erzeugt. Ungeachtet dessen muss auch in Produktionssystemen ein Kontrollfluss modelliert werden. Dies geschieht durch die Manipulation von Chunks im Arbeitsgedächtnis.

Für eine a-priori-Berechnung des Kontrollflusses ist die Berücksichtigung des komplexen Konfliktresolutionsmechanismus' von ACT-R zu aufwändig. Stattdessen wird die Zielverarbeitung (Matching und Modifizierung) verfolgt. Am Ziel-Chunk werden normalerweise die einzelnen Stufen der Problemlösung repräsentiert. Vorgeplante Produktionssequenzen werden über die gezielte Modifikation des aktuellen Ziels realisiert.

Die Visualisierung nutzt den verbreiteten und standardisierten Statechart-Formalismus (Harel 1987) als Basis der Visualisierung des Kontrollflusses. Die Statechart-Zustände werden verwendet, um den Zustand des aktuellen Ziels zu repräsentieren. Der Zustand eines Ziels ergibt sich aus seinem Typ und der Belegung seiner Slots. Die Statechart-Übergänge bilden die Anwendung von Produktionen ab. Jede Produktion im ACT-R-Modell wird durch genau einen Übergang visualisiert.

Um die Darstellung übersichtlich und aussagekräftig zu machen, müssen die Goal-Zustände geschickt generalisiert werden, indem passende Zustandsinvarianten

identifiziert werden. Dabei werden zur Zustandsidentifikation nicht nur Slot-Belegungen mit Werten, sondern auch mit `nil`, also das Nichtbelegtsein, die Negation der Belegung mit einem bestimmten Wert und die fehlende Einschränkung berücksichtigt. Die Generalisierung wird durch einen Unifikationsvorgang über diesen Wertebereich realisiert.

Abbildung 6-4 zeigt die Visualisierung eines einfachen Additionsmodells aus dem ACT-R 5.0 Tutorial[7], Unit 1. Das Ziel in diesem Modell ist die Addition zweier kleiner Zahlen `arg1` und `arg2` (Summe ≤ 10). Das Ergebnis soll im Slot `sum` gespeichert werden. Das Ziel ist als Statechart-Zustand dargestellt. Anfangs (oberster Zustand) ist `sum nil` und die beiden beliebigen Summanden sind an Variablen gebunden (und ≠ `nil`). Die einzige Produktion, die von diesem Zustand aus benutzt werden kann, ist `initialize-addition`. Die zusätzlichen Informationen am Übergang zeigen zusätzliche Bedingungen und Seiteneffekte der Produktion in verkürzter Form an.

Der nächste Zustand (in der Mitte) zeigt die Invariante der Slot-Belegungen, die hier zutreffen. Insbesondere hat der Slot `count` einen Wert und `sum` ist nicht mehr `nil`. Von diesem Zustand aus sind drei Produktionen anwendbar. `increment-count` und `increment-sum` unterscheiden sich im Test `count` ≠ `arg2`. `terminate-addition` kann angewendet werden, wenn `arg2` und `count` gleich sind. Die Anwendung dieser Produktion führt zum Endzustand wo `count nil` ist.

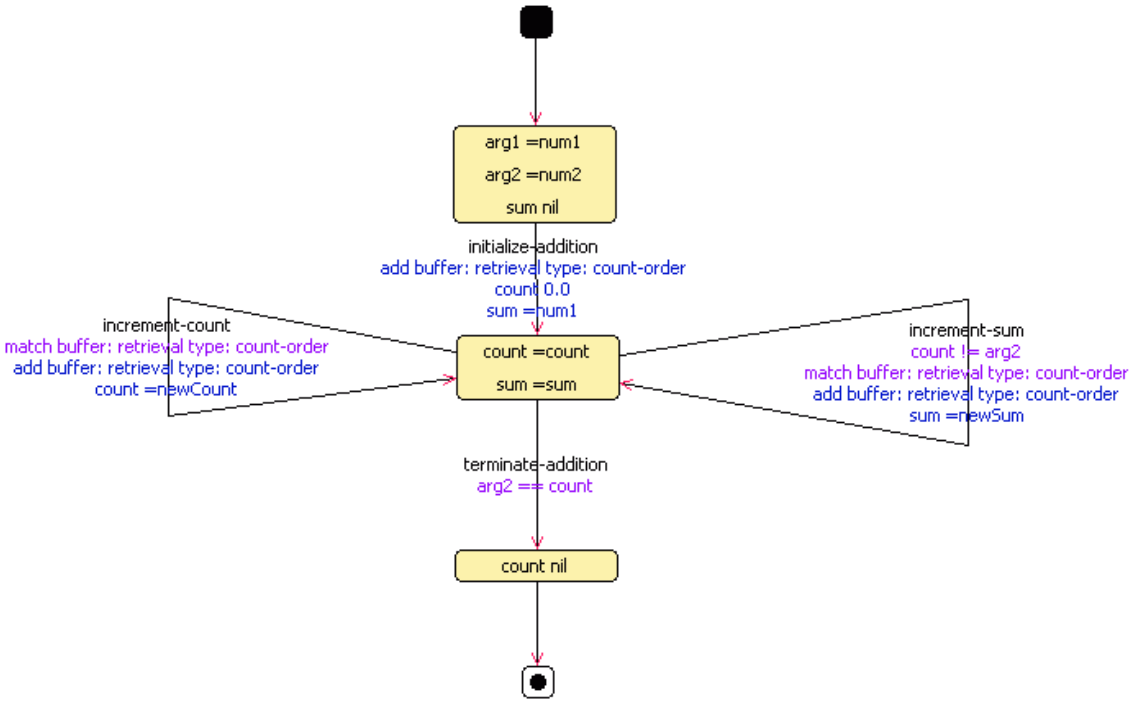


Abbildung 6-4: Visualisierung eines einfachen Additionsmodells aus der ACT-R Tutorial Unit 1

[7] http://act-r.psy.cmu.edu/tutorials/, Version vom Juni 2004

Für die Verarbeitung in ACT-R ist auch die Unterzielbildung ein wesentliches Element der Kontrollflusssteuerung. In der hier vorgestellten Statechart-Visualisierung wird jedes Ziel als *super-state* behandelt und dargestellt. Die Neben- und Unterzielbildung wird durch Transitionen zwischen *super-states* visualisiert (Bsp. s. Abbildung 6-5).

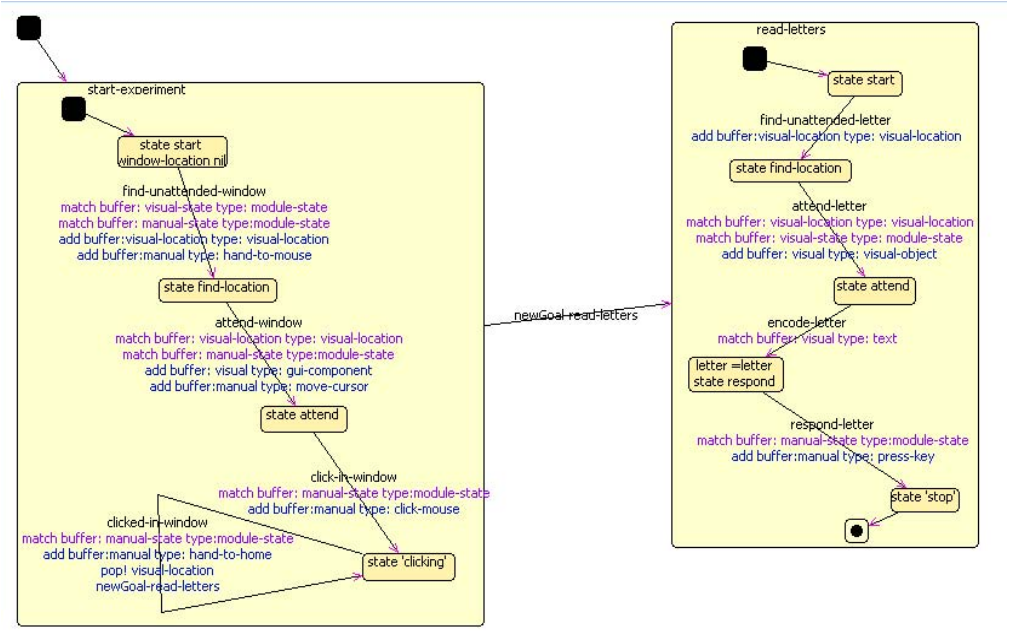


Abbildung 6-5: Visualisierung eines ACT-R-Modells mit Unterzielbildung

6.4.2. Implementierung

Die Analyse des ACT-R-Modells erfolgt in einem mehrstufigen Prozess (s. Abbildung 6-6). Zuerst wird der ACT-R Code, der Einfachheit halber in XML repräsentiert (DTD s. Anhang B), mit einem Parser in eine Zwischenrepräsentation gebracht. Aus dieser Zwischenrepräsentation wird das Objektmodell generiert, das mit einem Framework visualisiert wird.

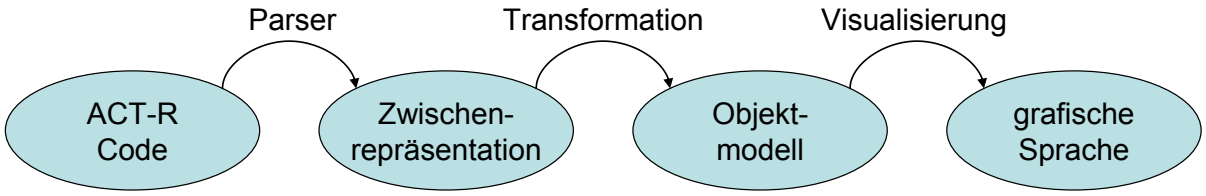


Abbildung 6-6: Flussmodell der ACT-R Visualisierung

Die Zwischenrepräsentation wird aus den Produktionen und den in ihnen spezifizierten Chunks generiert. Auf andere Modellkonstrukte (z.B. `Chunk-Type`, `add-dm`) wird nicht zurückgegriffen. Der Parser extrahiert aus dem Bedingungsteil jeder Produktion die Spezifikation des Goal-Buffers. Die Bedingungsspezifikationen der anderen Buffer wird für die Identifikation der Ablaufstruktur ignoriert und nur als zusätzliche Information verkürzt an den Kantenübergängen repräsentiert. Die

Spezifikation von Ziel-Chunk-Bedingungen Spec geschieht über eine Menge von Slot-Belegungen:

$$\text{Spec} \subset \boldsymbol{P}\{\,(\text{name, value, modifier}) \mid \text{value} \in \text{Symbol} \cup \text{Number} \cup \text{NIL} \cup \text{Variable},\ \text{modifier} \in \{\text{not}, \varnothing\}\,\}$$

Den konkreten Zielspezifikationen (s $\in$ Spec) werden für die Visualisierung von Neben- und Unterzielen *super-states* anhand der isa-Belegung zugeordnet. Dann werden innerhalb jedes *super-states* Klassen von Ziel-Spezifikationen mit einem Unifikationsalgorithmus gebildet. Diese Klassen werden als Goal-States gespeichert. Sie repräsentieren Statechart-Zustände, die später als Knoten im Diagramm dargestellt werden. Außerdem müssen Start- und Endknoten identifiziert werden.

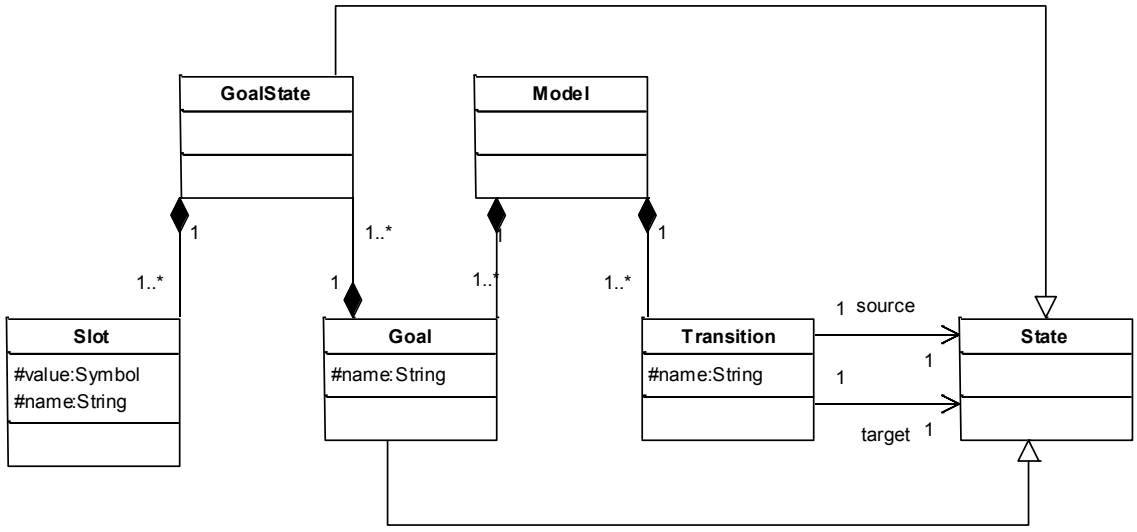


Abbildung 6-7: Klassendiagramm des Objektmodells ACT-R-Modell für Visualisierung

Das Objektmodell, das aus dieser Zwischenrepräsentation abgeleitet wird, entspricht dem Klassendiagramm in Abbildung 6-7). `GoalStates` mit ihren `Slots` und die dazugehörenden `Transitionen` werden dann visualisiert.

Zur Visualisierung wird eclipse mit dem GEF Framework verwendet. Eclipse ist eine verbreitete Entwicklungsumgebung für Java und andere Programmiersprachen. Es besitzt ein Framework zur Erweiterung durch Komponenten, die neue Funktionen innerhalb der Entwicklungsumgebung bereitstellen (Abbildung 6-8). In Eclipse gibt es eine Reihe von Basisdiensten, wie z.B. SWT und JFace zur Erzeugung und Verwaltung von GUI-Elementen, die als *Workbench* angeboten werden. Diese Basisfunktionalitäten können über definierte Interfaces der *Workbench* von Plugin-Komponenten (z.B. JDT, GEF) benutzt werden, auf deren Funktionalität wiederum andere Plugins zugreifen können.

Da das Framework auch mit GEF (*graphical editing framework*: Moore et al. 2004) ein Plugin mit einer Architektur für die grafische Modellierung beinhaltet, bietet es sich als Basis für die Implementierung der ACT-R Visualisierung an. GEF ist geeignet für die Implementierung neuer grafischer Editoren mit graph-artigen Darstellungen. Neue Editoren werden durch die Erweiterung vorhandener Klassen und die Implementierung von vorgegebenen Interfaces in neuen Plugins implementiert (s. Abbildung 6-9). Die Architektur von GEF basiert auf dem Model-View-Controller Design Pattern (Gamma et al. 1995). Das bedeutet, dass jedes Anzeige-Element eine zugeordnete Modellrepräsentation hat, deren Inhalt vom Anzeige-Element dargestellt wird. Änderungen in Anzeige oder Modell werden transparent im jeweils zugeordneten Objekt verwaltet.

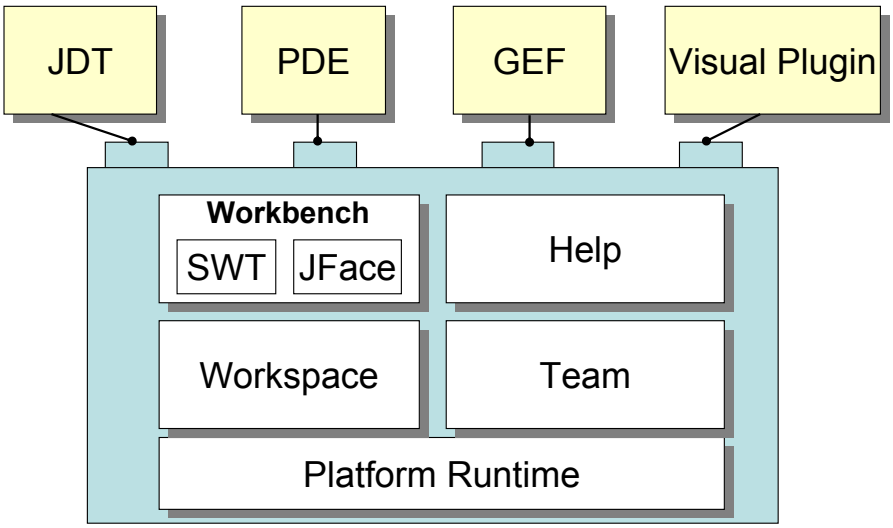


Abbildung 6-8: Software-Architektur von eclipse (nach Gallardo et al. 2003, S. 8)

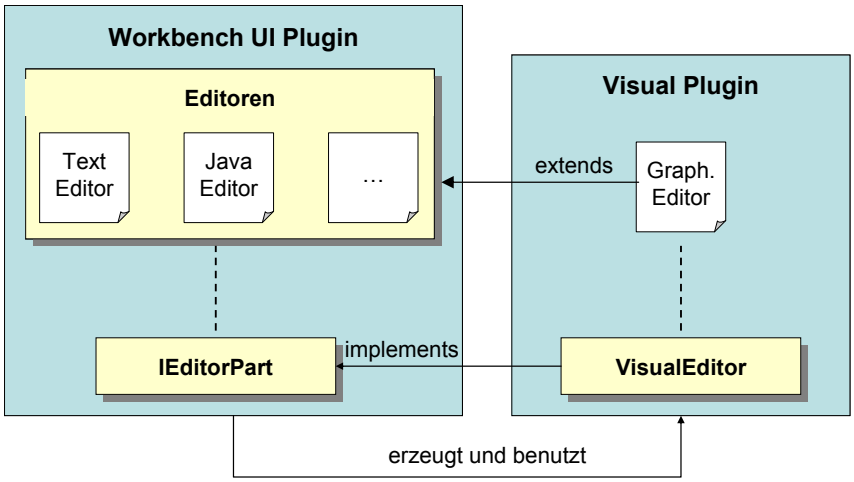


Abbildung 6-9: Nutzung von GEF in eclipse mit Plugin (nach Shavor et al. 2004, S. 201)

Mit diesem Ansatz wurde ein neues Plugin entwickelt, das die Visualisierung in eclipse zur Verfügung stellt (Nekrasova 2005). Das Plugin enthält das beschriebene

Objektmodell und mit dem Package `Figures` entsprechende Views für GEF. Controller werden von den Packages `Edit` und `Transform` bereitgestellt (s. Abbildung 6-10). Das Plugin stellt einen neuen Editor-Typ bereit, mit dem ACT-R-Modelle (in einer XML-Repräsentation) visualisiert werden können.

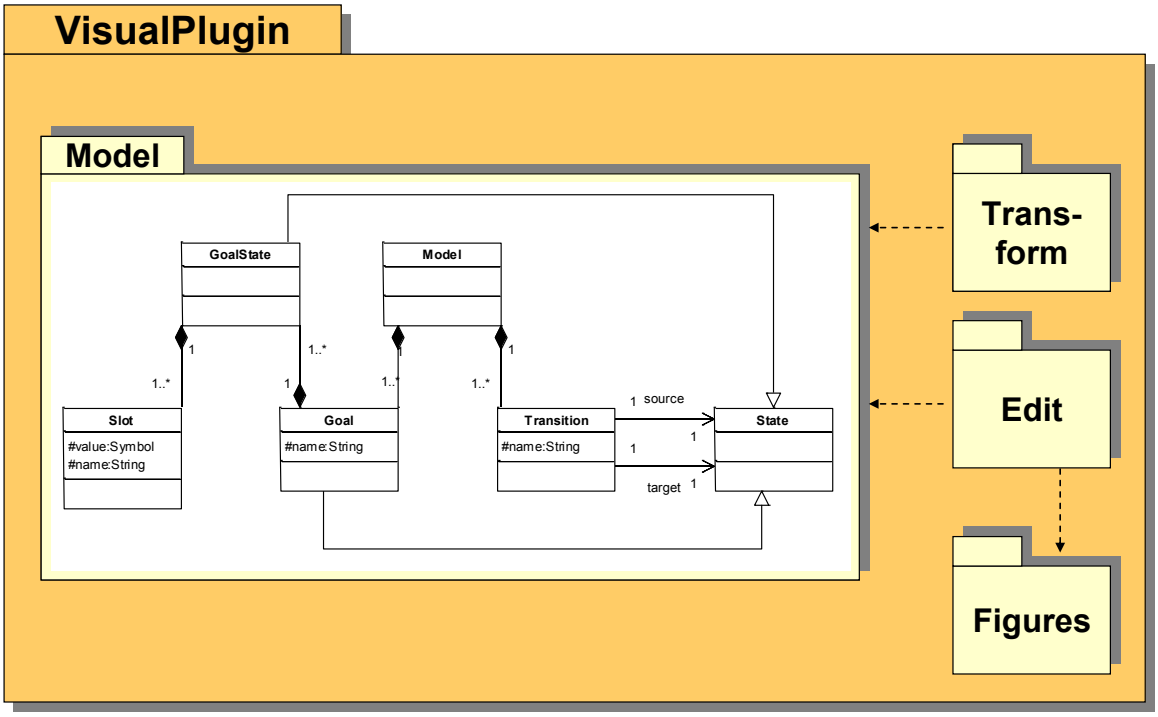


Abbildung 6-10: Packagestruktur des Visualisierungs-Plugins für eclipse

6.4.3. Abgrenzung

Es gibt eine Reihe von alternativen Visualisierungsansätzen bei kognitiven Modellierungswerkzeugen:

- COGENT (Cooper 2002) beinhaltet einen grafischen Editor, mit dem die Modellierungsbausteine miteinander verschaltet werden können. Diese Visualisierung ist jedoch nicht auf der Ebene des Kontrollflusses zwischen Produktionen (bzw. COGENT-Regeln) oder Chunks, sondern visualisiert die modellierte kognitive Architektur auf der höchsten Abstraktionsebene.
- In Soar (Laird et al. 1987, Newell et al 1993) ist seit den ersten Veröffentlichungen die grafische Notation von Problemräumen im Gebrauch. Damit wird der Prozess des Chunkings durch abgetrennte visuelle Elemente dargestellt. Innerhalb dieser Elemente können zusätzlich Zustandsübergänge mit Operatoren dargestellt werden. Die Notation der Zustandsübergänge ist nicht vereinheitlicht worden – insbesondere die Generalisierung von Zuständen wird nicht vorgenommen. Deshalb hilft diese Notation nicht bei der Dokumentation umfangreicher Modelle. Sie wurde insbesondere nicht bei der Dokumentation von Modellen in anderen kognitiven Architekturen übernommen.

- SoarML (Taylor & Crossman 2006) ist eine neue Modellierungssprache für Soar, die an UML angelehnt ist und eine Reihe von spezifischen Konstrukten enthält. Mit SoarML ist es auch möglich den Kontrollfluss eines Modells darzustellen. Bis jetzt ist allerdings ein Round-Trip Engineering, d.h. sowohl das Erzeugen von Soar-Code aus einem Modell, als auch die Generierung einer SoarML-Darstellung aus einem Soar-Modell nicht möglich. Als Notationssprache ist SoarML jedoch nützlich.
- In der Literatur über ACT-R-Modelle werden oft Zusammenhänge zwischen Zielen dargestellt. Eine spezielle Notation hat sich noch nicht herausgebildet. Oft werden konzeptuell Zielbäume mit und-/oder-Elementen verwendet. Mit ihnen ist jedoch keine Generalisierung über Invarianten oder die Darstellung von Produktionen möglich.

Neben diesen Visualisierungsansätzen, die auf die Dokumentation von Aspekten des Modellaufbaus abzielen, gibt es auch einige Systeme für die run-time Visualisierung:

- CaDaDis (Tor et al. 2004) ist ein System, um die Produktionsanwendung bei unterschiedlichen Systemen zu visualisieren, u.a. wird ACT-R unterstützt. Die Visualisierung erfolgt in Form eines PERT-Diagramms.
- In der ACT-R Umgebung der *Carnegie Mellon University* (s. S. 187) ist eine ähnliche Visualisierung wie CaDaDis integriert.

Der in dieser Arbeit vorgestellte Ansatz und das implementierte System sind einzigartig, da es das einzige System ist, das Modelle einliest und daraus eine Visualisierung des Kontrollflusses auf der Ebene der Zielverarbeitung und Produktionsanwendung generiert, in der Verarbeitungsschritte bzw. Invarianten der Chunk-Zustände generalisiert werden.

6.4.4. Einsatz in Fallstudie und Bewertung

Das vorgestellte System ist als Prototyp in einer Diplomarbeit implementiert worden (Nekrasova 2005). Es wurde nicht in einer Fallstudie erprobt, denn das System hat noch nicht die Funktion eines Editors. Neue Elemente (Produktionen oder `GoalStates`) können einem existierenden Modell in der Visualisierung also nicht hinzugefügt oder existierende Modellelemente geändert werden. Die Interaktion mit dem Editor beschränkt sich bislang auf eine Änderung der räumlichen Anordnung der Modellelemente. Insofern ist noch keine Modellierungsunterstützung oder sogar ein komplettes *Round-Trip Engineering* möglich. Das System eignet sich jedoch schon, um existierende ACT-R-Modelle zu visualisieren.

Dazu müssen sie erst in die XML-Repräsentation gebracht werden. Hierfür existiert ein CommonLISP-Programm, das die LISP-Makros eines ACT-R-Modells einliest und eine entsprechende XML-Repräsentation ausgibt, die mit dem vorgestellten System eingelesen werden kann.

Die Implementierung hat noch einige Schwächen, die die Benutzbarkeit einschränken. So ist lediglich ein sehr einfacher Layout-Manager realisiert, so dass das grafische Erscheinungsbild eines eingelesenen Modells von Hand angepasst werden muss. Das manuell verbesserte Layout kann jedoch mit dem Modell abgespeichert und wieder aufgerufen werden. Die Skalierbarkeit der Visualisierung leidet noch unter dem Fehlen einer Zoomfunktion und der fehlenden Möglichkeit, Submodelle zu bilden und getrennt zu bearbeiten.

Die Beschäftigung mit diesem Bereich hat jedoch gezeigt, dass die Visualisierung des Kontrollflusses eine kritische Komponente für den anwendungsorientierten Einsatz von ACT-R ist. Die Fähigkeit zum *Round-Trip Engineering* ist dabei von entscheidender Bedeutung. Die Verfügbarkeit von Werkzeugen zu Softwaresystem-modellierung mit UML, die diese Fähigkeit haben, hat zu einem Durchbruch der modell-orientierten Softwareentwicklung und einer Qualitätsverbesserung des Software Engineering geführt.

Die Verwendung von Eclipse als Basissoftware für die Implementierung der Visualisierung ist ein gangbarer Weg. Das GEF-Framework erlaubt die effiziente Realisierung von Modellanalyse und Visualisierung. Eine zukünftige Erweiterung des hier vorgestellten Visualisierungsansatzes um Elemente von SoarML oder Herbal wäre eine vielversprechende Entwicklungsrichtung.

6.5. ACT-R Shell

Im Rahmen der Fallstudie Modell der Fluglotsenleistungen – MoFL wurde ein komplexes kognitives Modell mit ACT-R Version 3.0 entwickelt (s. 5.1). Zu diesem Zeitpunkt gab es keine grafische Benutzungsoberfläche zur Arbeit mit dem ACT-R-System und auch die Anbindung von ACT-R an externe Prozesse war Pionierarbeit. Um mit der Komplexität des zu entwickelnden Modells umgehen zu können und die Integration in das heterogene Systemumfeld zu bewältigen, mussten neue Hilfsmittel entwickelt werden.

Das vorliegende Frontend „ACT-R Shell“ ist eine anwendungsspezifische IDE für die Arbeit mit dem ACT-R-Modell MoFL. Die Umgebung läuft unter Linux unter X11 mit ACT-R Version 3.0b4. Zum Betrieb werden Perl, GNU CommonLISP (gcl) und tixwish, einer Ablaufumgebung für die Programmiersprache Tcl/Tk mit der Erweiterung Tix, benötigt.

Die grafische Benutzungsoberfläche bietet Online-Zugriff auf unterschiedliche Sichten der internen Vorgänge der laufenden Simulation des kognitiven Modells. Zusätzlich ist die Steuerung der Simulation (sowohl des kognitiven Modells, als auch der Luftraumsimulation) von dort aus möglich. Dazu kann die Simulation über ein Menü gestartet, angehalten, fortgesetzt und beendet werden. In einem leicht erweiterbaren Dialog lassen sich Modell und ACT-R-Modell vor dem Simulationsstart parametrieren.

Die Oberfläche ist in Tcl/Tk (bzw. der Erweiterung Tix) programmiert. Das Programm hat einen Umfang von ca. 1300 LOC. Das Frontend ist über Message-Pipes bzw. alternativ über Sockets mit dem CommonLISP-Prozess der ACT-R-Simulation und der Luftraumsimulation gats verbunden. Kommandos zur Steuerung der Simulation (Anhalten, Fortsetzen, Beenden) erfolgen über das Senden von UNIX-Signalen. Die Oberfläche besteht aus einem *tabbed Widget*. Über die Reiter der Tabulatoransicht können Detailinformationen zu den abgedeckten Bereichen des Frontends abgerufen werden. Die einzelnen Tabs ermöglichen

- die grafische Konfiguration der vielfältigen ACT-R und MoFL-spezifischen Parameter (s. Abbildung 6-11),
- das Logging der Vorgänge während der Simulation (s. Abbildung 6-12),
- die Inspektion des aktuellen Zustands des Arbeitsgedächtnisses des kognitiven Modells (s. Abbildung 6-13) und
- die Visualisierung der räumlich analogen Anteile des Arbeitsgedächtnisses – des *pictures* (s. Abbildung 6-14).

6.5.1. Konfiguration

Der Dialog zur Konfiguration ist modular und leicht erweiterbar. Dazu werden Unter-Tabulatorfenster definiert. In der aktuellen Version der ACT-R Shell sind das zwei

Views zur Spezifikation von ACT-R Architekturparametern („ACT-R I“, „ACT-R II“), zwei Views zur Parametrierung der Ablaufumgebung („Traces“, „System“) und zwei Views zur Spezifikation modell-spezifischer Parameter („Communication“, „MoFL“).

```
(setf *mofl-register-t* 40)
(format *command-trace* "*mofl-register-t* set to 40~%")
;tcl .b.nbframe.conf.nb.nb.nbframe.m1.comt.border.frame.mofl-\\
            register-t.frame.entry delete 0 end
;tcl .b.nbframe.conf.nb.nb.nbframe.m1.comt.border.frame.mofl-\\
            register-t.frame.entry insert 0 40
```

Quelltext 6-9: CommonLISP/tcl-Code zur Parametrierung. Ausschnitt aus der Konfigurationsdatei customize.lsp

Die Konfigurationseinstellungen werden als Datei gespeichert. Als Repräsentationsformat wird CommonLISP-Code in die Datei `customize.lsp` geschrieben, der direkt von ACT-R bzw. dem CommonLISP-Interpreter, in dem ACT-R abläuft, gelesen und interpretiert wird. In den CommonLISP-Code sind Kommentare eingefügt, die tcl-Code enthalten, mit dem die Werte in die Oberfläche ACT-R Shell geladen und angezeigt werden.

In Quelltext 6-9 wird dieses Prinzip für einen Konfigurationsparameter gezeigt: Der Wert der globalen CommonLISP-Variablen `*mofl-register-t*` wird mit `setf` auf `40` gesetzt. Danach wird ein CommonLISP-Ausdruck mit `format` benutzt, um auf dem *Stream* `*command-trace*` zu Dokumentationszwecken auszugeben, dass der Wert auf `40` gesetzt wurde. Ein Kommentar, also Code, der nicht interpretiert wird, wird in CommonLISP mit dem Zeichen „`;`“ eingeleitet. Danach folgt ein Ausdruck, der das entsprechende Eingabefeld in der Oberfläche von Anfang (`0`) bis Ende (`end`) löscht (`delete`). Danach wird am Anfang (`0`) dieses Eingabefeldes der Wert `40` eingesetzt (`insert`).

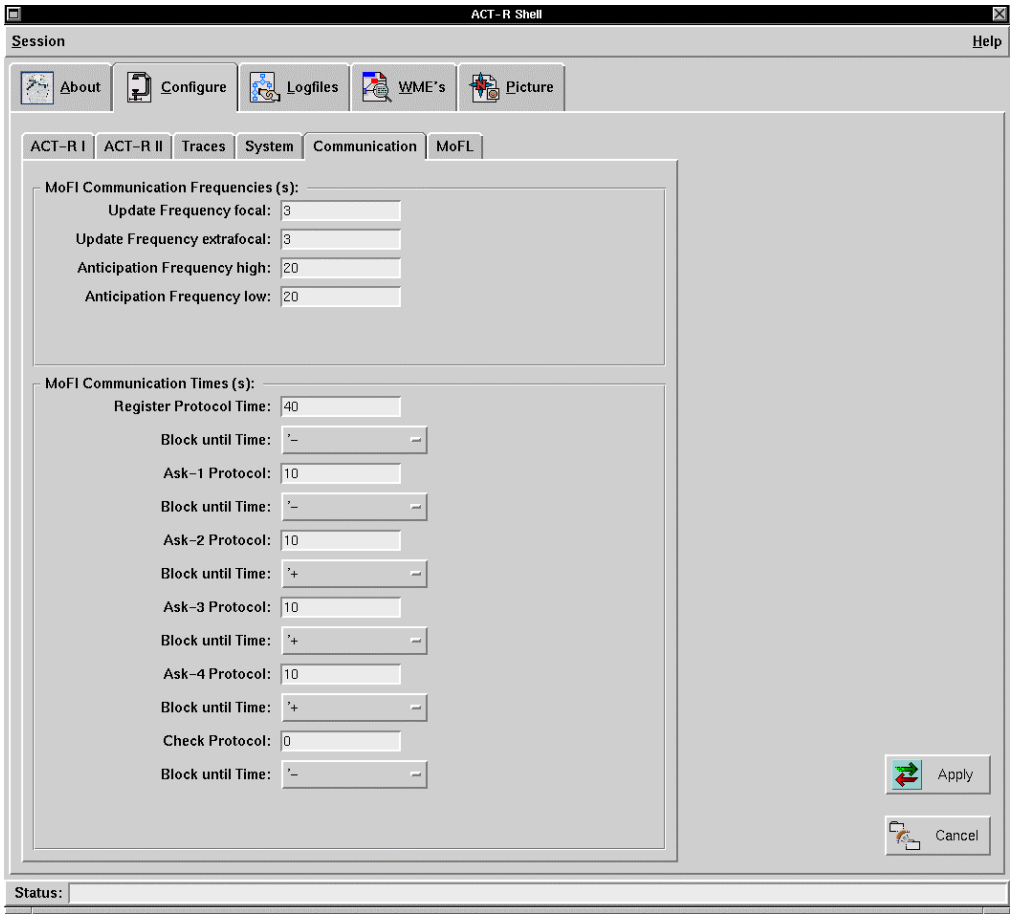


Abbildung 6-11: Konfiguration der Simulation. Hier Sicht auf die Kommunikationsparameter

Dieses Schema zur Repräsentation von Konfigurationsdaten mit den zwei verwendeten Programmiersprachen spart Programmieraufwand in den experimentellen Prototypen. Es kann verwendet werden, da die Syntax der beiden Sprachen hinreichend verschieden ist. Sollten in einem System noch mehr Programmiersprachen verwendet werden, ist dieser Weg nicht mehr gangbar. In einem Produktionssystem sollte eine standardisierte Sprache wie XML zur Repräsentation der Konfigurationsparameter verwendet werden.

Die Konfigurationsdialoge sind erweiterbar gehalten. Für jeden Eintrag wird in einer getrennten Konfigurationsdatei festgelegt, welcher Gruppe von Einträgen er angehört, welcher Erklärungstext im Fenster erscheinen soll und an welche LISP-Variable der eingegebene Wert gebunden werden soll. Jeder Eintrag kann von einem der drei Typen Zahl, Auswahl und freie Eingabe sein. Bei einer Zahl wird im Dialogfenster ein Skala-Widget, bei einer (eins-aus-n) Auswahl werden Radio-Buttons und bei einer freien Eingabe wird ein Entry-Widget benutzt.

6.5.2. Protokollierung

Während das Modell läuft, erzeugt ACT-R Ausgaben. Die Ausgaben werden je nach ihrer Art auf unterschiedliche *Streams* geschrieben. Die *Streams* werden für die ACT-R Shell auf Message-Pipes umgebogen und kontinuierlich von der Oberfläche

ausgelesen. Der Inhalt wird in den vier Sub-Views des Logfiles-Tabulator-Views ausgegeben (s. Abbildung 6-12).

Über den Kanal `logfiles/actr` werden Informationen zum Start und übergeordneten Ablauf des ACT-R-Kerns ausgegeben.

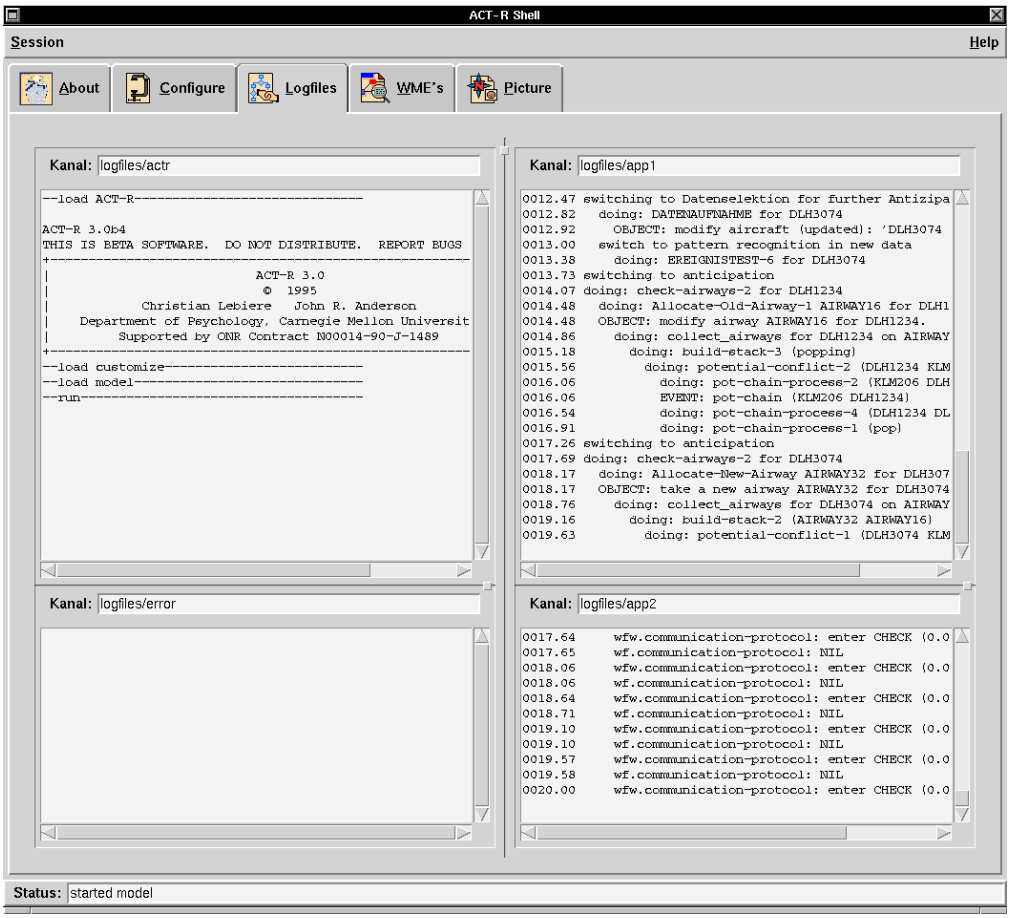


Abbildung 6-12: Anzeige der Logging-Kanäle

Über den Kanal `logfiles/error` werden Fehlermeldungen ausgegeben, die auf Systemebene von dem Simulationsprozess, dem CommonLISP-Interpreter oder von Hilfsprozessen erzeugt werden, deren Aufgabe die Integration der Teilsysteme ist. Anwendungsspezifische Fehlermeldungen, bei denen Fehlersituationen von der Anwendungslogik erkannt und darüber eine entsprechende Meldung ausgegeben wird, werden in einem der beiden anwendungsspezifischen Kanäle ausgegeben.

Die Kanäle `logfiles/app1` und `logfiles/app2` sind anwendungsspezifisch. Produktionen im ACT-R-Modell können entsprechende Ausgaben machen. Im Anwendungsbeispiel der Fallstudie „Modell der Fluglotsenleistungen MoFL“ wurden im ersten Kanal Ausgaben zur Produktionsauswahl und Parametrierung dargestellt. Im zweiten Kanal wurden Nachrichten zur Interprozesskommunikation zwischen kognitiven Modell und Aufgabensimulation und zur Abarbeitung des Kommunikationsprotokolls zwischen diesen beiden Systemen angezeigt.

6.5.3. Inhalt des Arbeitsgedächtnisses

Der Inhalt des Arbeitsgedächtnisses ist eine wichtige Größe bei der Arbeit mit kognitiven Modellen. Zugriff auf den aktuellen Zustand des Arbeitsgedächtnisses ist zum einen für das Debugging des Modells erforderlich (s. auch *situation awareness Monitor* der Firma SoarTech: Jones 1999). Zum anderen ist das Arbeitsgedächtnis auch für das Verständnis des konkret ablaufenden kognitiven Prozesses und daher für die Interpretation der Simulationsergebnisse erforderlich.

Abbildung 6-13 zeigt den Tabulator-View für den Zustand des Arbeitsgedächtnisses bzw. den deklarativen Speicher des ACT-R-Modells. Im linken Bereich gibt es zwei Sub-Views: Im oberen wird die Struktur des Arbeitsgedächtnisses angegeben. Man sieht die Chunk-Typen mit ihrer Vererbungstaxonomie in einer Baumdarstellung als Knoten. Die gespeicherten Chunks sind als konkrete Ausprägungen der Typen als Blätter des Baumes dargestellt.

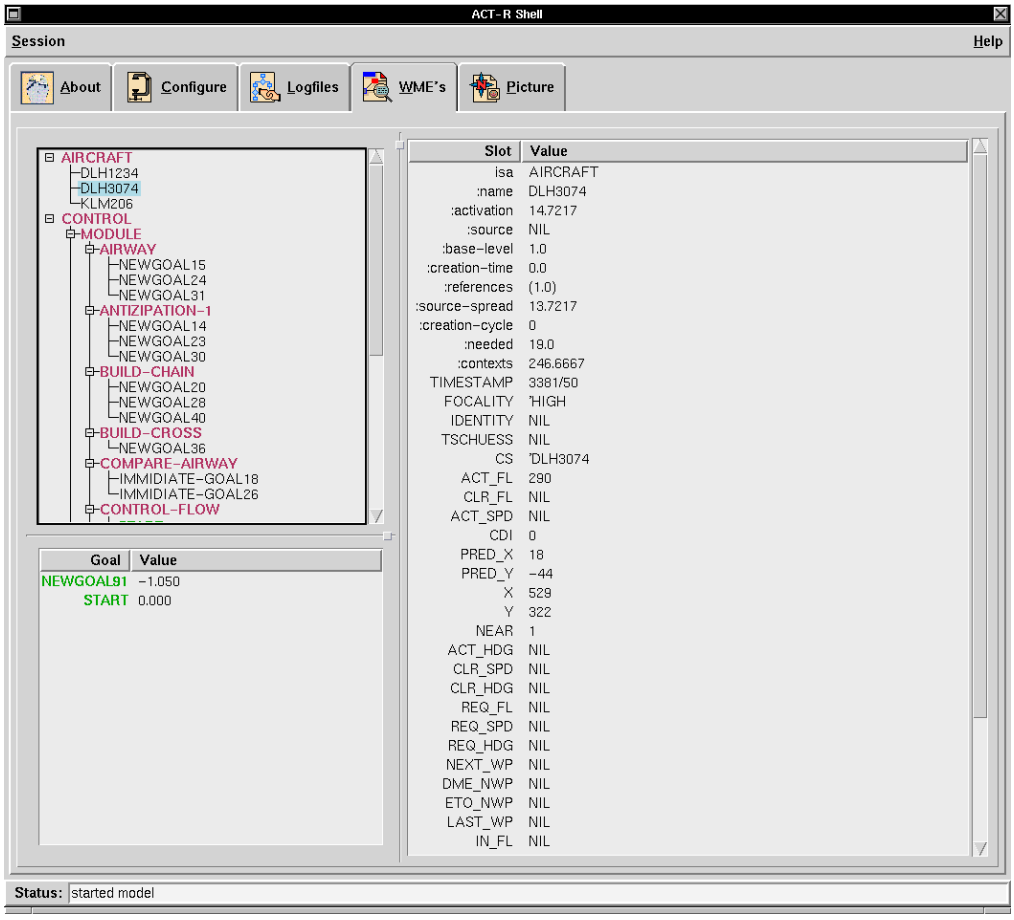


Abbildung 6-13: Sicht auf das deklarative Wissen in MoFL während eines Simulationslaufes

Dieser Sub-View dient gleichzeitig zur Navigation: Wenn ein Blatt mit der Maus aktiviert wird, wird im rechten Sub-View der ausgewählte Chunk in einer Detailsicht dargestellt. Man sieht alle Slots des Chunks mit ihren Werten und die sub-symbolischen Parameter des Chunks, die von der ACT-R Architektur angelegt und verwaltet werden. Im unteren View links wird der Goalstack angezeigt. Oben ist das

current-goal, darunter die übergeordneten Ziele. Diese Darstellung ist in der Softwaretechnik üblich für Stack-Datenstrukturen.

Im konkreten Beispiel der Anwendung in der Fallstudie „Modell der Fluglotsenleistungen MoFL“ gibt es neben der „digitalen“ Sicht in das Arbeitsgedächtnis noch eine visuelle Sicht auf die räumlich analogen Teile des *pictures*. Mit einer LISP-Funktion können Grafiken im *picture*-View erzeugt oder verändert werden.

Erkannte Luftwege, die Position von Luftfahrzeugen und deren prädizierte Wege auf den Luftstraßen werden visualisiert (s. Abbildung 6-14). Die Grenzen des kontrollierten Sektors werden nicht im mentalen Modell von MoFL in der visualisierten Form repräsentiert. Sie werden in dieser Ansicht nur als Hilfsmittel zur Interpretation des simulierten *pictures* dargestellt.

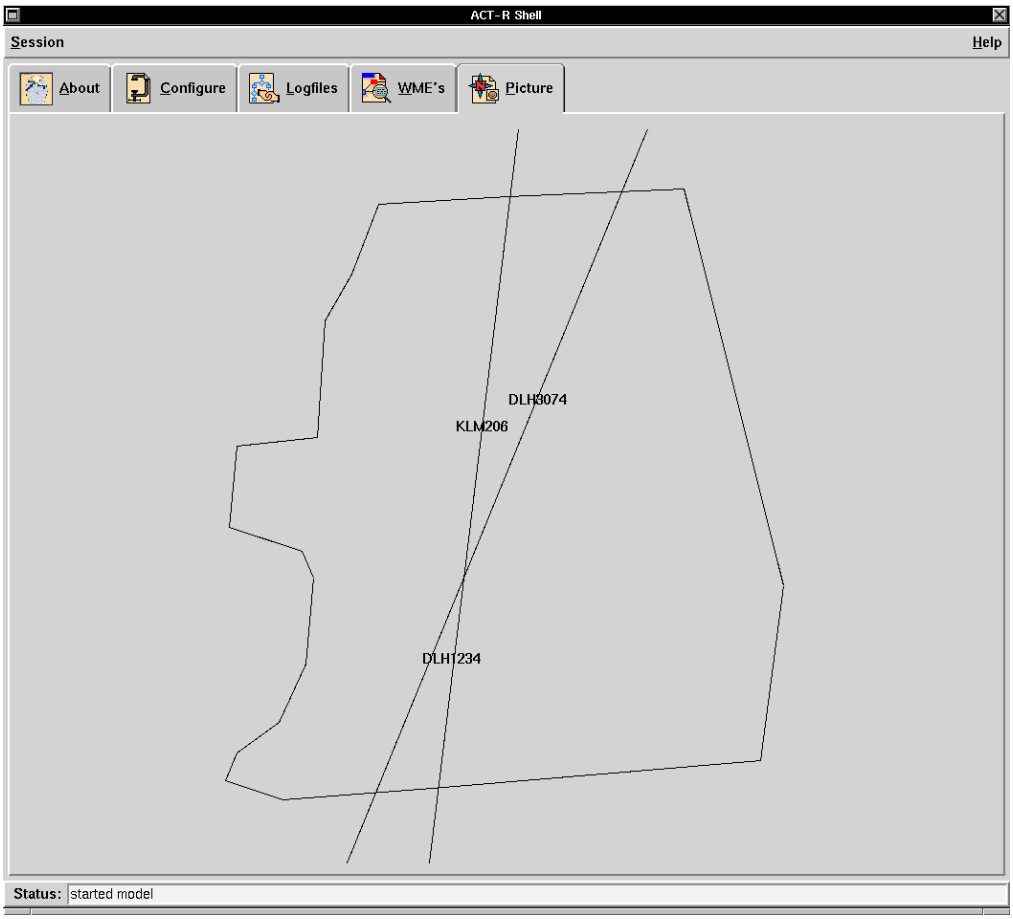


Abbildung 6-14: Anzeige des räumlich analogen Teils („picture“) des Arbeitsgedächtnisses

6.5.4. Abgrenzung

Seit der Entwicklung der ACT-R Shell sind einige neue integrierte Entwicklungumgebungen für die kognitive Modellierung erschienen. Im Bereich der Modellierung mit Soar gibt es neben dem SoarTech-internen *situation awareness Monitor* (Jones 1999) und anderen internen Werkzeugen den frei verfügbaren Soar-Debugger, der eine einfache Ablaufumgebung für Soar-Modelle darstellt. In der Soar-

Community sind Soar-Modelle tendenziell eigenständige Programme, die mit einer für ihren Zweck angepassten grafischen Benutzungsoberfläche ausgestattet werden. Dazu können als Standardschnittstellen Tcl/Tk-Bindings in Soar verwendet werden.

COGENT (Cooper & Fox 1998) ist eine weitere geschlossene Entwicklungsumgebung für Prolog-basierte kognitive Modelle (s. 3.2.2, S. 45). COGENT beinhaltet eine integrierte Entwicklungsumgebung, mit der Modellstrukturen erstellt und parametriert werden können. Es gibt auch einen Editor für Produktionsregeln. Als Ablaufumgebung unterstützt das System die Verwendung unterschiedlicher integrierter Protokoll- und Auswertungsmittel. COGENT ist jedoch sehr ungeeignet, um kognitive Modelle an externe Prozesse und Aufgabensimulationen anzuschließen.

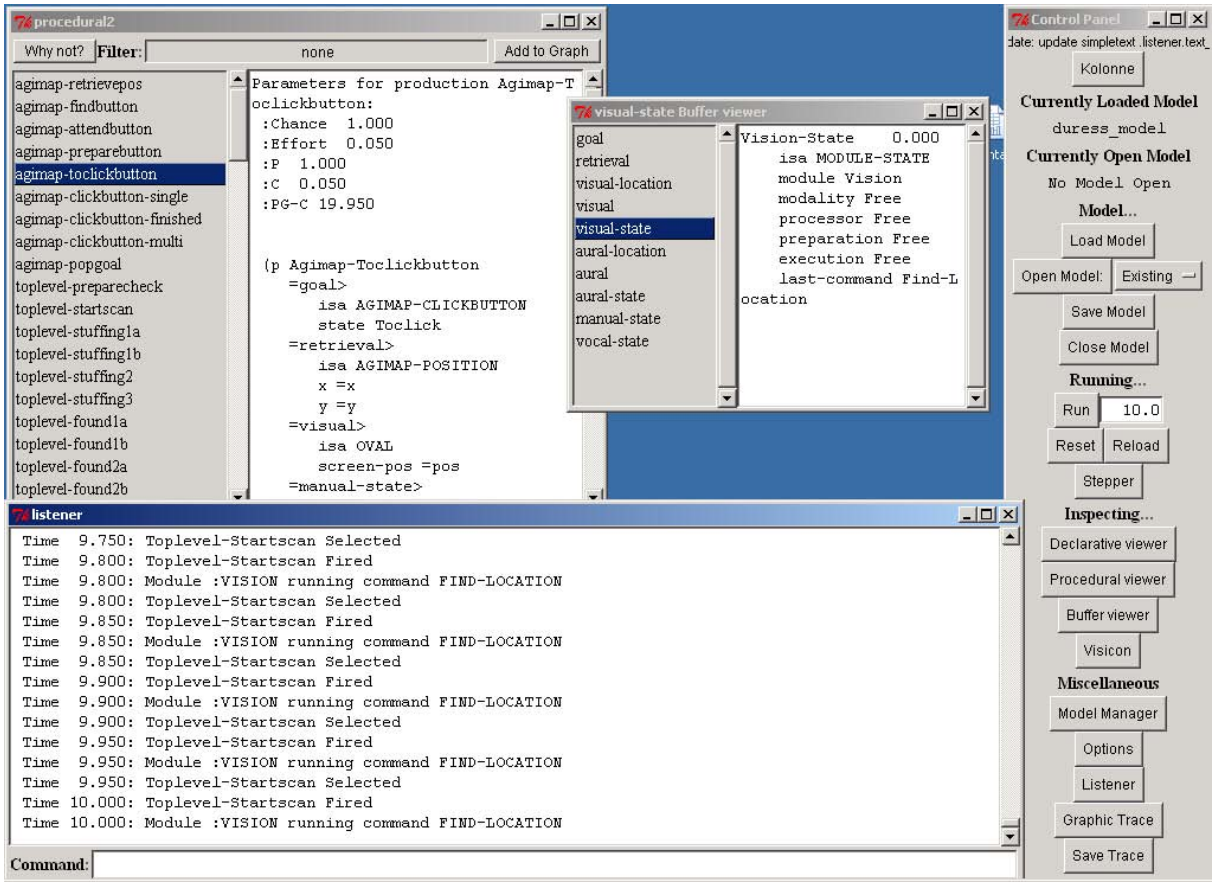


Abbildung 6-15: Screenshot der mitgelieferten Umgebung zu ACT-R/PM Version 5 (mit eigener Erweiterung „Kolonne“ als Button oben rechts)

In der ACT-R-Welt gibt es zwei Systeme. jACT-R ist eine experimentelle und nicht breit gepflegte Portierung von ACT-R von CommonLISP nach Java als Ablauf- bzw. Implementierungsumgebung (Harrison 2002). jACT-R beinhaltet eine grafische Oberfläche zur Formulierung von ACT-R-Produktionen und Chunk-Typen. Es handelt sich um einen reinen Modelleditor. Als Ablaufumgebung unterstützt jACT-R lediglich den Start des Modells und die Protokollierung des Standard-Ausgabestreams. Eine

Anbindung externer Prozesse ist aufgrund der Implementierungsplattform Java denkbar, aber in der aktuellen Version nicht unterstützt oder vorgesehen.

Die zweite ACT-R-Entwicklungsumgebung ist Teil des offiziellen ACT-R-Releases von der *Carnegie Mellon University* (CMU) und kommt von derselben Gruppe wie die ACT-R Kernkomponenten. ACT-R/PM wird seit Version 5 mit einer eigenen grafischen Umgebung (s. Abbildung 6-15) ausgeliefert, die sich ähnlicher Mechanismen wie die hier beschriebene ACT-R Shell bedient: Die Oberfläche ist in Tcl/Tk programmiert, die Interaktion mit dem CommonLISP-Prozess des ACT-R/PM-Kerns erfolgt über Pipes und wird auf ACT-R-Seite über Hooks bedient. Es steht ein ähnlicher Funktionsumfang zur Verfügung.

Das System bindet aber zusätzlich eine Visualisierung der Produktionskonfiguration und -anwendung und des neuen PM-Moduls zur Interaktion mit grafischen Oberflächen ein. Außerdem gibt es Unterstützung für das in dieser Version von ACT-R neu hinzugekommene Konzept der Buffer. Das System ist unter Windows und MacOS ablauffähig, ACT-R-Modelle müssen nicht angepasst werden, um in dieser Umgebung ablauffähig zu sein.

6.5.5. Einsatz in Fallstudien und Bewertung

Die ACT-R Shell wurde für die Entwicklung von MoFL eingesetzt. Sie ist der erste Vorläufer einer integrierten Entwicklungsumgebung (IDE) für die kognitive Modellierung mit ACT-R. Die Oberfläche ist als generische ACT-R Shell ausgelegt, in der beliebige ACT-R-Modellsimulationen ablaufen können. Zur Anpassung der generischen Shell können die Konfigurationsdialoge geändert und erweitert werden. Die beiden anwendungsspezifischen Kanäle `logfiles/app1` und `logfiles/app2` können über zwei vordefinierte Streams vom ACT-R-Modell aus mit der `!output!`-Direktive und der `format`-Funktion angesprochen werden. Zur Visualisierung räumlich analoger Inhalte im Arbeitsgedächtnis stehen LISP-Funktionen zur Verfügung, die in das Programm des kognitiven Modells eingebunden werden können. Zur Anzeige des Arbeitsgedächtnis-Inhalts in der Oberfläche muss das ACT-R-Modell manipuliert werden. Dazu wird eine entsprechende Funktion an den `Production-Cycle-Hook` von ACT-R angehängt, die dann zwischen jeder Produktionsanwendung ausgeführt wird. Der Anpassungsaufwand, um ein beliebiges ACT-R-Modell in dieser Oberfläche ablaufen zu lassen, ist gering.

Nur die ACT-R-Entwicklungsumgebung von der CMU ist sinnvoll mit der ACT-R Shell vergleichbar. Die neue ACT-R 5 Umgebung ist ein Standardwerkzeug, das nicht zur Anpassung gedacht ist. Es unterstützt in der Entwicklung durch Debugging Funktionen, bietet aber wenig Unterstützung bei der Interpretation und Dokumentation von Simulationsläufen. Es ist per se nicht für Simulationsverbünde mit mehreren Prozessen geeignet.

Die ACT-R Shell dient hingegen vorwiegend als Ablaufumgebung für ACT-R 3 Modelle in Simulationsverbünden aus mehreren unabhängigen Prozessen und zur

Unterstützung bei der Interpretation von Simulationsläufen. Erst durch dieses Werkzeug war es möglich, mit der Komplexität der kognitiven Modellierung für anwendungsorientierte Fragestellungen in *real-world*-Simulationsumgebungen umzugehen. Die ACT-R Shell ist darauf ausgelegt, für konkrete Anwendungen mit komplexen Systemumgebungen angepasst zu werden.

6.6. Agimap

Agimap ist zum Einen ein neues Framework zur Modellierung von Wahrnehmung und Handlung mit ACT-R/PM, zum Anderen beinhaltet Agimap eine neu entwickelte integrierte Entwicklungsumgebung zur Modellierung. Agimap besteht aus einer Abstraktionsschicht zur Kopplung von Ein-/Ausgabemechanismen von ACT-R/PM an externe Aufgabenumgebungen mit TCP/IP-Sockets. Da der Ansatz von Agimap mit den Compilationsansätzen (s. 3.4.1) vergleichbar ist und die Entwicklung mit Agimap Wahrnehmungsmodellierung als eine Art der Aufgabenmodellierung umsetzt, wird Agimap im Folgenden zur Analyseebene „Aufgabenmodellierung“ gezählt.

Die Möglichkeit der Abbildung der Interaktion von kognitivem Modell und Aufgabenumgebung ist eine kritische Anforderung für die Anwendung der kognitiven Modellierung für anwendungsorientierte Fragestellungen bei dynamischen Mensch-Maschine-Systemen. Ein neuer Ansatz zur Realisierung der dafür notwendigen Kopplung wurde bereits mit ACT-COM in Abschnitt 6.2 vorgestellt. Während ACT-COM aber Wahrnehmung und Handlung auf ein Kommunikationsprotokoll abbildet, ist es mit Agimap darüber hinaus möglich, auch ergonomische Fragestellungen von Wahrnehmbarkeit und Bedienbarkeit der Elemente der Benutzungsoberfläche zu beantworten. Der Modellierungsaufwand ist dadurch jedoch größer als beim ACT-COM-Ansatz.

Zur Simulation von Wahrnehmung und Handlung existiert in der kognitiven Architektur ACT-R das Subsystem PM (*perceptual/motor*: Byrne & Anderson 1998). PM beinhaltet Strategien und Eigenschaften der Wahrnehmung und der motorischen Handlung auf feingranularer Ebene. Eine ACT-R-eigene Schnittstelle zwischen dem PM-Subsystem und der Aufgabenumgebung ist AGI (ACT-R GUI Interface: ACT-R Research Group o.J.). Es erlaubt einem ACT-R-Modell die Elemente einer grafischen Benutzungsoberfläche (GUI) visuell wahrzunehmen und mit GUI-Widgets, den Ein-/Ausgabe-Elementen des GUIs (Button, Menü, Eingabefeld etc.), zu interagieren. Dadurch wird eine Bedienbarkeitsbewertung auf der ergonomischen Analyseebene der GUI-Elemente möglich.

6.6.1. Aufgabenmodellierung

Die Ebene, auf der ACT-R/PM mit AGI interagiert, ist jedoch sehr detailliert. Um einen Effekt zu erreichen, wie z.B. einen Button zu klicken oder um eine bestimmte Information an einem AGI-Widget zu finden, aufzunehmen und ACT-R zur Verfügung zu stellen, sind viele komplex interagierende Produktionen erforderlich. Diese niedrige Granularitätsebene erlaubt dann in Simulationen weitgehende Aussagen über die Bedieneffizienz einer Schnittstelle zu treffen.

Agimap ist eine Erweiterung, die ACT-R um eine höher aggregierte Schnittstelle zu AGI erweitert und so die ACT-R/PM-Modellierung um Größenordnungen effizienter und handhabbarer macht. Die Erweiterung geschieht nach dem

Paradigma der Compilationsansätze (s. 3.4.1), indem die Aktionen auf feingranularer Ebene hinter einer vereinfachenden Schnittstelle gekapselt werden.

AGI ist für die Bedienung von GUIs mit einem Standardsatz elementarer Widgets ausgelegt. Entsprechend beziehen sich die Möglichkeiten zur Informationsaufnahme lediglich auf basale Informationstypen wie Texte, Farben und Ziffern. Benutzungsschnittstellen von Mensch-Maschine-Systemen arbeiten aber zur Erleichterung der Interpretation durch die Bediener auch mit komplexeren und u.a. auch analogen Anzeigen (z.B. Tachometern oder Füllstandsanzeigen). Die von Agimap für die Interaktion über AGI zur Verfügung gestellten komplexen Widgets basieren darauf, dass die aus der technischen Simulation kommenden Werte von einem *Mapper* auf eine passende Repräsentation in AGI transformiert werden. Die Abbildung erfolgt auf die textuellen Mittel, die AGI zur Verfügung stellt (Texte an beliebigen Positionen in unterschiedlichen Farben). Durch die Kombination dieser Darstellungselemente wird eine adäquate visuelle Repräsentation als zusammengesetztes Widget erreicht.

Zur Definition eines komplexen Widgets werden mehrere passende Mapper instanziiert und konfiguriert. Tabelle 6-4 zeigt die zur Verfügung stehenden basalen GUI-Elemente und die bisher definierten Mapper.

Tabelle 6-4: Die gegenwärtig in Agimap definierten GUI-Elemente und Mapper zum Aufbau komplexer Widgets

GUI-Elemente	`Button`	Anzeige eines Widgets zum Auslösen einer Aktion mit vorgegebener Position, Größe
	`Text`	Darstellung einer Zeichenkette mit vorgegebener Position und Farbe
Mapper	`MapText`	Darstellung des empfangenen Wertes an fester Position in fester Farbe dargestellt
	`MapText2Color`	Darstellung einer festen Zeichenkette in einer von zwei Farben abhängig vom empfangenen Wert
	`MapNumber2Pos`	Darstellung einer festen Zeichenkette an Ort abhängig vom empfangenen Wert

Durch deren Kombination kann z.B. eine analoge grafische Füllstandsanzeige mit farbigen Minimum- und Maximumalarmen in eine gleichwertige textuelle Darstellung umgesetzt werden (s. Abbildung 6-16). Dazu wird der Wert des Füllstandes in eine Position gemappt (`MapNumber2Pos`) und ein Text (Zeichenkette `---`) angezeigt. Außerdem werden Alarmmeldungen bei Über- bzw. Unterschreiten von

vorgegebenen Füllstandsgrenzen in farbige Zeichensymbole (>>) gemappt (`MapText2Color`).

Die Strategien, mit denen die Informationsaufnahme an der AGI-Repräsentation des GUI-Elementes aus der originalen Benutzungsoberfläche erfolgt, sind so gestaltet, dass der analoge Charakter der Anzeige berücksichtigt wird. So wird beispielsweise der Fülltrend durch Abfrage und räumlichen Vergleich der Positionen des Flüssigkeitsspiegels ermittelt. Die in der Aufgabenumgebung intern repräsentierten Werte werden dazu nicht herangezogen.

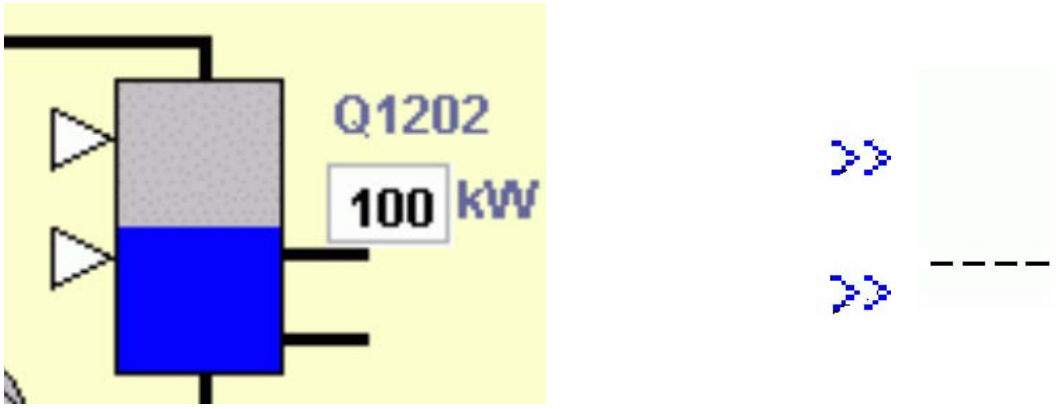


Abbildung 6-16: Repräsentation von grafisch analoger Anzeige im Original-GUI (links) als textuell analoge Anzeige in AGI (rechts)

6.6.2. Abgrenzung

Es gibt eine Reihe von Repräsentationssprachen für die Spezifikation von GUIs. Erste Ansätze stammen aus den *User Interface Management Systemen* (UIMS), konnten sich aber nicht durchsetzen. Aktuelle Systeme sind mit UseML, UIML, XAML, XUL, Glade XML und SwiXML sämtlich XML-basiert (s. Louridas 2007 für einen Überblick, zu UseML Reuther 2003). Sie eignen sich für den Einsatz in Agimap nicht, weil mit ihnen die Standard GUI-Elemente beschrieben werden. Ein Mapping auf die einfachere textuelle Repräsentation für AGI ist ad hoc nicht möglich.

6.6.3. Implementierung

Für jede Instanz eines solchen Widgets in AGI werden Produktionen und deklarative Strukturen in Form von Chunks automatisch erzeugt. Zur Benutzung der Strategien gibt es ein API, das über Zielstrukturen mit ACT-R-Produktionen verbunden ist. Im Agimap-Framework wird für jedes AGI-Element eine interne ID verwaltet, über die Chunks (Chunk-Slot `target`) und Produktionen dem GUI-Element zugeordnet werden. Alle Agimap-Elemente haben einen Typ und eine Position im AGI-Fenster (Chunk-Typ `agimap-position`).

Der Zugriff auf Werte (Simulation von Wahrnehmung) erfolgt über spezielle Ziele, die entsprechende Produktionsfolgen triggern. Quelltext 6-10 zeigt den Typ des Ziels, das für die Abfrage des Zustands eines Widgets benutzt wird. In `state` wird der

aktuelle Schritt in der Verarbeitungskette repräsentiert, `target` ist die ID des abzufragenden Widgets, in `result` wird am Ende der abgelesene Wert repräsentiert. Das kann `0/1`, `an/aus`, `oben/unten` sein.

```
(chunk-type agimap-checkState
                    state
                    target
                    result)
```

Quelltext 6-10: Definition des Ziels zur Abfrage des Zustands eines Widgets

Quelltext 6-11 zeigt, wie die Abfrage des Zustandes aus einer Produktion initiiert wird. Im Folgenden wird die dadurch initiierte Produktionsabfolge dargestellt:

1. Dabei ist `state` am Anfang `retrieve`.
2. Eine automatisch generierte Produktion aus dem Agimap-Framework matcht auf dieses Ziel und feuert. Das Ergebnis ist, dass die erwartete Bildschirmposition des Widgets aus dem Arbeitsgedächtnis abgerufen wird. Unter Umständen ist die Position noch nicht oder nicht mehr verfügbar, dann muss die Position im AGI-Fenster erst gesucht werden.
3. Im nächsten Schritt (`state` ist `find`) wird ein `visual-location` Chunk für PM neu erzeugt oder abgerufen.
4. Anschließend (`state` ist `attend`) wird über `=visual-state> modality free` der Zustand des visuellen Systems von ACT-R/PM geprüft und das Modul `visual` angesprochen.
5. Der nächste Schritt für `state` ist `read`. Hier wird über das `visual-location`- und das `retrieval`-Modul von ACT-R/PM der angezeigte Wert gelesen und interpretiert.
6. Schließlich wird der wahrgenommene Wert an den `result`-Slot gebunden, `state` auf `done` gesetzt und das ursprüngliche Ziel aus `topgoal` abgerufen und wieder zum aktuellen Ziel gemacht.

```
(P Produktionsname
    …
 ==>
   +goal>
     ISA agimap-checkState
     target =id
     state  retrieve                  ;; später: done
     topgoal =goal
     result nil)                      ;; später: on/off, ...
```

Quelltext 6-11: Initiierung des Zugriffs auf das GUI-Element `=id` über die Erzeugung des Ziels `agimap-checkState`

Diese Kette aus sechs Produktionsschritten wird automatisch durch das Agimap-Framework erzeugt. Die Schnittstelle für den Modellierer ist das `agimap-checkState`-Ziel, über das die Aktion initiiert und das wahrgenommene Ergebnis ausgelesen wird. Es gibt für alle Mapper und GUI-Element-Typen eine Reihe weiterer vorgefertigter Wahrnehmungsstrategien mit unterschiedlich komplexen vorgefertigten Produktionsketten (s. auch Quelltext 6-18).

Zur Definition der erforderlichen Chunk-Typen, Chunk-Namen und der zum Zugriff erforderlichen Prozeduren wird Metaprogrammierung benutzt. Dazu wurde eine Reihe von Makros in CommonLISP definiert und zur Laufzeitumgebung von ACT-R hinzugefügt. Quelltext 6-12 zeigt Ausschnitte aus dem Makro, das das Mapping von einem Wert `val` mit dem Element `var` (Agimap ID) realisiert. Dabei ist eine Reihe von Agimap-internen Variablen dargestellt, die dafür verwendet werden. Sie werden nach einer Agimap-eigenen Konvention über Metaprogrammierung erzeugt und benutzt.

```
(defmacro display-number-to-pos (var val) …
   … *AGIMAP-X-<VAR>* …
   … *AGIMAP-Y-<VAR>* …
   … *AGIMAP-XRange-<VAR>* …
   … *AGIMAP-YRange-<VAR>* …
   … *AGIMAP-SignText-<VAR>* …
   … *AGIMAP-MinValue-<VAR>* …
   … *AGIMAP-MaxValue-<VAR>* …
   … *AGIMAP-Color-<VAR>* …
… )
```

Quelltext 6-12: Ausschnitt aus der Definition des Makros für die Implementierung des Agimap-Mappers: MapNumber2Pos

6.6.4. Kopplung

Voraussetzung für die Verwendung von PM und AGI ist, dass die Aufgabenumgebung in Form eines Fensters innerhalb des ACT-R-Prozesses läuft. Für die kognitionswissenschaftliche Grundlagenforschung mit ACT-R/PM ist dieser Ansatz ausreichend, da die Aufgabenumgebungen i.A. wenig komplex sind und innerhalb der CommonLISP-Entwicklungsumgebung vom ACT-R Wirtssystem prototypisch implementiert werden können. Im anwendungsorientierten Kontext dieser Arbeit steht aber die Kopplung mit externen Prozessen, oft auf anderen Rechnern im Vordergrund. Zum einen, weil Aufgabenumgebungen in Form von Simulatoren oder zu testenden Systementwürfen vorliegen, zum anderen, weil das Timing der externen Aufgabenumgebung entscheidend ist, jedoch nicht ohne größeren Aufwand in einer programmtechnischen Nachbildung in einer anderen Programmiersprache (CommonLISP) und Ablaufumgebung (interpretiert, *Garbage Collection* indeterministisch) adäquat abgebildet werden kann. Deshalb stellt Agimap

auch eine Kopplungsschicht zur Verfügung, die AGI an externe Aufgabenumgebungen anschließt.

Voraussetzung für die Anwendbarkeit des in Agimap verfolgten Ansatzes zur Kopplung ist, dass die Aufgabenumgebung nach dem in der Softwaretechnik verbreiteten Observer Design-Pattern (Gamma et al. 1995) Änderungen von allen Prozessvariablen überträgt. In einem einfachen Protokoll werden dann Wertänderungen und Eingriffe per TCP/IP übertragen. Die jeweils aktuellen Werte werden als Chunk im Arbeitsgedächtnis von ACT-R repräsentiert.

Quelltext 6-13 zeigt die zentrale Funktion `update`, über die Agimap mit externen Aufgabenumgebungen lesend interagiert. Sie liest beim Vorliegen neuer Daten vom TCP/IP-Socket `*sock*` eine Zeile, die dann in die Bestandteile `r`, `l` (Variablenname und Wert) aufgespalten wird. Die weitere Verarbeitung erfolgt mit `display(r l)`, wobei zunächst der GUI-Element- oder Mapper-Typ festgestellt und dann zu einer spezialisierten Funktion verzweigt wird. Am Ende wird die Funktion `update` im ACT-R/PM-spezifischen Schedulingmechanismus für die nächste Ausführung in `*external-update-cycle*` Sekunden angemeldet. Der Schreibzugriff wird asynchron von diesem Zyklus durch Schreiben einer entsprechend formatierten Nachricht auf den Socket `*sock*` realisiert.

```
(defun update ()
  (if (listen *sock*)
    (prong
      (setf s (read-line *sock*))
      (setf r (evaluate "(first (list ~a))" s))
      (setf l (evaluate "(second (list ~a))" s))
      (evaluate "(display ~a ~a)" r l)))
  (pm-timed-event (+ (/ (pm-get-time) 1000) *external-update-cycle*)
            'update)))
```

Quelltext 6-13: Update-Routine des Agimap-Frameworks – am Ende wird die Funktion im PM-Scheduling-Mechanismus für die Ausführung in `*external-update-cycle*` Sek. angemeldet

6.6.5. Entwicklungsumgebung

Die Repräsentation des GUI-Modells mit den verwendeten Elementen und Mappern, und die Spezifikation der kopplungsspezifischen Daten erfolgt in einem grafischen Editor (s. Abbildung 6-17). Der Editor ist in Tcl/Tk programmiert. Er enthält keine Modellierungsspezifika sondern hat als einzige Aufgabe die Anzeige bzw. Änderung des GUI-Modells. Er ist deswegen sehr kompakt programmiert und hat einen Umfang von nur ca. 750 LOC Tcl/Tk-Code.

Der Editor benutzt XML zur Speicherung des GUI-Modells. Dazu wurde eine DTD erstellt, in der die Struktur des XML-Formates definiert ist (s. Anhang A). Diese XML-Repräsentation wird dann mit einem XSLT-Skript nach CommonLISP in das Metaprogrammierungsschema transformiert (*modell.agm LISP* in Abbildung 6-18).

Das XSLT-Skript bildet somit die modellierungsspezifischen Regeln ab. Der Umfang des XSLT-Skripts sind 120 LOC XSLT-Code. Die automatisch generierte Definition des Programmcodes zur Kopplung und der verwendeten Agimap-Elemente wird im ACT-R-Modell eingebunden (*model ACT-R* in Abbildung 6-18). Der ACT-R/PM-Interpreter beinhaltet die Agimap-Erweiterung (*agimap LISP* in Abbildung 6-18), die zum Ablauf des Modells erforderlich ist. Das Ergebnis des Simulationslaufs ist ein Protokoll (*Protokoll LOG* in Abbildung 6-18), das auf Benutzungsprobleme hin analysiert werden kann.

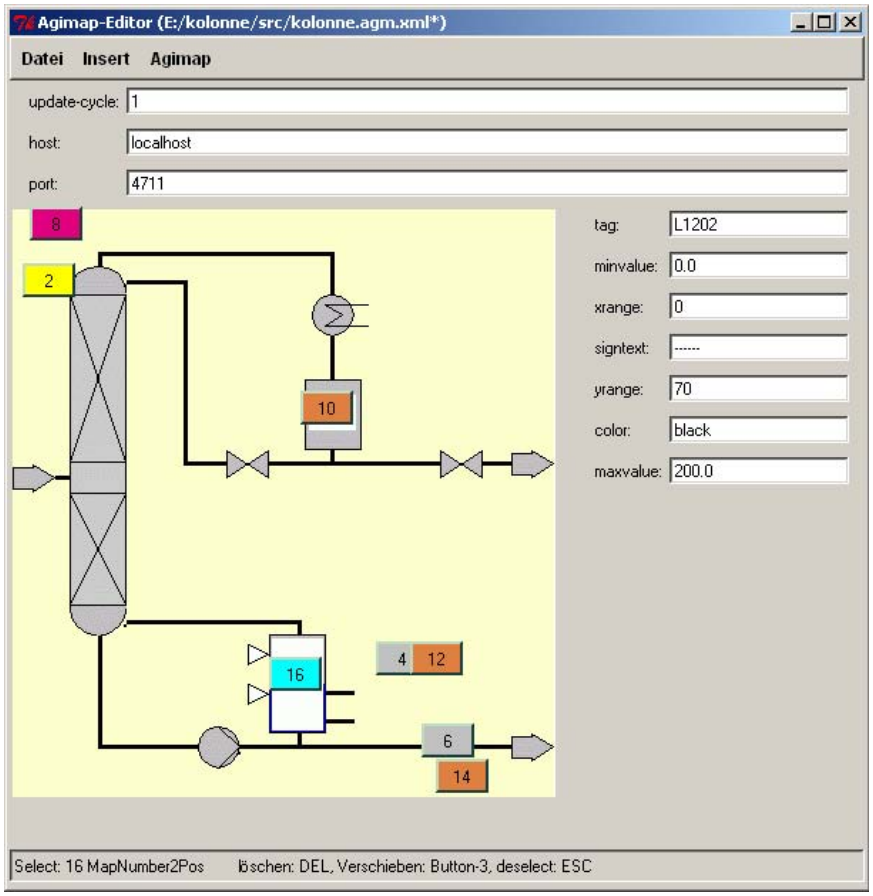


Abbildung 6-17: Grafischer Editor zum Erstellen des GUI-Modells und zur Konfiguration der Prozesskopplung

Durch die Verwendung von AGI als Basis und die Kopplung über Sockets bietet Agimap eine weitgehende Plattformunabhängigkeit. Die Simulation kann auf beliebigen Rechnerplattformen laufen. AGI ist sowohl unter MacOS, Linux als auch unter Windows lauffähig. Der Editor ist in Tcl/Tk, einer ebenfalls auf den meisten GUI-Plattformen verfügbaren Programmiersprache und Ablaufumgebung, portabel implementiert. Das Handling der Kopplung bei der Simulation wird durch Agimap weitgehend automatisiert.

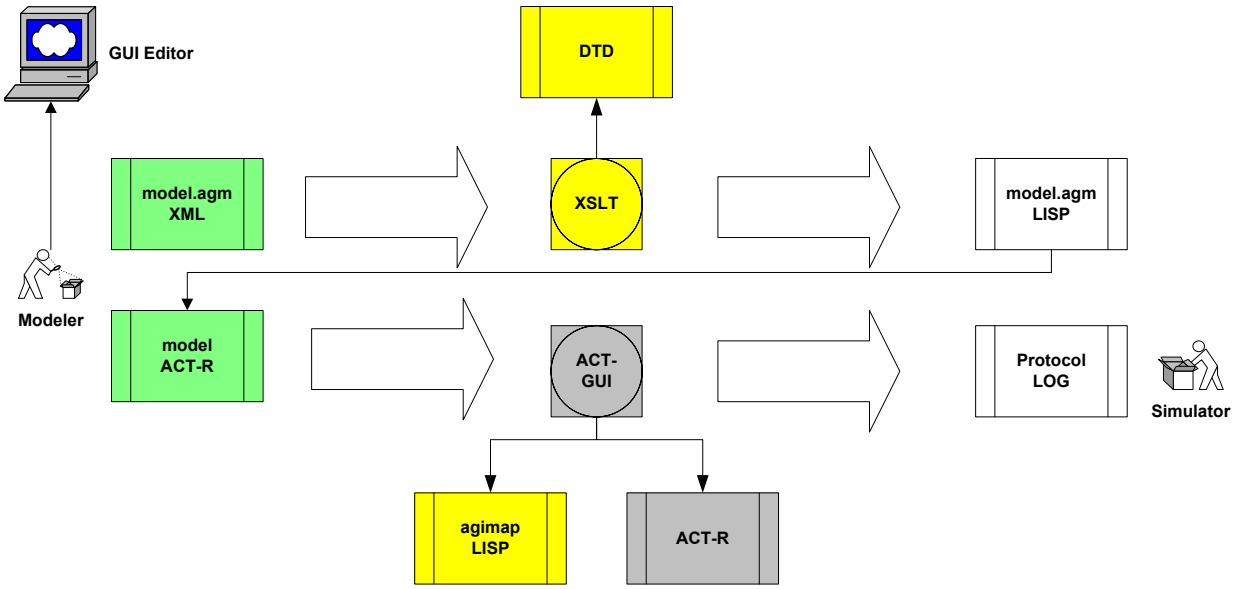


Abbildung 6-18: Werkzeugkette von Agimap

6.6.6. Einsatz in Fallstudien

Agimap wurde in den beiden Fallstudien DURESS (s. 5.2) und Rektifikationskolonne (s. 5.3) eingesetzt. In beiden Fällen wurde der gesamte Ein-/Ausgabe-spezifische Modellteil mit dem Agimap-Framework implementiert. Die Codegenerierung erfolgte komplett mit Hilfe des Editors und der Verarbeitungskette von Agimap. Dadurch konnten diese softwaretechnisch anspruchsvollen Aufgaben mit geringem Aufwand umgesetzt werden.

Der Umfang des generierten Codes macht die Komplexität und die Arbeitsersparnis deutlich: Bei der Fallstudie Rektifikationskolonne wurden 350 LOC ACT-R/PM bzw. CommonLISP-Code erzeugt. Dafür wurden drei GUI-Elemente und sieben Mapper mit dem Editor spezifiziert.

DURESS hat mehr Interaktionskomponenten. Es wurden 1060 LOC ACT-R/PM bzw. CommonLISP-Code automatisch aus 28 GUI-Elementen und 46 Mappern erzeugt. Die Spezifikation des GUIs mit dem Editor war auch in diesem deutlich aufwändigeren Fall unproblematisch und konnte innerhalb weniger Stunden aus Screenshots des DURESS-GUIs aufgebaut werden.

Zur Anbindung des Frameworks an die Aufgabenumgebung kann für jede relevante Prozessvariable ein Agimap-Mapper instanziiert werden, dessen Aufgabe es ist, bei einer Änderung „seines“ Wertes diesen Wert transformiert in AGI darzustellen, wodurch der Wert dem ACT-R-Modell zur Verfügung gestellt wird. Eine Erweiterung auf eine „1 zu n“-Zuordnung von einer Prozessvariablen zu mehreren Mappern wurde im Rahmen dieser Arbeit nicht realisiert, da sie für die Umsetzung der durchgeführten Fallstudien nicht erforderlich war. Inzwischen ist diese Erweiterung jedoch umgesetzt worden.

6.6.7. Bewertung

Agimap erlaubt dem Modellierer, von wahrnehmungsnahen Modellierungsaspekten und der technischen Anbindung an die Aufgabenumgebung zu abstrahieren, indem der grafische Editor benutzt wird, dessen Daten direkt in die Simulation eingebunden werden können. Durch die Verwendung von AGI wird die Aufgabensimulation jedoch auf nur ein GUI-Fenster beschränkt.

Durch die grafische Spezifikation und anschließende automatische Generierung von wahrnehmungs- und handlungsbezogenen Anteilen des ACT-R/PM-Modells wird die Modellprogrammierung wesentlich erleichtert. Bislang ist Agimap das einzige grafische Werkzeug für die softwaretechnische Anbindung von ACT-R/PM-Modellen an externe Aufgabenumgebungen. Die automatische Codegenerierung ist nicht nur effizienter als eine Handcodierung; durch die High-Level-Schnittstellen zu ACT-R/PM und die folgende Komplexitätsreduktion ist sie auch weniger fehleranfällig.

ACT-COM und Agimap sind für unterschiedliche Anwendungsbereiche einsetzbar und haben nebeneinander eine Daseinsberechtigung. ACT-COM ist mit ähnlichem Aufwand wie Agimap einsetzbar, jedoch kann Interaktion zwischen Modell und Aufgabensimulation nicht auf einer ergonomischen Analyseebene durchgeführt werden. Das Kommunikationsprotokoll zwischen Aufgabenumgebung und Modell ist dafür weniger stark eingeschränkt und kann leichter an existierende Schnittstellen angepasst werden. Es ist sogar eine Abbildung nicht computer-vermittelter Mensch-Maschine-Interaktion und von Mensch-Mensch Kommunikation möglich (z.B. Sprechfunk Lotse-Pilot in MoFL), was mit Agimap nicht möglich ist.

Agimap stellt eine Abstraktion der Modellierungselemente für die Nachbildung von Interaktionsvorgängen mit ACT-R/PM dar, indem der dafür erforderliche Modellanteil in einem Modul mit vordefinierten Mappern und GUI-Elementen gekapselt wird. Damit wird ein Konzept in ACT-R neu eingeführt, das analog zur Programm-Library mit definiertem API bei Programmiersprachen ist. Die Schnittstelle zum ACT-R Modul kann dabei effizient durch den Agimap Editor bedient werden. Die ACT-R/PM Strukturen und Verarbeitungsprozesse, die innerhalb des Moduls gekapselt sind, interagieren stark miteinander, die Schnittstelle für die Verwendung des Moduls wird über einen Chunk realisiert.

Wie beim Timer Modul wird das *information hiding* Prinzip aus der Softwaretechnik umgesetzt. Dadurch kann bei stabil bleibender Schnittstelle die Realisierung der Modul-internen Verarbeitung ausgetauscht werden, ohne dass bereits existierende kognitive Modelle, die Agimap nutzen, geändert werden müssten. Auch hier ist also Rückwärtskompatibilität gegeben.

6.7. Planex

Planex ist ein neues Framework zur Verarbeitung von KLM-GOMS-artigen Aufgabenbeschreibungen in ACT-R. Mit ihm ist es möglich, komplexe Aufgabenbeschreibungen ohne die Codierung von einzelnen Produktionen in ACT-R-Modellen umzusetzen. Mit Planex können vorliegende Aufgabenmodelle direkt in ACT-R-Modelle eingebunden werden.

Solch ein Framework ist geeignet für Aufgabenumgebungen mit SOP-artigen Aufgabenbeschreibungen, die beim Bediener nicht hochautomatisiert ablaufen, sondern als Handlungskette repräsentiert sind. Ein Zeichen für diese Art der Repräsentation ist die Fähigkeit, Handlungsstrategien verbalisieren zu können. Diese Art der Repräsentation kann zum Verwechseln und Vergessen von Schritten führen. Mit Planex können diese Eigenschaften modelliert und simuliert werden.

6.7.1. Entwurf und Implementierung

In Planex wird die SOP mit Chunks repräsentiert. Dabei wird das Schema von KLM-GOMS verwendet. Jede Methode wird als ein einzelner Chunk gespeichert. Der dazu verwendete Chunk-Typ `method` hat acht Slots. Im Slot `goal` wird das Ziel repräsentiert, das mit dieser Methode erreicht wird. In sieben weiteren Slots werden Operatoren gespeichert, die die Schritte der Methode sind. Dabei wird die Reihenfolge der Schritte durch die Slot-Namen gekennzeichnet (s. Quelltext 6-14). Ziel und Operatoren werden von anderen Produktionen referenziert. In Quelltext 6-14 ist auch ein Beispiel-Chunk für eine Methode gezeigt, die aus fünf Schritten besteht. Diese Methode hat den Namen `method_0564` und erfüllt das Ziel `increase_level_of_tankB`.

```
(chunk-type method goal step1 step2 step3 step4 step5 step6 step7)
(add-dm (method_0564 isa method goal increase_level_of_tankB
                     step1 valve1_increase
                     step2 pumpB_increase
                     step3 valve5_increase
                     step4 valve3_increase
                     step5 valve7_decrease))
(chunk-type goal state method target stepcounter operator …)
(add-dm (goal isa goal state checktasks))
(goal-focus goal)
```

Quelltext 6-14: Chunk-Typdefinitionen für Planex und Erzeugung von Beispiel-Chunks

Der zweite Chunk-Typ, der in Planex gebraucht wird, ist `goal`. Mit diesen Chunks werden jedoch nicht die Ziele von Planex-Methoden repräsentiert, sondern goal ist das ACT-R-Verarbeitungsziel. Dieses Ziel bleibt während der ganzen Planex-Verarbeitung aktiv. Es hat fünf Slots `state`, `method`, `target`, `stepcounter` und `operator`, die für die interne Verwaltung der Methoden-Abarbeitung benötigt werden. Das Ziel benötigt initial als einzige Belegung `state` mit einem

anwendungsspezifischen Wert (z.B. `checktasks` s. Quelltext 6-14). Jedoch ist die weitere Struktur des Ziels anwendungsspezifisch, da Operatoren dort Werte des Zustandes, den sie elaborieren, speichern.

Während der Verarbeitung wird die Situation diagnostiziert und ein Planex-Ziel zur Ausführung ausgewählt. Dieses Ziel, das durch die Abarbeitung einer Planex-Methode erreicht werden soll, wird im Slot `target` des ACT-R-Ziels `goal` gespeichert. Zusätzlich muss `state` auf `retrieve_method` gesetzt werden.

Daraufhin beginnt die Verarbeitung innerhalb des Planex-Frameworks. Einige vordefinierte Produktionen feuern jetzt. Es folgt zunächst ein kurzer Überblick über die Verarbeitungskette: Mit den Produktionen wird eine passende Methode ausgewählt und die Schritte, die zu ihrer Abarbeitung erforderlich sind, werden ausgeführt. Die Schritte sind in Form von Operatoren gespeichert. Die Operatoren werden in Planex durch Produktionen realisiert.

Der erste Schritt in der Planex-Verarbeitung ist der Abruf einer passenden Methode. Da die Methoden als Chunks repräsentiert werden, muss zuerst der passende Chunk in den Retrieval-Buffer transferiert werden (Abruf aus dem Arbeitsgedächtnis). Dazu wird die Produktion `retrieve_method` verwendet. Sie setzt den Slot `state` danach auf den Wert `retrieve_method_store` (s. Quelltext 6-15).

```
(P retrieve_method
   =goal>
     isa goal
     state retrieve_method
     target =target
     method nil
  ==>
   =goal>
     state retrieve_method_store
   +retrieval>
     isa method
     goal =target)
```

Quelltext 6-15: Planex-Produktion zum Abruf der passenden Methode

Danach wird der Methoden-Chunk aus dem Retrieval-Buffer abgerufen und an den Slot `method` des Ziels gebunden (s. Quelltext 6-16). Dabei wird gleichzeitig der Slot `stepcounter` auf 1 gesetzt, wenn noch nicht begonnen wurde, die Methode abzuarbeiten (Fall `retrieve_method_store` – dann ist der Slot `stepcounter` noch unbelegt, also `nil`). Im anderen Fall (`retrieve_method_store*`) wird `stepcounter` nicht geändert.

```
(P retrieve_method_store
   =goal>
     isa goal
     state retrieve_method_store
     target =target
     method nil
     stepcounter nil
   =retrieval>
     isa method
     goal =target
  ==>
   =goal>
     state method_nextstep
     method =retrieval
     stepcounter 1)
```

```
(P retrieve_method_store*
   =goal>
     isa goal
     state retrieve_method_store
     target =target
     method nil
     - stepcounter nil
   =retrieval>
     isa method
     goal =target
  ==>
   =goal>
     state method_nextstep
     method =retrieval
   )
```

Quelltext 6-16: Planex-Methode zur Speicherung des Methoden-Chunks im Ziel

Im nächsten Schritt wird der erste Schritt der Methode abgerufen. In folgenden Durchläufen werden die darauf folgenden Schritte der Methode abgerufen. Der aktuelle Schritt in der Methode wird im Slot `stepcounter` (initial 1) des Ziels gespeichert. Dazu werden mehrere Produktionen benötigt (s. Quelltext 6-17). Die erste Produktion ist `select_operator_retrieve_method`. Mit ihr wird gegebenenfalls die aktuelle Methode, die auch im goal im Slot `method` referenziert wird, wieder verfügbar gemacht. Außerdem gibt es für jeden Schritt eine eigene Produktion, die nach dem Schema von `select_operator_6` aufgebaut ist. Dabei wird lediglich die Schrittnummer angepasst. Mit diesen Produktionen wird der aktuell anstehende Schritt an den Slot `operator` des ACT-R-Ziels gebunden und `state` auf `operator_selected` gesetzt.

```
(P select_operator_retrieve_method
   =goal>
     isa goal
     state method_nextstep
     method =method




  ==>
   +retrieval> =method)
```

```
(P select_operator_6
   =goal>
     isa goal
     state method_nextstep
     stepcounter 6
   =retrieval>
     isa method
     step6 =step
  ==>
   =goal>
     isa goal
     state operator_selected
     stepcounter 7
     operator =step)
```

Quelltext 6-17: Planex-Produktionen für Auswahl und Abruf des nächsten Operators

Nachdem der nächste Operator ausgewählt wurde, wird er mit einer spezifischen Produktion ausgeführt. Im Beispiel in Quelltext 6-18 ist eine Produktion dargestellt, die den Operator `pumpA_increase` mit Hilfe von Agimap (s. 6.6) implementiert. In dieser Produktion werden weitere Aktionen ausgelöst, indem ein neues Ziel erzeugt wird. Das neue Ziel enthält eine Referenz auf das alte Ziel, damit das alte Ziel nach Abarbeitung des neuen Ziels wieder zum aktuellen gemacht werden kann. `state` im alten Ziel auf `operator_engaged` gesetzt als Zeichen dafür, dass der Operator dann abgearbeitet ist.

```
(P initiate_action_pumpA_increase
   =goal>
     isa goal
     state operator_selected
     operator pumpA_increase
  ==>
   =goal>
     state operator_engaged
   +goal>
     isa agimap-clickButton
     state retrieve
     target incpumpa
     numberofclicks 5
     topgoal =goal)
```

Quelltext 6-18: Planex-Produktion zur Initiierung des Operators pumpA_increase

Nachdem der Operator ausgeführt wurde, muss der Ausgangszustand wieder hergestellt werden. An dieser Stelle gibt es zwei alternative Implementierungen von Planex (s. Quelltext 6-19). `degage_actionA` bewirkt, dass der Methoden-Chunk neu abgerufen werden muss, da er nicht mehr zur Verfügung steht. Dadurch sind mehrere zusätzliche Produktionsanwendungen erforderlich. Bei `degage_actionB` wird auf diesen Zwischenschritt verzichtet. Der Abruf des nächsten Operators wird direkt vorbereitet.

```
(P degage_actionA                          (P degage_actionB
   =goal>                                     =goal>
     isa goal                                   isa goal
     state operator_engaged                     state operator_engaged
  ==>                                        ==>
   -visual-location>                          -visual-location>
   =goal>                                     =goal>
     state retrieve_method                      state method_nextstep)
     method nil)
```

Quelltext 6-19: Planex-Produktion zum Aufräumen des Zustandes nach Operatoranwendung

Nicht alle Methoden haben genau sieben Schritte. Deshalb gibt es Produktionen nach dem Muster von `select_operator_6_stop` (s. Quelltext 6-20), die in dem Fall feuern, wenn der nächste auszuführende Schritt in der aktuellen Methode nicht mit einem Operator belegt ist (`nil`). Dann wird das ACT-R-Ziel entsprechend geändert, d.h. `state` wird auf `method_finished`, `method`, `stepcounter` und `operator` werden auf `nil` gesetzt.

```
(P select_operator_6_stop
   =goal>
     isa goal
     state method_nextstep
     stepcounter 6
   =retrieval>
     isa method
     step6 nil
  ==>
   =goal>
     isa goal
     state method_finished
     method nil
     stepcounter nil
     operator nil)
```

Quelltext 6-20: Planex-Produktion zur Beendigung der Methode nach dem fünften Schritt

Während dieser Schritt auf Planex-Ebene ablief, wird im letzten Schritt auf Aufgabenebene ein neuer Ausgangszustand hergestellt (s. Quelltext 6-21). Insbesondere werden hier die aufgabenspezifischen Slots des ACT-R-Ziels ausgewertet oder zurückgesetzt.

```
(P degage_method
   =goal>
     isa goal
     state method_finished
  ==>
   -visual-location>
   =goal>
     state checktasks
     method nil
     target nil
     … )
```

Quelltext 6-21: Planex-Produktion zum Aufräumen des Zustandes nach Methodenabarbeitung

Abbildung 6-19 zeigt den oben geschilderten prinzipiellen Ablauf der Planex-Verarbeitung in Anlehnung an die Notation zur Visualisierung von ACT-R (s. 6.4).

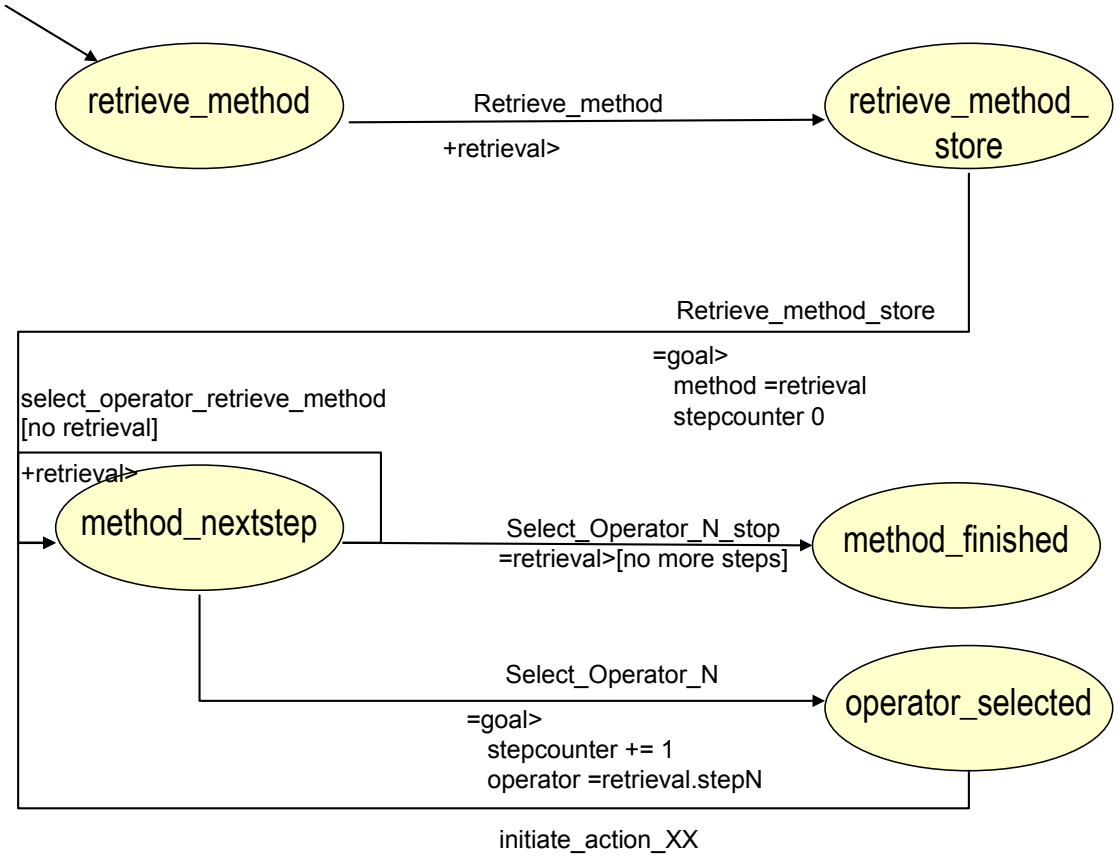


Abbildung 6-19: Visualisierung des prinzipiellen Ablaufs bei der Planex-Abarbeitung in Anlehnung an die Notation zur Visualisierung von ACT-R

6.7.2. Abgrenzung

Planex ist vergleichbar mit anderen Ansätzen (z.B. Schoppek & Boehm-Davis 2004). Es ist aber gegenwärtig das einzige Framework, das ohne Goalstackstrukturen auskommt und somit zur aktuellen ACT-R-Architektur passt.

Planex ist auch vergleichbar mit Systemen aus dem Bereich der Compilationsansätze (s. 3.4.1). Insbesondere hat GOMS2ACT (Amant et al. 2007) einen ähnlichen Sprachumfang. Planex unterscheidet sich von diesen Ansätzen aber dadurch, dass damit Pläne abgearbeitet werden können. GOMS2ACT und ähnliche Systeme unterstützen im Gegensatz dazu die Kapselung von Wahrnehmungs- und Handlungspatterns.

Planex ist ein Ansatz, um den perfekten Zugriff, der bei Produktionen immer besteht, einzuschränken. Jedoch ist es eine Designentscheidung, welche Elemente des Frameworks als Chunk und welche als Produktion modelliert werden. Methoden werden in Planex als Chunk repräsentiert. Durch unterschiedliche Aktivierung der auf dasselbe Ziel passenden Methoden-Chunks kann dadurch eine Auswahl der Handlungsstrategie auf subsymbolischer Ebene von ACT-R beeinflusst werden. Weiterhin sind auch das Verwechseln sowie das Vergessen von Chunks möglich, wodurch zusätzliche Produktionen benötigt werden, um einen Chunk wieder abrufbar zu machen. Dass Operatoren hingegen in Planex als Produktionen umgesetzt werden, ermöglicht hier eine Einbindung komplexerer Aktionen. Damit ist das Framework so flexibel, dass die Operatoren nicht alle dieselbe Granularität besitzen müssen wie bei GOMS2ACT.

Mit dieser Aufteilung im Planex-Ansatz ist es jedoch nicht möglich, komplexeren Methoden mit Fallunterscheidungen wie in NGOMSL auszudrücken. Stattdessen müssen Entscheidungen und Diagnoseprozesse, die zur Planex-Zielauswahl führen, in Planex-externen Produktionen vorgenommen werden.

Planex beinhaltet eine syntaktische Einschränkung der Anzahl der Methodenschritte. Diese Einschränkung ist mit unterschiedlichen Werten aus anderen Systemen bekannt (z.B. Schoppek & Boehm-Davis 2004). Die Begründung für diese Einschränkung liegt in der begrenzten Kapazität des Arbeitsgedächtnisses. Es gibt zwar keine formale Beschränkung der Anzahl der Slots eines Chunks in ACT-R, jedoch scheint die Faustregel, dass sieben ± zwei Chunks gleichzeitig verarbeitet werden können, anwendbar (z.B. Geisler 2006). Längere Methoden müssen über eine Verschaltung mehrerer Methoden abgebildet werden.

6.7.3. Planex und Lernen

Der Planex-Ansatz muss gegen *Instance Based Learning* (IBL) abgegrenzt werden. Beim IBL wird Lernen von Bedienstrategien durch die Speicherung von Chunks („Instanzen“) realisiert, in denen Problemkontext und eine bereits bekannte erfolgreiche Lösung (in Form einer Aktion und erforderlichen Parametern) repräsentiert werden. Während der Bedienung einer Aufgabenumgebung werden dadurch erfolgversprechende Eingriffe erlernt, die auch in der Zukunft angewandt werden können. Dazu wird dann der am besten passende gespeicherte Instanz-Chunk aus dem Arbeitsgedächtnis abgerufen, wobei ACT-R-Mechanismen wie *partial matching* (z.B. bei der Bedienung der Mikrowelt *sugar factory*: Taatgen &

Wallach 2002) oder *blending* (z.B. bei der Bedienung einer Wasseraufbereitungsanlagensimulation: Gonzales et al. 2003) verwendet werden, um flexibles Verhalten zu erzeugen.

Die IBL-Strategie kann nur bei reaktiven Aufgabenumgebungen benutzt werden, denn Instanzen stellen lediglich Operatoren dar. Eine Abfolge von Eingriffen in Form von Methoden bzw. einer Sequenz von Operationen, wie sie bei komplexeren Aufgaben erforderlich ist, kann so nicht realisiert werden. Obwohl auch die Chunks, die in Planex Methoden repräsentieren, im Arbeitsgedächtnis gespeichert sind und über einen zu spezifizierenden Problemkontext abgerufen werden können, der in Planex ein GOMS-goal ist, können solche Chunks nicht als Instanzen bei der Bedienung beobachtet und gelernt werden, da sie das Resultat einer Sequenz von Bedienschritten sind.

Stattdessen kann der ACT-R Mechanismus *production compilation* zum Lernen verwendet werden (s. z.B. Taatgen & Lee 2003): Eine neue Produktion wird dabei generiert, sobald zwei Produktionen nacheinander gefeuert haben. Die neue Produktion hat dieselbe Funktion wie die Kombination der beiden Eingangsproduktionen. Dabei wird auch die Benutzung von Chunks durch die beiden Produktionen berücksichtigt. Die neue Produktion ist spezieller (kann also in weniger Fällen ausgeführt werden) als die beiden Regeln jeweils für sich genommen.

Im Fall des Planex-Frameworks ist dieser Lernmechanismus voll anwendbar. Neue Produktionen entstehen so aus der Sequenz: `retrieve_method`, `retrieve_method_store`, `method_nextstep`, `operator_selected`, „Operatorausführung“ usw. Dabei werden die Produktionen bei jeder Anwendung weiter aggregiert (es werden immer Zweier-Sequenzen verschmolzen), bis für jede Methode genau eine komplexe Produktion erlernt wurde, die schneller abgerufen und ausgeführt werden kann als über den ursprünglichen Planex-Ablauf.

6.7.4. Multitasking

Das Planex-Framework harmoniert gut mit denkbaren Multitasking-Frameworks (z.B. Behandlung asynchroner Kommunikationsereignisse in ACT-COM, Abschnitt 6.2). Obwohl es sich um eine feste Sequenz von Produktionsaufrufen handelt, mit denen eine Planex-Methode realisiert wird, kann der Ablauf durch ein externes Ereignis unterbrochen werden. Das Multitasking-Framework muss in diesem Fall einen neuen Chunk erzeugen oder einen existierenden abrufen und zum aktuellen Ziel machen. Das aktuelle Ziel muss entweder eine Referenz auf das bis dahin gültige Planex-Ziel beinhalten oder bei früheren ACT-R Versionen als neueres (Unter-) Ziel auf dem Goalstack abgelegt werden, so dass das ältere Planex-Ziel nach der Bearbeitung der asynchronen Erfordernis wieder aktiviert werden kann und das System so wieder im alten Zustand bzw. im Task der Abarbeitung der Planex-Methode ist.

Der Wechsel zurück zum vorherigen Task kann entweder durch ein *pop* des neuen Ziels erfolgen oder indem das im neuen Ziel referenzierte *goal* des vorherigen

Tasks wieder aktiviert wird. Die letztere Möglichkeit ist zu bevorzugen (in aktuellen Versionen von ACT-R wurde das Goalstack-Konstrukt fallengelassen), da hierbei die Aktivierung des Planex-Ziel-Chunks berücksichtigt werden kann. Ist sie zu niedrig, weil die Bearbeitung des asynchronen Tasks zu viel Aufmerksamkeit erfordert hatte, kann die Abarbeitung der Planex-Methode nicht fortgesetzt werden, und die Methode muss erst neu abgerufen werden und somit deren Abarbeitung von vorne begonnen werden. Ein solches Schema wird auch in MT-GOMS (s. nächster Abschnitt) verwendet.

Daneben können vorgeplante Punkte zum Task-Switch zwischen komplett abgearbeiteten Planex-Methoden liegen. Dadurch entsteht – mit derselben Begründung wie dort – ein ähnliches Multitaskingkonzept wie beim Modell MoFL (s. 5.1).

6.7.5. Einsatz in Fallstudie und Bewertung

Planex wurde in der Fallstudie DURESS (s. 5.2) eingesetzt. Für diese Fallstudie lag bereits eine Sammlung von Kontrollstrategien in Form einer Planbibliothek vor. Die Planbibliothek bestand aus 770 Plänen, die in der logischen Programmiersprache Prolog codiert waren. Mit einem einfachen Perl-Skript konnte daraus der Planex-spezifische Code des Modells mit wenig Aufwand generiert werden. Dabei wurden die vorliegenden Pläne auf 12 Planex-Ziele und 22 Operatoren abgebildet. Mit dem Planex-Framework konnte das Kontrollwissen, das für die Bedienung der DURESS-Aufgabenumgebung erforderlich ist, leicht im ACT-R-Modell eingebettet werden. Die Parametrierung von Chunks hat durch den ACT-R Resolutionsmechanismus ebenso wie die Komplexität der Methoden-Chunks (Anzahl Schritte) Einfluss auf die Auswahl der Kontrollstrategien. Es konnte somit ein besseres Modell erstellt werden als es durch Repräsentation der Pläne als Produktionen möglich gewesen wäre.

Planex stellt ein Framework innerhalb von ACT-R dar, um KLM-artige Pläne zu repräsentieren und auszuführen. Das Planex-Framework ist generisch angelegt. Durch Austausch des KLM-Modells kann ein anderes Verhalten erreicht werden. Planex ist zudem in hohem Maß wiederverwendbar. Dazu müssen nur die wesentlichen Produktionen des Frameworks als externes Modul eingebunden werden. Die Schnittstelle zum Framework besteht aus einem Chunk dessen Typ ebenfalls im Modul definiert wird. Durch die explizite deklarative Repräsentation prozeduralen Wissens als KLM-Modell wird die Kommunizierbarkeit von Bedienermodellen unterstützt.

6.8. MT-GOMS

Zur prospektiven Analyse und Bewertung neuer Dialogsysteme im Kraftfahrzeug wurde die Methode MT-GOMS entwickelt. MT-GOMS, kurz für Multitasking GOMS, ist eine neue methodische Weiterentwicklung von GOMS (s. 3.3.5). Die Methode hat den Zweck, die Bedienbarkeit und das Ablenkungspotenzial von *In-Vehicle Information Systems* (IVIS) zu bewerten. Beispiele für IVIS sind Einzelsysteme in der Mittelkonsole wie Autoradio und Telefon, aber auch integrierte Fahrzeugbediensysteme wie iDrive von BMW. Die Bedienung von IVIS wird als Nebenaufgabe angesehen. Insbesondere in kritischen Fahrsituationen darf diese Nebenaufgabe nicht zu stark oder zu lange von der Hauptaufgabe, der Kraftfahrzeugführung, ablenken. Im Gegensatz zur heute üblichen experimentellen Untersuchung mit Versuchsfahrten oder im Fahrsimulator ist die Anwendung von MT-GOMS deutlich weniger aufwändig und lässt sich in bereits frühen Phasen der Produktentwicklung anhand von Spezifikationen und mit hoher Frequenz einsetzen.

6.8.1. Konzeption der Methode

MT-GOMS stellt Bedienermodellierungstechniken zur modellgestützten Bewertung von IVIS im Kraftfahrzeug zur Verfügung. Das Verfahren ist auch auf interaktive Systeme in anderen Verkehrsträgern und auf zukünftige Zulassungsprüfungen für Dialogsysteme im Kraftfahrzeug übertragbar. Die Methode kann prinzipiell auch auf andere Multitasking-Aufgabenumgebungen außerhalb der Kraftfahrzeugführung übertragen werden. Zielstellung der hier beschriebenen Entwicklung ist, Aussagen über die Bedienbarkeit neuer Dialogsysteme effizient und bereits in frühen Phasen der Systementwicklung machen zu können. Die Bedienbarkeit wird in Bezug gesetzt zu einer Auswahl kritischer Fahrsituationen mit unterschiedlichen Anforderungen an die Aufmerksamkeit des Fahrers. Um dies zu erreichen, wird die Modellierungsmethode GOMS erweitert.

Die Bedienung von IVIS bedingt eine Abwendung der Aufmerksamkeit von der Fahraufgabe (Hauptaufgabe) zur Gerätesteuerung (Nebenaufgabe). Während besonders anspruchsvoller Fahrmanöver oder bei hoher Komplexität der Benutzungsschnittstelle des Dienstes könnte es zu einer zu großen Ablenkung kommen. Daher müssen die zur Verfügung stehenden kognitiven Ressourcen effektiv zwischen Haupt- und Nebenaufgabe verteilt werden.

Die Verteilung bedingt eine Unterbrechung von Handlungen in Primär- und Sekundäraufgabe. Je nach Art der Handlung ist eine Aktion unterbrechbar oder nicht. Ein weiteres handlungsspezifisches Merkmal ist die Rückkehr zur ursprünglichen Tätigkeit nach einer Unterbrechung: Handlungen können entweder am Abbruchpunkt fortgesetzt werden, die Rückkehr zu einem früheren Einstiegspunkt einer Teilhandlung kann notwendig sein, oder es sind sogar zusätzliche Teilhandlungen einzufügen. Die Hauptaufgabe ist fest vorgegeben und bestimmt den Rahmen für die

Abarbeitung der Nebenaufgabe. IVIS müssen also so gestaltet werden, dass sowohl die Unterbrechbarkeit von erforderlichen Bedienhandlungen jederzeit möglich ist, als auch dass die Bedienung der Dienste durch die Unterbrechungen nicht wesentlich behindert wird.

Das Verfahren MT-GOMS basiert auf einer Erweiterung der existierenden Modellierungs- und Simulationstechnik NGOMSL. Zusätzlich werden Elemente von CPM-GOMS benutzt. Eine Einschränkung der GOMS-Methoden ist die Festlegung auf Einzelaufgaben: In Multitasking-Umgebungen müssen jedoch auch der Aufgabenwechsel und die Interaktion zwischen parallelen Aufgaben modelliert werden. Da das sehr aufwändig ist, wurden GOMS-Methoden selten in Multitasking-Umgebungen eingesetzt. Ein Einsatz für die Bewertung von IVIS erfordert die Berücksichtigung von Multitasking.

Die Ausgangsidee von MT-GOMS ist, die Hauptaufgabe nur prototypisch zu modellieren. Für einige ausgewählte Fahrsituationen wird dazu ein Ressourcenprofil experimentell erhoben, das die Aufmerksamkeitserfordernisse der Hauptaufgabe repräsentiert. Dabei werden zeitliche Ausprägung und die Art der benötigten Aufmerksamkeitsressourcen charakterisiert. Abbildung 6-20 zeigt beispielhaft für eine angenommene prototypische Situation die Belegung der drei Ressourcen „kognitiv“, „visuell“, „rechte Hand“ in einem Profil über 20 Sekunden.

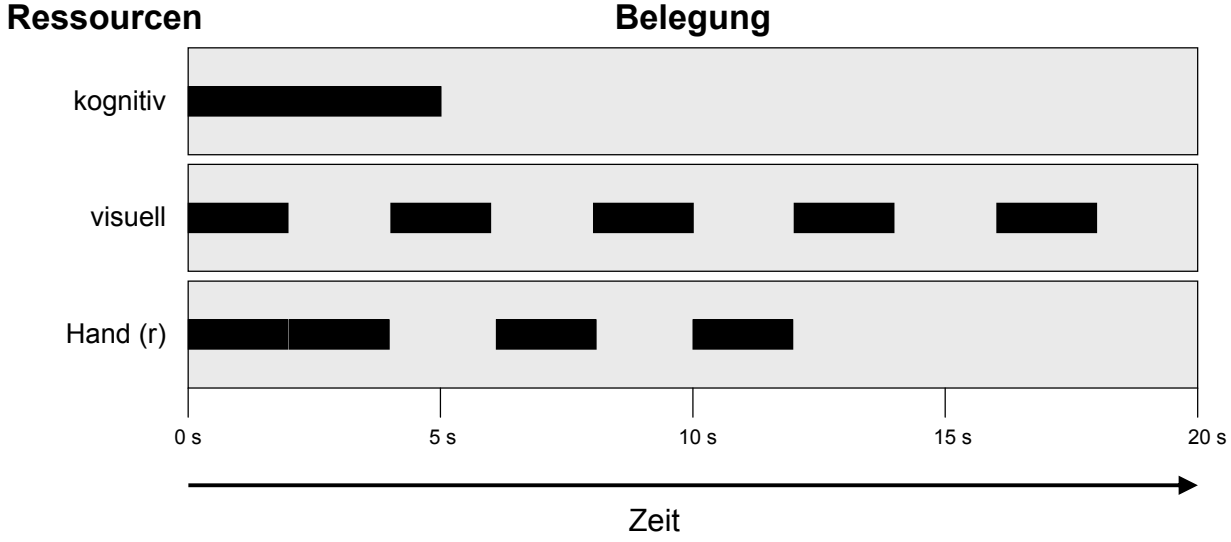


Abbildung 6-20: Profil der Belegung der Ressourcen in einer Fahrsituation

Für die Modellierung der Nebenaufgabe wird die Weiterentwicklung von GOMS verwendet. Sie basiert auf Konzepten von CPM-GOMS und NGOMSL. Aus CPM-GOMS wurde die Parallelisierung von Operatoren und deren Zuordnung zu ihrer Ressourcennutzung übernommen. Zur Behandlung von Zielen und komplexen Auswahlregeln werden die entsprechenden Elemente aus NGOMSL verwendet.

Zusätzlich werden multitasking-relevante Eigenschaften von Methoden und Operatoren der Nebenaufgabe erfasst, die beschreiben, welche Einschränkungen für

das Scheduling von Haupt- und Nebenaufgabe einzuhalten sind. Diese Aufgabeneigenschaften sind anwendungsspezifisch. Sie müssen bei der Modellierung zusätzlich zum GOMS-Aufgabenmodell ermittelt und im Modell repräsentiert werden. MT-GOMS wurde dafür um folgende Eigenschaften erweitert:

- Zur Bestimmung der Zeitpunkte, an denen der Scheduler zwischen Haupt- und Nebenaufgabe wechseln kann, müssen Unterbrechungspunkte definiert werden. Sie beziehen sich auf Schritte von Methoden. Manche Methoden können während ihrer Abarbeitung unterbrochen werden, jedoch immer nur zwischen der Bearbeitung von Schritten. Andere Methoden können gar nicht unterbrochen werden. Sie müssen bis zum Ende abgearbeitet werden. Dann kann ein Wechsel zur Hauptaufgabe stattfinden. Diese Eigenschaft kennzeichnet kritische Methoden, die der Bediener aus anwendungsspezifischen Gründen nicht unterbrechen kann.
- Nachdem von der Hauptaufgabe zurück zur Nebenaufgabe gewechselt wurde, gibt es mehrere Möglichkeiten, wie die Wiederaufnahme der unterbrochenen Methode geschehen kann. Manche Methoden können am Unterbrechungspunkt wiederaufgenommen werden. Unter Umständen darf dafür aber nur eine maximale Unterbrechungszeit verstrichen sein, in der diese Art der Wiederaufnahme noch möglich ist. Die Unterbrechungszeit bildet die Vergessensrate im Arbeitsgedächtnis nach.
- In der unterbrochenen Methode können weiterhin Wiederaufnahmepunkte definiert werden. Dann kann die Methode zwar nicht direkt am ursprünglichen Unterbrechungspunkt wieder aufgenommen werden, es muss aber auch nicht zum Anfang der Methode gesprungen werden, sondern zum letzten bearbeiteten Wiederaufnahmepunkt. Als Wiederaufnahmepunkte können alle Schritte der Methode markiert werden. Die Bedeutung der Wiederaufnahmepunkte sind wesentliche Zwischenschritte in der Aufgabenbearbeitung. Insbesondere nach der Erfüllung von Teil- bzw. Unterzielen sind Weideraufnahmepunkte anzunehmen.
- Nach der Wiederaufnahme kann es erforderlich sein, eine Methode auszuführen, die den Zustand des Bediensystems oder den kognitiven Zustand wiederherstellt. Daher kann jeder Methode eine Wiederaufnahmemethode zugeordnet werden.

In einem regelgeleiteten Transformationsprozess wird das MT-GOMS-Modell der Nebenaufgabenbearbeitung IVIS-Bedienung mit dem Ressourcen-Profil einer ausgewählten prototypischen Fahraufgabe gemischt. Das Resultat ist ein Aufgabenmodell, in dem die Aktivitäten der Haupt- und Nebenaufgabe gemischt sind. Damit steht ein GOMS-Modell der Nebenaufgabenbearbeitung zur Verfügung, das auch wesentliche Aspekte der Hauptaufgabe beinhaltet. Aus diesem Modell kann die Beanspruchung durch die Hauptaufgabe bei der Nebenaufgabenbearbeitung abgeleitet werden. Mögliche Wiederholungen von Teilen der Nebenaufgabe und

andere bekannte Phänomene bei der Bearbeitung von Doppelaufgaben werden von den Regeln des Transformationsprozesses berücksichtigt.

Eine Simulation des Modells der Bearbeitung der Gesamtaufgabe (Haupt- und Nebenaufgabe zusammen in ihrer Interaktion) zeigt dann etwaige Schwierigkeiten bei der Bedienung und damit die Kritizität des Ablenkungspotenzials einer Schnittstellengestaltung des Dialogsystems in der gewählten Fahrsituation auf.

Der Transformationsprozess ist generisch. Es können also unterschiedliche Regeln zur Mischung von Haupt- und Nebenaufgabe verwendet werden. Eine erste Regel, mit der eine maximale Umsetzung der Sicherheitsanforderungen aus der Hauptaufgabe umgesetzt wird, ist, das Ressourcenprofil aus der Hauptaufgabe komplett zu erfüllen. Die Aktivitäten zur Erfüllung der Nebenaufgabe werden dann entsprechend den Möglichkeiten von Eigenschaften zu Unterbrechbarkeit und Wiederaufnahme um diese Ressourcenanforderungen mit Standardverfahren des Scheduling herumgruppiert. Alternativ dazu kann eine höhere Risikobereitschaft im Verhalten durch Optimierung des Scheduling bei teilweiser Abweichung von den Anforderungen des Ressourcenprofils der Hauptaufgabe modelliert werden. Die Abweichung kann dabei für jede Ressource einzeln durch Angabe einer maximal erlaubten Verschiebung ausgedrückt werden.

6.8.2. Implementierung

Abbildung 6-21 zeigt das Objektmodell von MT-GOMS-Aufgabenbeschreibungen und Ressourcenprofilen anhand eines UML-Klassendiagramms. Die Klassen `Goal`, `Operator`, `Method`, `SelectionRule`, `Rule`, `Step`, `Task` und `Description` gehören zur GOMS-Basis von NGOMSL. Die Operatoren `NSG` (*new sub goal*) und `RGA` (*return with goal acomplished*) gehören zum strukturellen Teil von NGOMSL. Im Gegensatz zu anderen Operatoren realisieren sie die Zielverarbeitung. Die neuen Sprachelemente werden zur Notation von Ressourcen (`Ressource`), Unterbrechbarkeit (`ResumeInfo.interruptable`), Wiederaufnahmefähigkeit (`ResumeInfo.resumable` und `ResumeInfo.resumableUntil`) und Widerauf-nahmepunkten (`CheckPoint` bei einfacher Wiederaufnahme bzw. `ResumeMethod-Point` bei Wiederaufnahme mit Wiederherstellungsmethode) in Methoden von Methodenschritten definiert.

Zusätzlich ist im Klassendiagramm in Abbildung 6-21 das Objektmodell von Ressourcenprofilen dargestellt. Es besteht aus einer Reihe von Blockierungen einer bestimmten Ressource (`RessourceBlock`). Jeder Block beginnt zu einem bestimmten Zeitpunkt (`RessourceBlock.begin`), der relativ zum Start des Profils angegeben wird. Es gibt zwei Möglichkeiten zur Spezifikation der Ressourcennutzung. Es kann eine Liste von einzelnen Blockierungen bestimmter Dauer mit Beginn- und Endezeitpunkt (`ConstBlock`) angegeben werden und es können sich wiederholende Muster mit Beginnzeitpunkt und einer Zeitdauer für

Nutzung (`RepBlock.inUse`) und nachfolgende Freigabe (`RepBlock.notUsed`) der Ressource spezifiziert werden.

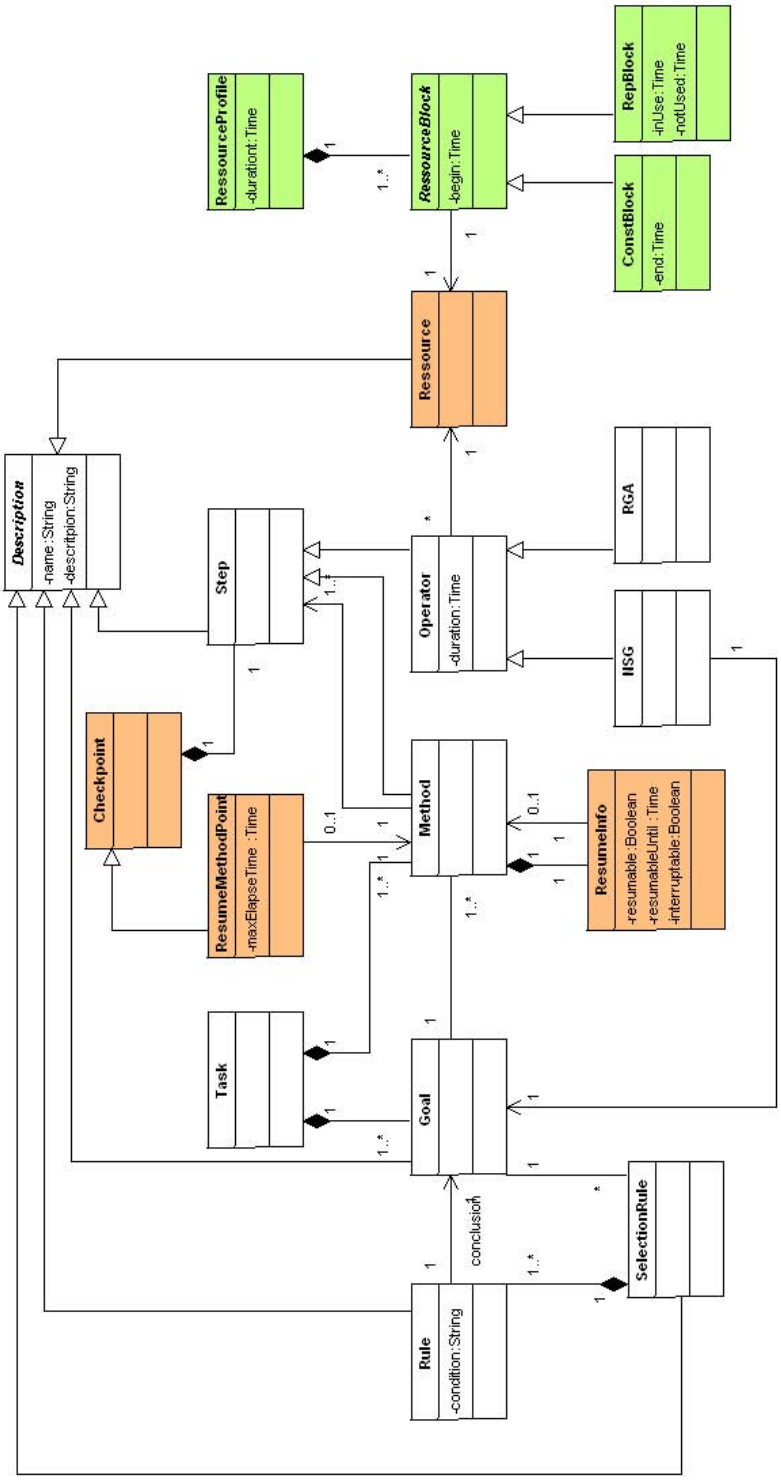


Abbildung 6-21: Klassendiagramm für Aufgabenbeschreibung in MT-GOMS (Erweiterungen von NGOMSL: `Checkpoint`**,** `ResumeMethodPoint`**,** `ResumeInfo` **und** `Ressource`**; Spezifikation von Ressourcen-Profilen:** `RessourceProfile`**,** `RessourceBlock`**,** `ConstBlock`**,** `RepBlock`**)**

MT-GOMS ist nicht mehr im Rahmen dieser Arbeit implementiert worden. Es gibt jedoch eine Implementierung auf der Basis von Java. In dieser Implementierung werden die Aufgabenmodelle und Ressourcenprofile mit XML repräsentiert und vom Schedulingalgorithmus zu einem integrierten Multitasking-Modell transformiert. Aus dem Vergleich der Vorhersagen von Einzelaufgaben- und Mehraufgabenmodell werden dann Kennzahlen generiert, mit deren Hilfe das Ablenkungspotential bewertet wird (Urbas et al. 2007).

6.8.3. Abgrenzung

Stand der Technik bei der Bewertung der Ablenkung von der Fahraufgabe durch die Benutzung von IVIS ist gegenwärtig der Einsatz von Blickbewegungsmessung (Sodhi et al. 2002) und Okklusionsmethode (Baumann et al. 2003) in Benutzertests. Dafür müssen fortgeschrittene Prototypen des zu untersuchenden IVIS vorliegen. Die Durchführung der Tests ist vergleichsweise aufwändig.

Eine Modellierung der Gesamtaufgabe Fahrzeugführung und IVIS-Bedienung ist bereits mit großem Aufwand mit ACT-R durchgeführt worden (Driver-ACT, Salvucci 2006). Dabei wurden Haupt- und Nebenaufgabe sowie deren Interaktion auf der Granularitätsebene von ACT-R modelliert. Für einen anwendungsorientierten Einsatz von Modellierungstechniken ist der Aufwand jedoch zu groß.

Für die GOMS-Aufgabenmodellierung ist ein Satz anwendungsspezifischer Operatoren erforderlich. Für unterschiedliche Domänen liegen passende Operator-Bibliotheken vor (z.B. Card et al. 1983). GOMS wurde auch bereits für die Modellierung von IVIS-Bedienung im Kraftfahrzeug eingesetzt, jedoch als Singletask im stehenden Auto (Hamacher & Hähnel 2002). Daher liegt ein Operatorensatz für die Mittelkonsolenbedienung vor.

GOMS+ bzw. PDL (*Procedure Definition Language,* Freed & Remington 2000) ist ein Ansatz zur GOMS-Modellierung von Multitasking. Dazu wurde GOMS um einige Konstrukte erweitert: `do-in-parallel` und `race` zur Parallelisierung von Aktivitäten, `wait-for`, `priority` und `requires` zur Formulierung von Abhängigkeiten zwischen parallelen Schritten. Mit APEX (Freed 1998b) können PDL Aufgabenmodelle simuliert werden. APEX enthält eine Scheduling-Komponente, mit der die Nebenläufigkeit umgesetzt wird. In APEX bzw. mit GOMS+ werden jedoch nicht Haupt- und Nebenaufgabe getrennt modelliert, sondern einzelne Methoden bzw. PDL *Procedures* können parallel ausführbare Schritte enthalten. Die Interaktion zwischen den parallelisierbaren Schritten wird dann von der APEX-Architektur aufgrund der modellierten Einschränkungen automatisch ermittelt.

MT-GOMS ist als einzige Modellierungstechnik in der Lage, Aufgaben in Multitaskingumgebungen ohne gleichzeitige Modellierung der komplexen Hauptaufgabe zu beschreiben. Damit sinkt der Modellierungsaufwand erheblich und ein entwicklungsbegleitender Einsatz ist möglich.

6.8.4. Einsatz in Fallstudie und Bewertung

Mit dem Ansatz von MT-GOMS ist es erstmals möglich, mit wenig Aufwand GOMS-Analysen in Multitasking-Aufgabenumgebungen durchzuführen. Die Komplexität des Scheduling ist dabei in die Simulationsumgebung verlagert worden. Die Anforderungen der Hauptaufgabe müssen dabei jedoch experimentell erhoben werden. Denn ein Aufgabenmodell, aus dem Ressourcenprofile ableitbar wären, müsste eine sehr hohe Komplexität, vergleichbar mit dem Driver-ACT-Modell (Salvucci 2006) haben. Es müssen jedoch nur einige wenige prototypische Ressourcenprofile erhoben werden, die dann in allen zukünftigen MT-GOMS Anwendungen einer Aufgabendomäne benutzt werden können.

Die Resultate von MT-GOMS-Simulationen sind zwar auf feingranularer Ebene, denn sie beziehen sich auf die Belegung einzelner kognitiver Ressourcen. Die vereinfachenden Annahmen der Simulationsumgebung lassen dagegen nur grobe Schätzungen der Bedienbarkeit und des Ablenkungspotenzials des modellierten IVIS zu.

Der MT-GOMS-Ansatz ist – jedoch nicht mehr im Rahmen dieser Arbeit – in einer Implementierung im Rahmen eines Anwendungsprojektes erprobt worden. In dem Projekt „Vernetztes Fahren“ (Kolrep et al. 2004) wurden die Anforderungen von prototypischen Fahraufgaben experimentell in einem Fahrsimulator erhoben (Urbas et al. 2005) und für die MT-GOMS-Verarbeitung als Ressourcenprofile repräsentiert. Im Rahmen des Projektes wurden Aufgabenmodelle mehrerer IVIS-Entwürfe mit MT-GOMS erstellt. Die Bedienbarkeit und das Ablenkungspotenzial der IVIS-Prototypen wurden experimentell ermittelt und mit den Simulationsergebnissen von MT-GOMS verglichen. Ergebnis ist, dass Modelle, die die Singletask-Bedingung bereits gut abbilden, auch das Ausmaß der Verlängerung der Bearbeitungszeit im Mehraufgabenkontext verlässlich vorhersagen. MT-GOMS ermöglichte somit kritische Stellen im Interaktionsverlauf zu identifizieren und die Entwicklungsarbeit auf diese Probleme zu fokussieren (Urbas et al. 2007).

MT-GOMS stellt eine Abstraktion der Bedienermodellierung dar, die spezifisch für den in dieser Arbeit angestrebten Anwendungszweck der Modellierung ist, da nur die Sekundäraufgabe explizit modelliert wird. Die Strategien zum Multitasking und die Ressourcenprofile interessierender Primäraufgaben werden dagegen in die Architektur des MT-GOMS Systems verlegt.

MT-GOMS ist dabei nicht auf Primäraufgaben aus der Kraftfahrzeugführung beschränkt, denn Ressourcenprofile können für beliebige Primäraufgaben erhoben und hinterlegt werden. MT-GOMS ist also ein generischer Ansatz zur Bedienermodellierung in Multitaskingumgebungen auf der Aufgabenanalyseebene. Die Taskwechselstrategien sind jedoch spezifisch für den Wechsel zwischen genau zwei Aufgaben, wovon die nicht explizit modellierte Primäraufgabe (z.B. Kraftfahrzeugführung) kritisch ist und die Sekundäraufgabe (z.B. IVIS-Bedienung) nicht kritisch ist.

MT-GOMS ist also für eine Vielzahl von Fragestellungen verwendbar. Damit ist ein hoher Wiederverwendungsgrad der in MT-GOMS spezifizierten Verarbeitungsprozesse und Strukturen gegeben.

6.9. Diskussion und Ausblick

Ein Ziel dieser Arbeit ist ACT-R unter Berücksichtigung von KLM und (N)GOMSL so zu erweitern, dass Bedienermodellierung in dynamischen Aufgabenumgebungen effizient ermöglicht wird. Während der Modellentwicklung für die Fallstudien wurden die neuen Werkzeuge und Erweiterungen parallel dazu konzipiert und realisiert. Deshalb ist es nicht möglich, den Nettomodellierungsaufwand der Bedienermodellierung der Fallstudien anzugeben. Zudem ist ein Vergleich von Fällen mit und ohne Unterstützung durch die neuen Erweiterungen unmöglich, da Bedienermodellierung in dynamischen Mensch-Maschine-Systemen diese Erweiterungen erst erfordert und der Versuch der Bedienermodellierung ohne diese Erweiterungen nicht zum gewünschten Ergebnis führen kann.

An dieser Stelle kann deshalb lediglich eine exemplarische Aufstellung des erforderlichen Aufwands zur Kopplung von Bedienermodell und externer Aufgabenumgebung mit ACT-COM und Agimap vorgenommen werden. Die anderen Erweiterungen Planex, Timer und MT-GOMS sind so eng mit Inhalt und Struktur der jeweiligen Bedienermodelle verwoben, dass eine Einzelbetrachtung des Integrationsaufwandes nicht sinnvoll ist.

ACT-COM wurde in der Fallstudie MoFL verwendet. Agimap wurde in DURESS und der Rektifikationskolonne benutzt. Die Einbindung des ACT-COM-Moduls und von Agimap ist ohne nennenswerten Aufwand möglich. Die Benutzung bedingt bei ACT-COM die Spezifikation des Protokolls und die Definition von Call Back Funktionen. Im Fall von MoFL wurden unterschiedliche externe Aufgabenumgebungen mit unterschiedlichen Kommunikationsprotokollen verwendet (s. Abschnitt 6.2.1). Die Anpassung der Protokollspezifikation war dabei ebenfalls ohne nennenswerten Aufwand möglich. Im Fall von Agimap wird das Framework über den Agimap-Editor in das Bedienermodell integriert. Dabei muss das GUI der externen Aufgabenumgebung mit dem Editor nachgebildet werden. Die Agimap Mappings von DURESS und der Rektifikationskolonne konnten dabei jeweils innerhalb weniger Stunden mit dem Editor aufgebaut werden.

Neben der Integration von ACT-COM und Agimap auf der Seite des kognitiven Modells muss der Aufwand für die Anbindung auf der Simulationsseite betrachtet werden. Im Fall von ACT-COM wurden zwei unterschiedliche externe Aufgabenumgebungen angebunden, die den Luftraum simulieren. EnCoRePLUS musste um ca. 700 Programmzeilen (*lines of code* – LOC) in C++ und gats/simco um ca. 900 LOC in C erweitert werden, damit die Anforderungen an die Anbindung der ACT-COM Schnittstelle erfüllt werden konnten. Die erforderlichen Änderungen waren komplex und erforderten einen Arbeitsaufwand von jeweils ca. fünf Personentagen. Im Fall von Agimap wurden prozesstechnische Simulationen für DURESS und die Rektifikationskolonne angeschlossen. Beide sind mit unterschiedlichen Frameworks in Java realisiert. Die Änderungen an beiden Aufgabenumgebungen folgen dem

Publisher/Subscriber Design Pattern. Der Aufwand zur Realisierung lag jeweils bei wenigen Personenstunden. Der größere Aufwand von ca. zwei Personentagen lag bei der Anbindung von DURESS in der Analyse der undokumentierten Software-Architektur. Im Fall der Rektifikationskolonne, das mit dem ProperEduct-Framework implementiert ist, war dafür kein Aufwand erforderlich.

Somit wird deutlich, dass das Ausmaß des Aufwands, der für die Kopplung von Bedienermodell und externer Aufgabenumgebung erforderlich ist, stärker von der Software-Architektur der Aufgabensimulation als von den Gegebenheiten der Erweiterungen ACT-COM und Agimap bestimmt wird.

Es hat sich gezeigt, dass manche der erforderlichen Neuentwicklungen zur Bedienermodellierung in dynamischen Mensch-Maschine-Systemen als Framework innerhalb der ACT-R-Theorie und -Architektur implementierbar waren (Planex, Umgang mit Multitasking in MoFL und Agimap). Nur Timer und in gewissem Ausmaß ACT-COM sind Erweiterungen von ACT-R in dem Sinn, dass der Fähigkeitsumfang der kognitiven Architektur erweitert wurde. MT-GOMS wiederum erweitert die Fähigkeiten von GOMSL.

Eine wichtige Anforderung an kognitive Architekturen ist, dass Neuentwicklungen rückwärtskompatibel sein müssen. Das bedeutet, dass alle Modelle, die bisher in einer Architektur umsetzbar waren und spezifische kognitive Phänomene erklärt hatten, auch in der weiterentwickelten Architektur möglich sein müssen. Die Implementierung als Framework innerhalb einer Architektur ist ein weniger starker Eingriff. Insbesondere besteht nicht die Gefahr, die erforderliche Rückwärts-kompatibilität zu verlieren. Da die Erweiterungen Timer und ACT-COM Bereiche betreffen, die bisher nicht von der ACT-R-Architektur abgedeckt wurden, entstehen hier keine Probleme mit der Rückwärtskompatibilität.

MT-GOMS-Modelle der Sekundäraufgabe sind im Prinzip erweiterte GOMSL-Aufgabenmodelle. Werden die neuen Sprachelemente zu Unterbrechbarkeit etc. aus einem MT-GOMS-Modell einer Sekundäraufgabe herausgefiltert, ergibt sich ein Standard GOMSL-Modell. Der MT-GOMS-Ansatz ist also ebenfalls in hohem Maß rückwärtskompatibel.

Der Weg über eine Transformation von Aufgabenmodell zum kognitiven Modell vor der Laufzeit der Modellsimulation im Sinne des Compilationsansatzes hat sich als sehr praktikabel erwiesen. Neueste Ergebnisse des Forschungsschwerpunktes Bedienermodellierung des Graduiertenkollegs prometei der TU Berlin bestätigen die Qualität und Stabilität des Ansatzes mit dem HTAMap-System. In dieser Arbeit wurde dieser Ansatz an mehreren Stellen angewendet:

- bei Agimap zur Generierung von ACT-R/PM Produktionen und Chunks
- bei MT-GOMS zur Generierung eines Multitaskingmodells aus dem Singletaskmodell und dem Ressourcenprofil

- bei DURESS zur Generierung von Planex-Code aus der vorliegenden Plan Library
- beim Werkzeug zur Visualisierung – hier jedoch als Transformation in eine grafische Sprache.

Das Thema Kopplung wurde auf unterschiedlichen Granularitätsebenen angegangen. ACT-COM und Agimap sind zwei unterschiedliche Ansätze, die auf verschiedene Anwendungen abzielen. Je nach Aufgabenstellung sind beide Ansätze erforderlich.

Zur Unterstützung des Modellierungsprozesses wurden mehrere Werkzeuge neu entwickelt. Die Erfahrungen mit der Bedienermodellierung zeigen, dass die Werkzeugunterstützung für die Anwendbarkeit kritisch ist. Dabei ist zu beachten, dass der ganze Modellierungsprozess unterstützt werden muss. Zum Entwicklungsprozess (Abbildung 2-4, S. 15) gibt es die folgende Zuordnung von neu entwickelten Werkzeugen:

1) konzeptuelle Modellbildung/Verifikation: Visualisierung/grafische Sprache
2) Formalisierung/Modellverifikation: Visualisierung
3) Programmerstellung/Programmverifikation: Agimap, MT-GOMS
4) Simulationsexperimente/Ergebnisverifikation: ACT-R Shell
5) Validierung: *noch keine Unterstützung*

Einen Ausblick auf zukünftige Weiterentwicklungen geben die Arbeiten im Forschungsschwerpunkt Bedienermodellierung im Graduiertenkolleg prometei der TU Berlin. Hier werden die folgenden Themen weitergeführt:

- Nutzung von Daten aus Bedienersimulationsexperimenten als neues Werkzeug für die Modellierungsphase Validierung (SimTrA, Dzaack 2008)
- Weiterentwicklung von Agimap zu HTAmap zur Verwendung von Design Patterns zur Bedienermodellierung von Mensch-Maschine-Interaktion (Heinath 2009)
- Weiterentwicklung von Timer (Pape 2009)

Weitere Zukunftsthemen für weiterführende Arbeiten sind eine weitere Integration der Werkzeuge und die Entwicklung einer verbesserten Visualisierung auf der Basis von SoarML. Außerdem ist die Fähigkeit zum Round-Trip Engineering eine zentrale Anforderung an eine zukünftige integrierte Bedienermodellierungsumgebung.

Neue Architekturerweiterungen sind im Bereich verteilter Mensch-Maschine-Systeme erforderlich. Insbesondere die dafür erforderliche Fähigkeit zur Teilnahme an vernetzten Simulationen setzt voraus, dass eine Zeitsynchronisation von Simulatoren der Aufgabenumgebung und kognitivem Modell möglich ist. Außerdem muss die Informationsaufnahme der kognitiven Architektur an die Kommunikations- und Anwendungsprotokolle, die in diesem Bereich verwendet werden (DIS, HLA) angepasst werden.

Zur Verbesserung der Mehrbenutzerentwicklung ist eine ACT-R-Spracherweiterung zur Modularisierung erforderlich. Eine Paketierung ist jedoch ungleich schwieriger als bei der Softwareentwicklung, da es viele Querbezüge und Seiteneffekte zwischen Modellteilen geben kann, die nicht explizit spezifiziert sind. Hier gibt es eine direkte Analogie zu den Problemen und Lösungsansätzen bei der Modularisierung von Ontologien (OWL-Erweiterung P-DL: *Package Based Desciption Logic*).

Die Literatur zur Bedienermodellierung ist hauptsächlich kognitionswissenschaftlich und grundlagenorientiert. Kognitive Modelle werden selten über Websites als zusätzliches Material zu Modellveröffentlichungen zugänglich gemacht. Für die Identifikation und Formalisierung von Design Patterns zur Bedienermodellierung fehlt indes noch ein Katalog von komplexen anwendungserprobten Modellen. Ein schwieriges Problem für den Aufbau und die Pflege einer solchen Modellsammlung sind die wenig standardisierten Methoden zur Qualitätssicherung und die damit fehlende Vergleichbarkeit von konkreten Modellen sowie die erforderliche Bereitstellung von komplexeren (und oft zugangsbeschränkten) Aufgabenumgebungen. Jedoch bietet ein solcher Modellkatalog die Möglichkeit Patterns mit formalen Methoden und erfahrungsbasiert abzuleiten.

Die Dynamik in der ACT-R Community und die gegenwärtigen Forschungsgegenstände einiger Arbeitsgruppen belegen, dass die Auswahl von ACT-R/PM als Grundlage der Bedienermodellierung auf kognitiver Analyseebene geeignet ist. Ebenso lassen die aktuellen Arbeiten an Compilationsansätzen darauf schließen, dass (N)GOMSL für die Aufgabenmodellierung geeignet ist. Weiterentwicklung und Austausch in einer Community sind also mittelfristig gesichert.

7. Zusammenfassung

In dieser Arbeit wurden softwaretechnische Methoden zur Bedienermodellierung in dynamischen Mensch-Maschine-Systemen untersucht und zu einem anwendungsorientierten Einsatz weiterentwickelt. Das übergeordnete Ziel war dabei der praktische Einsatz von Bedienermodellierung zur Unterstützung der Analyse und Gestaltung von Mensch-Maschine-Interaktion im Entwicklungsprozess. Durch die Simulation von Benutzungstests mit Bedienermodellen anstatt der Durchführung von Versuchen mit Probanden sollen *Human Factors*-bezogene Gestaltungsprobleme bei geringeren Kosten und früher im Entwicklungsprozess als bei herkömmlichen Benutzertests aufgedeckt werden. Die praktische Anwendbarkeit von Werkzeugen zur Bedienermodellierung ist gegenwärtig noch nicht gegeben. Durch softwaretechnische Methoden sollte deshalb die Anwendbarkeit für diesen Einsatzbereich durch diese Arbeit erhöht werden.

Im Rahmen dieser Arbeit wurde dazu der Stand der Forschung in Bezug auf die Bedienermodellierung mit kognitiven Architekturen und Werkzeugen zur kognitiven Simulation, zur hierarchischen Aufgabenanalyse und -modellierung, zur *Human Reliability Analysis*, zur Bedienermodellierung mit Aufgabennetzwerken und Simulationsumgebungen sowie der Synthese der damit verknüpften Analyseebenen „Kognition“ und „Aufgabe“ in Compilations- und hybriden Modellierungsansätzen aufgearbeitet. Diese Methoden und Werkzeuge wurden dann spezifisch auf ihre Eignung in Bezug zur anwendungsorientierten Bedienermodellierung in dynamischen Mensch-Maschine-Systemen hin untersucht, indem bisher modellierte Phänomene, Anwendungsfelder und ihre Anforderungen dargestellt wurden. Anhand dieses Überblicks über den aktuellen Stand der Forschung wurde ACT-R/PM als kognitive Architektur und NGOMSL zur Aufgabenmodellierung in diesem Bereich ausgewählt.

Mit diesen Werkzeugen wurden mehrere Fallstudien zur Bedienermodellierung durchgeführt. Die Fallstudien wurden so ausgewählt, dass sie einen breiten Bereich potenzieller Anforderungen an die Bedienermodellierung in dynamischen Mensch-Maschine-Systemen abdecken. Das wurde sichergestellt, indem ein neuer dazu passender Beschreibungsrahmen für Aufgabenumgebungen in dynamischen Mensch-Maschine-Systemen entwickelt und auf unterschiedliche Domänen angewendet wurde. Das Ergebnis war eine Klassifizierung unterschiedlicher Aufgabenumgebungen in drei Klassen: Command and Control, Prozesskontrolle und Kraftfahrzeugführung. Die wesentlichen Unterschiede in Bezug auf Anforderungen an die Bedienermodellierung zwischen diesen Klassen liegen in den benötigten Kontrollstrukturen und der Interaktion mit der Aufgabenumgebung, die aus deren Komplexität und Dynamik resultieren. Eine weitere Dimension bei der Auswahl der Fallstudien bildete der Grad der Deliberativität bzw. Reaktivität der Aufgabe.

Die Fallstudien zur Bedienermodellierung wurden durchgeführt, um die vorhandenen Lücken der bestehenden Theorien und Werkzeuge zu identifizieren

und gleichzeitig neue Werkzeuge und Erweiterungen der Theorien zu entwickeln und im Einsatz zu testen. Ergebnis dieser Arbeit sind in den Fallstudien erprobte neue Werkzeuge und Erweiterungen von Bedienermodellierungsmethoden auf den Analyseebenen Kognition und Aufgabe in den folgenden Bereichen: Guideline (Modellierungsrationale ACT-R/PM und NGOMSL), Modellierungswerkzeug (Visualisierung, Agimap, ACT-R Shell, MT-GOMS), Framework (ACT-Com, Agimap, Planex) und Erweiterung (Timer, MT-GOMS). Alle diese neuen Entwicklungen haben Eigenschaften, die sie von allen anderweitig verfügbaren Arbeiten abheben und die die Bedienermodellierung in dynamischen Mensch-Maschine-Systemen effizienter und effektiver machen.

7.1. Zielerreichung und Beantwortung der Forschungsfragen

Um die Ziele dieser Arbeit zu erreichen, mussten folgende Fragen beantwortet werden:

1) *Für welche Aufgabensituation (Anwendungsdomäne) ist die Bedienermodellierung anwendbar?*

   Durch das Framework zur Beschreibung von Aufgabenumgebungen und der Zuordnung von Anwendungsdomänen zu unterschiedlichen Anforderungen an Kontrollstrukturen und Interaktion wird diese Frage differenziert beantwortet. Zusätzlich muss betrachtet werden, ob die Aufgabe deliberative oder reaktive Bedienung erfordert.

2) *Welche Erweiterungen sind an den kognitionswissenschaftlichen Frameworks erforderlich, um mit dynamischen Mensch-Maschine-Systemen umgehen zu können?*

   Es hat sich gezeigt, dass Neuentwicklungen in Form von Frameworks und Architekturerweiterungen in folgenden Bereichen notwendig waren: Kontrollstrukturen für Multitasking und SOPs, Zeitwahrnehmung und Interaktion mit externen Aufgabenumgebungen.

3) *Mit welchen Entwicklungswerkzeugen kann die Bedienermodellierung unterstützt werden?*

   Die Bedienermodellierung konnte durch Entwicklung und Bereitstellung von Werkzeugen zur Unterstützung der konzeptuellen Modellbildung (Visualisierung/grafische Sprache), der Formalisierung (Visualisierung), der Modellprogrammerstellung (Agimap, MT-GOMS) und der Durchführung von Simulationsexperimenten (ACT-R Shell) unterstützt werden.

7.2. Fazit

Mit ACT-R/PM, KLM und NGOMSL wurden nützliche Architekturen bzw. Formalismen für die Bedienermodellierung identifiziert und durch neue Erweiterungen für Fragestellungen der Mensch-Maschine-Interaktion zugänglich

gemacht. Durch neue Werkzeuge ist der praktische Einsatz effizienter geworden. Eine zukünftig stärkere Anwendungsorientierung beim Einsatz von kognitiven Architekturen ist abzusehen, da sie zunehmend Verbreitung außerhalb der erkenntnisorientierten Kognitionswissenschaft finden (z.B. autonomes Fahren bei Langley 2006).

Die Wirtschaftlichkeit der Bedienermodellierung als einem von vielen Instrumenten im Entwicklungsprozess von Mensch-Maschine-Systemen ist letztlich das einzige Argument für oder gegen einen Einsatz. Es bleibt festzuhalten, dass durch die Entwicklungen in dieser Arbeit die Kosten eines Einsatzes reduziert wurden. Trotzdem bleibt es grundsätzlich bei einer Verlagerung von variablen Kosten für die wiederholte Durchführung von Benutzertests gegen Ende der Entwicklung hin zu einem festen Kostenblock zur Modellerstellung in frühen Phasen der Entwicklung. Die – auch wiederholte – Durchführung und Auswertung von Simulationsexperimenten ist hingegen vernachlässigbar. Ein wirtschaftliches Argument für die Bedienermodellierung ist also insbesondere dann gegeben, wenn Benutzertests wie bei verteilten Mensch-Maschine-Systemen sehr aufwändig durchzuführen sind.

Als Entwicklung zeichnet sich eine zunehmende „Reife“ der Bedienermodellierung als Engineeringwerkzeug ab. Um das zu belegen, kann eine Analogie zur Softwareentwicklung gezogen werden: Bis zur Lösung der Softwarekrise in den siebziger Jahren waren Programme als Softwareprodukte wissenschaftliche oder sogar kunsthandwerkliche Objekte, die in sehr mühseliger Kleinarbeit codiert werden mussten. Dann wurden Hochsprachen und die strukturierte, später die objektorientierte Programmierung eingeführt. Durch eine fortschreitende Professionalisierung war es möglich, einen Kanon allgemein anerkannter Design Patterns und Software-Architekturen zu definieren, deren Anwendung mit hoher Wahrscheinlichkeit zu qualitativ guten Softwareprodukten führt. Die Reifung der Programmierung zum Software Engineering ermöglicht heute die Entwicklung von missions- und sicherheitskritischen Softwaresystemen, die um Größenordnungen komplexer als frühere sind.

Diese Entwicklung „von Assembler zu C++ und UML“ muss analog in der Bedienermodellierung nachvollzogen werden, um zu einem ähnlichen Reifegrad zu gelangen. Kognitive Architekturen und Compilationsansätze versprechen diese Entwicklung zu durchlaufen, und auch die Entwicklung integrierter Modellierungswerkzeuge ist bei beherrschbarem Aufwand durch Nachvollziehen der Entwicklungen der Softwaretechnik machbar. Die Hoffnung ist also begründet, dass die Bedienermodellierung in absehbarer Zeit zu einem ähnlich hohen Anwendungspotenzial wie die Softwareentwicklung kommen kann.

Literaturverzeichnis

Aasmann, J. (1995). *Modelling Driver Behaviour In Soar*. Leidschendam: KPN Research.

ACT-R Research Group (o.J.). *AGI - ACT-R GUI Interface*. Pitsburgh, PA: Department of Psychology, Carnegie Mellon University. Online verfügbar unter: http://act-r.psy.cmu.edu/tutorials/AGI.pdf (letzter Zugriff: 2008-05-17).

Amant, R. St., Horton, T. E. & Ritter, F. E. (2007). Model-Based Evaluation of Expert Cell Phone Menu Interaction. *ACM Transactions on Computer-Human Interaction, 14* (1), 1-24.

Amant, R. St. & Riedl, M. O. (2001). A perception/action substrate for cognitive modeling in HCI. *International Journal Human-Computer Studies, 55*, 15-39.

Amant, R. St. & Ritter, F. (2004). Automated GOMS-to-ACT-R model generation. In: *Proceedings of the sixth International Conference on Cognitive Modeling*. Pittsburgh, PA: Carnegie Mellon University/University of Pittsburgh.

Anderson, J. R. (2007). *How Can the Human Mind Occur in the Physical Universe*? New York: Oxford University Press.

Anderson, J. R. (2002). Spanning seven orders of magnitude: a challenge for cognitive modeling. *Cognitive Science, 26* (1), 85-112.

Anderson, J. R. (1993). *Rules of the Mind*. Hillsdale, NJ: Erlbaum.

Anderson, J. R. (1983). *The Architecture of Cognition*. Cambridge, MA: Havard University Press.

Anderson, J. R., Bothell, D., Lebiere, C. & Matessa, M. (1998). An Integrated Theory of List Memory. *Journal of Memory and Language, 38*, 341-380.

Anderson, J. R. & Lebiere, C. (1998). *Atomic Components of Thought*. Hillsdale, N.J.: Erlbaum.

Anderson, J. R., Matessa, M. & Lebiere, C. (1997). ACT-R: A theory of higher level cognition and its relation to visual attention. *Human-Computer Interaction, 12* (4), 439-462.

Annett, J. & Duncan, K. D. (1967). Task Analysis and Training Design. *Occupational Psychology, 41*, 211-221.

Barnard, P. J. & May, J. (1999). Representing cognitive activity in complex tasks. *Human Computer Interaction, 14* (1 & 2), 93-158.

Bass, E., Baxter, G. & Ritter, F. (1995). Creating models to control simulations: A generic approach. *AI and Simulation of Behaviour Quarterly, 93*, 18-25.

Baumann, M., Rösler, D., Jahn, G. & Krems, J. F. (2003). Assessing driver distraction using Occlusion Method and Peripheral Detection Task. In: H. Strasser, K. Kluth, H. Rausch & H. Bubb (Hrsg.), *Quality of work and products in enterprises of the future* (S. 53-56). Stuttgart: Ergonomia Verlag.

Baxter, G. D. & Harrison, M. (o.J.). *Timing Issues in Application Domains*. Technical Report. Human-Computer Interaction Group, The University of York.

Beard, D. V., Smith, D. K. & Denelsbeck, K. M. (1996). Quick and Dirty GOMS: A Case Study of Computed Tomography Interpretation. *Human-Computer Interaction, 11* (2), 157-180.

Bierwagen, T. (1996). *Programmsystem EnCoRe-PLus: Über die Möglichkeiten der programmierbaren Luftraumsimulation*. ZMMS-Forschungsbericht (96-2). Zentrum Mensch-Maschine-Systeme, TU Berlin.

Bierwagen, T., Eyferth, K. & Niessen, C. (1997). *Bericht über die Arbeit des DFG-Projektes Modellierung von Fluglotsenleistungen in der Streckenflugkontrolle vom 15.09.1994 bis zum 14.09.1996*. ZMMS-Forschungsbericht (97-2). Zentrum Mensch-Maschine-Systeme, Technische Universität Berlin.

Bierwagen, T., Helbing, H. & Kolrep, H. (1995). *Report on the work carried out 15.09.1992 to 14.09.1994*. Forschungsbericht (95-1). Institut für Psychologie, Technische Universität Berlin.

Block, R. A., George, E. J. & Reed, M. A. (1980). A Watched Pot Sometimes Boils: A Study of Duration Experience. *Acta-Psychologica, 46* (2), 81-94.

Bloechle, W. K. & Schunk, D. (2003). Micro SAINT Sharp Simulation Software. In: *Proceedings of the 2003 Winter Simulation Conference* (S. 182-187).

Bomsdorf, B. & Szwillus, G. (2002). „UML und Aufgabenmodellierung: Softwaretechnik und HCI im Dialog“ auf der Konferenz Mensch & Computer 2001. *i-com,* 2/2002, 49-52.

Booch, G. (1991). *Object Oriented Design with Applications.* Redwood City, CA: Benjamin/Cummings Publishing Company.

Bösser, T. & Melchior, E. (1992). The SANE Toolkit for cognitive modelling and user-centered design. In: M. Gabler, S. Harker & J. Ziegler (Hrsg.), *Methods and Tools in User-Centered Design for Information Technology* (S. 93-125). North-Holland: Elsevier Science Publishers.

Bösser, T. & Melchior, E. (1990). User Centered Design, Prototyping, and Cognitive Modelling with the SANE Toolkit. In: *ESPRIT '90 Conference Proceedings* (S. 589-605). Commission of the European Community.

Busse, D. K. & Johnson, C. W. (1999). Investigating Pilots' Cognitive Processes in Aviation Accidents. In: *Proceedings of the International Conference on Human Interfaces in Control Rooms, Cockpits, and Command Centres, Bath, UK, June 1999*.

Byrne, M. D. (2001). ACT-R /PM and menu selection: applying a cognitive architecture to HCI. *International Journal Human-Computer Studies, 55*, 1-44.

Byrne, M. D. & Anderson, J. R. (1998). Perception and Action. In: J. R. Anderson & C. Lebiere (Hrsg.), *The Atomic Components of Thought* (S. 167-200). Mahwah, NJ: Lawrence Erlbaum.

Cacciabue, P. C., Decortis, F., Drozdowicz, B., Masson, M. & Nordvik, J. (1992). COSIMO: A cognitive simulation model of human decision making and behavior in accident

management of complex plants. *IEEE Transactions on Systems, Man, and Cybernetics, 22* (5), 1058-1074.

Card, S. K., Moran, T. P. & Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum.

Chong, R. S. (1995). *EPIC-Soar. Perception, Cognition, and Action: One Small Step Towards Unification.* Report. Ann Arbor: Artificial Intelligence Laborytory, University of Michigan.

Chong, R. S. & Wray, R. E. (2005). Inheriting Constraint in Hybrid Cognitive Architectures: Applying the EASE Architecture to Performance and Learning in a Simplified Air Traffic Control Task. In: K. A. Gluck & R. W. Pew (Hrsg.), *Modeling Human Behavior with Integrated Cognitive Architectures. Comparison, Evaluation, and Validation* (S. 237-304). Mahwah, NJ: Lawrence Erlbaum Associates.

Cnossen, F. (2000). *Adaptive strategies and goal management in car driving*. Proefschrift. University of Groningen. Online verfügbar unter: http://irs.ub.rug.nl/ppn/238175715 (letzter Zugriff: 2008-05-17).

Cohen, M. A., Ritter, F. E. & Haynes, S. R. (2005). Herbal: A high-level language and development environment for developing cognitive models in Soar. In: *Proceedings of the 14th Conference on Behavior Representation in Modeling and Simulation* (177-182). Orlando, FL: University of Central Florida.

Cooper, R. (2002). *Modelling High-Level Cognitive Processes*. Mahwah, NJ: Lawrence Erlbaum Associates.

Cooper, R. & Farringdon, J. (1993). *Sceptic Version 4 User Manual*. Technical Report (UCL-PSY-ADREM-TR6). Department of Psychology, University College London. Online verfügbar unter: http://cogent.psyc.bbk.ac.uk/publications/cooperfarringdon93.html (letzter Zugriff: 2008-05-17).

Cooper, R. & Fox, J. (1998). COGENT: A visual design environment for cognitive modeling. *Behavior Research Methods, Instruments & Computers, 30*, 553-564.

Cooper, R., Yule, P., Fox, J. & Sutton, D. (1998). COGENT: An environment for the development of cognitive models. In: U. Schmid, J. Krems & F. Wysotzki (Hrsg.), *A Cognitive Science Approach to Reasoning, Learning and Discovery* (S. 55-82). Lengerich: Pabst Science Publishers.

Corker, K. (2000). Cognitive Models and Control: Human and System Dynamics in Advanced Airspace Operations. In: N. B. Sarter & R. Amalberti (Hrsg.), *Cognitive Engineering in the Aviation Domain* (S. 13-42). Mahwa, NJ: Lawrence Erlbaum Associates.

Craig, K., Doyal, J., Brett, B., Lebiere, C., Biefeld, E. & Martin, E. (2002). Development of a hybrid model of tactical fighter pilot behavior using IMPRINT task network model and ACT-R. In: *Proceedings of the Eleventh Conference on Computer Generated Forces and Behavior Representation. Orlando, FL*.

Diaper, D. (2001). Task Analysis for Knowledge Descriptions (TAKD): A Requiem for a Method. *Behaviour and Information Technology, 20* (3), 199-212.

Dzaack, J. (2008). *Analyse kognitiver Benutzermodelle für die Evaluation von Mensch-Maschine-Systemen*. Dissertation. Technische Universität Berlin. Online verfügbar unter: http://opus.kobv.de/tuberlin/volltexte/2008/1787/ (letzter Zugriff: 2008-05-17).

Dzaack, J., Trösterer, S., Pape, N. & Urbas, L. (2006). A computational model of retrospective time duration estimation. In: *Proceedings of the 7th International Conference on Cognitive Modeling* (S. 100-104). Trieste, Italy:

Eisenhardt, K. M. & Graebner, M. E. (2007). Theory Building from Cases: Opportunities and Challenges. *Academy of Management Journal 50* (1), 25-32.

Endsley, M. R. (1995). Toward a theory of situation awareness in dynamic systems. *Human Factors, 37* (1), 32-64.

Ezzedine, H. & Kolski, C. (2005). Modelling of cognitive activity during normal and abnormal situations using Object Petri Nets, application to a supervision system. *Cognition, Technology & Work, 7* (3), 167-181.

Faber, G. (2002). System Awareness im Glass Cockpit. In: M. Grandt & K.-P. Gärtner (Hrsg.), Situation Awareness in der Fahrzeug- und Prozessführung (S. 1-10). Bonn: Deutsche Gesellschaft für Luft- und Raumfahrt e.V. (DGLR-Berlicht; 2002-04).

Flicker, R. (2001). *Eingabeinstrumente für Data-Link-Anweisungen zum Ersatz des Sprechfunks an modernen Fluglotsenarbeitsplätzen*. Studienarbeit. Institut für Luft- und Raumfahrt, Technische Universität Berlin.

Foltz, C., Killich, S. & Wolf, M. (2000). *K3 User Guide. Version 0.1, 21.11.2000*. Institut für Arbeitswissenschaft, RWTH Aachen. Online verfügbar unter: http://www.iaw.rwth-aachen.de/download/produkte/k3_userguide_2000-11-21.pdf (letzter Zugriff: 2008-05-17).

Freed, M. A. (1998a). *Simulating human performance in complex, dynamic environments.* Ph.D. Thesis. Evanston, Illinois: Northwestern University. Online verfügbar unter: http://interruptions.net/literature/Freed-Dissertation.pdf (letzter Zugriff: 2008-05-17).

Freed, M. A. (1998b). Managing multiple tasks in complex, dynamic environments. In: Ch. Rich & J. Mostow (Hrsg.), *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference: AAAI 98, IAAI 98* (S. 921-927). Menlo Park, CA: AAAI Press.

Freed, M. A. & Remington, R. W. (2000). GOMS, GOMS+, and PDL. In: *Working Notes of the AAAI 2000 Fall symposium on Simulating Human Agents, Falmouth, MA*.

Freed, M. A., Shafto, M. G. & Remington, R. W. (1998). Using simulation to evaluate designs: The APEX approach. In: *Proceedings of the 1998 Conference on Engineering for Human-Computer Interaction*.

Furuta, K. & Kondo, S. (1993). An approach to assessment of plant man-machine systems by computer simulation of an operator's cognitive behavior. *International Journal of Man-Machine Studies, 39*, 473-493.

Furuta, K., Shimada, T. & Kondo, S. (1999). Behavioral simulation of a nuclear power plant operator crew for human-machine system design. *Nuclear Engineering and Design, 188*, 97-109.

Furuta, K., Takahashi, M., Yoshikawa, H., Sasaki, K., Itoh, T., Matsumiya, M., Sakaue, T., Kiyokawa, K. & Hasegawa, A. (1995). Analysis of operarots's diagnostic Behavior Using Computer Simulation. In: Y. Anzai, K. Ogawa & H. Mori (Hrsg.), *Symbiosis of Human and Artifact*. Amsterdam: Elsevier.

Gallardo, D., Burnette, E. & McGovern, R. (2003). *Eclipse in Action. A Guide for Java Developers*. Greenwich, CT: Manning.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). *Design Patterns. Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley.

Geisler, J. (2006). *Leistung des Menschen am Bildschirmarbeitsplatz: Das Kurzzeitgedächtnis als Schranke menschlicher Belastbarkeit in der Konkurrenz von Arbeitsaufgabe und Systembedienung*. Dissertation. Universität Karlsruhe.

Giesa, H.-G. (2003). *Die Bewertung der Verlässlichkeit von Mensch-Maschine-Systemen*. Düsseldorf: VDI Verlag (Fortschrittberichte Reihe 22; 13).

Gluck, K. A., Ball, J. T. & Krusmark, M. A. (2007). Cognitive Control in a Computational Model of the Predator Pilot. In: W. D. Gray (Hrsg.), *Integrated Models of Cognitive Systems* (S. 13-28). New York: Oxford University Press.

Gluck, K. A., Ball, J. T., Krusmark, M. A., Rodgers, S. M. & Purtee, M. D. (2003). A Computational Process Model of Basic Aircraft Maneuvering. In: F. Detje, D. Dörner & H. Schaub (Hrsg.), *The Logic of Cognitive Systems. Proceedings of the Fifth International Conference on Cognitive Modeling* (S. 117-122). Bamberg: Universitäts-Verlag.

Gluck, K. A. & Pew, R. W. (Hrsg.) (2005). *Modeling Human Behavior with Integrated Cognitive Architectures. Comparison, Evaluation, and Validation*. Mahwah, NJ: Lawrence Erlbaum Associates.

Gonzalez, C., Lerch, F. & Lebiere, C. (2003). Instance-based learning in dynamic decision making. *Cognitive Science, 27* (4), 591-635.

Gray, W. D. (2000). The nature and processing of errors in interactive behavior. *Cognitive Science, 24* (2), 205-248.

Gray, W. D., John, B. E. & Atwood, M. E. (1993). Project Ernestine: Validating a GOMS Analysis for Predicting and Explaining Real-World Task Performance. *Human-Computer Interaction, 8*, 237-309.

Gray, W. D., Kirschenbaum, S. S. & Ehret, B. D. (1997). The precis of Project Nemo, phase 1: Subgoaling and subschemas for submariners. In: *Proceedings of the Nineteenth*

Annual Conference of the Cognitive Science Society (S. 283-288). Hillsdale, NJ: Erlbaum.

Hamacher, N. & Hähnel, M. (2002). Formale Bewertung unterschiedlicher Fahrer-Informationssysteme. In: *IIR-Fachkonferenz, Volume C 2336, D 10*. Nürtingen: IIR GmbH.

Hamacher, N. & Hähnel, M. (2001). Konzept für die automatische Generierung von Komplexitätsmaen zur Evaluierung interaktiver Geräte. In: K.-P. Gärtner & M. Grandt (Hrsg.), *Human Factors bei der Entwicklung von Fahrzeugen. 43. DGLR-Fachausschusssitzung Anthropotechnik, 23.-24.10.2001 Hamburg* (S. 117-127). Bonn: DGLR (DGLR-Bericht; 2001-06).

Hamacher, N., Kraiss, K.-F. & Marrenbach, J. (2002a). Einsatz formaler Methoden zur Evaluierung der Gebrauchsfähigkeit interaktiver Geräte. *it + ti Informationstechnik und Technische Informatik, 44*, 49-55.

Hamacher, N., Zieren, J., Marrenbach, J. & Kraiss, K.-F. (2002b). Generierung normativer Benutzermodelle aus SDL-Spezifikationen. In: R. Marzi, V. Karavezyris, H. Erbe & K.-P. Timpe (Hrsg.), *Bedienen und Verstehen. 4. Berliner Werkstatt Mensch-Maschine-Systeme. 10.-12.10.2001* (S. 86-100). Düsseldorf: VDI-Verlag.

Harel, D. (1987). Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming, 8* (3), 231-274.

Harrison, A. (2002). *jACT-R: Java ACT-R.* 8th Annual ACT-R Workshop, August 2-4. 2002. Pittsburgh, PA: Carnegie Mellon University. Online verfügbar unter: http://act-r.psy.cmu.edu/workshops/workshop-2002/talks/AnthonyHarrison.pdf (letzter Zugriff: 2008-05-17).

Hart, S. & Staveland, L. (1988). Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In: P. Hancock & N. Meshkati (Hrsg.), *Human mental workload* (S. 139-183). Amsterdam: North Holland B.V.

Hasselbring, W. (2006). Software-Architektur. *Informatik-Spektrum, 29* (1), 48-51.

Hauß, Y. & Timpe, K. (2000). Automatisierung und Unterstützung im Mensch-Maschine-System. In: K.-P. Timpe, T. Jürgensohn & H. Kolrep (Hrsg.), *Mensch-Maschine-Systemtechnik: Konzepte, Modellierung, Gestaltung, Evaluation* (S. 41-62). Düsseldorf: Symposion.

Heinath, M. (2009). *Hierarchical Task Analysis Mapper: Mustergestützte Erstellung kognitiver Modelle zur Evaluation von Mensch-Maschine-Systemen*. Dissertation. Technische Universität Berlin.

Hilburn, B. (2004). *Cognitive Complexity In Air Traffic Control: A Literature Review*. Technical Report. The Netherlands: Center for Human Performance Research.

Hockey, G. R. J. (1997). Compensatory Control in the Regulation of Human Performance Under Stress and High Workload. A Cognitive-Energetical Framework. *Biological Psychology, 45*, 72-93.

Höfer, A. & Tichy, W. F. (2007). Status of Empirical Research in Software Engineering. In: V. R. Basili, D. Rombach, K. Schneider, B. Kitchenham, D. Pfahl & R. W. Selby (Hrsg.), *Empirical Software Engineering Issues. Critical Assessment and Future Directions. International Workshop, Dagstuhl Castle, June 26-30, 2006. Revised Papers*. Berlin: Springer (LNCS; 4336).

Hollnagel, E. (1998). *Cognitive Reliability and Error Analysis Method (CREAM).* Oxford: Elsevier Science Ltd.

Hollnagel, E. (1993). *Human reliability analysis: Context and control*. London: Academic.

Hudson, S. E., John, B. E., Knudsen, K. & Byrne, M. D. (1999). A Tool for Creating Predictive Performance Models from User Interface Demonstrations. In: *Proceedings UIST'99* (S. 93-102).

IEEE 1278 (1993). Standard for Distributed Simulation – Application protocols.

Jacobson, I., Booch, G. & Rumbaugh, J. (1998). *The Unified Software Development Process*. Reading, MA: Addison Wesley.

John, B. E. & Gray, W. D. (1995). CPM-GOMS: An Analysis Method for Tasks with Parallel Activities. In: I. R. Katz, R. Mack, L. Marks, M. B. Rosson & J. Nielsen (Hrsg.), *CHI 95: Conference on Human Factors in Computing Systems. Mosaic of Creativity* (S. 393-394). New York: ACM Press.

John, B. E. & Kieras, D. E. (1997). Using GOMS for User Interface Design and Evaluation: Which Technique? *ACM Transactions on Computer-Human Interaction, 3* (4), 287-319.

John, B. E. & Kieras, D. E. (1994). The GOMS Family of Analysis Techniques: Tools for Design and Evaluation. Technical Report (CMU-CS-94-181). Pittsburg, PA: School of Computer Science, Carnegie Mellon University. Online verfügbar unter: ftp://www.eecs.umich.edu/people/kieras/GOMS/John-Kieras-TR94.pdf (letzter Zugriff: 2008-05-17).

John, B. E., Prevas, K., Salvucci, D. D. & Koedinger, K. (2004). Predictive Human Performance Modeling Made Easy. In: *Human Factors in Computing Systems: CHI 2004 Conference Proceedings. Vienna, Austria, April 24-29, 2004*. New York: ACM Press.

Johnson, P., Diaper, D. & Long, J. B. (1984). Tasks, skills and knowledge: task analysis for knowledge based descriptions. In: B. Shackel (Hrsg.), *Human-Computer Interaction - INTERACT'84* (S. 499-503). Amsterdam: North-Holland.

Jones, R. M. (1999). Graphical Visualization of Situational Awareness and Mental State for Intelligent Computer-Generated Forces. In: *Proceedings of the Eighth Conference on Computer Generated Forces and Behavioral Representation. Orlando, FL.* (S. 219-222).

Jones, R. M., Crossman, J. A. L., Lebiere, C. & Best, B. J. (2006). An abstract language for cognitive modeling. In: *Proceedings of the 7th International Conference on Cognitive Modelling*. Trieste, Italy: Edizioni Goliardiche.

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P. & Koss, F. V. (1999). Automated Intelligent Pilots for Combat Flight Siumlation. *AI Magazine, 20* (1), 27-41.

Jongman, G. M. G. (1998). How to fatigue ACT-R? In: F. E. Ritter & R. M. Young (Hrsg.), *Proceedings of the second European conference on cognition modelling* (S. 52-57). Nottingham: University Press.

Jürgensohn, T. (2000). Bedienermodellierung. In: K.-P. Timpe, T. Jürgensohn & H. Kolrep (Hrsg.), *Mensch-Maschine-Systemtechnik. Konzepte, Modellierung, Gestaltung, Evaluation* (S. 107-147). Düsseldorf: Symposion.

Jürgensohn, T., Niessen, C. & Leuchter, S. (2000). Bedienermodellierung: Beispiele. In: K.-P. Timpe, T. Jürgensohn & H. Kolrep (Hrsg.), *Mensch-Maschine-Systemtechnik. Konzepte, Modellierung, Gestaltung, Evaluation* (S. 149-177). Düsseldorf: Symposion.

Kanno, T., Nakata, K. & Furuta, K. (2003). A method for team intention inference. *International Journal of Human-Computer Studies, 58*, 393-413.

Kanno, T., Shimizu, T. & Furuta, K. (2006). Modeling and simulation of residents' response in nuclear disaster. *Cognition, Technology & Work, 8* (2), 124-136.

Kieras, D. (1999). *A Guide to GOMS Model Usability Evaluation using GOMSL and GLEAN3.* Technical Report. Ann Arbor, MI: Artificial Intelligence Laboratory, Electrical Engineering and Computer Science Department, University of Michigan.

Kieras, D. E. (1988). Towards a practical GOMS model methodology for user interface design. In: M. Helander (Hrsg.), *The handbook of human-computer interaction* (S. 135-158). Amsterdam: North-Holland.

Kieras, D. E., Ballas, J. & Meyer, D. E. (2001). *Computational Models for the Effects of Localized Sound Cuing in a Complex Dual Task.* EPIC Report (TR-01/ONR-EPIC-13). Ann Arbour: University of Michigan. Online verfügbar unter: ftp://www.eecs.umich.edu/people/kieras/EPIC/TR-EPIC-13.pdf (letzter Zugriff: 2008-05-17).

Kieras, D. E. & Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction, 12* (4), 391-438.

Kieras, D. E. & Polson, G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man Machine Studies, 22*, 365-394.

Kieras, D. E., Wood, S. D., Abotel, K. & Hornof, A. (1995). GLEAN: A Computer-Based Tool for Rapid GOMS Model Usability Evaluation of User Interface Designs. In: *Proceedings of the ACM Symposium on User Interface Software and Technology 1995* (S. 91-100).

Kieras, D. E., Wood, S. D. & Meyer, D. E. (1997). Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. *ACM Transactions on CHI, 4*, 230-275.

Kindsmüller, M. C., Leuchter, S., Schulze-Kissing, D. & Urbas, L. (2004). Modellierung und Simulation menschlichen Verhaltens als Methode der Mensch-Maschine-System-Forschung. *MMI Interaktiv, 7*, 4-16.

Kirschenbaum, S. S. & Gray, W. D. (2000). The Precis of Project Nemo, Phase 2: Levels of Expertise. In: *Twenty-second Annual Conference of the Cognitive Science Society* (S. 753-758). Mahwah, NJ: Erlbaum.

Kobsa, A. & Wahlster, W. (Hrsg.) (1989). *User Models in Dialog Systems*. Berlin: Springer.

Kolrep, H., Gruhlke, F., Röse, K. & Urbas, L. (2004). Mensch-Maschine-Interaktion für mobile Anwedungen im Fahrzeug. In: C. Steffens, M. Thüring & L. Urbas (Hrsg.), *Entwerfen und Gestalten. 5. Berliner Werkstatt Mensch-Maschine-Systeme* (S. 406-415). Düsseldorf: VDI Verlag.

Kontogiannis, T. (2005). Integration of task networks and cognitive user models using coloured Petri nets and its application to job design for safety and productivity. *Cognition, Technology & Work, 7* (4), 241-261.

Kraiss, K.-F. (1998). Benutzergerechte Automatisierung – Grundlagen und Realisierungskonzepte. *at – Automatisierungstechnik 46*, 10/1998, 457-467.

Krems, J. (1996). Kognitive Architektur. In: G. Strube, B. Becker, C. Freksa, U. Hahn, K. Opwis & G. Palm (Hrsg.), *Wörterbuch der Kognitionswissenschaft* (S. 38). Stuttgart: Klett-Cotta.

Künzer, A., Ohmann, F. & Schmidt, L. (2004). Antizipative Modellierung des Benutzerverhaltens mit Hilfe von Aktionsvorhersage-Algorithmen. *MMI-Interaktiv, 7*, 61-83.

Laird, J. E. (2001). It knows what you're going to do: Adding anticipation to a Quakebot. In: *Proceedings of the fifth international conference on Autonomous agents 2001* (S. 385-392). New York, NY: ACM Press.

Laird, J. E., Newell, A. & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33*, 1-64.

Laird, J. E., Rosenbloom, P. S. & Newell, A. (1986). Chunking in Soar: The Anatomy of a General Learning Mechanism. *Machine Learning, 1* (1), 11-46.

Laird, J. E., Yager, E. S., Hucka, M. & Tuck, C. M. (1993). Robo-Soar: an integration of external interaction, planning and learning using Soar. In: *Toward learning robots* (S. 113-129). Cambridge, MA: MIT Press.

Langley, P. (2006). Cognitive architectures and general intelligent systems. *AI Magazine, 27* (2), 33-44.

Lebiere, C. (2005). Constrained Functionality: ACT-R Model. In: K. A. Gluck & R. W. Pew (Hrsg.), *Modeling Human Behavior With Integrated Cognitive Architectures. Comparison, Evaluation, and Validation* (S. 63-111). Mahwah, New Jersey: Lawrence Erlbaum.

Lebiere, C., Archer, R., Schunk, D., Biefeld, E., T., K. & Allender, L. (2003). Using an integrated task network model with a cognitive architecture to assess the impact of technological aids on pilot performance. In: D. Foyle, A. Goodman & B. Hooey (Hrsg.), *Proceedings of the 2003 Conference on Human Performance Modeling of Approach and Landing with Augmented Displays* (S. 165-187). Moffett Field, CA: NASA.

Lebiere, C., Biefeld, E., Archer, R., Archer, S., Allender, L. & Kelley, T. (2002). IMPRINT/ACT-R: Integration of a task network modeling architecture with a cognitive architecture and its application to human error modeling. In: *Proceedings of the Advanced Technologies Simulation Conference*.

Lee, A. Y., Polson, P. G. & Bailey, W. A. (1989). Learning and transfer of measurement tasks. In: *CHI '89: Proceedings of the SIGCHI conference on Human factors in computing systems* (S. 115 - 120). New York: ACM Press.

Lee, S. M., Ravinder, U. & Johnston, J. C. (2005). Developing an agent model of human performance in air traffic control operations using Apex cognitive architecture. In: *WSC '05: Proceedings of the 37th conference on Winter simulation* (S. 979-987). Orlando, FL: Winter Simulation Conference.

Leontiev, A. (1972). *Le Dévelopment du Psychisme*. Paris: Editions Sociales.

Leplat, J. (1988). Task complexity in work situations. In: L. Goodstein, H. Anderson & S. Olsen (Hrsg.), *Tasks, Errors and Mental Models* (S. 105-115). London: Taylor & Francis.

Leuchter, S. (1999). *Integration eines kognitiven Modells in ein Simulationssystem zum Training der Luftverkehrsüberwachung.* Diplomarbeit. Fachbereich Informatik, Technische Universität Berlin.

Leuchter, S. & Jürgensohn, T. (2000). A tutoring system for air traffic control on the basis of a cognitive model. In: J. L. Alty (Hrsg.), *Proceedings of the XVIII European Conference on Human Decision Making and Manual Control* (S. 275-281). Loughborough: Group D.

Leuchter, S., Niessen, C., Eyferth, K. & Bierwagen, T. (1997). Modelling mental Processes of experienced operators during control of a dynamic man-machine system. In: B. B. Borys, G. Johannsen, C. Wittenberg & G. Sätz (Hrsg.), *Proceedings of the XVI. European Annual Conference on Human Decision Making and Manual Control* (S. 268-276). Kassel, Germany: University of Kassel, Institute for Measurement and Automation (IMAT-MMS).

Leuchter, S. & Urbas, L. (2004a). Integrierter agiler Entwicklungsprozess für softwareintensive Mensch-Maschine-Systeme. In: Ch. Steffens, M. Thüring & L. Urbas (Hrsg.), *Entwerfen und Gestalten. 5. Berliner Werkstatt Mensch-Maschine-Systeme* (S. 53-68). Düsseldorf: VDI.

Leuchter, S. & Urbas, L. (2004b). Extending ACT-R for modeling dynamics and timing for operating human-machine-systems. *MMI-Interaktiv, 7*, 38-46.

Leuchter, S. & Urbas, L. (2004c). Useware Engineering mit kognitiven Architekturen. *atp - Automatisierungstechnische Praxis, 46* (9), 68-72.

Leuchter, S. & Urbas, L. (2003). Modeling dynamics and timing for operating human-machine systems. In: F. Detje, D. Dörner & H. Schaub (Hrsg.), *The Logic of Cognitive Systems. Proceedings of the Fifth International Conference on Cognitive Modeling* (S. 279-280). Bamberg: Universitäts-Verlag.

Liu, Y., Feyen, R. & Tsimhoni, O. (2006). Queueing Network-Model Human Processor (QN-MHP): A Computational Architecture for Multitask Performance in Human-Machine Systems. *ACM Transactions on Computer-Human Interaction, 13* (1), 37-70.

Lonsdale, P. R. & Ritter, F. E. (2000). Extending Tcl-Tk to provide a functional exe and hand for the Soar cognitive modelling architecture. In: N. Taatgen & J. Aasman (Hrsg.), *Proceedings of the 3rd International Conference on Cognitive Modelling* (S. 202-209). Veenendaal, The Netherlands: Universal Press.

Louridas, P. (2007). Declarative GUI Programming in Microsoft Windows. *IEEE Software, 4/2007*, 16-19.

Lüdtke, A. (2005). *Kognitive Analyse Formaler Sicherheitskritischer Steuerungssysteme auf der Basis eines integrierten Mensch-Maschine-Modells*. Berlin: Akademische Verlagsgesellschaft.

Lüdtke, A., Möbus, C. & Thole, H.-J. (2002). Cognitive Modelling Approach to Diagnose Over-Simplification in Simulation-Based Training. In: S. A. Cerri, G. Gouarderes & F. Paraguacu (Hrsg.), *Intelligent Tutoring Systems 2002* (S. 469-506). Berlin: Springer-Verlag.

Lugner, P. & Bub, W. (1990). Modellbildung und Modellreduktion: Systematik der konzeptionellen Modellbildung. In: F. Breitenecker (Hrsg.), *Simulationstechnik. 6. Symposium in Wien, September 1990*. Braunschweig: Vieweg.

Martens, A. (2004). *Verteilte Geschäftsprozesse. Modellierung und Verifikation mit Hilfe von Web Services*. Stuttgart: Wiku.

McClain, L. (1983). Interval estimation: Effect of processing demands on prospective and retrospective reports. *Perception and Psychophysics, 34* (2), 185-189.

Meyer, B. (1992). Applying Design by Contract. *IEEE Computer 25* (10), 40-51.

Meyer, D. E., Kieras, D. E., Lauber, E., Schumacher, E. H., Glass, J., Zurbriggen, E., Gmeindl, L. & Apfelblatt, D. (1995). Adaptive Executive Control: Flexible Multiple-Task Performance Without Pervasive Immutable Response-Selection Bottlenecks. *Acta Psychologica, 90*, 163-190.

Misker, J., Taatgen, N. A. & Aasman, J. (2001). Validating a tool for simulating user interaction. In: *Proceedings of the Fourth International Conference on Cognitive Modeling* (S. 163-168). Mahwah, NJ: Lawrence Erlbaum.

Mistrzyk, T. & Szwillus, G. (2008). Modellierung sicherheitskritischer Kommunikation in Aufgabenmodellen. *i-com* , 1/2008. 39-42.

Mitchell, D. K. (2000). *Mental Workload and ARL Workload Modeling Tools*. Technical Note (ARL-TN-161). Aberdeen Proving Ground, MD 21005-5425: Army Research Laboratory.

Möbus, C., Schröder, O. & Thole, H. (1994). Diagnosing and Evaluating the Acquisition Process of Programming Schemata. In: J. Greer & G. McCalla (Hrsg.), *Student Modelling: The Key to Individualized Instruction, Proceedings of the NATO Advanced Research Workshop on Student Modelling* (S. 211-264). Berlin: Springer.

Moore, B., Dean, D., Gerber, A., Wagenknecht, G. & Vanderheyden, Ph. (2004). *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*. IBM (Redbooks).

Moran, T. P. (1981). The Command Language Grammar: A representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies, 15*, 3-50.

Morgan, G. P., Cohen, M. A., Haynes, S. R. & Ritter, F. (2005). Increasing efficiency of the development of user models. In: *Proceedings of the IEEE System Information and Engineering Design Symposium*.

Mori, G., Paterno, F. & Santoro, C. (2002). CTTE: support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering, 28* (8), 797- 813.

Naumann, A. & Rötting, M. (2007). Digital Human Modeling for Design and Evaluation of Human-Machine Systems. *MMI-Interaktiv, 12*, 27-35.

Naur, P. & Randell, B. (1969). *Software Engineering.* Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th Oc-tober 1968 (Brussels: Scientific Affairs Division, NATO).

Nekrasova, L. (2005). *Programmvisualisierung von Produktionssystemen am Beispiel der kognitiven Modellierungssprache ACT-R.* Diplomarbeit. Institut für Softwaretechnik und theoretische Informatik, Technische Universität Berlin.

Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.

Newell, A. (1980). Physical symbol systems. *Cognitive Science, 4*, 135-183.

Newell, A., Rosenbloom, P. S. & Laird, J. E. (1993). Symbolic architectures for cognition. In: *Foundations of cognitive neuroscience* (S. 93-131). Cambridge, MA: MIT Press.

Newell, A. & Simon, H. A. (1976). Computer science as empirical inquiry: Symbols and search. *Communications of the ACM, 19*, 113-126.

Niessen, C. & Eyferth, K. (2001). A model of the air traffic controller's picture. *Safety Science, 37*, 187-202.

Niessen, C., Leuchter, S. & Eyferth, K. (1998). A psychological model of air traffic control and its implementation. In: F. E. Ritter & R. M. Young (Hrsg.), *Proceedings of the Second European Conference on Cognitive Modelling (ECCM-98)* (S. 104-111). Nottingham: Nottingham University Press.

OMG (2006). *Business Process Modeling Notation Specification*. OMG Final Adopted Specification (dtc/06-02-01). OMG. Online verfügbar unter: http://www.omg.org/docs/dtc/06-02-01.pdf (letzter Zugriff: 2008-05-17).

OMG (2000). *OMG Unified Modeling Language Specification. Version 1.3*. Document (formal/00-03-01). OMG. Online verfügbar unter: http://www.omg.org/docs/formal/00-03-01.pdf (letzter Zugriff: 2008-05-17).

Onken, R., Otto, H. & von Garrel, U. (2001). Adaptive Modellierung des Fahrerverhaltens - Ein Kernelement für die kognitive Kooperation bei zukünftiger Fahrerassistenz. In: T. Jürgensohn & K.-P. Timpe (Hrsg.), *Kraftfahrzeugführung* (S. 81-93). Berlin: Springer-Verlag.

Opwis, K. & Spada, H. (1994). Modellierung mit Hilfe Wissensbasierter Systeme. In: *Enzyklopädie der Psychologie: Forschungsmethoden* (S. 200-248).

Pahl, G. & Beitz, W. (1993). *Konstruktionslehre. Methoden und Anwendungen. Dritte, neubearbeitete und erweiterte Auflage*. Berlin: Springer-Verlag.

Pape, N. (2009). *Berechenbare quantitative Modelle der Dauerschätzung*. Dissertation. Technische Universität Berlin.

Pape, N., Dzaack, J., Leuchter, S. & Urbas, L. (2005). How to integrate time-duration estimation in ACT-R/PM. In: *Proceedings of the 12th Annual ACT-R Workshop*. Trieste, Italy.

Parnas, D. L. (1972a). A Technique for Software Module Specification with Examples. *Communications of the ACM*, 15 (5), 330-336.

Parnas, D. L. (1972b). On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15 (12), 1053-1058.

Paternò, F. (1999). *Model-Based Design and Evaluation of Interactive Applications*. Berlin: Springer.

Payne, S. J. & Green, T. R. G. (1986). Task-Action Grammars: a model of the mental representation of task languages. *Human-Computer Interaction, 2* (2), 93-133.

Pew, R. W. (2007). Some History of Human Performance Modeling. In: W. D. Gray (Hrsg.), *Integrated Models of Cognitive Systems* (S. 29-44). New York: Oxford University Press.

Pew, R. W. & Mavor, A. S. (Hrsg.) (1998). *Modeling Human and Organizational Behavior. Applications to Military Simulations*. Washington, D.C.: National Academic Press.

Post, E. (1943). Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics, 65*, 197-268.

Pree, W. (1997). *Komponentenbasierte Softwareentwicklung mit Frameworks*. Heidelberg: dpunkt-Verlag.

Price, B., Baecker, R. & Small, I. (1993). A Principled Taxonomy of Software Visualization. *Journal of Visual Languages and Computing, 4* (3), 211-266.

Rasmussen, J. (1986). *Information Processing and Human-Machine Interaction*. New York: Elsevier.

Rasmussen, J. (1983). Skills, Rules, and Knowledge; Signals, Signs and Symbols, and Other Distinctions in Human Performance Models. *IEEE Transactions on Systems, Man, and Cybernetics, 13* (3), 257-266.

Reichart, G. (1992). THERP (Technique for Human Error Rate Prediction). In: H. Bubb (Hrsg.), *Menschliche Zuverlässigkeit. Definitionen, Zusammenhänge, Bewertung* (S. 106-115). Landsberg: ecomed-Fachverlag.

Remington, R., Matessa, M., Freed, M. & Lee, S. M. (2003). Using APEX/CPM-GOMS to develop Human-Like Software Agents. In: *Workshop on Humans, Multi-Agent Systems (AAMAS-03). Melbourne, Australia, July 14, 2003*.

Reuther, A. (2003). *useML - Systematische Entwicklung von Maschienenbediensystemen mit XML.* Dissertation. Universität Kaiserslautern.

Ritter, F. E., Baxter, G. D., Jones, G. & Young, R. M. (2001). User interface evaluation: How cognitive models can help. In: J. Carroll (Hrsg.), *Human-computer interaction in the new millenium* (S. 125-147). Reading, MA: Addison-Wesley.

Ritter, F. E., Baxter, G. D., Jones, G. & Young, R. M. (2000). Supporting Cognitive Models as Users. *ACM Transactions on Computer-Human Interaction, 7* (2), 141-173.

Ritter, F. E., Haynes, S. R., Cohen, M., Howes, A., John, B., Best, B., Lebiere, C., Jones, R. M., Crossman, J., Lewis, R. L., Amant, R. S., McBride, S. P., Urbas, L., Leuchter, S. & Vera, A. (2006a). High-level Behavior Representation Languages Revisited. In: *Proceedings of the 7th International Conference on Cognitive Modelling*. Trieste, Italy: Edizioni Goliardiche.

Ritter, F. E., van Rooy, D., Amant, R. S. & Simpson, K. (2006b). Providing User Models Direct Access to Interfaces: An Exploratory Study of a Simple Interface With Implications for HRI and HCI. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 36* (3), 592-601.

Ritter, F. E., Shadbolt, N. R., Elliman, D., Young, R., Gobet, F. & Baxter, G. D. (2003). *Techniques for modeling human performance in synthetic environments: A supplementary review*. State of the Art Reports. Wright-Patterson Air Force Base, OH: Human Systems Information Analysis Center.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM, 12* (1), 23-41.

Salvucci, D. D. (2006). Modeling driver behavior in a cognitive architecture. *Human Factors, 48*, 362-380.

Salvucci, D. D. (2001). An integrated model of eye movements and visual encoding. *Cognitive Systems Research, 1*, 201-220.

Salvucci, D. D. & Lee, F. J. (2003). Simple cognitive modeling in a complex cognitive architecture. In: *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'03)* (S. 265-272). New Yorck: ACM Press.

Santoro, T., Kieras, D. & Campbell, G. (2000). GOMS modeling application to watchstation design using the GLEAN tool. In: *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference* (S. 964-973). Arlington, VA: National Training Systems Association.

Schäfer, D. (2001). *Context-Sensitive Speech Recognition in the Air Traffic Control Simulation*. Ph.D. Thesis and ECC Note (02/2001). EUROCONTROL Experimental Centre.

Schmidtke, H. (1976). *Handbuch der Ergonomie*. München: Carl Hanser Verlag.

Schoelles, M. J. (2002). *Simulating Human Users in Dynamic Environments.* Unpublished doctoral dissertation. Fairfax, Virginia: George Mason University. Online verfügbar unter: http://www.rpi.edu/~grayw/pubs/papers/Schoelles02_diss.pdf (letzter Zugriff: 2008-05-17).

Schoppek, W. & Boehm-Davis, D. A. (2004). Opportunities and challenges of modeling user behavior in complex real world tasks. *MMI interaktiv, 7*, 47-60.

Schoppek, W., Holt, R. W., Diez, M. S. & Boehm-Davis, D. A. (2001). Modeling behavior in complex situations - the example of flying an automated aircraft pilot. In: *Proceedings of the Fourth International Conference on Cognitive Modeling*. Mahwah, NJ: Lawrence Erlbaum Associates.

Schulze-Kissing, D., Urbas, L. & van der Meer, E. (2005). Befunde zur Zeitdauerschätzung in der Mensch-Maschine-Interaktion. In: K. Karrer, B. Gauss & C. Steffens (Hrsg.), *Mensch-Maschine-Systemtechnik aus Forschung und Praxis* (S. 265-282). Düsseldorf: Symposion.

Scott, E. & Marshall, S. P. (1998). *A Hybrid Model of Novices' Performance on a Simulated CIC Task.* ACT-R Workshop 1998. Pitsburgh, PA: Carnegie Mellon University.

Shah, K., Rajyaguru, S., Amant, R. S. & Ritter, F. E. (2003). Image processing for cognitive modeling in dynamic gaming environments. In: F. Detje, D. Doerner & H. Schaub (Hrsg.), *Proceedings of the Fifth International Conference on Cognitive Modeling*. Bamberg, Germany: Universitäts-Verlag.

Shavor, S., D'Anjou, J., Fairbrother, S., Kehn, D., Kellerman, J. & McCarthy, P. (2004). *Eclipse Anwendungen und Plug-Ins mit Java entwickeln*. München: Addison-Wesley.

Shu, Y. & Furuta, K. (2005). An inference method of team situation awareness based on mutual awareness. *Cognition, Technology & Work, 7* (4), 272-287.

Smith, B. R. & Tyler, S. W. (1997). The Design and Application of MIDAS: A Constructive Simulation for Human-System Analysis. In: *Proceedings of the 2nd Technology and Training (SIMTECT) Conference, Canberra, Australia*. Online verfügbar unter:

http://human-factors.arc.nasa.gov/dev/www-midas/documents/Smith_Tyler_97.pdf (letzter Zugriff: 2008-05-17).

Soar Community (o.J.). *SML FAQ.* Ann Arbor: University of Michigan. Online verfügbar unter: http://winter.eecs.umich.edu/soarwiki/SML_FAQ (letzter Zugriff: 2008-05-17).

Sodhi, M., Reimer, B. & Llamazares, I. (2002). Glance Analysis of Driver Eye Movements to Evaluate Distraction. *Behavior Research Methods, Instruments and Computing, 34* (4), 529-538.

Stanton, N. A. & Baber, C. (2006). The ergonomics of command and control. *Ergonomics, 49* (12 & 13), 1131-1138.

Strube, G. (1996). Kognition. In: G. Strube, B. Becker, C. Freksa, U. Hahn, K. Opwis & G. Palm (Hrsg.), *Wörterbuch der Kognitionswissenschaft* (S. 303-317). Stuttgart: Klett-Cotta.

Swain, A. & Guttmann, H. (1983). *Handbook of Human Reliability Analysis Procedure*. Report (NUREG/CR-4772). Washington: U.S. Nuclear Regulatory Commission.

Taatgen, N. A. (2002). A model of individual differences in skill acquisition in the Kanfer-Ackerman Air Traffic Control Task. *Cognitive Systems Research, 3* (1), 103-112.

Taatgen, N. A. (2001). A model of individual differences in learning Air Traffic Control. In: *Proceedings of the Fourth International Conference on Cognitive Modeling* (S. 211-216). Mahwah, NJ: Lawrence Erlbaum Associates.

Taatgen, N. A., Anderson, J., Dickison, D. & van Rijn, H. (2005). Time Interval Estimation: Internal Clock or Attentional Mechanism? In: B. Bara, L. Barsalou & M. Bucciarelli (Hrsg.), *Proceedings of the 27th Annual Conference of the Cognitive Science Society* (S. 2122-2127). Mahwah, NJ: Lawrence Erlbaum Associates.

Taatgen, N. A. & Lee, F. J. (2003). Production Compilation: A Simple Mechanism to Model Complex Skill Acquisition. *Human Factors, 45* (1), 61-76.

Taatgen, N. A., van Rijn, H. & Anderson, J. (2004). Time Perception: Beyond Simple Interval Estimation. In: *Proceedings of the 6th Interantional Conference on Cognitive Modeling* (S. 296-301).

Taatgen, N. A. & Wallach, D. (2002). Whether skill acquisition is rule or instance based is determined by the structure of the task. *Cognitive Science Quarterly, 2* (2), 163-204.

Tauber, M. (1990). ETAG: Extended Task Action Grammar: a language for the description of the user's task language. In: D. Diaper, D. Gilmore, G. Cockton & B. Shackel (Hrsg.), *Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction* (S. 163-168). Amsterdam: North-Holland.

Taylor, G. & Crossman, J. (2006). *SoarML: A Graphical Modeling Language for Agents.* 26th Soar Workshop, May 22-26, 2006. Ann Arbor Michigan. Online verfügbar unter: http://www.eecs.umich.edu/~soar/sitemaker/workshop/26/proceedings/Crossman.pdf (letzter Zugriff: 2008-05-17).

Timpe, K.-P. & Jürgensohn, T. (2000). Perspektiven der Mensch-Maschine-Systemtechnik. In: K.-P. Timpe, T. Jürgensohn & H. Kolrep (Hrsg.), *Mensch-Maschine-Systemtechnik. Konzepte, Modellierung, Gestaltung, Evaluation* (S. 337-347). Düsseldorf: Symposion.

Timpe, K. & Kolrep, H. (2000). Das Mensch-Maschine-System als interdisziplinärer Gegenstand. In: K.-P. Timpe, T. Jürgensohn & H. Kolrep (Hrsg.), *Mensch-Maschine-Systemtechnik: Konzepte, Modellierung, Gestaltung, Evaluation* (S. 9-40). Düsseldorf: Symposion.

Tor, K., Haynes, S. R., Ritter, F. E. & Cohen, M. A. (2004). Categorical data displays generated from three cognitive architectures illustrate their behavior. In: *Proceedings of the International Conference on Cognitive Modeling* (S. 302-307). Mahwah, NJ: Lawrence Erlbaum.

Urbas, L. (1999). *Entwicklung und Realisierung einer Trainings- und Ausbildungsumgebung zur Schulung der Prozessdynamik und des Anlagenbetriebs im Internet.* Düsseldorf: VDI Verlag.

Urbas, L., Heinath, M. & Leuchter, S. (2007). Bedienermodellgestützte Bewertung des Ablenkungspotenzials von Komfortsystemen im KFZ in frühen Phasen der Systementwicklung. *i-com, 2/2007*, 21-29.

Urbas, L. & Leuchter, S. (2002). Modellgestütztes situation awareness-Training für komplexe und dynamische Mensch-Maschine-Systeme. In: R. Marzi, V. Karavezyris, H. .. H. Erbe & K.-P. Timpe (Hrsg.), *Bedienen und Verstehen. 4. Berliner Werkstatt Mensch-Maschine-Systeme. 10.-12. Oktober 2001* (S. 71-85). Düsseldorf: VDI-Verlag (Fortschritt-Berichte, Reihe 22; 8).

Urbas, L., Schulze-Kissing, D. & Leuchter, S. (2005). Werkzeuge für die Erstellung kognitiver Nutzermodelle. In: K. Karrer, B. Gauss & C. Steffens (Hrsg.), *Mensch-Maschine-Systemtechnik aus Forschung und Praxis* (S. 247-264). Düsseldorf: Symposion.

Urbas, L., Schulze-Kissing, D., Leuchter, S., Kiefer, J., Dzaack, J. & Heinath, M. (2005). Erfassung residualer Ressourcen Profile zur Modellierung von Fahrerassistenzsystem-Nutzung unter Fahrbedingungen. In: *TEAP 2005*.

VDI 2221 (1993). *Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte*.

VDI 3633, Blatt 6 (2001). *Simulation von Logistik-, Materialfluss- und Produktionssystemen. Abbildung des Personals in Simulationsmodellen.*

Vicente, K. (1995). Cognitive work analysis: implications for microworld research on human-machine interaction. In: *IEEE International Conference on Systems, Man and Cybernetics, 1995. 'Intelligent Systems for the 21st Century'* (Vol. 4, S. 3432-3436). IEEE Systems, Man, and Cybernetics Society.

Vicente, K., Christoffersen, K. & Pereklita, A. (1995). Supporting operator problem solving through ecological interface design. *IEEE Transactions on Systems Man and Cybernetics, 25* (4), 529-545.

Wallach, D. (1996). *Komplexe Regelungsprozesse: Eine kognitionswissenschaftliche Analyse*. Wiesbaden: DUV.

Wandmacher, J. (2002). *GOMS-Analysen mit GOMSED*. Institut für Psychologie, Technische Universität Darmstadt. Online verfügbar unter: http://www.tu-darmstadt.de/fb/fb3/psy/kogpsy/GOMS-Analysen%20mit%20%20GOMSED.pdf (letzter Zugriff: 2008-05-17).

Werther, B. (2006). *Kognitive Modellierung mit Farbigen Petrinetzen zur Analyse menschlichen Verhaltens*. Dissertation. Fachbereich Maschinenbau, TU Carolo Wilhelmina Braunschweig.

Werther, B. & Lorenz, B. (2003). Modellbasierte Bewertung menschlicher Informationsverarbeitung mit höheren Petrinetzen. In: Ch. Steffens, M. Thüring & L. Urbas (Hrsg.), Entwerfen und Gestalten. *5. Berliner Werkstatt Mensch-Maschine-Systeme* (S. 181-196). Düsseldorf: VDI-Verlag (Fortschritt-Berichte, Reihe 22; 16).

Wickens, C. D. (1984). *Engineering Psychology and Human Performance*. Columbus, OH: Charles E. Merrill Publishing Co.

Wintermute, S. (2006). Visual Attention for a Real-Time Strategy Game. *26th Soar Workshop May 22-26, 2006. Ann Arbor, Michigan*. Online verfügbar unter: http://www.eecs.umich.edu/~soar/sitemaker/workshop/26/proceedings/Wintermute.pdf (letzter Zugriff: 2008-05-17).

Wirth, N. (1971). Program Development by Stepwise Refinement. *Communications of the ACM 14* (4), 221-227.

Wood, S. D. (2000). *Extending GOMS to human error and applying it to error-tolerant design.* Ph.D. Thesis. Ann Arbor: University of Michigan.

Wood, S. D. & Kieras, D. E. (2002). Modeling Human Error for Experimentation, Training, and Error-Tolerant Design. In: *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC), Orlando, FL*.

Xu, J. (2006). ORTS: A Case Study of Multitasking in Soar. *26th Soar Workshop May 22-26, 2006. Ann Arbor, Michigan*. Online verfügbar unter: http://www.eecs.umich.edu/~soar/sitemaker/workshop/26/proceedings/Xu2.pdf (letzter Zugriff: 2008-05-17).

Zachary, W., Ryder, J., Hicinbothom, J. & Bracken, K. (1997). The Use of Executable Cognitive Models in Simulation-based Intelligent Embedded Training. In: *Proceedings of Human Factors and Ergonomics Society 41st Annual Meeting* (S. 1118-1122). Santa Monica, CA: Human Factors and Ergonomics Society.

Zachary, W., Ryder, J., Stokes, J., Glenn, F., Mentec, J. L. & Santarelli, T. (2005). A COGNET/IGEN Cognitive Model That Mimics Human Performance and Learning in a Simulated Work Environment. In: K. A. Gluck & R. W. Pew (Hrsg.), *Modeling Human Behavior with Integrated Cognitive Architectures. Comparison, Evaluation, and Validation* (S. 113-175). Mahwah, NJ: Lawrence Erlbaum.

Zakay, D. & Block, R. A. (1997). Temporal cognition. *Current Directions in Psychological Science, 6* (1), 12-16.

Anhang A
XML-Repräsentationsformat für GUI-Beschreibungen in Agimap (DTD)

Modelle für Agimap GUIs (s. Abschnitt 6.6) werden als XML anhand der Document Type Definition (DTD) in Quelltext Anhang-1 repräsentiert.

```
<!ELEMENT AGIMAP (GUI, MAP)>
<!ATTLIST AGIMAP        bgimg CDATA #REQUIRED
                        width CDATA #REQUIRED
                        height CDATA #REQUIRED
                        update-cycle CDATA #REQUIRED
                        host CDATA #REQUIRED
                        port CDATA #REQUIRED>
<!ELEMENT GUI (Button|Text)*>
<!ELEMENT MAP (MapText|MapText2Color|MapNumber2Pos)*>
<!ELEMENT Button EMPTY>
<!ATTLIST Button        actid CDATA #REQUIRED
                        action CDATA #REQUIRED
                        text CDATA #IMPLIED
                        x CDATA #REQUIRED
                        y CDATA #REQUIRED
                        width CDATA #REQUIRED
                        height CDATA #REQUIRED>
<!ELEMENT Text EMPTY>
<!ATTLIST Text          actid CDATA #REQUIRED
                        text CDATA #REQUIRED
                        x CDATA #REQUIRED
                        y CDATA #REQUIRED>
<!ELEMENT MapText EMPTY>
<!ATTLIST MapText       actid CDATA #REQUIRED
                        tag CDATA #REQUIRED
                        x CDATA #REQUIRED
                        y CDATA #REQUIRED
                        color CDATA #REQUIRED>
<!ELEMENT MapNumber2Pos EMPTY>
<!ATTLIST MapNumber2Pos actid CDATA #REQUIRED
                        tag CDATA #REQUIRED
                        x CDATA #REQUIRED
                        y CDATA #REQUIRED
                        xrange CDATA #REQUIRED
                        yrange CDATA #REQUIRED
                        signtext CDATA #REQUIRED
                        minvalue CDATA #REQUIRED
                        maxvalue CDATA #REQUIRED
                        color CDATA #REQUIRED>
<!ELEMENT MapText2Color EMPTY>
<!ATTLIST MapText2Color actid CDATA #REQUIRED
                        tag CDATA #REQUIRED
                        x CDATA #REQUIRED
                        y CDATA #REQUIRED
                        signtext CDATA #REQUIRED
                        offtext CDATA #REQUIRED
                        offcolor CDATA #REQUIRED
                        oncolor CDATA #REQUIRED>
```

Quelltext Anhang-1: DTD zur Repräsentation von Agimap GUI-Beschreibungen

Anhang B
XML-Repräsentationsformat für ACT-R-Modelle (DTD)

ACT-R-Modelle werden für VisualPlugin (s. Abschnitt 6.4) als XML anhand der Document Type Definition (DTD) Quelltext Anhang-2 repräsentiert. Es handelt sich um eine Modifikation der Sprache, die Harrison (2002) für jACT-R verwendet.

```
<!ELEMENT model ( declarative-memory, procedural-memory, buffer+)>
<!ELEMENT declarative-memory ( chunk-type+, chunk+)>
<!ELEMENT procedural-memory ( production+)>
<!ELEMENT chunk-type (slot+)>
<!ATTLIST chunk-type name CDATA #REQUIRED>
<!ELEMENT chunk (slot+)>
<!ATTLIST chunk name CDATA #REQUIRED type CDATA #REQUIRED>
<!ELEMENT production ( condition, action)>
<!ATTLIST production name PCDATA #REQUIRED>
<!ELEMENT condition (match+)>
<!ELEMENT match (slot+)>
<!ATTLIST match buffer CDATA #REQUIRED type CDATA #REQUIRED>
<!ELEMENT slot EMPTY>
<!ATTLIST slot name CDATA #REQUIRED equals CDATA #IMPLIED not CDATA
          #IMPLIED>
<!ELEMENT action (modify*, add*, remove*)>
<!ELEMENT modify ( slot+)>
<!ATTLIST modify buffer CDATA #REQUIRED>
<!ELEMENT add (slot*)>
<!ATTLIST add buffer CDATA #REQUIRED chunk CDATA #IMPLIED type CDATA
          #IMPLIED>
<!ELEMENT remove EMPTY>
<!ATTLIST remove buffer CDATA #REQUIRED>
<!ELEMENT buffer EMPTY>
<!ATTLIST buffer name CDATA #REQUIRED chunk CDATA #REQUIRED>
```

Quelltext Anhang-2: DTD zur Repräsentation von ACT-R-Modellen als XML

Anhang C
Socketanbindung von CommonLISP[8]

Ein im BSD-Unix eingeführter Mechanismus, mit dem Protokoll TCP/IP zu kommunizieren, sind Sockets. Ein Socket wird auf Benutzerebene wie eine Datei behandelt. Konzeptionell hat man dann einen Deskriptor, der die Verbindung zu einem Lese- und einem Schreibstrom ermöglicht, wodurch zwei Prozesse über ein Netzwerk verbunden sind. Ein logischer Socket wird von einem Server bedient und von mehreren Clients genutzt. Dazu öffnet der Server einen Socket auf seiner Seite, der über eine Portnummer auf seinem Rechner identifiziert wird. Ein Client, der sich auf diesem Socket mit dem Server verbinden möchte, öffnet eine Verbindung zu diesem logischen Socket, also der Portnummer auf diesem Rechner. Das registriert der Server, spaltet einen Kindprozess ab und erzeugt eine physikalische Verbindung, mit der ab jetzt Client und Server Kind kommunizieren. Der logische Socket (Port auf Server-Rechner) ist für den nächsten Client verfügbar. Für das Ansprechen des Sockets werden analog zu Filedeskriptoren Socketdeskriptoren erzeugt, auf die neben den socketspezifischen Systemcalls (`accept`, `bind`, `connect`, `getsockname`) die für Dateien üblichen Aufrufe (`open`, `read`, `write`, `close`) angewendet werden können. Aufgrund der Privatheit der Simulation ist es nicht erforderlich, dass sich mehrere Clients gleichzeitig mit dem Server verbinden. Normalerweise ist es unter Unix/Linux einfach, mit Sockets umzugehen. Es gibt eine einheitliche Schnittstelle zu entsprechenden Funktionen des Betriebssystems. Leider sind solche systemnahen Funktionen auch nur mit systemnahen Programmierumgebungen wie C direkt benutzbar. ACT-R setzt jedoch auf LISP auf, einer stark von der Systemumgebung abstrahierenden funktionalen Programmiersprache. Unter dem Standard CommonLISP gibt es keine Möglichkeit, direkt auf Sockets oder beliebige Systemcalls im Allgemeinen zuzugreifen.

Ein Ansatz, um Kommunikationsmechanismen über TCP/IP-Sockets von systemfernen Programmiersprachen aus zu benutzen, besteht darin, systemnahe Funktionen über ein *Foreign Interface* in die LISP Umgebung einzubinden. Der Nachteil dieser Lösung ist, dass solche Interfaces für CommonLISP nicht standardisiert sind und die Implementierung nicht mehr zwischen unterschiedlichen LISP Systemen und Rechnerarchitekturen portabel ist. Dieses Problem kann durch die Nutzung von UNIX-Mechanismen und -Konventionen umgangen werden: In nahezu jeder Programmierumgebung besteht die Möglichkeit, von einem Standarddeskriptor Daten zu lesen (Standardeingabe) und auf einen Standarddeskriptor Daten auszugeben (Standardausgabe). Durch Standardmechanismen (`close` + `dup`) werden diese Kanäle mit einem neuen systemnahen Programm mit einem zuvor erzeugten Socketkanal verbunden. Danach

[8] Aus Leuchter (1999)

wird die LISP-Umgebung in diesen Prozessraum geladen. Nun kann von der LISP-Umgebung aus über Standardein- und -ausgabe transparent auf den Socket zugegriffen werden. Diese Technik wird für MoFL (s. 5.1) mit einem Perl-Programm realisiert. Diese Technik ist sowohl zwischen allen unterschiedlichen Programmierumgebungen wie CommonLISP als auch zwischen vielen unterschiedlichen Rechnerarchitekturen und Betriebssystemen portabel.

Das Konzept nutzt die Struktur der Prozessverwaltung in UNIX-Systemen aus. Jeder Prozess läuft in einem Kontext ab. Der Kontext ist eine Datenstruktur im Speicher des Betriebssystems. Er hat einen statischen und einen dynamischen Teil, in dem Informationen zur Prozesskontrolle und zum Zustand der sequentialisierten Abarbeitung der Prozesse gespeichert sind. Im statischen Teil des Prozess-Kontextes wird zwischen dem Benutzerkontext, in dem Programmcode, Daten und Stack des Prozesses repräsentiert sind, und dem Systemkontext unterschieden. Der Systemkontext beinhaltet einen Prozesstabelleneintrag, der Informationen über den Prozesszustand, Steuerinformationen und einen Verweis in den dynamischen Teil des Prozesskontextes enthält, den u-Bereich und die p-Regionstabelle. Die p-Regionstabelle enthält Verweise auf die für den jeweiligen Prozess gültigen Teile des Benutzerkontextes. Der u-Bereich enthält insbesondere Verweise auf die Dateitabelle. Beim Benutzen des Systemaufrufes `exec` wird ein neuer Benutzerkontext angelegt und die p-Regionstabelle entsprechend geändert. Der Effekt dieses Systemaufrufes ist, dass das Programm, das in dem aufrufenden Prozess abläuft, durch ein anderes Programm ersetzt wird. Im Kindprozess ist am Anfang das Programm des Elternprozesses. Über den `exec`-Systemaufruf wird das eigentlich zu startende Programm in den Prozess des Kindes geladen.

Eine Eigenschaft dieser Methode ist, dass der u-Bereich beim exec-Systemaufruf nicht geändert wird. Dadurch bleiben Modifikationen an der Dateitabelle erhalten. Die Dateitabelle enthält Verweise auf geöffnete Dateien. Sie ist als Array implementiert. Zugriffe auf Dateien sind über den Index des entsprechenden Eintrages in der Dateitabelle zugreifbar. Wird eine Datei mit dem Systemaufruf close geschlossen, bleibt der Eintrag in der Dateitabelle leer, bis eine neue Datei geöffnet wird. Ihr Dateideskriptor wird in die erste leere Zelle der Dateideskriptortabelle geschrieben.

Der Systemaufruf `dup` bewirkt, dass ein Dateideskriptor, der über den Index in der Dateitabelle angegeben wird, dupliziert und seine Dateireferenz in die erste leere Zelle der Dateitabelle kopiert wird. Sockets werden in der Dateitabelle mit Dateideskriptoren eingetragen. In einer systemnahen Programmiersprache, die Funktionen zur Netzwerkkommunikation bereitstellt, kann daher ein Programm geschrieben werden, das eine Socketverbindung öffnet. Durch Öffnen des Sockets und Manipulation der Einträge in der Dateitabelle mit den Systemaufrufen `close` und `dup` werden Dateideskriptoren so verbogen, dass vor dem Aufruf des `exec` die Standardeingabe des Prozesses vom Socket liest und die Standardausgabe auf den Dateideskriptor des Sockets schreibt.

Im Quelltext Anhang-3 wird das Prinzip dieses Verfahrens in Perl umgesetzt.

```
#!/usr/bin/perl
require 5.002;
use Socket; use Carp; use POSIX ":sys_wait_h";
$ServerMachine = "server.domain.tld";
$ServerPort    = 4711;
socket(Sock, PF_INET, SOCK_STREAM, getprotobyname('tcp')) ||
            die "$0 socket: $!";
print(STDERR "Trying to connect to $ServerMachine on
            $ServerPort...\n");
connect(Sock,sockaddr_in($ServerPort,inet_aton($ServerMachine))) ||
            die "$0 connect: $!";
print(STDERR "Connection to $ServerMachine on $ServerPort\n");
system("/bin/rm mofl.out");
if (!defined ($pid = fork())) {
    die "$0 fork: $!\n";
} elsif (!$pid) {
    $socket = fileno(Sock);
    open(STDOUT,">&$socket"); $| = 1;
    open(STDIN,"<&$socket");
    exec 'gcl -f MoFL.lsp' || die "$0 exec: $!";
} else {
    while (! -e "mofl.out") {}
    open(MOFLLOG,"<mofl.out");
    $SIG{'INT'} = sub {
            print STDERR "trying to kill mofl*\n";
            print Sock "\r\nMOFL*: quit\r\n";
            close(MOFLLOG);
            close(Sock);
            print (STDERR "Connection closed.\n");
            kill 'KILL', $pid;
            print (STDERR "killed mofl*.\n");
    }
    while (waitpid($pid,&WNOHANG) == 0) {
            if (defined($line = <MOFLLOG>)) {
                print STDERR $line;
            }
    }
    print Sock "\r\nMOFL*: quit\r\n";
    close(MOFLLOG);
    close(Sock);
    print (STDERR "Connection closed.\n");
}
```

Quelltext Anhang-3: Perl-Programm zur Herstellung einer Socketverbindung, die von ACT-R (gcl: GNU CommonLISP) aus über StdInput/StdOutput angesprochen werden kann (Modell MoFL)

Anhang D
Kommunikationsprotokoll MoFL

Die Kommunikation spielt sich in zwei voneinander getrennten Phasen ab. In einem ersten Schritt muss der aktuelle Zustand des Luftraums zum kognitiven Modell übertragen werden („Startphase“), damit eine Repräsentation des simulierten *pictures* initialisiert werden kann. Bei den operationellen Abläufen in der Flugsicherung gibt es auch eine Übergabephase zwischen zwei Schichten. Dabei geht der frische Flugloste den Radarschirm systematisch durch und beginnt, den aktuellen Verkehr zu verstehen. Die anschließende „Betriebsphase“ ist durch das Lotsen des Verkehrs im engeren Sinne geprägt. Für beide Phasen wurden Sprechakte und ein Protokollablauf identifiziert. Tabelle Anhang-1 zeigt die Sprechakte getrennt nach Phase.

Tabelle Anhang-1: Sprechakte für die Kommunikation zwischen kognitiven Modell und Aufgabensimulation

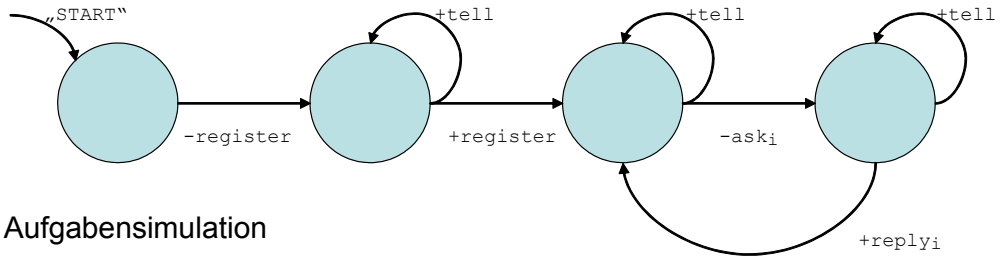
Startphase	Betriebsphase
`Register`: Anmelden Client/Server	`ask-with-reply`: Frage nach Luftfahrzeug-Parameter, Rauminformation; Anweisung an ein Luftfahrzeug
`tell`: Melden eines Luftfahrzeuges	`tell`: Anmelden eines Luftfahrzeuges
`register`: Ende der Startphase signalisieren	

Das kognitive Modell initiiert die Kommunikation mit der Luftraumsimulation unabhängig von der Art des Verbindungsaufbaus. Es ist also egal, welcher der Kommunikationspartner als Client oder als Server agiert. Nachdem eine Verbindung aufgebaut ist, muss sich das kognitive Modell bei der Luftraumsimulation anmelden (`register`). Danach sendet die Luftraumsimulation Informationen über Luftfahrzeuge (`tell`). Dieser Kommunikationsfluss von Aufgabensimulation zu kognitivem Modell kann von beiden Seiten aus nicht unterbrochen werden. Ein anderer Kommunikationsbedarf besteht zu diesem Zeitpunkt nicht. Das Ende der Startphase muss für das kognitive Modell erkennbar sein. Dazu sendet die Luftraumsimulation ein Token, das nicht in den Informationen des `tell`-Stroms vorkommen kann (`register`). `ask-with-reply` wird in der Startphase nicht gesendet. Ein `ask-with-`

`reply` kann von beiden Seiten aus initiiert werden (`ask`) und muss von der anderen Seite beantwortet werden (`reply`). Eine Antwort kann je nach konkreter Ausprägung durch eine Antwort oder eine Bestätigung erfolgen.

In der Betriebsphase muss zwischen synchroner und asynchroner Kommunikation unterschieden werden. Der Großteil der Kommunikation zwischen Fluglotsen und Piloten wird vom Lotsen initiiert und vom angesprochenen Piloten beantwortet. Es muss jedoch Ausnahmen geben, da sich neu einfliegende Luftfahrzeuge ungefragt beim Lotsen über Sprechfunk anmelden. Diese `tell`-Sprechakte verlaufen aus der Sicht des Lotsen asynchron. Ein `ask-with-reply` wird ausschließlich von der Seite des kognitiven Modells aus initiiert und von der Luftraumsimulation beantwortet. Vor dem nächsten `ask-with-reply` muss der Sprechakt abgeschlossen sein. Dazu ist notwendig, dass das Ende von `reply` erkennbar ist. Wird kein neues `ask-with-reply` erzeugt, während ein anderes noch unbeantwortet ist, kann auf eine aufwändige Verwaltung ausstehender Antworten verzichtet werden. `tell` kann ausschließlich von der Seite der Luftraumsimulation aus initiiert werden und wird vom kognitiven Modell nicht beantwortet. Auf `tell` muss das kognitive Modell also in jeder Situation reagieren können. `register` kann in der Betriebsphase nicht gesendet werden.

Wenn sich Sprechakte nicht überschneiden und eindeutig identifizierbar sind, ist eine eindeutige und verklemmungsfreie Kooperation durch diese Protokolldefinition gesichert. Sie ist zudem einfach zu implementieren. Zur formalen Spezifikation des Protokollablaufes werden zwei Zustandsübergangsdiagramme benutzt (s. Abbildung Anhang-1). In einem Zustandsübergangsdiagramm werden die möglichen Zustände, die ein System annehmen kann, als Knoten eines Graphen dargestellt. Dabei ist ein Knoten besonders ausgezeichnet, um den initialen Zustand des Systems zu markieren. Die Knoten sind durch gerichtete Kanten verbunden, die die Übergänge von einem Zustand in den nächsten repräsentieren. Im Kontext einer Protokollspezifikation werden die Kanten mit den Ein- bzw. Ausgaben, die den Zustandsübergang bewirken, bezeichnet. Es wird hier die Konvention benutzt, dass ein vorgestelltes «-» das Lesen (Empfangen) und ein «+» das Schreiben (Senden) bezeichnet. Da getrennte Systeme (Luftraumsimulation und kognitives Modell), die jeweils einen eigenen Zustand haben, betrachtet werden, werden zwei Zustands-übergangsdiagramm benötigt.



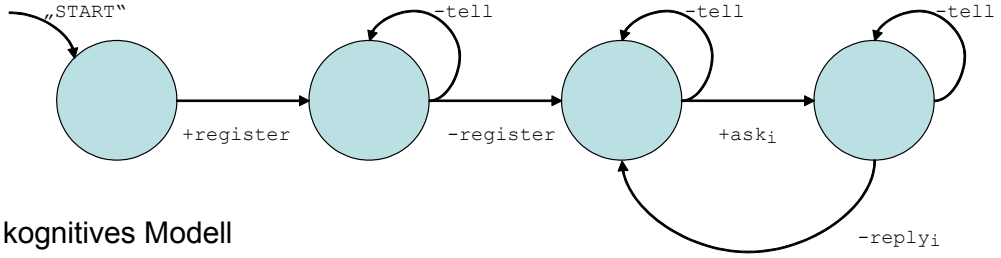


Abbildung Anhang-1: Zustandsübergangsdiagramm zur Spezifikation des Kommunikationsprotokolls

Ein Kommunikationsprotokoll kann als formale Sprache aufgefasst werden. Eine formale Sprache lässt sich leicht durch eine Grammatik eindeutig spezifizieren. Als Notation für die Protokollgrammatik wird hier BNF, die Backus Naur Form, benutzt (s. Quelltext Anhang-4). Mit diesem Formalismus lässt sich eine bestimmte Teilmenge der formalen Sprachen, die Menge der kontextfreien oder Typ-2 Sprachen in der Hierarchie von Chomsky, beschreiben.

```
<register>   ::=  register<NEWLINE>

<ask-1>      ::=  ask(<CALLSIGN>(,      <PARAMETER>)*,<PARAMETER>)<NL>

<ask-2>      ::=  ask(sector,<SECTOR>)<NL>

<ask-3>      ::=  ask(point,<KOORD>,<KOORD>,<RADIUS>)<NL>

<ask-4>      ::=  ask(<CALLSIGN>(,<zuweisung>)*,<zuweisung>)<NL>

<reply-1>    ::=  reply-1(<CALLSIGN>(,<wert>)*,<wert>)<NL>

                | reply-1(<CALLSIGN>)<NL>

<reply-2>    ::=  reply-2(<lfz>(,<lfz>)*)<NL>

                | reply-2()<NL>

<reply-3>    ::=  reply-3(<lfz>(,<lfz>)*)<NL>

                | reply-3()<NL>

<reply-4>    ::=  reply-4(<CALLSIGN>,ok)<NL>

<tell>       ::=  tell(<CALLSIGN>,<KOORD>,<KOORD>,

                      <FLIGHTLEVEL>,<NEAR>)<NL>

<zuweisung>  ::=  <PARAMETERNAME>=<PARAMETERWERT>

<wert>       ::=  <PARAMETERNAME>,<PARAMETERWERT>

<lfz>        ::=  <CALLSIGN>,<KOORD>,<KOORD>,
                  <FLIGHTLEVEL>,<PREDICTOR>,<PREDICTOR>
```

Quelltext Anhang-4: BNF-Spezifikation Kommunikationsprotokoll

Die passende Antwort auf <ask-n> ist <reply-n>. <ask-1>/<reply-1> bezieht sich auf Parameter eines Luftfahrzeuges. <ask-2>/<reply-2><ask-3>/<reply-3> sind Fragen nach Luftfahrzeugen in einem bestimmten Raum des Radarschirmes. <ask-4>/<reply-4> sind Anweisungen an ein Luftfahrzeug. Die Syntax für die einzelnen Symbole, wie <CALLSIGN>, <SECTOR> etc. sind entsprechend definiert.

Anhang E
Abkürzungen

- ACT-COM – ACT-R communication
- ACT-R – adaptive control of thought - rational
- ACT-R/PM – ACT-R/ Perceptual Motor
- AGI – ACT-R GUI Interface
- AMBOSS – *Eigenname, kein Akronym*
- AMBR – agent-based modeling and behavioral representation
- APEX – architecture for procedure execution
- API – application programming interface
- AS – action scheduling score
- ASN.1 – abstract syntax notation.1
- ASTERIX – all purpose structured Eurocontrol radar information exchange
- ATC – air traffic control
- AWT – abstract windowing toolkit
- BATON – blackboard architecture for task-oriented networking
- BC – belief creation score
- BMW – Bayerische Motorenwerke
- BNF – Backus-Naur Form
- BPEL – business process execution language
- BPEL4WS – BPEL for web services
- BPMN – business process modeling notation
- BREAM – bridge reliability and error analysis method
- BSD – Berkley system distribution
- BuB – Bedienen und Beobachten
- C2 – command and control
- CAD – computer aided design
- CaDaDis – categorical data display
- CCT – cognitive complexity theory
- CLG – command language grammar
- CMN-GOMS – Card Moran Newell GOMS
- CMU – Carnegie Mellon University
- COCOM – contextual control model
- COGENT – cognitive objects environment
- CORBA – common object request broker architecture
- COSIMO – cognitive simulation model (of human decision making and behavior in accident management of complex plants)
- CPM-GOMS – critical path method GOMS
- CREAM – cognitive reliability and error analysis method
- CSE – cognitive systems engineering lab
- CTT – concur task trees
- CWP – controller working position kernel system
- DAAD – Deutscher Akademischer Austauschdienst
- DCS – distributed control system
- DFG – Deutsche Forschungsgemeinschaft
- DFS – Deutsche Flugsicherung GmbH
- DIS – distributed interactive simulation
- DREAM – driver reliability and error analysis method
- DTD – document type definition

- DURESS – dual reservoir system simulation
- DuressJ – dual reservoir system simulation implementation in Java
- EnCoRe – en-route controllers' representation
- EPIC – executive process – interactive control
- ETAG – extended task-action grammar
- FüWES – Führungs- und Waffeneinsatzsystem
- G2A – GOMS to ACT-R
- gats – generic air traffic simulation
- gcl – GNU Common LISP
- GEF – graphical editing framework
- GLEAN – GOMS language evaluation and analysis
- GNU – GNU is not UNIX
- GOMS – goals, operators, methods, and selection rules
- GOMSL – GOMS language
- GOMS2ACT – GOMS to ACT-R
- GUI – Graphical User Interface
- HLA – high level architecture
- HLSR – high level symbolic representation language
- HRA – human reliability analysis
- I/F – interface
- IA – inference application score
- IBL – instance based learning
- ICS – interacting cognitive subsystems
- IDE – integrated development environment
- IEEE – Institute of Electrical and Electronics Engineers
- IES – interface editor system
- IMPRINT – improved performance research integration tool
- ISP-DL – impasse success problem solving driven learning
- IVIS – in vehicle information systems
- jACT-R – Java ACT-R
- JDT – Java development tools
- J-DURESS – Java implementation of DURESS
- JSPS – Japan Society for Promoting the Sciences
- K3 – Koordination, Kooperation, Kommunikation
- KI – künstliche Intelligenz
- KLM – keystroke level model
- KR – knowledge retrieval score
- LAN – local area network
- LISP – list processing
- LOC – lines of code
- MHP – model human processor
- MHP-GOMS – model human processor - GOMS
- MIDAS – man machine interactive design and analysis system
- MMS – Mensch-Maschine-System
- MoDyS – modelling of user behaviour in dynamic systems
- MoFL – Modell der Fluglotsenleistungen
- MT-GOMS – multi tasking GOMS
- MVC – model view controller
- NASA – National Aeronautics and Space Administration
- (N)GOMSL – (natural) GOMS language
- NIL – nihil (lat. nichts)

- NYNEX – New York/New England Exchange Corporation
- OCCS – operator cognitive crew cognitive simulation
- ODID – operational display and input development
- ODS – operational display system
- OMG – object management group
- ORTS – open real time strategy
- OSF – open software foundation
- OWL – web ontology language
- PDA – personal digital assistant
- PDL – procedure definition language
- P-DL – package based desciption logic
- PERT – program evaluation and review technique
- PM – perceptual motor
- PRA – probabilistic risk assessments
- PSA – probabilitic safety assessment
- PW – physical work score
- QN-MHP – queueing network-model human processor
- RPC – remote procedure call
- SANE Toolkit – skill acquisition network toolkit
- SHE – simulated hand and eye
- simco – sophisticated interface for monitoring, controlling and organizing
- SML – Soar markup language
- Soar – state, operators, actions, and result
- SoarML – soar markup language
- SOP – standard operation procedure
- SORTS – Soar/ORTS
- SWIG – simplified wrapper and interface generator
- SwiXML – Swing XML
- SWT – standard widget toolkit
- TAG – task action grammar
- TAKD – task analysis by knowledge description
- Tcl – tool command language
- Tcl/Tk – tool command language / toolkit
- TclSoar – tool command language – state, opreators, actions, and result
- TCP/IP – transmission control protocol / internet protocol
- THERP – technique for human error rate prediction
- Tix – Tk interface extension
- Tixwish – tix windowing shell
- TMS – truth maintenance system
- UDP – user datagram protocol
- UIMS – user interface management system
- UML – unified modeling language
- USD – US Dollar
- UseML – user interface markup language
- UVM – user's virtual machine
- UWR – updateable world representation
- VP – visual perception score
- WR2 – West Radar 2
- XAML – extensible application markup language
- XML – extensible markup language
- XSLT – extensible stylesheet transformation language
- XUL – extensible user-interface language