Kognitive Modelle

Jan-Martin Pulwitt und Claas Oppitz

Proseminar: Human-Computer-Interaction SS2007
Lehrstuhl i10 für Medieninformatik – Prof. Dr. Jan Borchers
Betreuer: Daniel Spelmezan

Inhaltsverzeichnis

Einleitung

Kognitive Modelle stammen ursprünglich aus der Psychologie.
Sie wurden aber schon bald darauf von Informatikern angewendet und teilweise weiterentwickelt und finden im Forschungsgebiet Mensch-Maschine-Interaktion (HumanComputerInteraction) und demzufolge bei der Entwicklung von Schnittstellen zwischen Benutzern und Computersystemen regen Gebrauch.
Diese Modelle ermöglichen es nämlich die Interaktion eines Benutzers mit einem System zu beschreiben oder zu simulieren.
Dadurch lässt sich in erster Linie der kognitive Aufwand eines Benutzers, der nötig ist um eine konkrete Aufgabe mit dem Computersystem zu lösen quantifizieren. Die meisten Modelle tun das, indem sie zeitliche Aufwände abschätzen, andere graduieren zusätzlich beispielsweise die Schwierigkeit die Benutzung des Systems zu erlernen. Allen gemeinsam ist das Ziel die Effektivität der Benutzerschnittstelle zu bewerten. Der Große Vorteil kognitiver Modelle ist, dass sie bereits vor der Implementierung eingesetzt werden können und somit durch frühzeitiges Testen Kosten klein gehalten werden können.

Wir unterteilen Kognitive Modelle in vier weitere Sinneinheiten:

1. Hierarchische Modelle, die der Aufgaben- und Ziel-Modellierung dienen
2. Linguistische Modelle, die die Interaktion auf sprachlicher Ebene modellieren
3. Modelle der Physis, die körperliche Handlungen der Benutzer beschreiben
4. Kognitive Architekturen, die Grundannahmen zur kognitiven Funktionsweise modellhaft darstellen

Hierarchische Modelle

GOMS

GOMS ist an Acronym aus Goals, Operators, Methods, Selection. Es wurde von Stuart Card, Thomas P. Moran und Allen Newell entwickelt und veröffentlicht (S.K. Card 2003).
Goal steht dabei für die Ziele, die ein Benutzer erreichen will.
Operators sind ausführbare Basisaktionen.
Methods bedeutet verschiedene Methoden um eine Aktion auszuführen.
Selection gibt Aufschluss über die Wahl der Methode.

Ziele sind zum einen das Hauptziel, also die Aufgabe die der Benutzer bewältigen möchte, und Unterziele, die wiederum in weitere Unterziele unterteilt sein können. Diese stellen evaluierbare Teilerfolge dar, so dass eine Verfehlung eines Ziels festgestellt werden kann, um in solch einem Fall die Bearbeitung des Unterziels zu wiederholen. Ziele und Unterziele werden strukturiert notiert, so dass die Ziele-Unterziele-Hierarchie erkennbar wird.
Operatoren sind Handlungen, die sich sowohl auf die Kognition als auch auf die Physis beziehen können. Beispiele hierfür sind etwa zum einen die Handlung eine Textbox zu lesen und im Vergleich dazu die Leertaste zu drücken oder eine Schaltfläche anzuklicken. Hierbei kann der Grad der Abstrahierung beziehungsweise der Konkretisierung variieren, zum Beispiel zwischen „Menüpunkt auswählen“ und „Maus über Schaltfläche bewegen und linke Maustaste drücken“. Allzu genaue Umschreibungen müssten aber streng genommen schon der nächsten Kategorie, den Methoden, zugeschrieben werden.
Diese werden verschiedene Umsetzungen einer Operation notiert. Zum Beispiel lässt sich ein Fenster einer Standardanwendung unter Windows stets auf mindestens zwei Arten schließen:

Zum einen über die entsprechende Schaltfläche, nämlich den X-Knopf in der Titelleiste des Anwendungsfensters, zum anderen über das Tastenkürzel Alt-F4, also das Drücken der F4-Taste bei gehaltener Alt-Taste.
In der vierten und letzten Kategorie, die Auswahl heißt, werden die Methoden festgelegt, beziehungsweise Auswahlregeln oder Prioritäten notiert. Ein Beispiel für eine Auswahlregel ist etwa: wenn möglich immer Tastenkürzel benutzen.
Die komplette Notation der in GOMS modellierten Aufgabe, das Fenster einer Standard-Windows-Anwendung zu schließen, sieht wie folgt aus (vgl. A. Dix 2003):

Ziel: Fenster schließen
. [Ziel: Benutze Schließen-Methode
. . Bewege Mauszeiger auf Titelleiste
. . Öffne Pop-up-Menu ‚Datei'
. . Bewege Mauszeiger auf Option ‚Beenden'
. . Klicke auf ‚Beenden'
. Ziel: Benutze X-Knopf
. . Bewege Mauszeiger auf Titelleiste
. . Klicke auf X-Knopf
. Auswahl Ziel: Benutze Alt-F4 Methode
. . Drücke Alt-F4]

GOMS werden sehr rege benutzt. Das liegt vor allem an der Einfachheit des Modells, die eine gewisse Universalität erzeugt. Dass das Modell auf viele Probleme anwendbar ist aufgrund seiner einfachen Struktur ist zwar ein Vorteil, führt aber auch dazu, dass eine Vielzahl von Aspekten außer acht gelassen wird. Training und Transfer spielen im Modell keine Rolle, also wird nicht darauf eingegangen, wie schnell die Benuzung des Systems erlernt werden kann und inwieweit es von Nutzen sein kann, Erfahrung in der Benutzung ähnlicher Systeme zu haben. GOMS geht nämlich von Expertenbenutzern aus, also Benutzern, die die Bedienung des Systems perfekt beherrschen.
Es gibt aber zahlreiche Variationen und hybride Modelle, also Modelle die Grundzüge des GOMS-Modells enthalten, es erweitern oder mit andern Modellen kombinieren. Besonders nennenswert sind Natural GOMS Language, kurz NGOMSL, das 1988 von David Kieras veröffentlicht wurde und das zusätzlich Lernerfolge von Benutzern bewerten kann, und Cognition-Perception-Motor-GOMS, kurz CPM-GOMS, das von John und Atwood Gray1993 veröffentlicht wurde und dadurch realistischer wird, dass Ziele und Operatoren parallelisiert werden und Operatoren-Kombinations-Templates für Wahrnehmung (Perception), Erkennung (Cognition) und Motorik inbegriffen sind.
Darüber hinaus ist auch noch KL-GOMS, KeystrokeLevelGOMS, nennenswert. Dieses Modell wird später unter der Kategorie Modelle der Physis als Keystroke-Level-Modell vorgestellt.

Cognitive Complexity Theory

Cognitive Complexity Theory, kurz CCT, wurde 1985 von David Kieras und Peter Polson vorgestellt (Polson 22 (1985)). Ziel beim Entwurf von CCT war, im Vergleich zu GOMS größere Präzision und Umfang der möglichen Vorhersagen zu erreichen.
Die Grundidee Aufgaben in Ziele und Unterziele zu unterteilen entspricht der Idee von GOMS. Die konkrete Modellierung der Handlung erfolgt jedoch, anders als bei GOMS, durch sogenannte Produktionsregeln. Produktionsregeln sind als if-Klauseln nach dem Schema „if Bedingung then Aktion" aufgebaut.

Der Momentan-Zustand des Anwenders wird durch den Modellstatus beschrieben, in dem sich auch die hierarchische Ziel-Struktur wiederspiegelt.
Zur Verdeutlichung führen wir als Beispiel (vgl. A. Dix 2003) einen Schritt des Einfügens eines vergessenen Leerzeichens in einem Texteditor an.

Ausgangs-Modellstatus:
(*GOAL* Text schreiben)
(*TEXT* Aufgabe: Leerzeichen einfügen)
(*TEXT* Aufgabe ist bei Zeile 2, Zeichen 16)
(*CURSOR* 3 29)

In jedem Zyklus werden die if-Bedingungen aller Produktionsregeln durchlaufen und ggf. die entsprechenden Aktionen nacheinander ausgeführt. Die Aktion einer Produktionsregel wird ausgeführt, wenn die Bedingung der if-Klausel zutrifft, also wenn sich der Wahrheitswert TRUE ergibt. Im Folgenden ist eine im Beispiel passende Produktionsregel angeführt:

…
(ZIELSETZUNG_LEERZEICHEN_EINFÜGEN
IF (*AND* (*TEST-GOAL* Text schreiben)
(*TEST-TEXT* Aufgabe: Leerzeichen einfügen)
(*NOT* (*TEST-GOAL* Leerzeichen einfügen))
(*NOT* (*TEST-NOTE* bearbeite
"Leerzeichen einfügen"))))
THEN ((ADD-GOAL Leerzeichen einfügen)
(ADD-NOTE bearbeite "Leerzeichen
einfügen")
(LOOK-TEXT Aufgabe ist bei Zeile %LINE,
Zeichen %COLUMN)))
…

Die Ausführung der Produktionsregel führt zu einer Aktualisierung des Modellstatus.

(*GOAL* Text schreiben)
(*TEXT* Aufgabe: Leerzeichen einfügen)
(*TEXT* Aufgabe ist bei Zeile 2, Zeichen 16)
(*NOTE* bearbeite "Leerzeichen einfügen")
(*GOAL* Leerzeichen einfügen)
(*LINE* 2)
(*COL* 16)
(*CURSOR* 3 29)

Im Ergebnis hat das Durchlaufen der Produktionsregeln also dazu geführt, das momentane Ziel vom Schreiben eines Textes zum Einfügen eines Leerzeichens abzuändern.
In den nachfolgenden Schritten könnten z.B. Produktionsregeln aufgerufen werden, die den Cursor an die entsprechende Position im Dokument bringen und dann das Leerzeichen tatsächlich einfügen.

Aus diesem Beispiel lässt sich leicht ersehen, dass die Anwendung von CCT im Vergleich zu GOMS wesentlich komplexer ist.

Der Nutzen im Vergleich zu GOMS liegt in der höheren Aussagekraft des Modells. Zum einen kann der vom Anwender zu erbringende Lernaufwand abgeschätzt werden, der aufgebracht werden muss, um unter gewissen Voraussetzungen Aufgaben mit einem für ihn neuem System zu bearbeiten. Beispielsweise könnten für die Modellierung einer Aufgabe mit Microsoft Word die bekannten Handlungs-Schemata, über die ein Benutzer aufgrund der Kenntnis eines Texteditors verfügt, als Produktionsregeln vorausgesetzt werden. So würde durch das Modell klar werden, dass der Anwender praktisch direkt in der Lage ist, bekannte Aufgaben aus einem Standard-Texteditor auch in Microsoft Word zu erledigen.
Weiterhin ist es mit CCT möglich Fehlersituationen zu simulieren. Dazu bedarf es nicht mehr, als den Modellstatus manuell auf den gewünschten fehlerhaften Zustand zu manipulieren. Die zyklische Ausführung der Produktionsregeln simuliert dann das Verhalten des Benutzers in dieser fehlerhaften Situation auf Basis der vorhandenen Handlungs-Schemata.
Dadurch, dass zyklisch alle Produktionsregeln durchlaufen werden, und sämtliche Passenden ausgeführt werden, lassen sich durch CCT auch parallele Handlungsstränge modellieren. Beispielsweise kann das oben angeführte Texteditor-Beispiel so ergänzt werden, dass eine weitere Produktionsregel stetig für eine Korrektheitsüberprüfung in der Nähe der aktuellen Cursorposition sorgt. So könnte das natürliche Benutzerverhalten simuliert werden beim Navigieren durch den Text auch stetig die Korrektheit zu überprüfen – unabhängig vom eigentlichen Grund der Navigation durch den Text.
Durch „Matching“ mit einem Modell der Benutzerschnittstelle lassen sich mittels CCT auch Probleme von vorhandenen Bearbeitungs-Schemata zusammen mit bestimmten Schnittstellen-Layouts vorhersehen und simulieren. Dies ist möglich, da (vgl. A. Dix 2003) die grundsätzliche Struktur üblicher Formalismen zur Beschreibung von Benutzerschnittstellen zum CCT Modell kompatibel ist.

Linguistische Modelle

Backus-Naur-Form

Dieses Modell verdankt seinen Namen seinen beiden Begründern John Backus[1] und Peter Naur, die es 1959 veröffentlichten. Es dient allgemein der Beschreibung von Dialog Grammatiken. Dabei wird ausschließlich die Syntax beschrieben, was ja der Sinn einer Grammatik ist, und kein Augenmerk auf die Semantik gelegt. In den 90er Jahren wurde die BNF von Phyllis Reisner im Forschungsgebiet Human-Computer-Interaction dann auch auf die Problematik der Beschreibung von Benutzerschnittstellen angewandt.

Die Backus-Naur-Form modelliert die Benutzung eines Systems, indem eine Aufgabe unterteilt wird in verschiedene Teilaufgaben, die gegebenenfalls wieder unterteilt wird, und so weiter bis zu einer Ebene maximaler Konkretisierung. In dieser befinden sich die sogenannten Terminale, die Benutzeraktionen wie etwa Mausklicks als Teillösungen eines komplexeren, übergeordneten Problems wie etwa eine Schaltfläche anklicken sind.
Dabei ist jede Zeile der Gestalt, dass links ein Name steht dem ein Ausdruck zugewiesen wird. Ausdrücke enthalten wiederum Namen, die durch UND- und ODER-Operatoren verknüpft sind.
Am besten lässt sich das an einem Beispiel (A. Dix 2003) veranschaulichen. Darum wird die BNF im Folgenden dazu verwandt die Benutzung der Linien-Zeichnen-Funktion eines

[1] John Backus erhielt Ende 1953 von IBM das OK zur Entwicklung einer Programmiersprache für den neu erschienenen IBM Computer, den 704er.
Backus stellte ein Team von Programmierern und Mathematikern zusammen, und realisierte FORTRAN.

Grafikprogramms zu beschreiben. Mit der im Beispiel benutzten Funktion kann man eine Linie zwischen einem Start- einem End- und beliebig vielen Zwischenpunkten zeichnen. Dazu muss man im geöffneten Grafikprogramm die Linien-Zeichnen-Funktion durch Anklicken der entsprechenden Schaltfläche auswählen. Daraufhin wird der Startpunkt des Polygonzugs mit einem Mausklick festgelegt. Jeder weitere Punkt wird ebenfalls mit einem Mausklick notiert und der Endpunkt wird durch einen Doppelklick bestimmt.

zeichne-linie ::= wähle-linie + wähle-punkte + letzter-punkt
wähle-linie ::= positioniere-maus + KLICK-MAUS
wähle-punkte ::= wähle-einen | wähle-einen + wähle-punkte
wähle-einen ::= positioniere-maus + KLICK-MAUS
letzter-punkt ::= positioniere-maus + DOPPELKLICK-MAUS
positioniere-maus ::= leer | BEWEGE-MAUS + positioniere-maus

zeichne-linie heißt die gesamte Aufgabe. Es beinhaltet wähle-linie, die Anwahl der Linien-Zeichnen-Funktion, wähle-punkte, die Auswahl beliebig vieler Punkte, und letzter-punkt, die Auswahl des Endpunkts durch positioniere-maus und einen Doppelklick.
wähle-punkte setzt sich zusammen aus wähle-einen, was wiederum positioniere-maus und einen Mausklick beinhaltet oder wähle-einen und wiederum wähle-punkte selbst. Auf diese Weise formuliert man das ein bis beliebig viele Punkte gesetzt werden können. Ähnlich trickreich ist auch positioniere-maus aufgebaut. Es setzt sich zusammen aus nichts, also keiner Aktion, oder der Basisaktion BEWEGE-MAUS und dem erneuten Aufruf von positioniere-maus. So können beliebig viele Mausbewegungen hintereinander geschaltet werden und es wird gewährleistet, dass das Modell erfasst, dass im Falle die Maus ist schon richtig positioniert keine Bewegung der Maus von Nöten ist.

Eine Analysemöglichkeit, die das Modell bietet ist die Anzahl der Regeln zu ermitteln, da sie eine Kennzahl für die Komplexität des Systems ist. Die Verschachtelungstiefe der Systemgrammatik, also die Anzahl der Regeln, ist aber im Grunde willkürlich, da man komplexere Ausdrücke bilden könnte um damit mehrere Regeln zusammenzufassen. Deshalb wird oft die Anzahl der Zuweisungs-, UND- und ODER-Operatoren ermittelt. Ein anderer Ansatz ist die Anzahl der benötigten Basisaktionen zur Lösung einer Aufgabe mit dem System als Komplexitätsmaß zu verwenden.
Ein Nachteil des Modells ist, dass kein Augenmerk auf die Wahrnehmung des Benutzers, ihre Schwächen und die daraus resultierenden Probleme gelegt wird. Es lässt sich also nicht evaluieren, ob beispielsweise bestimmte Buttons auf der Benutzeroberfläche schnell gefunden werden, was ja ein wichtiger Aspekt der Benutzerfreundlichkeit ist. Um dieser Problematik gerecht zu werden, hat Phyllis Reisner das Modell um Information-Suchen-Aktionen erweitert.

Task Action Grammar

Die Task Action Grammar wurde 1986 als Erweiterung der BNF von Stephen Payne und Thomas Green vorgestellt (Payne 2 (1986)). Ziel beim Entwurf war es, im Vergleich zur BNF stärkeren Fokus auf die kognitiven Aspekte zu legen und so präzisere Aufwands-Schätzungen machen zu können. Dabei erfolgt die Modellierung, wie bei der BNF, weitgehend durch textliche Beschreibung der Funktionsweise der Benutzerschnittstelle.
Die Entwurfs-Ziele werden dadurch realisiert, dass zum einen die Konsistenz und zum anderen das vorhandene Vorwissen des Benutzers mit in das Modell einfließen.

Das Vorwissen gewisser Funktionen und die Fähigkeit vergleichbare Funktionen aus dem Zusammenhang direkt zu erschließen wird durch sogenannte Wortfelder angegeben. Ein Beispiel solcher aus dem Kontext erschließbarer Funktionen sind z.B. die Vor- und Rückwärts Buttons in heutigen Webbrowsern. Für einige Anwender könnten diese Bedienelemente intuitiv Bedienbar modelliert werden. Für andere Anwender hingegen gilt, dass sobald die Funktion eines der Buttons gelernt wurde, auch direkt die Funktion des anderen Buttons ersichtlich wird. Entsprechende Zusammenhänge werden in TAG durch die BNF ergänzende Schlüsselwörter angegeben.
Um die Einbeziehung der Konsistenz von Benutzerschnittstellen in das Modell zu veranschaulichen, führen wir als Beispiel (vgl. A. Dix 2003) die Konsistenz von Datei-Operationen an der Linux Shell an.
Die von uns betrachteten Shell-Operationen sind *Kopieren, Verschieben* und *Verknüpfen*. In BNF können die Aktionen wie folgt modelliert werden:

Kopieren ::= cp + *Dateiname* + *Dateiname* | cp + *Dateinamen* + *Verzeichnis*
Verschieben ::= mv + *Dateiname* + *Dateiname* | mv + *Dateinamen* + *Verzeichnis*
Verknüpfen **::= ln +** ***Dateiname + Dateiname*** **| ln +** ***Dateinamen + Verzeichnis***

Dieses BNF ist gleichwertig zu folgendem inkonsistenten BNF Modell.
Kopieren ::= cp + *Dateiname* + *Dateiname* | cp + *Dateinamen*+ *Verzeichnis*
Verschieben ::= mv + *Dateiname* + *Dateiname* | mv + *Dateinamen* + *Verzeichnis*
Verknüpfen **::= ln +** ***Dateiname + Dateiname*** **| ln +** ***Verzeichnis + Dateinamen***

Selbstverständlich ist letztere Syntax auf Grund der Inkonsistenz beim Verknüpfen mehrerer Dateien schwerer zu erlernen. Um aber auch die Konsistenz zu berücksichtigen, werden in einem entsprechenden Modell Datei-Operationen gruppiert. Die spezifische Operation wird zum Parameter einer allgemeinen Datei-Operation.

*Datei-Operation[**Kopieren**],*
*Datei-Operation[**Verschieben**],*
*Datei-Operation[**Verknüpfen**]*

*Befehl[**Aktion= Kopieren**] ::= cp*
*Befehl[**Aktion = Verschieben**] ::= mv*
*Befehl[**Aktion = Verknüpfen**] ::= ln*

Anhand dieses Beispiels wird klar, dass also die Konsistenz von Benutzerschnittstellen in TAG Modellen berücksichtigt wird. Wie zuvor erläutert, kann außerdem das Vorwissen der Benutzer in gewissen Grenzen berücksichtigt werden. Daraus resultiert also die Möglichkeit, eine im Vergleich zur BNF präzisere Abschätzung des kognitiven Aufwands beim Anwenden und zu dem Erlernen einer spezifischen Benutzerschnittstelle. Genau wie bei der BNF ist jedoch leider kritisch anzumerken, dass sich Modelle der TAG nur schwer für grafische Applikationen erstellen lassen. Der Grund dafür liegt im zur Beschreibung von grafischen Benutzerschnittstellen unnatürlichen Ansatz der Modellierung auf Basis von formalen Sprachen.

Modelle der Physis

Keystroke-Level-Model

Dieses Modell kann auch wie eingangs erwähnt als ein stark abstrahiertes GOMS aufgefasst werden. Es ermittelt die zeitliche Komplexität einer Aufgabe, beziehungsweise der Lösung des Problems mit dem System, das bewertet werden soll.
Hierzu werden die Benutzeraktionen, die an einer grafischen Oberfläche als Benutzerschnittstelle getätigt werden können, in Klassen eingeteilt, die Anzahl der Aktionen je Klasse ermittelt und die den Klassen zugeteilten zeitlichen Aufwände summiert. Es gibt dabei fünf physisch motorische Operationstypen:
nämlich Tastendruck (keystroke), Maustastendruck (button), Mauszeiger auf Ziel bewegen (pointing), Hand an Maus oder Tastatur bewegen (homing) und mit der Maus eine Linie zeichnen (drawing),
einen mentalen Operationstyp: Mentale Vorbereitung (mentally preparation)
und einen dem System zuzuordnenden Operationstyp: Warten auf Systemantwort (system response).
Diese werden im folgenden Beispiel (vgl. A. Dix 2003) mit dem Anfangsbuchstaben der englischen Bezeichnung abgekürzt.
Die vom Benutzer zu lösende Aufgabe ist in einem Texteditor ein fehlerhaftes Zeichen zu korrigieren. Dazu bewegt der Benutzer zuerst die Hand an die Maus (H) und dann mit dieser den Mauszeiger links neben das fehlerhafte Zeichen (P) und klickt dorthin (B) um den Cursor dort zu positionieren. Dann bewegt er die Hand zurück zur Tastatur (H) und löscht das Zeichen nach einer mentalen Vorbereitungsphase (M) durch das Drücken der Rücktaste (K). Schließlich gibt er die Korrektur über die Tastatur ein (K) und positioniert den Cursor am Textende, indem er die Hand zur Maus bewegt (H), sich mental vorbereitet (M), den Mauszeiger ans Ende des Textes bringt (P) und einen Mausklick tätigt (B).
Es ergibt sich für diese Aktionen ein zeitlicher Aufwand von 6,74 Sekunden. Dabei ist der Wert für das Bewegen des Mauszeigers (pointing) gemittelt und deshalb für kürzere Aufgaben mit wenigen Benutzeraktionen mitunter recht ungenau. Bei dem Wert für das Drücken einer Taste der Tastatur gehen wir hier von einem geübten Maschinenschreiber aus, der etwa 90 Wörter in der Minute tippt.
So ergeben sich folgende Werte:

Aktion	***K***	***B***	***P***	***H***	***D***	***M***	***R***
Zeit	0.12	0.20	1.10	0.40	-	1.35	-
Anzahl	2	2	2	3	0	2	0
Zeit	0.24	0.40	2.20	1.20	0	2.70	0

Abschließend ist noch zu erwähnen, dass die Werte für das Zeichnen einer Linie (drawing) und für das Warten auf Systemantwort (system response) so stark variieren, dass selbst gerundete Zeitangaben zu ungenau und somit nicht sinnvoll wären.

Three-State-Model

Das Three-State-Model ist eine auf Zeigegeräte wie Maus oder Tablet-PC-Stift bezogene Ergänzung des KLM. Die deutsche wörtliche Übersetzung Drei-Zustands-Modell ist geeignet, nur nicht gebräuchlich.
Im Three-State-Model wird kein Bezug auf die relative Position des Benutzers zum Zeigegerät genommen (dem notwendigen „Homing“, nach KLM), sondern der Zustand des

Zeigegeräts an sich wird betrachtet. Nach dem Three-State-Model gibt es drei Grundzustände, die nach empirischen Studien zu unterschiedlichen Reaktions- und Bearbeitungszeiten führen. Die drei unterschiedenen Grundzustände sind:

- Die Position des Zeigegeräts ist dem Computer bekannt, es wird kein Button gedrückt. Das entspricht dem Zustand, den eine Computermaus hat, wenn sie an einen PC angeschlossen ist und nur zum Zeigen verwendet wird.
- Die Position des Zeigegeräts ist dem Computer bekannt, es wird ein Button gedrückt. Das entspricht dem Zustand, den die Maus z.B. beim Drag-and-Drop hat, während also ein Objekt gezogen wird.
- Der Computer hat keine Verbindung zum Gerät. Dieser Zustand ist für Computer-Mäuse eher unüblich, da dieser nur von einer defekten oder nicht angeschlossenen Maus erzeugt wird. Im Bezug auf Zeigegeräte wie einen Tablet-PC Stift ist dieser Zustand aber durchaus üblich: Der Stift ist „zu weit“ vom Display entfernt, als dass die Position oder der Druck auf Buttons registriert werden könnten.

Die Aussage des Three-State-Model lässt sich auf die Kernaussage reduzieren, dass eine allgemeine Betrachtung gemäß des KLM von Ausführungszeiten mit Zeigegeräten weniger präzise ist, als wenn der Status der jeweiligen Zeigegeräte zusätzlich berücksichtigt wird.

Kognitive Architekturen

Problem Space Model

Das Problem Space Model erzeugt Erkenntnis über das nötige Wissen und Können potentieller Benutzer eines Systems. Es führt zu einem programmierten Benutzermodell (englisch: Programmed User Model - PUM). Wenn ein solches PUM aufgrund mangelnder Kenntnisse über das System und dessen Benutzung eine Aufgabe nicht lösen kann, ist das System unzulänglich implementiert und muss überarbeitet werden.
Das Problem Space Model basiert auf einem Agenten, der in einer Umwelt eingebettet existiert. Die Umwelt ist das zu untersuchende System, der Agent ist eine virtuelle Einheit des Modells, die man sich wie ein intelligentes Computerprogramm vorstellen kann. Der Agent scannt seine Umwelt und sammelt Informationen über sie. Dabei bringt er alle möglichen veränderbaren Zustände, wie zum Beispiel wo sich der Mauszeiger befindet, und alle ausführbaren Operationen in Erfahrung. Die Auswahl und Ausführung einer dieser Operationen verändert die Umwelt des Agenten und überführt das System in einen anderen Zustand. In diesem neuen Zustand geht der Agent dann wieder genauso vor wie zuvor, bis irgendwann die Aufgabe erfüllt ist, also der erwünschte Zustand erreicht ist.
Daraus ergibt sich ein Aktivitätskreisel, der vier Komponenten enthält:

1. Zielformulierungsprozess, der einen Satz von gewollten Status und den ersten Status für die Aufgabe definiert
2. Operationsauswahlprozess, der eine zielnähernde Operation vorschlägt
3. Operationanwendungsprozess, der eine Operation auswählt und ausführt und den Status und somit auch die Umwelt verändert
4. Zielerfüllungsprozess, der prüft, ob der aktuelle Status der angestrebte Status ist, und sobald dies der Fall ist den Agenten deaktiviert

Man kann das Problem also zu einer Suche eines optimalen Weges durch ein Set von Status abstrahieren. Alle Sets aller Status stellen also den Problemraum dar. Kanten bestehen in Form von Operationen zwischen einem Status und dem Status, in den das System durch Ausführung der Operation überführt wird.
Aufgrund seiner Komplexität ist das Modell jedoch nicht implementierbar.

Es führt allerdings zu einem Modell namens SOAR (State, Operator And Result), das 1987 veröffentlicht wurde (Laird 33 (1987)).

Interactive Cognitive Subsystems

Das Modell der Interactive Cognitive Subsystems (A. Dix 2003) geht davon aus, dass die Bearbeitung von sämtlichen Prozessen im menschlichen Gehirn durch 9 interagierende Teilsysteme geschieht. Die wörtliche Übersetzung „interaktive kognitive Teilsysteme“ ist also durchaus treffend.
Die neun Teilsysteme fokussieren jeweils einen separaten Bereich, für dessen Bearbeitung sie zuständig sind. Die grobe Unterscheidung erfolgt dabei in 5 den Sinnen zuzuordnende Teilsysteme (Hören, Riechen, Fühlen, Schmecken, Sehen). Für heutige übliche Computersysteme sind nicht alle dieser Teilsysteme relevant, womit direkt auch schon die Generalität dieser modellhaften Betrachtungsweise klar wird.
Die anderen vier kognitiven Teilsysteme dienen der Verarbeitung des Wissens, bzw. dem Verknüpfen der Informationen. Beispielsweise dient eines der Systeme nach dem ICS Modell ausschließlich dem Erkennen von Zusammenhängen zwischen den Wahrnehmungen und vorhandenem Wissen.
Aus dieser detaillierten Betrachtungsweise, aus der genauen Aufteilung von Teilfunktionen sowie aus dem Modell des Zusammenspiels lassen sich neue Faktoren ableiten, deren Berücksichtigung zwar relevant ist, die aber in bisher aufgeführten Modellen keine oder nur eine untergeordnete Rolle spielen. Ein konkretes Beispiel für eine Ableitbare Schlussfolgerung ist z.B. die fehlende Möglichkeit in GOMS parallele kognitive Abläufe zu modellieren. Das komplexere Modell CCT ermöglicht dies. Gemäß der ICS wird jedoch klar, dass CCT wiederum auch keinen Bezug auf die konkreten erkenntnistheoretischen Lernprozesse des Benutzers nimmt.

Quellenverzeichnis

A. Dix, J. Finlay, G. D. Abowd, R. Beale. *Human Computer Interaction.* Prentice Hall, 2003.

Card, S.K.; T.P. Thomas & A. Newell. *The Psychology of Human-Computer Interaction*, London: Lawrence Erbaum Associates, 1983.

Payne, S.J. & Green, T.R.G. „Task-action grammars: A model of the mental representation of task languages." *Human-Computer Interaction*, 2 (1986): 93-133.

Polson, D.E. Kieras and P.G. „An approach to the formal analysis of user complexity." *International Journal of Man-Machine Studies*, 22 (1985): 365-394.

Laird, Rosenbloom, Newell, John and Paul, Allen (1987).Soar: „An Architecture for General Intelligence". *Artificial Intelligence*, 33: 1-64