Institut für Visualisierung und interaktive Systeme

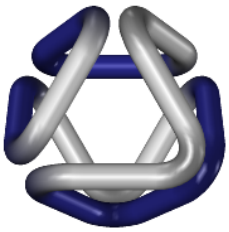
Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Fachstudie Nr. 134

Kognitions-Frameworks I

Christoph Bergmann
Christian Dittrich
Michael Kircher

Studiengang:	Softwaretechnik
Prüfer:	Prof. Thomas Ertl
Betreuer:	Michael Raschke
begonnen am:	21.12.2010
beendet am:	29.07.2011
CR-Klassifikation:	H.1.2, I.2.0, I.2.4, I.2.10, J.4



Universität Stuttgart

Institut für Visualisierung und Interaktive Systeme

Kognitions-Frameworks I

Ausarbeitung

Fachstudie (WS10, SS11)

Betreuer: Michael Raschke, Prof. Thomas Ertl

Christoph Bergmann, Christian Dittrich, Michael Kircher

Stuttgart, 10. August 2011

Kurzbeschreibung

Kognitions-Frameworks stellen einen interessanten Ansatz dar, um die Prozesse des menschlichen Gehirns im Computer nachzubilden. Eine Suche im Internet fördert eine Vielzahl an verschiedenen Frameworks zu Tage, welche sich mehr oder weniger stark auf bestimmte Einsatzbereiche spezialisieren oder auch den Anspruch haben, beliebige Probleme zu simulieren.

Diese Fachstudie untersucht die beiden Frameworks ACT-R und CogTool und prüft diese, auf deren Einsetzbarkeit zur Interaktionsanalyse eines Benutzer mit einer GUI oder Visualisierung. Dabei werden zunächst die beiden Frameworks im Detail vorgestellt, während anschließend herausgearbeitet wird, welche grundlegende Eigenschaften für ein Framework nötig sind, um kognitive Prozesse beim Arbeiten mit GUIs und Visualisierungen simulieren zu können.

Abstract

Cognition-Framworks are an interesting approach to simulate the processes of the human mind. The internet offers a huge amount of different frameworks. Some of them are more specialized to a certain field of application, while others claim to simulate arbitrary problems.

The goal of this work was to evaluate the cognition frameworks ACT-R and CogTool. These two frameworks were analyzed for a simulation of cognitive processes while working with a GUI or visualization. First, both frameworks will be presented in detail. Then, fundamental properties of GUIs and visualization will be given, which are neccessary for using a cognitive framework to simulate cognitive processes while working with GUIs and visualization.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Die Fachstudie untersucht die Einsetzbarkeit von Kognitions-Frameworks im Hinblick auf die Interaktionsanalyse eines Benutzers mit GUIs und Visualisierungen. Es geht also um die Frage, wie mentale Prozesse simuliert werden können, die ein menschlicher Betrachter bei der Verwendung einer GUI oder Visualisierung durchläuft. Das Thema trifft hierbei die Gebiete künstliche Intelligenz, Kognitionswissenschaft und Visualisierung. Der Nutzen, die Kognitionsframeworks unter dieser Fragestellung anzuwenden, besteht zum einen darin, unterschiedliche GUIs oder Visualisierungen automatisch bewerten zu können, was bisher meist nur unter Durchführung aufwendiger Benutzerstudien möglich war. Zusätzlich bieten die Frameworks einen Ansatzpunkt, um die Frage „Wie gut ist eine GUI/Visualisierung?“ auf Basis wissenschaftlicher Erkenntnisse der Perzeption und Kognition fundierter bzw. objektiver zu beantworten.

Hierfür werden in Kapitel 2 zunächst einige Grundlagen, insbesondere in Bezug auf die menschliche Perzeption und Kognition, als auch allgemeine Grundlagen zu Visualisierungen an sich erläutert.

Kapitel 3 stellt anschließend ausgewählte Vertreter der Kognitions-Frameworks vor: „ACT-R“ als allgemeines Kognitions-Framework, mit dem Anspruch beliebige Szenarien durch Modelle abbilden zu können. Weiterhin „CogTool“, als spezialisiertes, auf ACT-R aufbauendes Framework zur Interaktionsanalyse mit GUIs. Zuletzt folgt ein Blick auf „CAEVA“, eine Architektur zur Auswertung von Visualisierungsanwendungen.

Kapitel 4 entfernt sich nun wieder etwas von den konkreten Frameworks und versucht die Informationen aus den Frameworks zu abstrahieren, zu der Frage, was ist eigentlich allgemein nötig, um eine GUI oder Visualisierung kognitiv simulieren zu können? Hierbei ist auch die maschinelle Zugänglichkeit von GUIs und Visualisierungen ein wichtiger Punkt. Inbesondere der benötigte Grad der Zugänglichkeit spielt dabei eine wichtige Rolle.

Zuletzt fasst Kapitel 5 die Ergebnisse zusammen und zieht ein Fazit der Recherchen.

Einen visuellen Überblick über den Inhalt bietet die folgende Abbildung 1.

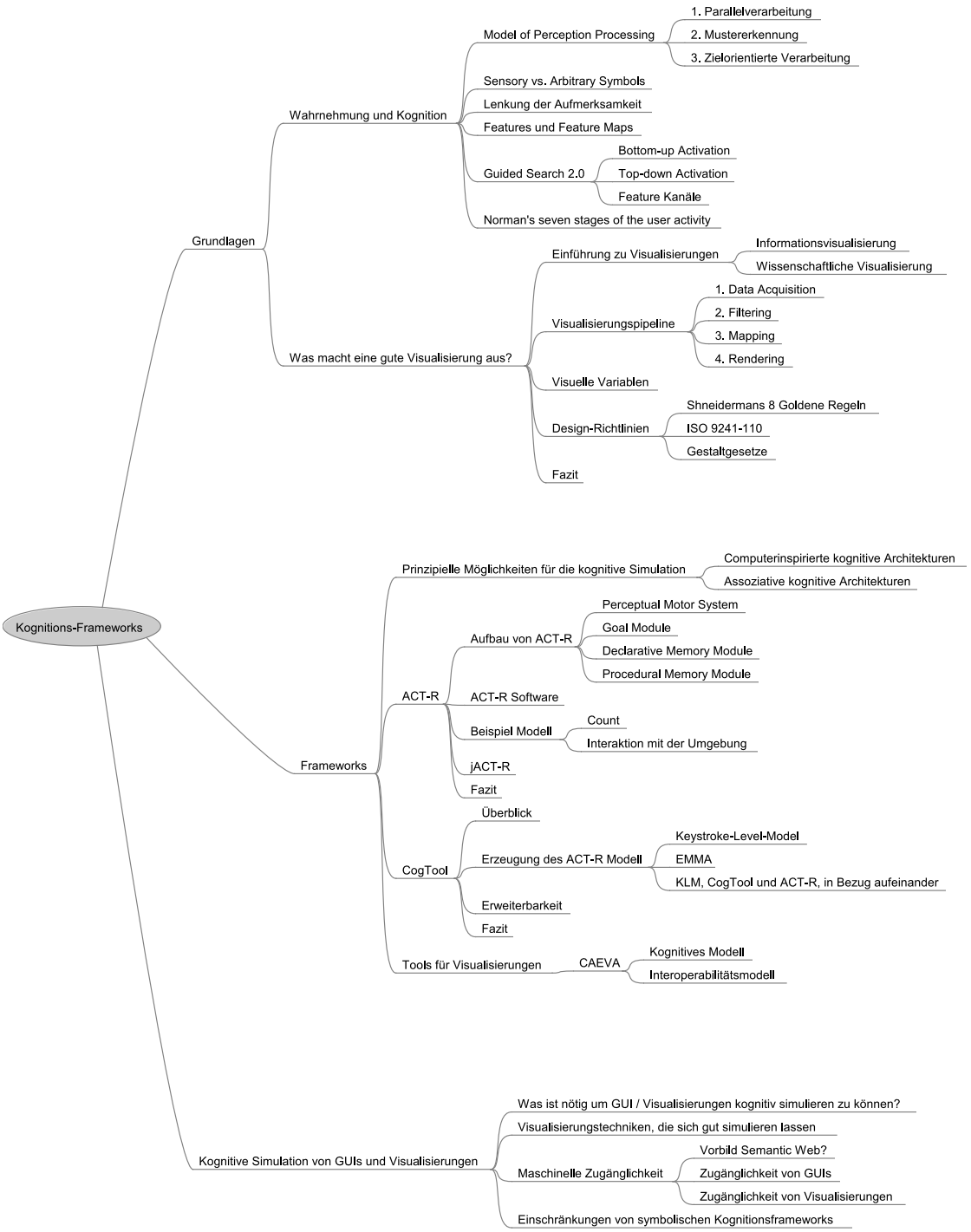


Abbildung 1: Überblick der behandelten Themen

2. Grundlagen

Das Thema der Fachstudie steht interdisziplinär zwischen den Themen künstliche Intelligenz, Kognitionswissenschaft und Visualisierung. Zu allen Bereichen sollen in diesem Kapitel zunächst einige grundlegende Aspekte, die wir für das weitere Verständnis als wichtig erachten, erläutert werden.

2.1. Wahrnehmung und Kognition

Zur Simulation der Wahrnehmung einer GUI oder Visualisierung durch einen Menschen, stellen sich zunächst einige Frage:

- Wie nimmt ein Mensch seine Umgebung auf kognitiver Ebene wahr?
- Wie werden Sinneseindrücke weiter verarbeitet?
- Wie entstehen aus der eingehenden „Datenflut“ der Sinnesorgane semantische Entitäten, die ein Mensch auf abstrakter Ebene wahrnehmen, kategorisieren, verarbeiten und bewerten kann?

Das Kapitel stellt ausgewählte Aspekte dieses Verarbeitungsprozesses vor, wobei sich diese zum Teil überlappen bzw. ergänzen, da es kein geschlossenes „Standardmodell“ der Perzeption bzw. Kognition gibt. Das Kapitel umfasst ein dreistufiges Phasenmodell der menschlichen Wahrnehmungsverarbeitung, einen Abschnitt über sogenannte sensorische und arbiträre Symbole, ein Kapitel zu „Features und Feature Maps“, sowie „Guided Search“ und zuletzt einen Abschnitt über Norman's seven stages.

2.1.1. Model of Perceptual Processing

Das Modell entstammt [31] (S.20 - S.22) und wird in diesem Kapitel zusammengefasst. Es unterteilt die menschliche Wahrnehmung in drei Phasen, die jeweils nacheinander ablaufen und einen ansteigenden Grad der Informationsabstraktion aufweisen. Abbildung 2 zeigt hierzu den grundlegenden Ablauf, wobei pro Stufe die eingehende Informationsmenge reduziert wird und gleichzeitig der Abstraktionsgrad von primitiven Features bis hin zu größeren Objekten ansteigt.

Phase 1: Parallelverarbeitung
Beteiligt sind Milliarden von Nervenzellen im Auge und im visuellen Kortex des Gehirns. Jede Nervenzelle ist dabei auf die Erkennung eines bestimmten Merkmals ausgerichtet, z.B. die Orientierung von Kanten, Erkennung von einzelnen Farben und einzelne Bestandteile von Texturen, oder die Erkennung von Bewegungen. Die Verarbeitung ist

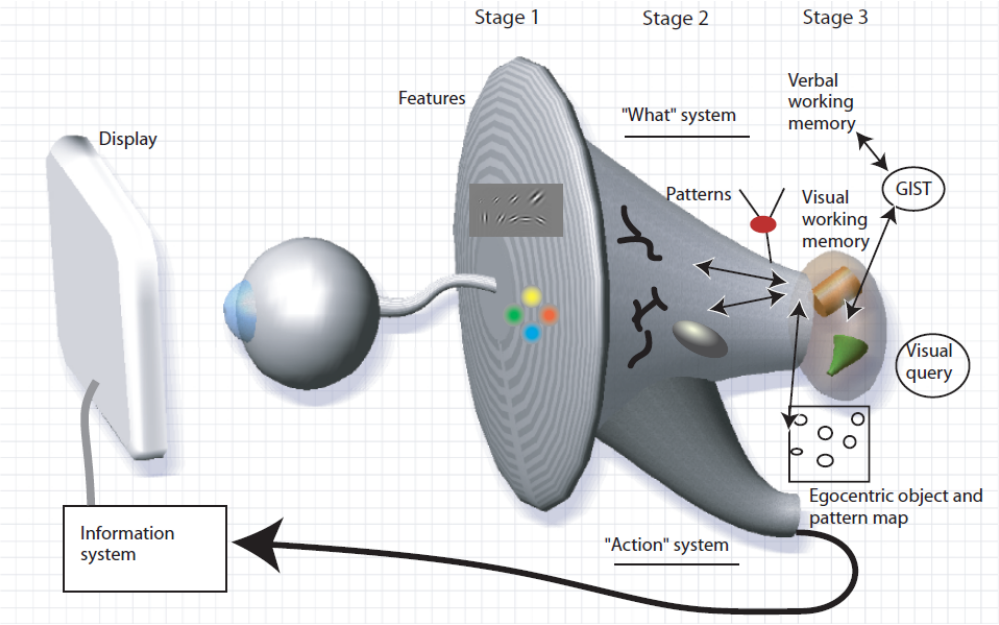


Abbildung 2: Three-Stage Modell der menschlichen Informationsverarbeitung. In Phase 1 werden gleichzeitig sehr viele, jedoch nur primitive Features wahrgenommen, während in den weiteren Phasen diese Features zu komplexeren Objekten kombiniert werden, wovon nur noch wenige im Arbeitsgedächtnis gehalten werden können [31].

dabei ein massiv paralleler Prozess aller Nervenzellen, der ständig abläuft und nicht von der Aufmerksamkeit des Menschen auf bestimmte Bereiche des visuellen Sichtfelds abhängt. Aufgrund der schnellen Parallelverarbeitung ist es für Visualisierungen besonders interessant diese Phase der Perzeption anzusprechen, um eine besonders effiziente Informationsvermittlung zu ermöglichen.

Phase 2: Mustererkennung
Die zweite Phase unterteilt das Sichtfeld in Regionen, also unterschiedliche Bereiche, und Muster. Dies können beispielsweise kontinuerliche Umrisse oder ganze Texturen sein. Die Phase ist sehr flexibel, da sie auf den großen Informationsbestand der ersten Phase zugreift, als auch die Aufmerksamkeit des Menschen beachtet. Aufmerksamkeit bezeichnet in diesem Zusammenhang nicht den Fokus-Punkt der Augen beim Betrachten, sondern den „Fokus-Punkt“ der Gedanken, also die Fragestellung unter der die visuellen Informationen durch das Gehirn ausgewertet werden (auch „Visual Queries“ genannt, [31] S.22). Die Verarbeitung verläuft im Gegensatz zur ersten Phase sequentiell und stellt einen Kreuzungspunkt zwischen den Informationsflüssen aus der ersten Phase und den abstrakten Aufmerksamkeitsmechanismen der dritten Phase dar.

Phase 3: Zielorientierte Verarbeitung
Repräsentiert die höchste Ebene der Perzeption, auf der die Objekte des visuellen Arbeitsgedächtnisses liegen, von denen nur wenige gleichzeitig gehalten bzw. erzeugt werden können (in der Größenordnung von 3-5 Objekten, [31] S.188). Ein Objekt ist das Ergebnis einer Fragestellung, die sich aus der Aufmerksamkeit des Menschen ergibt. Beantwortet wird die Fragestellung durch eine Sequenz aus „Visual Queries“, die mit Hilfe der Mustererkennung der zweiten Phase gelöst werden. Als Beispiel könnte für ein Objekt des Arbeitsgedächtnisses die Frage nach einer Route auf einer Karte stehen. Die zugehörige Visual Query entspricht demnach einer möglichen Straße, welche zwei Städte miteinander verbindet (Abbildung 3).

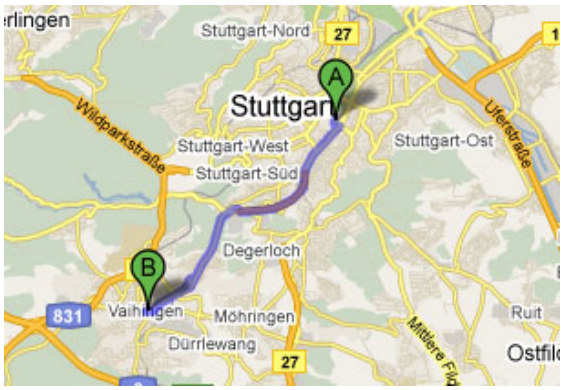


Abbildung 3: Route zwischen zwei Städten: ein Objekt im Arbeitsgedächtnis, zusammengesetzt durch eine Visual Query.

Die Konsequenz des Verarbeitungsmodells, im Hinblick auf die Effizienz einer Visualisierung und in Bezug auf einen Betrachter, ist nun vor allem die Fragestellung, wie stark eine Visualisierung die unterschiedlichen Phasen fordert. Einfach ausgedrückt: ein menschlicher Betrachter sollte demnach besonders schnell eine Fragestellung beantworten können, wenn möglichst wenige Objekte des visuellen Arbeitsgedächtnisses und wenige Visual Queries benötigt werden, da diese der sequentiellen Verarbeitung unterliegen. Noch weiter abstrahiert könnte man sagen, dass die Erkennung der Elemente einer Visualisierung auf einer möglichst niedrigen Stufe des Verarbeitungsmodells stattfinden sollte.

2.1.2. Sensory vs. Arbitrary Symbols

Das Kapitel stellt eine Zusammenfassung von [31] (S.10 - S.20) dar. Inhalt ist eine Taxonomie visueller Repräsentationen in zwei Kategorien: Sensorische Symbole, die rein aus den Perzeptionsmechanismen des Gehirns semantisch erkannt werden können und

arbiträre Symbole, deren semantische Erkennung erlernt werden muss. Ein Beispiel für beide Kategorien liefert Abbildung 4.

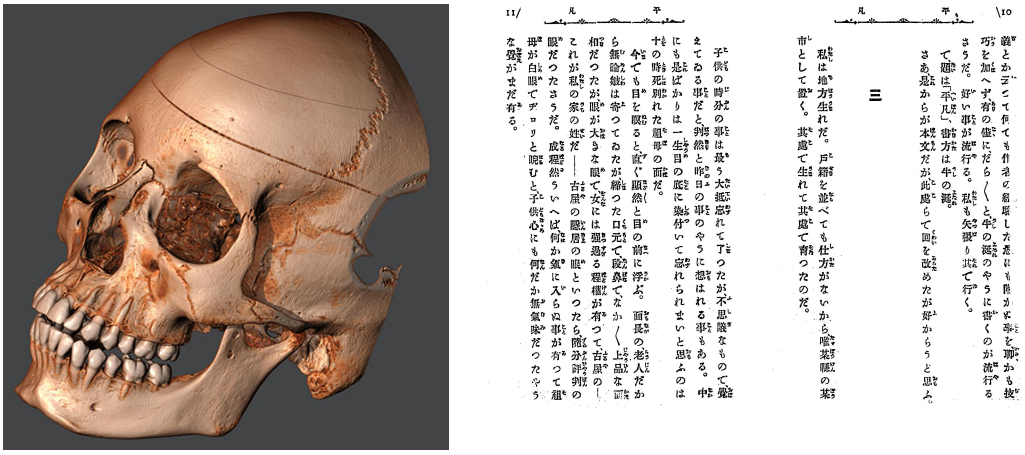


Abbildung 4: Sensorisches Symbol: Visualisierung der Form eines Schädels, unterstützt durch Beleuchtung, Schattierung, Verdeckung [44]. Arbiträre Symbole: japanische Schriftzeichen [37].

Sensorische Symbole:
Grundlage der sensorischen Symbole ist die Annahme, dass das visuelle System des Menschen aus der Evolution heraus, als Werkzeug zur Wahrnehmung der Umgebung entstanden ist. Umgebung bezeichnet dabei die typische physikalische Umgebung in der sich ein Mensch befindet. Sie unterliegt den physikalischen Gesetzen (z.B. der Art wie sich Licht ausbreitet) und hat „typische“ Eigenschaften, wie z.B. Gegenstände, die sich durch ihre Oberflächen, bzw. der Übertragung der Informationen darüber (in Form von Lichtwellen) dem menschlichen visuellen System bemerkbar machen. Das menschliche Gehirn wird in dieser Unterscheidung als ein Zentrum spezialisierter, paralleler Verarbeitungsmaschinen dargestellt. Da die Spezialisierung auf die Wahrnehmung der „typischen“ Umgebung eines Menschen ausgerichtet ist und evolutionär begründet wird, ist die Konsequenz daraus, dass alle Menschen im Prinzip das gleiche visuelle System haben und auf die gleiche Art „sehen“, also die Verarbeitung von eintreffenden Lichtstrahlen im Auge bis zum Sinneseindruck im Allgemeinen gleich verläuft und somit relativ einheitliche Begriffe, wie „Farbe“ oder „Form“ entstehen.

Ein sensorisches Symbol stellt eine visuelle Entität dar, die ohne, dass die Erkennungsmerkmale des Symbols vorab erlernt werden mussten, durch das visuelle System des Menschen erkannt werden können. Ein Beispiel dafür wäre eine 3D-Darstellung eines Objekts, inklusive Schattierung und Beleuchtung, unter der Voraussetzung dass es die Form des Objekts selbst ist, die visualisiert und erkannt werden soll. Das Objekt selbst

muss nichts alltägliches darstellen, trotzdem kann bei der Erkennung das volle Potenzial der Oberflächen- bzw. Objekterkennung des visuellen Systems zum Einsatz kommen. Sensorische Symbole sind für alle Menschen gleich effektiv, ihre Erkennung kann nicht einfach durch Willen oder Anweisungen verhindert werden. Sie sind oft unabhängig von Individuen, Kulturen und Zeit ([31], S.10). Dies bedeutet aber nicht, dass auf ein sensorisches Symbol auch immer eine gleiche, fest aufgesetzte Bedeutung trifft. Die Zuordnung einer Bedeutung zu einem sensorischen Symbol ist sehr wohl abhängig von Kultur bzw. Konvention.

Arbiträre Symbole:
Ein visuelles Symbol ist arbiträr, wenn seine Repräsentation nicht aus einer perzeptuellen Basis heraus entstanden ist. Ein Beispiel für arbiträre Symbole ist Schrift. Arbiträre Symbole leiten sich aus einer Kultur ab, bzw. sind in diese eingebettet. Sie sind schwer zu erlernen und leicht zu vergessen, außer sie werden exzessiv wiederholt und die Verwendung trainiert. Aufgrund der Einbettung in eine Kultur lassen sich diese Symbole nicht einfach verändern. Dafür besitzen sie aber das Potenzial, abstrakte Konzepte zu transportieren. Bei der Erschaffung neuer Symbole kann die Darstellung frei erstellt werden, insofern sie sich nicht ungünstig mit existierenden Symbolen überschneidet.

Bei einer Bewertung eines Symbols bezüglich dem Eignungsgrad als visuelles Informationstransportmittel innerhalb einer Visualisierung müsste theoretisch zunächst die Kategorie unterschieden werden. Bei der Analyse eines sensorischen Symbols kann die Bewertung danach erfolgen, wie gut die Erkennung im Sinne des menschlichen visuellen Systems geschieht. Bei einem arbiträren Symbol muss die Effektivität primär danach bewertet werden, wie gut sich das Symbol an bzw. in die Konventionen des Kulturkreises des Betrachters integriert („Culture influences cognition“, [31], S.17). Demnach benötigt die Bewertung eines arbiträren Symbols einen „Referenzbetrachter“, während bezüglich eines sensorischen Symbols lediglich von einem menschlichen Betrachter ausgegangen werden muss, ggf. unter Beachtung von Abweichungen wie z.B. Farbenblindheit. Erschwert wird die Bewertung einer Visualisierung dadurch, das praktisch die meisten Visualisierungen hybrid aufgebaut sind und Elemente beider Kategorien enthalten. Folglich besitzen die meisten Repräsentationen sowohl Aspekte, die gelernten Konventionen zuzuordnen sind, als auch Bereiche, die von der visuellen Verarbeitung im Gehirn abhängig sind.

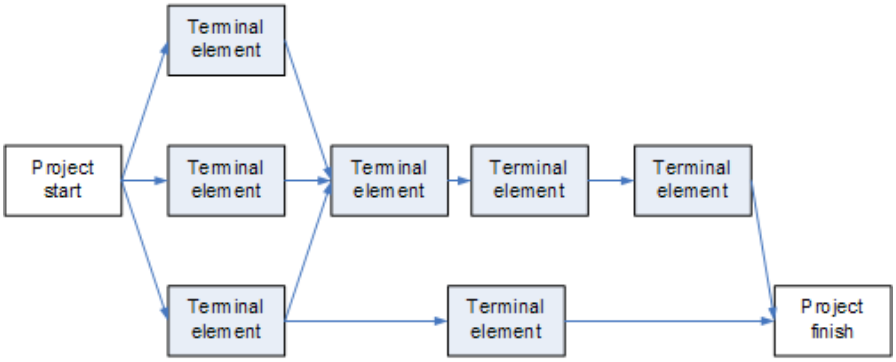


Abbildung 5: Hybride Visualisierung: In diesem Beispiel werden visuelle Entitäten mittels durchgehenden Konturen verbunden (sensorische Symbole) und mittels Schrift identifiziert (arbiträre Symbole).

2.1.3. Lenkung der Aufmerksamkeit

Die Lenkung der Aufmerksamkeit auf bestimmte Stimuli ist ein wichtiger Schritt bei der Erkennung von Objekten. Ohne Lenkung der Aufmerksamkeit auf einen Stimulus ist dessen bewusste Wahrnehmung gar nicht erst möglich. Hierzu unterscheidet Posner in [23]: Das Ausrichten der Aufmerksamkeit auf einen Stimulus als *Orientieren* und dem bewussten Wahrnehmen eines Stimulus als *Detektieren*. Das Orientieren kann sowohl von einer externen Kontrolle, als auch vom zentralen Nervensystem des Subjekts selbst beeinflusst werden. Damit lässt sich erklären, dass wir uns von Äußerungen wie „Guck mal da!“ beeinflussen lassen und unsere Aufmerksamkeit in die angezeigte Richtung lenken.

2.1.4. Features und Feature Maps

Features sind die Eigenschaften von Objekten im Sichtfeld, zum Beispiel Farbe, Größe oder Richtung. Die Erkennung von einzelnen Features kann über große Teile des Sichtfelds parallel durchgeführt werden, da für jedes Feature ein eigener Teil des Wahrnehmungssystems zuständig ist (siehe [30]). Werden die erkannten Features mit ihrer Position im Sichtfeld verknüpft, dann entstehen Feature Maps. Diese sind für die Erkennung von Objekten nötig, die einer Kombination bestimmter Features an der selben Position entsprechen. Diese gleichzeitige Suche nach verschieden Eigenschaften eines Objektes wird auch Conjunction Search genannt.

2.1.5. Guided Search 2.0

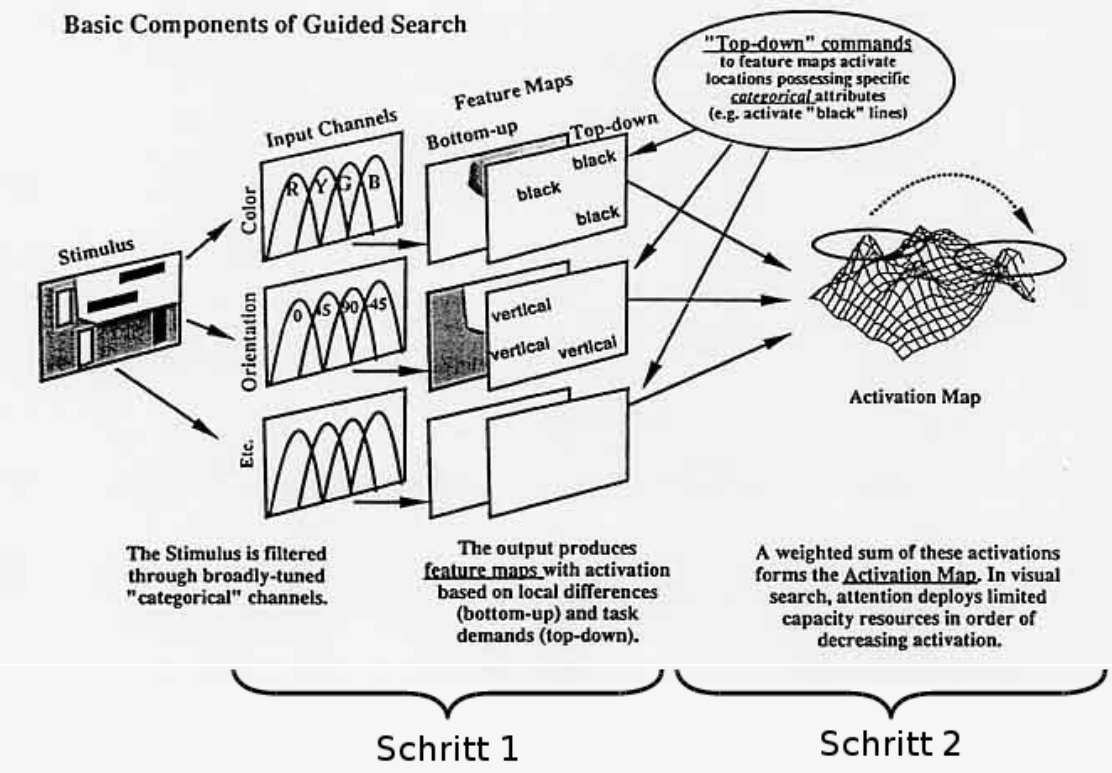


Abbildung 6: Architektur von Guided Search 2.0: unterteilt die Wahrnehmung von Objekten in zwei Schritte.

Das Guided Search 2.0 Modell [47] beschreibt die Suche nach Objekten in zwei Schritten. Der erste Schritt ist die hoch parallel ablaufende Erkennung von Features und Erstellung von Feature Maps. Der zweite Schritt ist die seriell oder zumindest begrenzt parallel ablaufende Suche anhand einer Activation Map. Die Activation Map gibt die Wahrscheinlichkeit an, ob sich das gesuchte Objekt an einer bestimmten Stelle im Sichtfeld befindet, und kann somit für die Lenkung der Aufmerksamkeit auf relevante Objekte im Sichtfeld genutzt werden. Die Activation Map setzt ich aus zwei Teilen zusammen: Der *Bottom-Up-Activation* und der *Top-Down-Activation*.

Mit der Bezeichnung „2.0“ möchte Wolfe ausdrücklich darauf hinweisen, dass sich dieses Modell der Wahrnehmung weiterentwickelt und mittlerweile schon die erste Überarbeitung hinter sich hat.

Bottom-Up-Activation:

Die Bottom-Up-Activation ist ein Maß für die Ungewöhnlichkeit eines Features an einer Position im aktuellen Kontext. Je mehr sich ein Feature an einer Position von seiner Umgebung abhebt, desto höher ist seine Aktivierung. Da sich die Bottom-Up-Activation nur darauf bezieht, wie sehr sich ein Feature an einer Position von seiner Umgebung abhebt, ist sie unabhängig vom Ziel und Wissen des Betrachters über das Problem und das Ziel.

Top-Down-Activation:

Die Top-Down-Activation eines Features wird von dem Ziel beeinflusst, das der Betrachter verfolgt. Diese zweite Art der Aktivierung existiert, um Objekte auch dann noch zu finden, wenn alle Features im Sichtfeld gleich ungewöhnlich sind, also die Bottom-Up-Activation für alle Positionen ungefähr gleich ist, und, um zu vermeiden, dass den Objekten, denen die Aufmerksamkeit zugeteilt wird, der Bezug zum eigentlichen Ziel des Betrachters fehlt. Beispielsweise ist die Top-Down-Activation für ein grünes Objekte gering, wenn der Benutzer tatsächlich nach einem roten Objekt sucht, auch wenn sich das grüne Objekt inmitten von blauen Objekten befindet. In Guided Search 2.0 wird die Top-Down-Activation dazu benutzt, die Werte der Bottom-Up-Activation bezüglich der Relevanz für das Ziel des Betrachters zu gewichten.

Feature-Kanäle

Aktuelle Forschungen ([12], [13], [48]) haben gezeigt, dass sich die Aktivierung eines bestimmten Features für die Suche besser verstehen lässt, wenn man annimmt, dass einige Features, wie zum Beispiel Farbe oder Ausrichtung, in verschiedene Kanäle geteilt sind. So gibt es laut Wolfe für das Feature *Farbe* die Kanäle Rot, Gelb, Grün und Blau. In Abbildung 6 sind diese als Input Channels gekennzeichnet. Die Bottom-Up-Activation wird innerhalb dieser Kanäle bestimmt. So hat ein orangefarbenes Objekt, umgeben von gelben Objekten im Kanal Gelb, nur eine sehr geringe, im Kanal Rot dagegen eine sehr hohe Bottom-Up-Activation. Durch die Top-Down-Activation würde dann für die Suche nach einem orangefarbenen Objekt der Kanal Rot stärker gewichtet werden, als der Kanal Gelb.

2.1.6. Norman's seven stages of the user activity

Die „Seven Stages of Action“ sind ein Modell des Usability-Wissenschaftlers Donald Norman. Das Modell versucht die Bewältigung einer Aufgabe durch eine Person anhand

von 7 Phasen zu beschreiben. Es wurde erstmals 1990 in Normans Buch „The Design of Everyday Things“ [22] veröffentlicht.

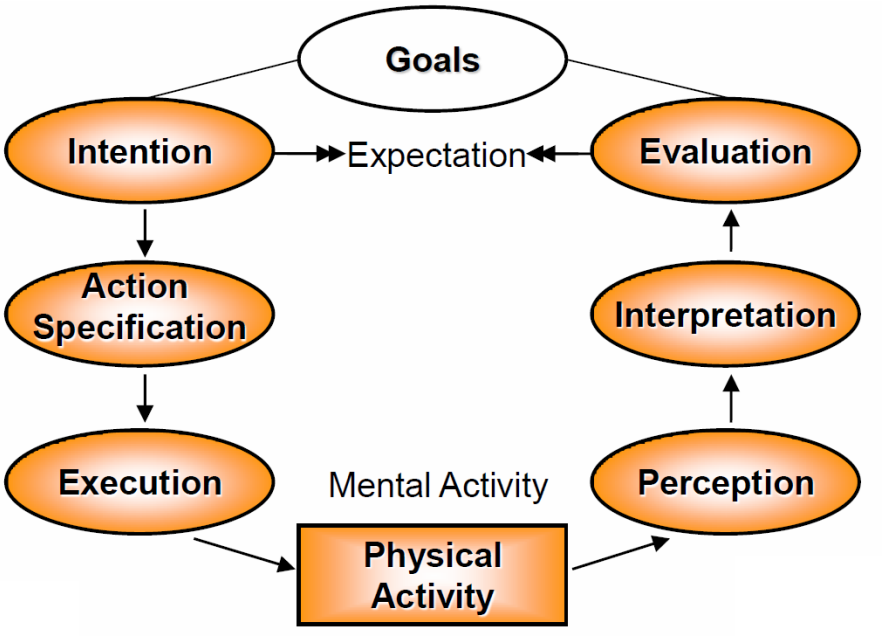


Abbildung 7: Norman’s seven stages of the user activity. Abbildung enstammt [32], Kapitel 2.2

Die Phasen haben dabei folgende Bedeutung

1. Intention: Was soll getan werden?
2. Action Specification: Wie soll dies geschehen?
3. Execution: Ausführung des Plans (Mental)
4. Physical Activity: Ausführung des Plans (Physisch)
5. Perception: Wahrnehmung der Ausführung
6. Interpretation (der Ausführung)
7. Evaluation: Stimmt die Ausführung mit den Erwartungen überein?

Die 7 Phasen aus Normans Modell lassen sich im Allgemeinen drei Bereichen der menschlichen Informationsverarbeitung zuordnen. Die Zuordnung geschieht über die vorherige Nummerierung ([32], Kapitel 2.3).

- Perzeption: Informationsaquise mittels Sinnesorganen (Norman: 5)
- Kognition (Norman 1,2,3,6,7):
  - Gedächtnis: Speichern von Informationen
  - Kommunikation: Austausch von Informationen
  - Denken/Entscheiden: Verarbeiten von Informationen
- Motorische Funktionen: Ausführen und Manipulation der Umwelt (Norman: 4)

Das Gedächtnis des Gehirns, wird in drei Bereiche unterteilt ([32], Kapitel 2.3):

- *Sensorisches Gedächtnis*: Kurzzeitiger Speicher für eingehende sensorische Informationen. Hohe Kapazität, aber aufgrund ständiger Aktualisierung nur sehr kurze Speicherdauer (im Millisekunden-Bereich).
- *Kurzzeitgedächtnis*: Speicher für symbolische Informationen. Speicherdauer ungefähr 15 Sekunden. Kapazität ungefähr 7 +/- 2 Elemente.
- *Langzeitgedächtnis*: Speicher für semantische, lang vorgehaltene Informationen. Hohe Kapazität, aber langsamer Zugriff.

2.2. Was macht eine gute Visualisierung aus?

Eine Datenmenge lässt sich auf nahezu beliebige Art und Weise in eine Visualisierung überführen, wobei manche Darstellungen besser geeignet sind als andere. In diesem Kapitel soll darauf eingegangen werden, welche grundsätzlichen Möglichkeiten es gibt Visualisierungen zu erzeugen. Abschnitt 2.2.1 zeigt typische Visualisierungstechniken abhängig von den zugrunde liegenden Daten auf, in 2.2.2 wird der Visualisierungsprozess dargestellt, welcher die Schritte aufzeigt, um Daten in eine graphische Darstellung zu überführen und in 2.2.3 welche Optionen dabei zur Verfügung stehen. Abschließend werden vorhandene Standards und Richtlinien für die Erstellung grafischer Oberflächen behandelt (2.2.4), welche als Basis für eine Bewertung verwendet werden können.

2.2.1. Einführung zu Visualisierungen

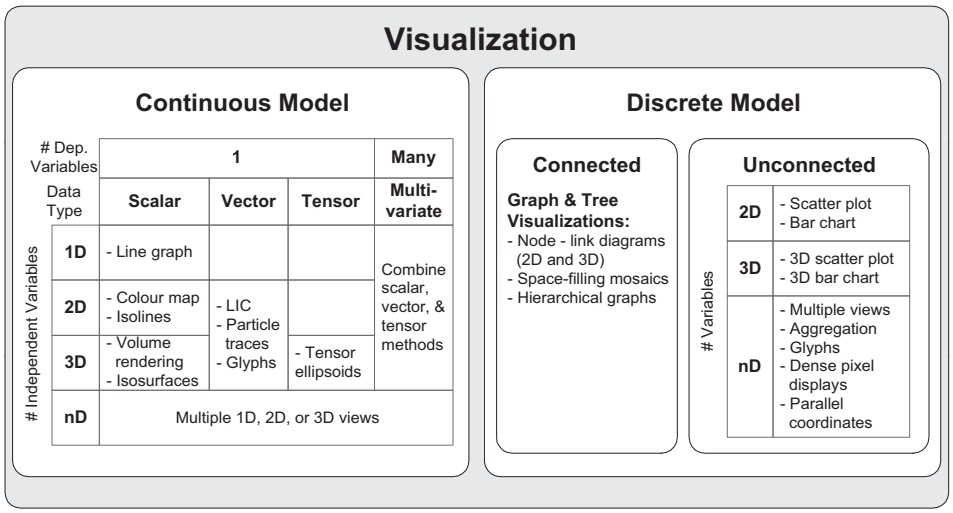


Abbildung 8: Klassifizierung von Visualisierungstechniken: Das Modell unterteilt Visualisierungstechniken nach den zu visualisierenden Daten. [11]

Visualisierungen lassen sich nach dem Schema aus Abbildung 8 einteilen [11]. Dieses Modell klassifiziert Visualisierung auf der ersten Ebene nach den darunter liegenden Daten, wobei hier zwischen

- kontinuierlichen
- diskreten

Daten unterschieden wird. Dann werden die jeweils typischen Visualisierungstechniken eingeordnet:

- kontinuierliche Datenmodelle:
  - Liniengraphen
  - Isolinien
  - Volumenrendering
  - Glyphendarstellungen
  - Tensorvisualisierungen

- diskrete Datenmodelle:
  - Graph- und Baumvisualsierungen
  - 2D- / 3D-Scatterplots
  - Parallele Koordinaten

Betrachten wir den kontinuierlichen Fall, dabei wird die Dimension der Daten (Spalten in Abb. 8) mit der Dimension des Anwendungsbereichs (Zeilen in Abb. 8) gegenübergestellt. So gibt es z.B. bei einer Landkarte mit Höhenlinien zu jedem Gitterpunkt einen skalaren Wert, die Höhe, welche auf einer zweidimensionalen Karte durch Linien aus Punkten mit dem selben Höhenwert dargestellt werden (Abb. 9 links). Für skalare Daten im 3D Anwendungsbereich werden Techniken des Volumenrenderings eingesetzt. Typischer Einsatz dieser Technik ist z.B. die Visualisierung von Daten aus medizinischen Scan-Verfahren wie Magnetresonanztomographie oder Computertomographie (Abb. 9 mitte). Vektorwertige Daten treten hauptsächlich in der Strömungsvisualisierung auf, wobei das Bewegungsverhalten von Gasen und Flüssigkeiten untersucht wird (Abb. 9 rechts).

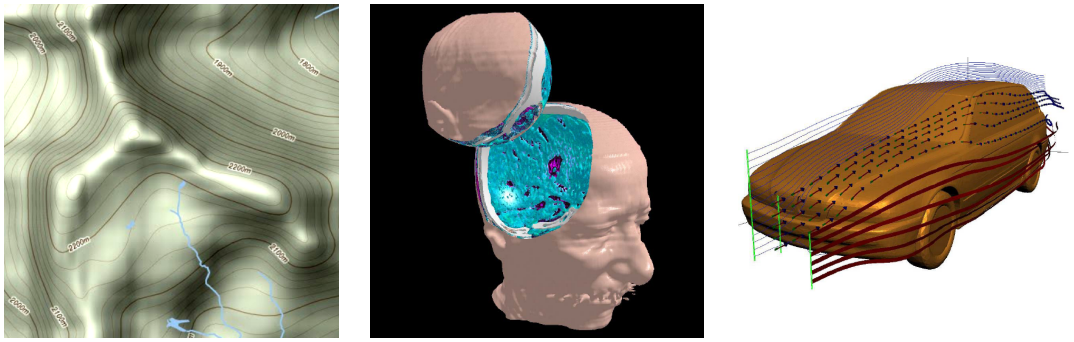


Abbildung 9: Beispiele unterschiedlicher Visualisierungstechniken mit kontinuierlichen Daten: Höhenfeldlinien einer Landkarte (links) [14], Visualisierung eines CT-Scans (Mitte) [11], Strömungsvisualisierung (rechts) [11]

Die Kategorie der diskreten Daten lässt sich in zwei Bereiche unterteilen, wobei unterschieden wird, ob die einzelnen Datenpunkte miteinander verbunden oder nicht verbunden sind. Im ersten Fall können hauptsächlich Graph- und Hierarchie-Visualisierungen eingesetzt werden, worunter z.B. Treemaps fallen, wie sie auch in Abbildung 11 zu sehen sind. Dabei wird die Baumstruktur durch ineinander geschachtelte Rechtecke dargestellt. Das äußerste Rechteck stellt die Wurzel dar, wobei dessen Fläche entsprechend der Hierarchie weiter unterteilt wird. Nicht mehr weiter unterteilte Flächen stehen für die Blätter, bzw. in einem Dateisystem für Dateien.

Die zweite Unterkategorie gruppiert Visualisierungstechniken, deren Datenpunkte untereinander nicht verbunden sind. Eine sehr häufig eingesetzte Technik sind hierbei Scatter Plots (Abbildung 11), welche direkt die Datenwerte in einem zweidimensionalen oder auch dreidimensionalen Diagramm darstellen, um so Korrelationen in den Datensätzen erkennen zu können. Für höher dimensionale Daten eignen sich Verfahren wie Parallele Koordinaten oder Glyphen Darstellungen. Parallele Koordinaten besitzen entsprechend der Dimension der Daten mehrere nebeneinander angeordnete Achsen, wobei sich die Skalen auf den Achsen auch unterscheiden können. Pro Datensatz wird der Wert auf jeder Achse eingetragen und über eine Profillinie mit benachbarten Achsen verbunden, so dass die Zugehörigkeit der einzelnen Werte zu einem Datum erkennbar bleibt (Abb. 10 links). Bei Glyphen dagegen werden mehrere Datenattribute auf ein graphisches Objekt abgebildet. So könnte z.B. ein Verkäufer seine Absatzmärkte mit Hilfe von Glyphen visualisieren (Abb. 10 rechts) [31]: Hierbei könnten die Glyphen entsprechend den Ländern auf einer Karte positioniert werden, das geschätzte Einkommen der potentiellen Käufer wird über die Größe der Glyphen kodiert und die Farbe könnte den durchschnittlichen Bildungsgrad anzeigen (siehe auch 2.2.3).

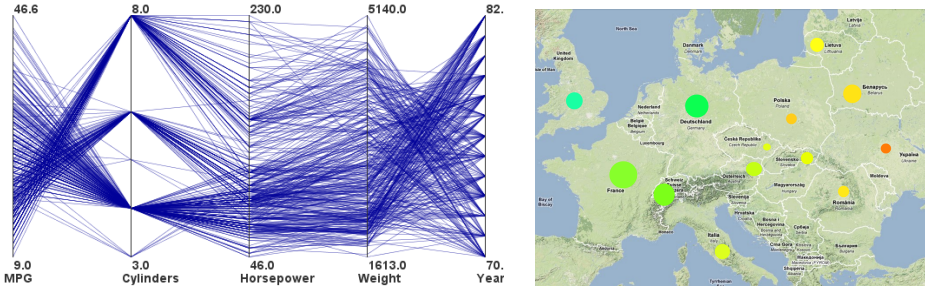


Abbildung 10: Visualisierung hochdimensionaler Daten: Visualisierung eines Autodatensatzen über Parallele Koordinaten (links) [21], Glyphenvisualisierung der Absatzmärkte eines Verkäufers (rechts).

Diese Klassifizierung kann als Grundlage für die Einteilung des Visualisierungsgebiets in Informations- und Wissenschaftliche Visualisierung gesehen werden:

Informationsvisualisierung

Die Daten im Bereich der Informationsvisualisierung bestehen typischerweise aus diskreten Datenpunkten, welche exakt vorliegen bzw. erfasst werden können, also keine Interpolation erfordern. Es gibt keinen räumlichen Bezug und oftmals besitzen die Daten sehr viele Dimensionen. Verbindungen oder Hierarchien in den Daten werden oft durch Graph- oder Baumstrukturen dargestellt [11].

Wichtige Anforderungen an eine Visualisierung sind unter anderem die Expressivität

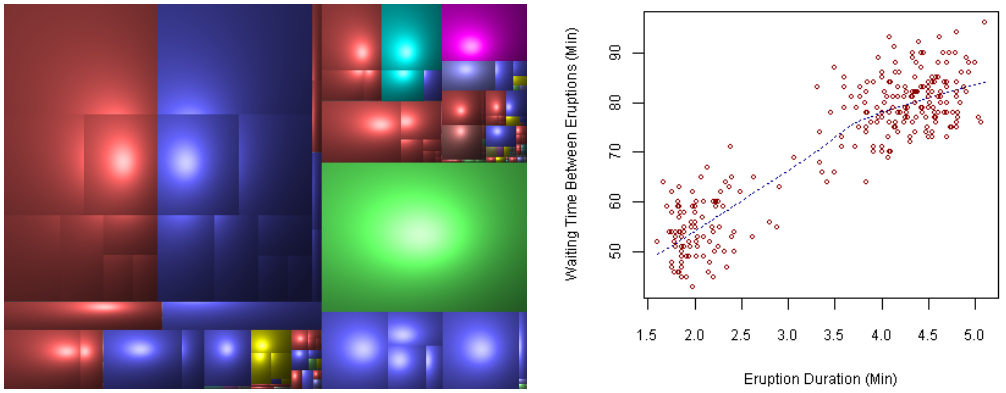


Abbildung 11: Beispiele unterschiedlicher Visualisierungstechniken mit diskreten Daten: Visualisierung eines Dateisystems als Treemap (links) [46], Scatterplot-Visualisierung der Eruptionsdauer und -abstände eines Geiser im Yellowstone Nationalpark (rechts) [39].

und Effektivität [28, 36]. Von einer expressiven Visualisierung wird gefordert, dass keine Informationen dargestellt werden, welche nicht in den Daten enthalten sind, also keine falschen Annahmen daraus abgeleitet werden können. Wohingegen die Effektivität angibt, wie schnell sich der Betrachter in einer Visualisierung zurecht findet und sein Ziel damit erreichen kann.

Anwendungsbereiche sind unter anderem die Visualisierung von Dateisystemen (Abbildung 11), Softwarekomponenten oder auch Netzwerkverbindungen im Internet. Die bekanntesten Darstellungen im Bereich Softwarevisualisierung sind UML-Diagramme. Diese decken ein recht breites Spektrum zur Darstellung von Softwarekomponenten ab, wie z.B. Komponentendiagramme, welche die Architektur eines Softwaresystems auf einer sehr abstrakten Ebene beschreiben. Einen etwas detaillierteren Blick auf die Struktur einer Software geben Klassendiagramme, worüber sich Beziehungen und Eigenschaften von einzelnen Klassen darstellen lassen. Für noch tiefere Einblicke eignen sich z.B. Aktivitätsdiagramme, um den Ablauf eines Algorithmus graphisch darzustellen.

Wissenschaftliche Visualisierung

Der Bereich der wissenschaftlichen Visualisierung beschäftigt sich hauptsächlich mit kontinuierlichen Datenmodellen. Bedingt durch eine begrenzte Anzahl an Sensoren erfolgt die Datenerfassung räumlich und zeitlich diskret, sodass für eine kontinuierliche Darstellung die Daten zusätzlich interpoliert werden müssen. Der Anwendungsbereich beschränkt sich hier meist auf zwei oder drei Dimensionen und besteht in der Regel aus mehr Datenpunkten als in der Informationsvisualisierung [11].

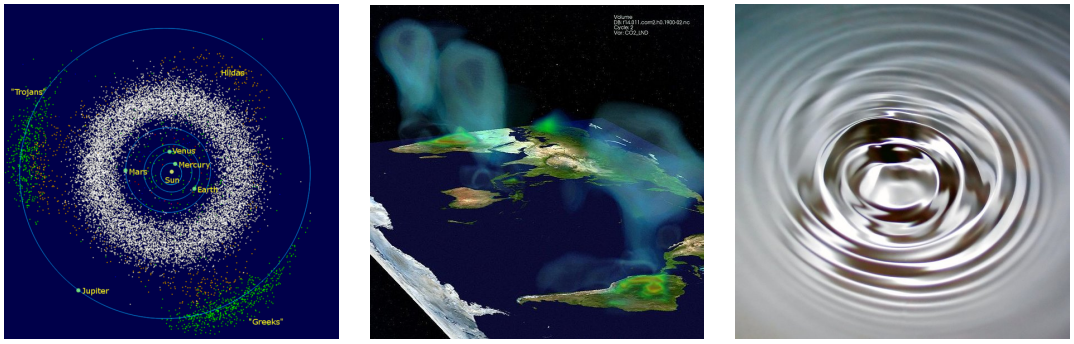


Abbildung 12: Beispiele wissenschaftlicher Visualisierungen: Visualisierung des Asteroidengürtels im inneren Sonnensystems (links) [40], Tornado Simulation (Mitte) [40], Wellensimulation auf einer Wasseroberfläche (rechts) [40]

2.2.2. Visualisierungspipeline

Die Visualisierungspipeline (Abbildung 13) beschreibt den Prozess, der durchlaufen wird, um eine Datenmenge in eine visuelle Darstellung zu überführen [11]:

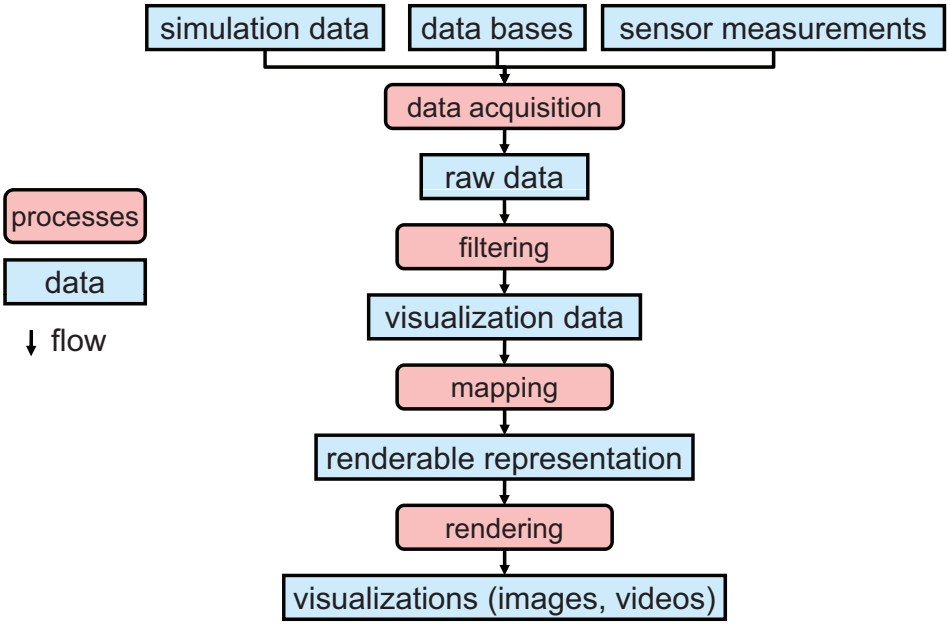


Abbildung 13: Visualisierungspipeline: Die Pipeline beschreibt den zu durchlaufenden Prozess, um eine Menge an Rohdaten in eine Visualisierung zu überführen. [11]

1. **Data Acquisition:** Der erste Schritt in der Pipeline beschreibt die Erfassung der Daten. Diese können von einer beliebigen Quelle stammen und sich stark in ihrer Größe unterscheiden. So können z.B. Ereignisse aus der realen Welt, wie Wetterdaten, astronomische Beobachtungen oder seismische Aktivitäten verwendet werden. Im theoretischen Bereich lassen sich Daten über Computersimulationen, wie Molekülmodellierung oder aus CAD Anwendungen gewinnen. Riesige Datenmengen bietet auch das Internet selbst oder die Daten zu Aktienmärkten. So beträgt die Datenmenge aus medizinischen Scannern einige Megabytes, während sich astronomische Beobachtungen schnell im Terrabyte-Bereich bewegen können.

2. **Filtering:** Das Filtering beschreibt eine Daten-zu-Daten Abbildung. Hierbei werden die Rohdaten der ersten Stufen verwendet, um daraus die zu visualisierenden Daten zu erstellen. Mögliche Transformationen, die hier durchgeführt werden sind unter anderem die Interpolation bzw. Approximation der Daten. Dies ist oft nötig, da Sensoren entsprechende Werte nur an diskreten Gitterpunkten erfassen können, die Visualisierung aber ein kontinuierliches Bild darstellen soll. Weiterhin können hier nicht benötigte Daten verworfen werden oder das Format der Daten kann in eine für die Visualisierung besser geeignete Darstellung konvertiert werden.

3. **Mapping:** Im dritten Schritt werden die Visualisierungsdaten durch das Mapping auf graphische Primitive abgebildet. Dies können abhängig von der Dimension Punkte, Linien, Flächen oder auch Volumenelemente sein, wobei diese zusätzlich Attribute für Farbe und Texturen erhalten.

4. **Rendering:** Zum Schluss werden die graphischen Primitive gerendert, wobei noch zusätzlicher Realismus eingefügt werden kann, wie z.B. Beleuchtungseffekte, Schatten oder Schraffierungen.

2.2.3. Visuelle Variablen

Im Mapping-Schritt der Visualisierungspipeline 2.2.2 werden die zu visualisierenden Daten auf graphische Elemente abgebildet. Hierfür lassen sich verschiedene visuelle Variablen identifizieren, auf welche die Daten abgebildet werden können [11]:

- Position x, y, (z)
- Größe
- Orientierung
- Form
- Helligkeit

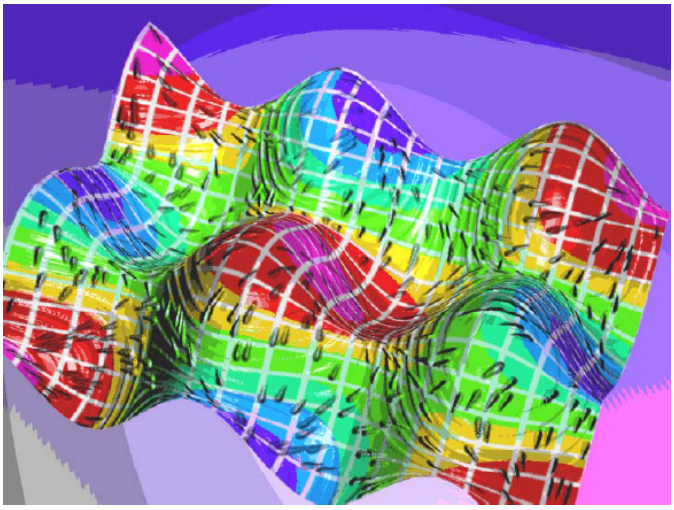


Abbildung 14: Abstrakte Visualisierung: Beispiel für die Kombination mehrerer visuelle Variablen. Die Gitterstruktur legt die Position im dreidimensionalen Raum fest, über den Farbverlauf wird ein skalarer Wert kodiert und zusätzlich zeigen Glyphen über ihre Orientierung und Größe weitere Daten an. [11]

- Farbe
- Textur

Die Effektivität der einzelnen Variablen, in Bezug auf die Wahrnehmungsfähigkeit des Benutzers, hängt stark von den Daten ab, welche visualisiert werden sollen. So ist die Position allgemein eine sehr effektive Variable und eignet sich für quantitative als auch für ordinale Daten sehr gut. Dies findet z.B. Anwendung bei Scatter Plots, wie in Abbildung 11 zu sehen ist. Während z.B. die Farbe schlecht geeignet ist, um quantitative Daten unterscheiden zu können, eignet sie sich für nominale Daten dagegen sehr gut. Zudem lässt sich Farbe mit den meisten anderen Techniken kombinieren, da sie bei gezieltem Einsatz, also sofern die Darstellung nicht zu überladen wird, als Warnsignal oder zur Gruppierung von Elementen eingesetzt werden kann.

Abbildung 14 zeigt eine Visualisierung, in der mehrere dieser Variablen kombiniert wurden. So wird durch die Gitterstruktur die Position im dreidimensionalen Raum festgelegt. Über eine Regenbogen-Farbtabelle wird ein zusätzlicher Wert pro Datenpunkt dargestellt, welcher sich von der Höhe unterscheidet. Zusätzlich befinden sich auf der Fläche Glyphen, welche eine unterschiedliche Orientierung besitzen und sich in ihrer Größe und ihrem Helligkeitswert unterscheiden.

2.2.4. Design-Richtlinien

Mögliche Bewertungskriterien für graphische Benutzungsoberflächen oder auch Visualisierungen lassen sich aus bereits vorhandenen Richtlinien bzw. Gesetzmäßigkeiten der menschlichen Wahrnehmung ableiten. Allgemein sollen diese Richtlinien bei der Erstellung einer graphischen Ausgabe helfen, um Fähigkeiten der Wahrnehmung zu unterstützen. Damit lässt sich auch eine bereits vorhandene GUI bzw. Visualisierung evaluieren. Im Folgenden werden einige dieser Standards näher betrachtet:

Shneidermans 8 Goldene Regeln

Die 8 Goldenen Regeln der Dialoggestaltung [17] von Ben Shneiderman umfassen:

1. Streben nach Konsistenz
2. Abkürzungen für erfahrene Benutzer anbieten
3. Informatives Feedback
4. Abgeschlossenheit von Dialogen
5. Einfache Fehlerbehandlung
6. Einfache Umkehrbarkeit von Aktionen
7. Kontrolle des Systems durch den Benutzer
8. Belastungen des Kurzzeitgedächtnisses reduzieren

Shneiderman konzentriert sich mit seinen 8 Goldenen Regeln hauptsächlich auf das Design von GUIs. Die meisten dieser Regeln beziehen sich auf die Interaktion mit einer graphischen Oberfläche. So besagt z.B. die erste Regel *Streben nach Konsistenz*, dass ähnliche Situationen durch eine Folge von konsistenten Aktionen behandelt werden können, oder Aktionen, die zum selben Ergebnis führen auch gleich benannt werden sollten. Die Regeln berücksichtigen sowohl Anfänger als auch fortgeschrittene Benutzer gleichermaßen. So besagt die zweite Regel *Abkürzungen für erfahrene Benutzer anbieten*, dass häufig benutzte Funktionen über Abkürzungen erreichbar sein sollten, um so die Effizienz zu erhöhen. Die Funktionen müssen aber dennoch über den normalen Weg aufgerufen werden können. Weitere Regeln gehen auf informatives Feedback, einfache Fehlerbehandlung oder auch die Umkehrung von Aktionen ein. Als interessant stellt sich noch die achte Regel dar, welche besagt, dass die Oberfläche möglichst einfach und übersichtlich gehalten und nicht mit unnötigen Komponenten oder gar Animationen überladen werden sollte.

Die 8 Goldenen Regeln sind alle sehr allgemein gehalten. So stellt es sich wohl als schwierig heraus, automatisch zu überprüfen, ob ein bestimmter Dialog nun ausreichend Rückmeldung auf ein Problem gibt oder nicht. Selbst bei der Evaluation durch einen Menschen, hängt eine solche Beurteilung noch stark von dessen Vorwissen und Erfahrung ab. So kann für einen IT-Experten die Anzeige eines Fehlercodes bereits ausreichend sein, während ein anderer Endanwender detaillierte Anweisungen benötigt, um fortfahren zu können. Weitere Regeln, wie die Abgeschlossenheit von Dialogen, lassen noch viel Spielraum für eine konkrete Umsetzung bzw. Bewertung.

ISO 9241-110 Grundsätze der Dialoggestaltung

Eine weitere wichtige Richtlinie stellt der Standard *ISO 9241-110 Grundsätze der Dialoggestaltung* [17] dar. Darin werden insgesamt sieben Grundsätze definiert, welche das Design von Dialogsystemen unterstützen soll:

1. Aufgabenangemessenheit
2. Selbstbeschreibungsfähigkeit
3. Steuerbarkeit
4. Erwartungskonformität
5. Fehlertoleranz
6. Individualisierbarkeit
7. Lernförderlichkeit

Die *Aufgabenangemessenheit* besagt, dass die graphische Oberfläche entsprechend der Arbeitsaufgabe aufgebaut sein soll. Dabei steht nicht primär im Vordergrund wie etwas technisch gelöst werden könnte, sondern wie es vom Benutzer nachvollzogen werden kann. Zudem sollte der Benutzer bei seiner Aufgabe unterstützt werden, um schnell und effektiv arbeiten zu können. Ein weiterer Grundsatz ist die *Selbstbeschreibungsfähigkeit*, wonach jeder Dialog Auskunft geben muss, welche Aktionen von diesem ausführbar sind. Die *Steuerbarkeit* von Dialogabläufen, beschreibt die Möglichkeit, dass Benutzer die Richtung und Geschwindigkeit beeinflussen können. Ein Dialogsystem ist *Erwartungskonform*, sofern es sich an allgemein anerkannte Konventionen hält. Zusätzlich sollte das System *fehlertolerant* sein und dem Benutzer die Möglichkeit bieten, bei falschen Eingaben eine Korrektur vorzunehmen, ohne mit der Dateneingabe von Neuem beginnen zu müssen. Die *Individualisierbarkeit* beschreibt die Anpassung der Darstellung an individuelle Fähigkeiten und Bedürfnisse. Eine GUI ist *lernförderlich*, sofern sie den Benutzer bei der Interaktion unterstützt und anleitet.

Allgemein lassen sich hier sehr viele Parallelen zu den Regeln von Shneiderman erkennen. So deckt sich z.B. der Grundsatz zur *Erwartungskonformität* mit der ersten Regel *Streben nach Konsistenz* von Shneiderman. Analoges gilt für die *Fehlertoleranz* oder auch die *Selbstbeschreibungsfähigkeit*. Daher können hier, in Bezug auf die Evaluation einer GUI dieselben Aussagen, wie bereits oben aufgeführt getroffen werden.

Für eine GUI geben diese Regeln gute Richtlinien vor, an die man sich beim Entwerfen halten sollte. Allerdings sind diese für eine Visualisierung nicht mehr ausreichend. Hier könnte der Punkt Selbstbeschreibungsfähigkeit entsprechend erweitert werden, welcher das Effektivitäts-Kriterium (siehe 2.2.1) berücksichtigt und damit Auskunft gibt, wie schnell der Benutzer die Absicht der Visualisierung erkennen kann. Ein weiteres Kriterium, welches vor allem bei dreidimensionalen Visualisierungen zum tragen kommt, ist das Verdeckungsproblem. So kann es sein, dass unter bestimmten Betrachtungswinkeln wichtige Informationen im Hintergrund liegen und von davor liegenden Objekten verdeckt werden, wodurch falsche Annahmen getroffen werden können. Andererseits verlieren auch manche Punkte, welche bei GUIs wichtig sind, hier an Bedeutung. Bei einer Visualisierung ist z.B. die Fehlertoleranz oder Individualisierbarkeit nebensächlich. Die Dateneingabe in einer Visualisierung beschränkt sich hauptsächlich auf die Auswahl von bestimmten Werten oder Bereichen, da die eigentlichen Daten bereits durch die Filtering-Stufe (siehe 2.2.2) eingegeben wurden.

Gestaltgesetze

Das Kapitel stellt eine Zusammenfassung von [31], S.189 - 198 dar.

Einen wesentlichen Beitrag zur Beschreibung der Wahrnehmung von Mustern lieferten Max Westheimer, Kurt Koffka und Wolfgang Kohler in den sogenannten „Gestaltgesetzen“. Die folgenden Phänomene beschreiben unterschiedliche Muster, die vom menschlichen visuellen System erkannt werden können.

Nähe: Das visuelle System gruppiert visuelle Entitäten, die räumlich nah beieinander liegen und es gruppiert Regionen mit vergleichbarer Elementdichte. Die räumliche Nähe stellt das einfachste, aber gleichzeitig effizienteste Mittel dar, um Beziehungen zwischen visuellen Elementen auszudrücken. Siehe Abbildung 15.

Abbildung 15: Gestaltgesetz: Gruppierung durch Nähe [31], S.189

Ähnlichkeit: Ähnliche Elemente erscheinen gruppiert. Die „Ähnlichkeit“ kann über unterschiedliche Dimensionen erfolgen, z.B. Form, Farbe oder Textur. Auch ist es möglich mehrere Dimensionen zu überlagern. Siehe Abbildung 16.

Abbildung 16: Gestaltgesetz: Gruppierung durch Ähnlichkeit [32]

Verbundene Elemente: Elemente, die visuell verbunden sind, z.B. durch Linien, erscheinen gruppiert. Je nach Art der Verbindung kann diese sogar stärker wirken als Nähe oder Ähnlichkeit. Siehe Abbildung 17.

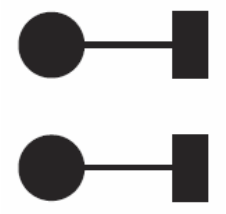


Abbildung 17: Gestaltgesetz: Gruppierung durch Verbundenheit [31], S.192

Kontinuität: Visuelle Elemente werden stärker als eine Entität wahrgenommen, wenn ihre Form fließend verläuft, als jene deren Form abrupte Richtungswechsel beinhalten. Siehe Abbildung 18.

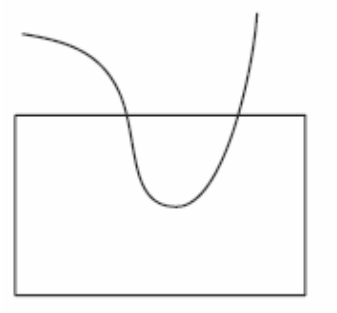


Abbildung 18: Gestaltgesetz Kontinuität: Die Linie wird als eigenständiges, dem Rechteck überlagertes Objekt wahrgenommen [31], S.192

Symmetrie: Symmetrie zwischen Formen, oder auch Abweichungen in der Symmetrie können gut erkannt werden. Ein Beispiel dafür ist eine Altersstrukturpyramide, getrennt nach Geschlecht, in der man sehr leicht erkennt in welchen Altersgruppen ähnliche Mengen vorhanden sind und wo besonders große Abweichungen sind. Siehe Abbildung 19.

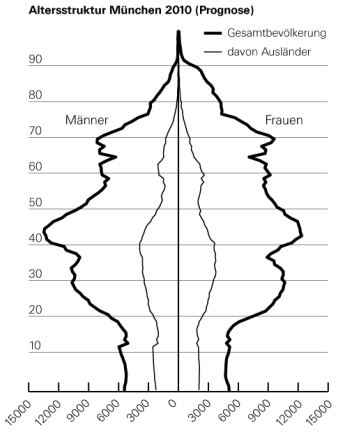


Abbildung 19: Gestaltgesetz Symmetrie: Abweichungen zwischen beiden Gruppen werden leicht erkannt [33]

Geschlossenheit: Eine geschlossene Kontur wird sehr wahrscheinlich als einzelnes Objekt wahrgenommen. Zusätzlich gibt es eine perzeptionelle Tendenz, visuelle Regionen durch geschlossene Konturen in „Außen“ und „Innen“ zu unterteilen. Ein offensichtliches Beispiel hierfür ist das Fenster-Konzept in Benutzungsschnittstellen. Diese werden oft auch mehrfach geschachtelt. Siehe Abbildung 20.

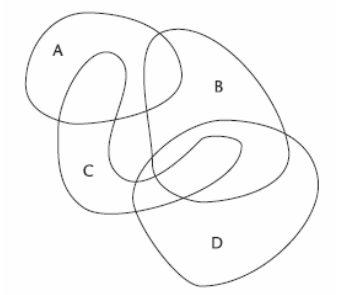


Abbildung 20: Gestaltgesetz: Geschlossene Formen [31], S.195

Relative Größe: Im Allgemeinen werden kleine Teile von Mustern stärker als Objekt wahrgenommen als große. Siehe Abbildung 21.

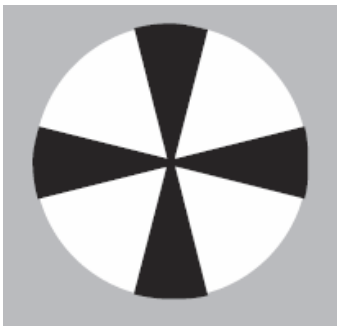


Abbildung 21: Gestaltgesetz Relative Größe: Die schwarzen Bereiche sind kleiner als die weißen und werden eher als Objekt wahrgenommen [31], S.197

Gestalt und Hintergrund: Visuelle Objekte, die als solche erkannt werden, scheinen eher im Vordergrund zu liegen. Siehe Abbildung 22.

Abbildung 22: Gestaltgesetz zu Gestalt und Hintergrund: Sieht man die Köpfe, scheint die schwarze Fläche kein definiertes Objekt zu sein. Sieht man umgekehrt die Vase, sind die Köpfe nur noch eine weiße Fläche im Hintergrund [31], S.198

Die Gestaltgesetze sind abstrakt genug, um sie auf die meisten Klassen von Visualisierungen anzuwenden. „Abstrakt“ soll in diesem Sinne bedeuten, dass sich für beliebige visuelle Objekte ein Zusammenhang nach einem Gestaltgesetz überprüfen lässt, ohne, dass diese in einen bestimmten Kontext eingebettet sein müssen. Es ist egal, ob die visuellen Objekte nun Teil eines technischen Diagramms, eines Gemäldes oder einer Volumenvisualisierung sind. Diese „Kontextfreiheit“ ist aber nicht völlig problemfrei. Für uns ist primär die Analyse einer Visualisierung nach den Gesetzen, primär Nähe, Ähnlichkeit, Verbundenheit und Kontinuität von Interesse. Eine grundsätzliche Frage ist hierbei: Angenommen eine Visualisierung ist vorgegeben, ließe sich durch Prüfung der Gestaltgesetze entscheiden, ob zwischen zwei visuellen Elementen eine Relation zum Ausdruck gebracht wird? Der Nachweis einer solchen Relation bedeutet aber nicht, dass die Relation auch semantisch korrekt in Bezug auf eine vorgegebene Fragestellung bezüglich der Visualisierung ist, ob der dargestellte Zusammenhang also überhaupt korrekt ist. Eine Prüfung macht demnach nur Sinn, wenn maschinell entscheidbar ist, ob die Relation korrekt ist. Folglich ist es schwer eine Visualisierung maschinell bewerten zu wollen, wenn nicht schon vorher zumindest auf abstrakter Ebene beschreibbar ist, welche Zusammenhänge Sinn ergeben.

Braths Metriken und Richtlinien

Brath stellt in [5] und [6] Metriken und Richtlinien für die Entwicklung von effizienten Visualisierungen vor, die auf seinen Erfahrungen aus ca. 130 Projekten aufbauen. Vorschläge für Metriken sind zum Beispiel die Anzahl der gleichzeitig dargestellten Dimensionen, die Anzahl der dargestellten Datenpunkte oder das Verhältnis von verdeckten Datenpunkten zu den sichtbaren. All diese Metriken können zur Bewertung der Komplexität einer Visualisierung herangezogen werden, wobei Brath davon ausgeht, dass eine

komplexe Visualisierung schwerer zu verstehen ist. Brath rät diese Metriken nicht unreflektiert einzusetzen, weil die Aussagekraft nicht für alle Visualisierungen gleich gegeben ist und sogar für einige Visualisierungen das Gegenteil der Realität widerspiegeln.

Braths Richtlinien orientieren sich an den Grundlagen der Perzeption, wie sie weiter oben beschrieben wurden. Ein wichtiger Punkt in diesen Richtlinien ist die möglichst intensive Nutzung der frühen Stadien der Perzeption, weil diese hoch-parallel arbeiten und durch effiziente Nutzung eine Suche in (nahezu) konstanter Zeit ermöglichen. Diese frühen Stadien der Perzeption würden im Guided-Search-Modell von Wolfe der Erkennung von Features und der Erstellung der Activation-Map entsprechen. Für die Richtlinien von Brath bedeutet das also, dass die Features der einzelnen Objekte so gewählt werden sollten, dass die Aktivierung an den Stellen am höchsten ist, an denen die interessanten Dinge in der Visualisierung passieren. Eine Richtlinie, die sich daraus ableitet ist zum Beispiel: „Use the most general form, color, orientation and texture to represent the typical or expected case. Use derivation from the general case to draw attention to the unexpected or unique information.“ ([6], S. 25).

Beim Einsatz von Braths Metriken und Richtlinien ist außerdem zu beachten, dass diese für statische 3D-Informationsvisualisierungen gedacht sind. Weiterhin wird spezielle Hardware, bzw. Anzeigetechniken außen vor gelassen, zum Beispiel stereoskopische Darstellungen und Eingabegeräte wie Datenhandschuhe. Die Beschränkung auf statische Visualisierungen kommt daher, dass viele Visualisierungen auch ausgedruckt oder als Screenshot verteilt werden und auf diesen Wegen die Möglichkeit der Navigation in der Visualisierung verloren geht.

2.2.5. Fazit

Zusammenfassend, auf die Fragestellung „*Was macht eine gute Visualisierung aus?*“ lassen sich folgende Punkte aus den oben behandelten Themen gewinnen:

- Eine Fragestellung zu einer Visualisierung lässt sich um so schneller beantworten, je weniger Objekte im Arbeitsgedächtnis gehalten werden müssen und je weniger Visual Queries benötigt werden (siehe 2.1.1).

- Da die Effektivität von arbiträren Symbolen stark vom Kulturkreis abhängt, muss stets die Zielgruppe im Auge behalten werden (siehe 2.1.2).

- Die Eignung unterschiedlicher visueller Variablen hängt stark vom Datentyp ab. Im Allgemeinen lässt sich sagen, dass nominale Daten sich gut auf Farben abbilden lassen, während quantitative Daten besser über Positionsunterschiede abgelesen werden können (siehe 2.2.3).

- Die Visualisierung sollte nicht überladen werden, um somit das Kurzzeitgedächtnis

nicht zu stark zu belasten (siehe Shneidermans 8 Goldene Regeln in 2.2.4 und 2.1.1).

- Beachten der Gestaltgesetze, um so den Wahrnehmungsprozess zu unterstützen (siehe Gestaltgesetze in 2.2.4).

3. Frameworks

Eine Suche im Internet fördert eine Vielzahl verschiedener kognitiver Frameworks, Theorien und Architekturen zu Tage. Bei einer genaueren Betrachtung stellt sich dann aber heraus, dass zu den meisten Theorien und Architekturen keine oder nur eine veraltete Implementierung existiert, wodurch sie sich für den praktischen Einsatz im Umfeld dieser Fachstudie weniger eignen. Bei bestehenden Frameworks haben wir festgestellt, dass es zwei Extrema gibt. Das eine Extrem sind Frameworks, die schon seit einiger Zeit im praktischen Einsatz etabliert sind, sich aber auf die Lösung spezieller Probleme beschränken, beispielsweise Frameworks zur Steuerung von Robotern. Das andere Extrem sind Frameworks, deren Ziel es ist, den menschlichen Verstand zu simulieren, die sich aber noch in der Konzeptionsphase oder am Beginn der Implementierung befinden. Es gibt wenige Frameworks, die für die Lösung einer großen Menge von Problemen geeignet sind und deren Implementierung sich in einem Stadium befindet, in dem sie für reale Anwendungen benutzbar sind, und deren Entwicklung immer noch weitergeführt wird. Die bekanntesten dieser Frameworks sind ACT-R (Adaptive Control of Thought-Rational) und Soar (füher SOAR, als Akronym für State, Operator, Apply, Result).

Im Folgenden wird die kognitive Architektur ACT-R im Detail vorgestellt, wobei die darin verwendeten Konzepte und praktische Beispiele aufgezeigt werden. In Kapitel 3.3 wird CogTool behandelt, welches eine Anwendung ist, mit der sich GUI-Prototypen erstellen und anschließend automatisiert evaluieren lassen. Hierfür setzt CogTool auf ACT-R auf, wobei die kognitive Simulation der GUI-Prototypen von ACT-R übernommen wird. Die Architektur Soar wird in der Parallelfachstudie *Kognitions-Frameworks II* von unseren Kommilitonen untersucht.

3.1. Prinzipielle Möglichkeiten für die kognitive Simulation

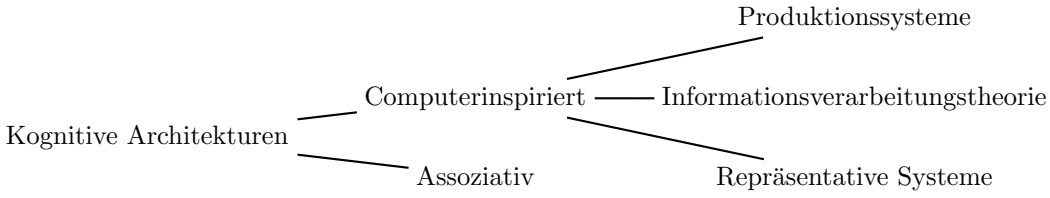


Abbildung 23: Hierarchie der verschiedenen Klassen von kognitiven Architekturen

Kognitive Architekturen lassen sich in zwei Arten unterteilen (siehe Abbildung 23): Die computerinspirierten Architekturen und die assoziativen Architekturen [45]. Moderne Architekturen vereinen Ansätze aus beiden Arten und werden daher als hybride Archi-

tekturen bezeichnet.

3.1.1. Computerinspirierte Kognitive Architekturen

Computerinspirierte kognitive Architekturen sind nach dem Vorbild eines von-Neumann-Rechners aufgebaut. Daher bestehen diese Architekturen aus einer zentralen Verarbeitungseinheit, Speicher und Eingabe- und Ausgabeeinheiten.

Die erste und bekannteste Klasse dieser Art von kognitiven Architekturen sind die sogenannten Produktionssysteme (Production Systems). In Produktionssystemen wird der menschliche Verstand als ein System modelliert, das aus einem Arbeitsspeicher und einer großen Menge von Produktions- und Prioritätsregeln, die die Reihenfolge der Produktionsregeln vorgeben, besteht. Eine Produktionsregel besteht dabei aus einer Bedingung und einer Aktion, die ausgeführt werden soll, wenn die Bedingung wahr wird. In jedem Verarbeitungsschritt werden alle Produktionsregeln bestimmt, bei denen die Bedingung wahr wird und die Aktion derjenigen ausgeführt, die nach den Prioritätsregeln an erster Stelle kommt. Eine solche Aktion modifiziert im Allgemeinen den Zustand des Arbeitsspeichers. Neuere Produktionssysteme sind innerhalb solcher Aktionen in der Lage, auch die Produktionsregeln zu verändern, neue hinzuzufügen oder bestehende zu löschen. Damit wird es möglich Lernprozesse in Produktionssystemen abzubilden. Beispiele für Produktionssysteme sind EPIC und SOAR.

Die zweite Klasse der computerinspirierten kognitiven Architekturen ist die Informationsverarbeitungstheorie. Sie geht davon aus, dass Informationen im menschlichen Gehirn die Verarbeitungsschritte Eingabe, Kodieren, Abspeichern, Auslesen und Ausgabe durchlaufen. Dabei interessieren insbesondere die Abläufe innerhalb der einzelnen Schritte. Eine kognitive Architektur, die sich ausschließlich auf die Informationsverarbeitungstheorie stützt ist uns nicht bekannt, aber man findet Teile davon zum Beispiel in CHREST wieder.

Die dritte Klasse der computerinspirierten kognitiven Architekturen sind die repräsentativen Systeme. Sie konzentriert sich auf die wahrheitsgemäße Abbildung der Struktur des menschlichen Wissens. In solchen Architekturen kommen komplexe Datenformate zum Einsatz, um das Wissen zu organisieren. Dabei wird angenommen, dass es zwei Arten von Speicher gibt. Den Arbeitsspeicher und einen Speicher für die strukturierten Daten. Beispiele solcher Datenformate sind Frames und Skripte. Im Allgemeinen können die Datenformate mittels Variablen von Objekten abstrahieren und Beziehungen zwischen Klassen repräsentieren. Weiterhin können sie in sich selbst eingebettet werden, also hierarchisch organisiert werden, und dadurch die Welt in verschiedenen Abstraktionsebenen repräsentieren. Ein solches repräsentatives System ist zum Beispiel PreACT.

3.1.2. Assoziative Kognitive Architekturen

Assoziative kognitive Architekturen basieren auf der Annahme, dass die Verarbeitung im Gehirn durch viele parallel arbeitende Einheiten geschieht, die Teil-Ganzes-Beziehungen herzustellen. Wurde zum Beispiel der erste Teil eines bestimmten Ablaufs erkannt und besteht für diesen Teil eine Assoziation mit dem gesamten Ablauf, dann lässt sich von dem Teil auf den gesamten Ablauf schließen. Die Architekturen an sich unterscheiden sich hauptsächlich darin, zwischen welchen Objekten sie die Assoziationen abbilden. Ein Beispiel für assoziative kognitive Architekturen sind neuronale Netze.

3.2. ACT-R

ACT-R (Adaptive Control of Thought-Rational) [2] ist eine kognitive Architektur, welche auf der Theorie Adaptive Control of Thought (ACT) aufbaut. Ziel dieses Projekts ist es, die Prozesse der menschlichen Kognition nachzubilden. ACT-R stellt hierfür ein Framework bereit, mit dem kognitive Modelle, bestehend aus Regeln und Fakten, erstellt und ausgeführt werden können. Die Regeln lassen sich testen, indem die Ergebnisse des Modells mit den Ergebnissen von Menschen verglichen werden, welche die selbe Aufgabe ausgeführt haben. Die Ergebnisse können hierbei aus folgenden Messungen bestehen (Beispiel-Setup siehe 3.2.3):

- Benötigte Zeit, um eine bestimmte Aufgabe durchzuführen
- Genauigkeit, die beim Durchführen der Aufgabe erreicht wurde
- Vergleich mit neurologischen Daten, welche aus FMRI gewonnen wurden

Der Vorteil von ACT-R ist hierbei, dass aus der Simulation quantitative Werte gewonnen werden können, welche sich direkt mit den Messungen aus Experimenten mit Menschen vergleichen lassen.

Anwendungsbereiche von ACT-R sind die Untersuchung von Problemlösungs- und Entscheidungsfindungsaufgaben, Lernen und Erinnern, Verstehen von natürlicher Sprache oder allgemeine kognitive Aufgaben. Konkrete Einsatzbereiche finden sich z.B. in der Mensch-Rechner-Interaktion, um verschiedene GUIs zu beurteilen (siehe Kapitel 3.3) oder im Bildungsbereich (Cognitive Tutoring System), wobei die Schwierigkeiten einer bestimmten Aufgabe abgeschätzt werden können, um so den Studenten gezielte Hilfestellungen geben zu können [2].

3.2.1. Aufbau von ACT-R

ACT-R ist in mehrere Module aufgeteilt, wobei jedes dieser Module auf eine bestimmte Informationsart, wie z.B. visuelle oder akustische Reize spezialisiert ist [18]. Abbildung 24 zeigt hierzu die grundlegende Architektur mit einigen Modulen des Systems. Die Theorie zu ACT-R trifft allerdings keine Aussage, wie viele Module existieren müssen. Einige sind schon implementiert und Teil des Frameworks. Für eine komplette Liste der implementierten Module mit ihren entsprechenden Puffern, siehe Abschnitt 3.2.2.

Im Folgenden werden die vier wichtigsten Module näher betrachtet:

- **Perceptual Motor System:** Besteht aus dem Visual und Manual Modul und ermöglicht die Interaktion mit der Umgebung.
- **Goal Module:** beinhaltet das aktuell zu erreichende Ziel.
- **Declarative Memory Module:** um Informationen aus dem Speicher abzurufen.
- **Procedural Memory Module:** Das Kernsystem von ACT-R. Hält Regeln für die Übergänge zwischen Zuständen.

Diesen Module lassen sich folgende Phasen aus Norman's seven stages zuordnen (siehe Abschnitt 2.1.6):

- Goal Module $\Leftrightarrow$ Phase 1 - Intention
- Declarative Memory Module $\Leftrightarrow$ Phase 2 - Action Specification
- Procedural Memory Module $\Leftrightarrow$ Phase 3 - Execution (Mental)
- Perceptual Motor System $\Leftrightarrow$ Phase 4 - Physical Activity und Phase 5 - Perception

Alle diese spezialisierten Module werden über das zentrale Produktionssystem integriert (siehe Abbildung 24 *Productions*), um ein bestimmtes Verhalten zu erzeugen. Das Kernsystem kann jedoch nicht auf beliebige Informationen aus den Modulen zugreifen, sondern zwischen jedem Modul und dem Kern befindet sich ein Puffer, welcher nur eine begrenzte Informationsmenge aufnehmen kann (auch Chunk genannt). Dieses Prinzip ist dem Menschen nachempfunden, da dieser sich zu einem bestimmten Zeitpunkt auch nicht über alle Informationen, welche in seinem Langzeitgedächtnis gespeichert sind bewusst ist. Analoges gilt für Objekte im visuellen Feld. Hier wird meist auch nur ein kleiner Teil der vorhandenen Objekte wahrgenommen, welche gerade als relevant erachtet werden. Der Informationsaustausch erfolgt somit nur über diese Puffer, wobei das Produktionssystem diese verwendet, um Muster in den Daten zu erkennen oder Änderungen an ein entsprechendes Modul weiterzuleiten. Dies kann z.B. auch das Absetzen

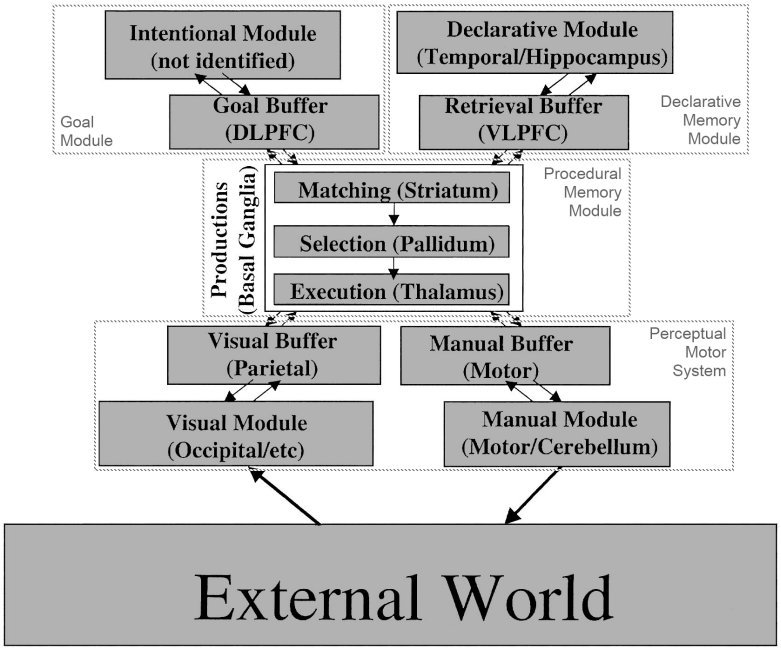


Abbildung 24: Grundlegende Architektur von ACT-R. ACT-R ist in mehrere Module aufgeteilt, welche alle über das Procedural Memory Modul integriert werden. [18]

eines Kommandos an den Manual Puffer sein, um somit das Durchführen einer Aktion im Manual Module anzustoßen.

Den einzelnen Komponenten des Systems können bestimmte Regionen des menschlichen Gehirns zugewiesen werden. In Abbildung 24 sind jeweils in Klammern die entsprechenden Areale angegeben, welche nach [18] am Besten dazu passen. So hält z.B. der Goal Puffer den aktuellen Zustand, um ein Problem zu lösen. Dieser wird mit dem *Dorsolateral Prefrontal Cortex* (DLPFC) assoziiert, welcher den Brodmann Arealen 9 und 46 entspricht (Abbildung 25). Die Funktionen des DLPFC umfassen unter anderem die Planung und Organisation von motorischen Fähigkeiten, Integration von sensorischen Informationen und er wird außerdem als Arbeitsgedächtnis genutzt. Der Manual Puffer ist zuständig für die Bewegungen der Hände und wird mit motorischen und somatosensorischen Regionen verknüpft. Diese entsprechen den Brodmann-Arealen 1-4. Für eine detailliertere Beschreibung der einzelnen Brodmann-Areale, siehe [35]. Das Kernsystem von ACT-R wird mit den Basalganglien des Gehirns in Verbindung gebracht. Diese sind Gehirnareale, welche unterhalb der Großhirnrinde liegen und wichtige Funktionen im Bereich Motorik, Kognition und limbische Regelungen (Verarbeitung von Emotionen) erfüllen [34].

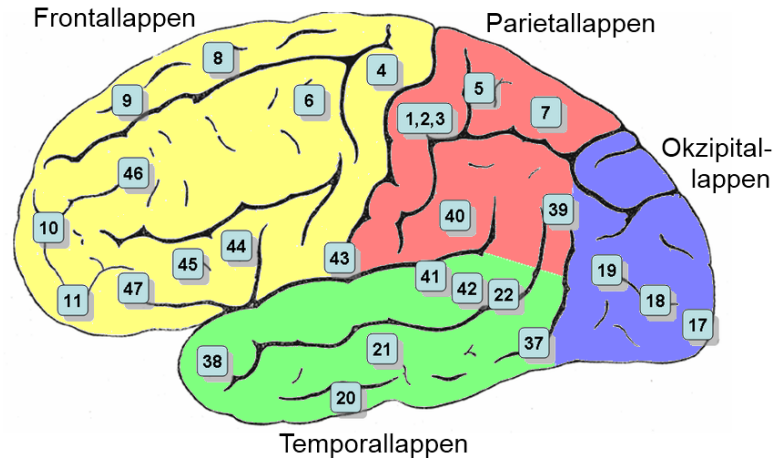


Abbildung 25: Regionen des menschlichen Gehirns: die Nummern entsprechen der Unterteilung der Gehirnareale nach Brodmann, [35]

Das Produktionssystem wird in drei Schritte unterteilt: das Matching dient zur Mustererkennung und sucht nach Produktionsregeln, welche zum aktuellen Zustand passend sind. Der Zustand von ACT-R ist hierbei der Inhalt aller Puffer zu einem bestimmten Zeitpunkt. Anschließend agiert der Selection-Schritt, um Konflikte zu erkennen und aufzulösen. Zum Schluss kontrolliert der Execution-Schritt die Ausführung der geplanten Aktionen. Eine wichtige Funktion der Produktionsregeln ist es, die Puffer zu aktualisieren, damit diese mit neuem Inhalt im nächsten Zyklus wiederverwendet werden können. ACT-R geht davon aus, dass die Durchführung eines solchen Zyklus 50 ms dauert. Diese Zahl hat sich im Laufe der Zeit entwickelt und wird auch in anderen kognitiven Architekturen wie Soar oder EPIC verwendet [18].

Die Architektur von ACT-R ist eine Mischung aus paralleler und serieller Verarbeitung. Zwischen den Modulen lässt sich ein hoher Grad an Parallelität erreichen, da z.B. das Visual Modul das Sichtfeld abarbeiten kann, während das Declarative Modul eine parallele Suche in verschiedenen Speichern durchführt. Dennoch gibt es zwei serialisierende Faktoren im System. Zum einen sind die Puffer begrenzt auf genau einen Chunk, sodass zu einem gegebenen Zeitpunkt immer nur genau eine Einheit verarbeitet werden kann. Zum anderen wird im Kernsystem immer nur genau eine Produktionsregel pro Zyklus ausgewählt und ausgeführt.

Im Folgenden werden die Komponenten aus Abbildung 24 im Detail vorgestellt:

Perceptual Motor System

Das Wahrnehmungs- und Handlungssystem in ACT-R ist der *Model Human Processor* Theorie aus [29] nachempfunden (siehe Abbildung 26) und wird auch erfolgreich in der kognitiven Architektur EPIC [20] eingesetzt. Dabei wird der menschliche Verstand als ein informationsverarbeitendes System aufgefasst. Ähnlich wie ein IT-Experte ein PC-System, mit Speicherkomponenten, Prozessor und den Verbindungen dazwischen, beschreiben würde, unterteilt das Modell den menschlichen Verstand in drei Komponenten:

1. Perceptual System (entspricht dem *Visual-Module* in Abbildung 24)

2. Cognitive System (entspricht dem *Produktionssystem* in Abbildung 24)

3. Motor System (entspricht dem *Manual-Module* in Abbildung 24)

Das Perceptual System (1) nimmt über Sensoren visuelle und akustische Signale wahr und speichert diese symbolisch in Puffern ab. Das Cognitive System (2) greift nun auf diese Informationen zu und verknüpft diese mit Inhalten aus dem Langzeitgedächtnis, um daraus Entscheidungen zu treffen. Aus diesen Entscheidungen werden Reaktionen generiert, welche an das Motor System (3) übergeben und dort ausgeführt werden. Hierbei wird abhängig von der Aufgabe unterschieden, ob diese Komponenten seriell oder parallel verschaltet sind. So wäre z.B. das Betätigen eines Lichtschalters infolge von zu wenig wahrgenommenem Licht eine serielle Ausführung, während Aufgaben wie Tippen, Lesen und Übersetzen gleichzeitig durchgeführt werden können.

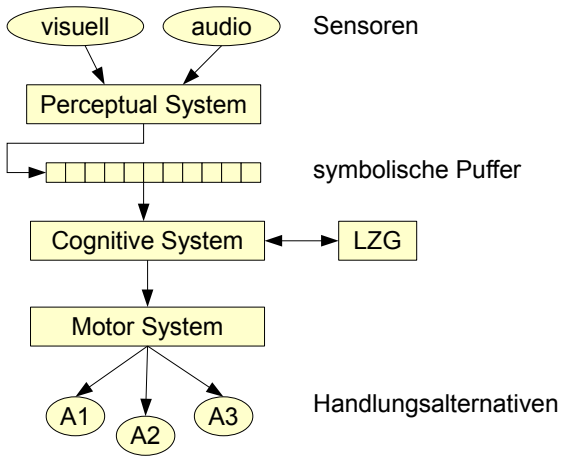


Abbildung 26: Schematische Darstellung der Model Human Processor Theorie. Dabei wird der menschliche Verstand in die drei Komponenten *Perceptual System*, *Cognitive System* und *Motor System* unterteilt.

Das Perceptual-Motor System stellt das Interface zur realen Welt dar. In ACT-R wird dies durch das Visual- und Manual-Modul abgedeckt. Das Visual-Modul ist nach der *Two Stream Hypothesis* [43] in zwei Subsysteme unterteilt, wobei jeweils ein eigener Puffer existiert. Die Two Stream Hypothesis unterteilt die visuelle Wahrnehmung in zwei Pfade: das dorosal System ist zuständig für die Erkennung der Position von Objekten im Raum (Wo-Dimension), während das ventral System Objekte eindeutig identifizieren kann (Was-Dimension).

Der Visual-Location-Puffer stellt die Wo-Dimension und der Visual-Object-Puffer die Was-Dimension dar. Wird z.B. auf einem Display ein grünes Objekt in einer Menge von blauen Objekten angezeigt, so ist die Zeit um die Position des grünen Objekts zu erkennen konstant, unabhängig von der Anzahl der blauen Objekte (Abbildung 27).

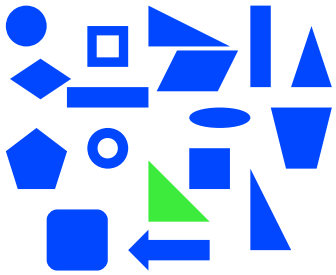


Abbildung 27: Die Position des grünen Objekts kann in konstanter Zeit ermittelt werden.

Das Produktionssystem kann über den Visual-Location-Puffer Einschränkungen angeben, welche das Suchfeld reduzieren (vgl. Abbildung 28). Diese Einschränkungen bestehen aus Attribut-Wert-Paaren, welche visuelle Eigenschaften der Objekte beschreiben, wie z.B. *color:green* oder *vertical:bottom*. ACT-R hat somit Wissen über die Position der Objekte, also *wo* diese sich befinden, und deren grundlegenden Eigenschaften. Um jedoch ein Objekt genau zu identifizieren, muss das Produktionssystem noch eine Anfrage an das *Was*-System stellen. Die Anfrage enthält dabei die Position des Objekts, sodass das System seine Aufmerksamkeit dorthin richten und das dort befindliche Objekt verarbeiten kann. Aus diesem Objekt wird nun ein Chunk in deklarativer Form erstellt. Für den Wechsel der Aufmerksamkeit werden zwei Modelle unterstützt: Beim Ersten wird von einer konstanten Zeit von 185 ms pro Objekt ausgegangen, was ähnlich zur Guided Search Theorie von Wolfe ist (siehe 2.1.5). Im zweiten Ansatz wird EMMA (Eye Movements and Movement of Attention), eine Theorie über Augenbewegungen implementiert. In EMMA ist die Zeit, die Aufmerksamkeit auf ein anderes Objekt zu lenken, davon abhängig, wie weit dieses Objekt vom aktuell fokussierten Punkt entfernt ist (siehe auch Abschnitt EMMA in 3.3) [18].

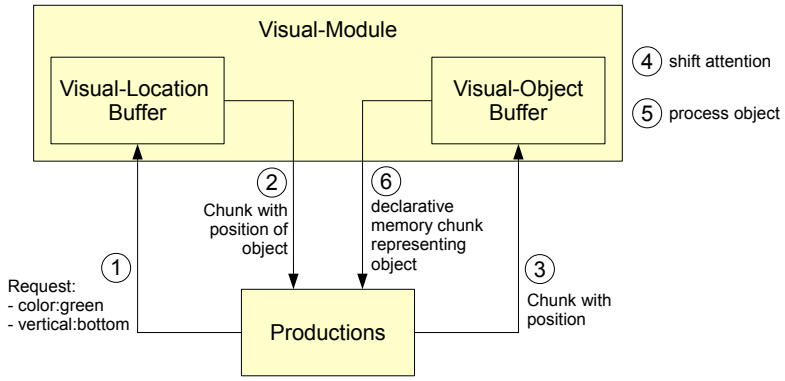


Abbildung 28: Ablauf der Erkennung eines Objekts im Perceptual System von ACT-R.

Goal Module

Das Goal Module hält die aktuellen Ziele, welche mit den wahrgenommenen Daten verfolgt werden. Werden z.B. die Zahlen 85 und 43 wahrgenommen, so könnten diese addiert, subtrahiert oder auf einem Telefon gewählt werden. Die menschliche Fähigkeit unterschiedlich auf diese Eingabe zu reagieren hängt also sehr stark von dem jeweiligen Ziel ab. Sollen die Zahlen addiert werden, so werden in der Regel eine Reihe von Schritten durchlaufen, wobei weitere Teilziele und Zwischenergebnisse anfallen, wie z.B. die Einerstellen zu addieren und anschließend die Zehner mit eventuellem Übertrag. Ein weiteres Beispiel sind die Türme von Hanoi (Abbildung 29). Hier werden sehr viele Teilziele erzeugt, wie z.B. „um Scheibe 4 auf Sockel C zu verschieben, muss zuerst Scheibe 3 auf Sockel B verschoben werden, und dafür Scheibe 2 auf C usw.“ um schließlich das Gesamtziel zu erreichen. Bei Versuchen wurde hierbei gezeigt, dass die Genauigkeit und Latenz eine solche Aufgabe durchzuführen sehr stark mit der Anzahl der Teilziele korreliert.

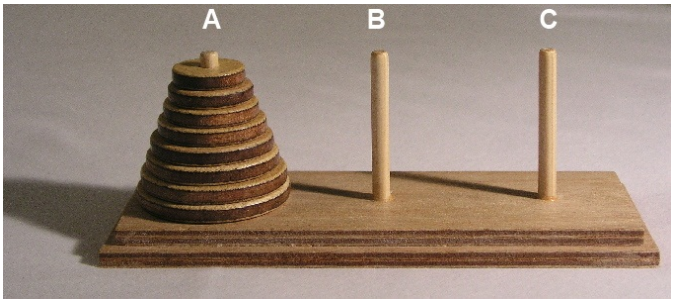


Abbildung 29: Türme von Hanoi: Der Turm muss vollständig auf den rechten Sockel verschoben werden, wobei nie eine kleinere Scheibe unter einer größeren liegen darf [42].

Declarative Memory Module

Das deklarative Gedächtnismodul bildet zusammen mit dem prozeduralen Gedächtnismodul den Kern von ACT-R und ist für die Speicherung und das Abrufen von symbolischem Wissen zuständig und gewährleistet, dass das gezeigte Verhalten über die Zeit konsistent bleibt. Der Zugriff auf das deklarative Gedächtnis benötigt Zeit, die von verschiedenen Faktoren beeinflusst wird.

Das deklarative Gedächtnis kann große Mengen von Informationen halten, wodurch es unmöglich wird diese alle gleichzeitig für die Lösung einer Aufgabe heranzuziehen, zumal Chunks sich gegenseitig ausschließen können oder für die Lösung irrelevant sein können. Pylyshyn hat dies als das sogenannte Roboter-Dilemma bezeichnet [24]. Daher muss eine Auswahl der wichtigsten Informationen getroffen werden. In ACT-R wird diese Auswahl als Chunk-Aktivierung A_i bezeichnet und ist abhängig von der allgemeinen Nützlichkeit bei der Lösung vergangener Probleme B_i und der Relevanz im aktuellen Kontext. Die Relevanz eines Elementes j im aktuellen Kontext ist von der Aufmerksamkeitsgewichtung W_j und der Assoziationsstärke S_{ij} mit dem Chunk abhängig.

$$A_i = B_i + \sum_{j=1}^{n_i} W_{ij} S_{ij}$$

Dabei gilt, je mehr Elemente an der Aktivierung eines Chunks beteiligt sind, desto geringer ist die Aufmerksamkeitsgewichtung eines einzelnen Elements und mit je mehr Chunks ein Element assoziiert ist, desto geringer ist die Assoziationsstärke des Elements mit einem einzelnen dieser Chunk.

$$W_{ij} = \frac{1}{n_i}$$
$$S_{ij} = S - \ln\left(\text{fan}_j\right)$$

wobei fan_j die Anzahl der Fakten ist, die mit dem Element j assoziiert sind, und S ein Parameter, der im Allgemeinen ungefähr 2 ist. Die allgemeine Nützlichkeit ist maßgebend vom Potenzgesetz des Vergessens geprägt, das besagt, dass ein Chunk für die Lösung eines Problems umso irrelevanter ist, je weiter die letzte Aktivierung in der Vergangenheit liegt. Im Gegensatz dazu wird ein Chunk allerdings umso relevanter, je öfter er aktiviert wurde.

$$B_i = \ln\left(\sum_{k=1}^{m_i} t_k^{-d}\right)$$

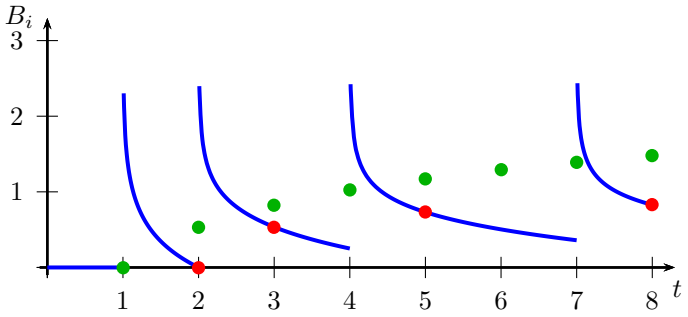


Abbildung 30: B_i in Abhängigkeit von der Zeit. Blau: Allgemeine Nützlichkeit B_i eines Chunks nach Aktivierungen zu den Zeitpunkten 1, 2, 4 und 7. Rot: Allgemeine Nützlichkeit B_i jeweils eine Zeiteinheit nach den Aktivierungen. Grün: Allgemeine Nützlichkeit B_i jeweils eine Zeiteinheit nach den Aktivierungen wenn der Chunk im Abstand von einer Zeiteinheit aktiviert wird.

wobei t_k die Zeit ist, die seit der k-ten Aktivierung vergangen ist und d ein Parameter für die Vergessensrate, für den sich 0.5 als guter Wert für eine große Menge an Problemen herausgestellt hat. Damit würde sich die Aktivierung wie folgt berechnen:

$$A_i = \ln\left(\sum_{k=1}^{m_i} t_k^{-d}\right) + \sum_{j=1}^{n_i} \frac{1}{n_i} \cdot (S - \ln(\text{fan}_j))$$

Die Aktivierung eines Chunks beeinflusst nun die Abrufwahrscheinlichkeit P_i und die Abrufverzögerung T_i für diesen Chunk. Dabei wird angenommen, dass Chunks erst ab einer bestimmten Aktivierung überhaupt abgerufen werden können. Ist dieser Schwellwert τ überschritten, so verhält sich die Abrufwahrscheinlichkeit in Abhängigkeit von der Aktivierung nach einer logistischen Funktion und die Abrufverzögerung nimmt invers

exponentiell zur Aktivierung ab.

$$P_i = \frac{1}{1 + e^{-\frac{A_i - \tau}{s}}}$$
$$T_i = Fe^{-A_i}$$

wobei s das Rauschen in den Aktivierungsleveln kontrolliert und üblicherweise um 0.4 liegt und F der Latenzfaktor ist, der zum Aktivierungsschwellwert τ in folgender Beziehung steht:

$$F \approx 0.35e^{\tau}$$

was bedeutet, dass die Abrufverzögerung bei einer Aktivierung in der Größe des Schwellwerts ($A_i = \tau$) ca. 0.35 Sekunden entspricht.

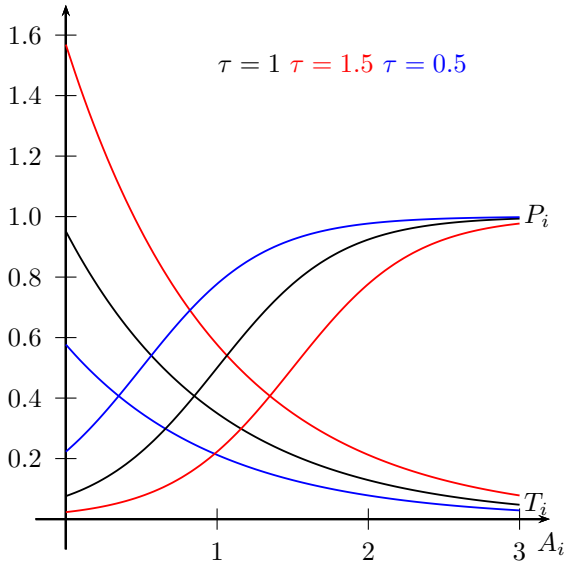


Abbildung 31: P_i und T_i in Abhängigkeit von A_i für verschiedene τ.

Procedural Memory Module

Das prozedurale Gedächtnismodul hält Regeln für die Übergänge zwischen Zuständen bereit. Technisch werden diese Zustände in ACT-R mittels Puffern zu den verschiedenen anderen Modulen realisiert und die Regeln geben an, wann sie angewendet werden können und wie die Puffer durch die Anwendung der Regel verändert werden. Diese Regeln heißen Produktionen. Ziel ist es dann, durch Anwendung von Produktionen von einem Startzustand in einen Endzustand zu gelangen.

Das grundlegende Problem dabei ist, dass zu einem beliebigen Zeitpunkt mehrere Produktionen angewendet werden können. Damit ACT-R hier eine Produktion auswäh-

len kann wird für Produktionen ein Nutzen U_i bestimmt, ähnlich der Aktivierung für Chunks.

$$U_i = P_i G - C_i$$

wobei P_i ein Schätzwert für die Wahrscheinlichkeit ist, dass das derzeitige Ziel erreicht wird, wenn die Produktion gewählt wird, G der Wert des derzeitigen Ziels ist und C_i ein Schätzwert für die Kosten (typischerweise die Zeit) ist, um das derzeitige Ziel zu erreichen. P_i und C_i unterliegen einem Lernprozess, der diese anhand von früheren Erfahrungen mit der Produktion anpasst.

Der Nutzen von Produktionen ist großen Schwankungen unterworfen, was dazu führt, dass eine Produktion in einigen Versuchen einen zufällig höheren Nutzen hat als eine andere und daher nicht immer die selbe Produktion gewählt wird.

Die Wahrscheinlichkeit, dass eine Produktion gewählt wird, kann dann mit folgender Formel bestimmt werden:

$$\Pi_i = \frac{e^{\frac{U_i}{t}}}{\sum_{j \in \mathcal{P}} e^{\frac{U_j}{t}}} \qquad \text{mit } \mathcal{P} = \{j \mid \text{Produktion } j \text{ kann angewendet werden}\}$$

für alle Produktionen j, die angewendet werden können, wobei der Parameter t die Schwankungen in den Nutzen reguliert und üblicherweise um 0.5 gewählt wird.

Die P_i als Schätzwerte für die Erfolgswahrscheinlichkeit werden einfach durch das Verhältnis von Anzahl der erfolgreichen Anwendung (m) zur Summe aus den Anzahlen der erfolgreichen und fehlgeschlagenen Anwendungen ($m + n$) bestimmt.

$$P_i = \frac{m}{m+n}$$

Ein Problem ergibt sich dabei allerdings zu Beginn der Berechnungen, wo m und n üblicherweise 0 sind und P_i daher nicht definiert ist. Ebenso würde P_i nach der ersten Anwendung zu einem der Extrema 1 (100% erfolgreich) oder 0 (gar nicht erfolgreich) ausschlagen. Um diese Probleme zu umgehen, wird ein vorheriger Wert θ festgelegt, der die Anzahl der erfolgreichen zu $\theta V + m$ und die Anzahl der fehlgeschlagenen Anwendungen zu $(1 - \theta) V + n$ verändert, wobei V die Stärke des Einflusses von θ auf spätere Schätzwerte angibt. Damit startet P_i bei θ und konvergiert mit der Zeit gegen $\frac{m}{m+n}$.

$$P_i = \frac{\theta V + m}{\theta V + m + (1 - \theta) V + n} = \frac{\theta V + m}{V + m + n}$$

Ist kein Vorwissen über die Wahrscheinlichkeiten vorhanden, so sollte $\theta = 0.5$ und $V = 2$ angesetzt werden. Für C_i wird analog verfahren.

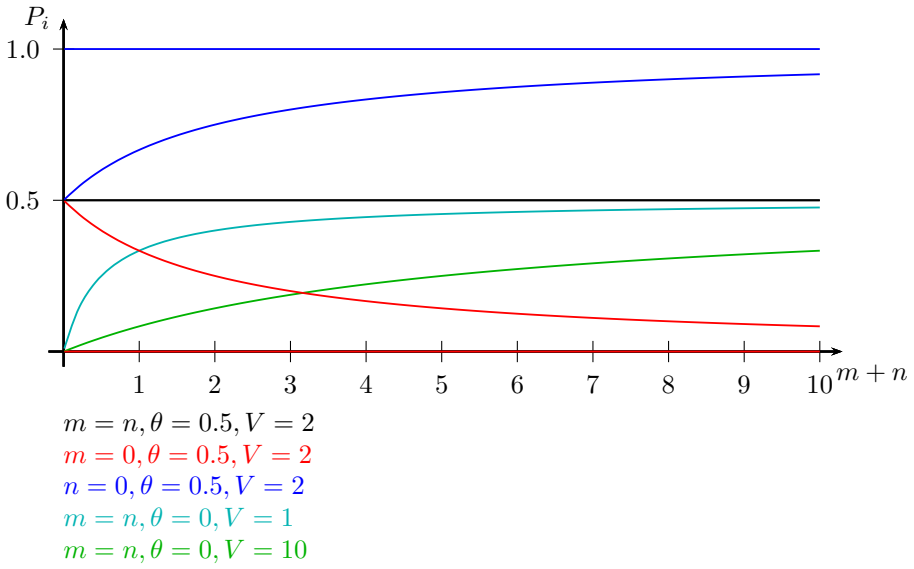


Abbildung 32: Vergleich der beiden Versionen für P_i. Diese Funktionen sind eigentlich diskret, weil m und n jeweils ganzzahlig sind, aber hier wurde für eine bessere Visualisierung der Effekte eine kontinuierliche Darstellung gewählt.

3.2.2. ACT-R Software

ACT-R liegt derzeit in der Version 6 vor und kann als Standalone-Anwendung oder als Archiv mit Quellcodedateien bezogen werden [2]. Die Standalone Varianten sind für Windows und Mac OS X verfügbar und können ohne Installation direkt verwendet werden. Um die Quellcodedateien verwenden zu können, muss Common Lisp auf dem System installiert sein, wodurch sich ACT-R auch auf Linux/Unix Systemen verwenden lässt. ACT-R 6 wird unter LGPL zur Verfügung gestellt. Im Folgenden wird von der Standalone-Variante für Windows ausgegangen, welche als Zip-Archiv verfügbar ist und lediglich entpackt werden muss, um ACT-R zu verwenden. Beschreibungen den Quellcode zu kompilieren sind unter [2] vorhanden.

In ACT-R 6 stehen folgende Puffer zur Verfügung, welche jeweils ein entsprechendes Modul in ACT-R integrieren (eine Beschreibung der wichtigsten Module ist im Abschnitt 3.2.1 zu finden):

- **visual-location:** Teil des Perceptual Systems für die Wo-Dimension (3.2.1).
- **visual:** Teil des Perceptual Systems für die Was-Dimension (3.2.1).
- **manual:** Nimmt Befehle zur Steuerung von Händen und Fingern entgegen.

- **retrieval:** Schnittstelle zum Declarative Memory Modul.
- **goal:** Hält das aktuelle Ziel.
- **production:** Dient nur zur Analyse des Zustands des Production-Moduls, erlaubt es nicht Chunks darin zu platzieren.
- **imaginal:** Hält visuelles Bild des Problemzustands. Ähnlich zum Goal-Puffer, jedoch vergeht hier eine bestimmte Zeit beim Manipulieren von Chunks.
- **imaginal-action:** Kann vom Benutzer verwendet werden, um das Imaginal-Modul zu erweitern. Standardmäßig keine Funktionalität darin implementiert.
- **temporal:** Gibt den Zählerstand aus dem Temporal-Modul an, wodurch Zeitabschnitte gemessen werden können.
- **vocal:** Schnittstelle zur rudimentär implementierten Sprachausgabe (Funktionsweise analog zum Manual-Modul).
- **aural-location:** Lokalisierung von Sound Ereignissen anhand bestimmter Bedingungen, wie z.B. Tonhöhe.
- **aural:** Lenkt Aufmerksamkeit auf eine bestimmte Sound-Quelle, verarbeitet den Sound und erzeugt daraus einen Chunk.

Die Oberfläche von ACT-R ist aus mehreren separaten Fenstern aufgebaut, die wichtigsten davon werden in Abbildung 33 gezeigt[1]:

- **Control Panel:** Bietet Zugang zu sämtlichen Funktionen von ACT-R.
- **Stepper:** Ermöglicht die Ausführung eines Modells Schritt-für-Schritt durchzugehen.
- **Listener:** Gibt Statusinformationen auf der Konsole aus und ermöglicht Befehle einzugeben.

Das Control Panel stellt den Einstiegspunkt zu allen weiteren Funktionen von ACT-R dar und ist in mehrere Bereiche unterteilt: als erstes wird dort das aktuell geladene Modell angezeigt, welches sich über die entsprechenden Buttons direkt darunter laden lässt. Durch Öffnen des Steppers lässt sich das geladene Modell Schritt-für-Schritt ausführen, wobei der Stepper alle möglichen Produktionsregeln aufzeigt, welche zum aktuellen Zustand passen und zusätzlich Variablenbelegungen darstellt. Im Bereich *Inspecting* des Control Panels lassen sich über entsprechende Buttons die Inhalte der Declarative und

[1] Eine detaillierte Beschreibung über alle Funktionen von ACT-R befindet sich auch in der mitgelieferten Dokumentation *EnvironmentManual.doc* des Zip-Archivs.

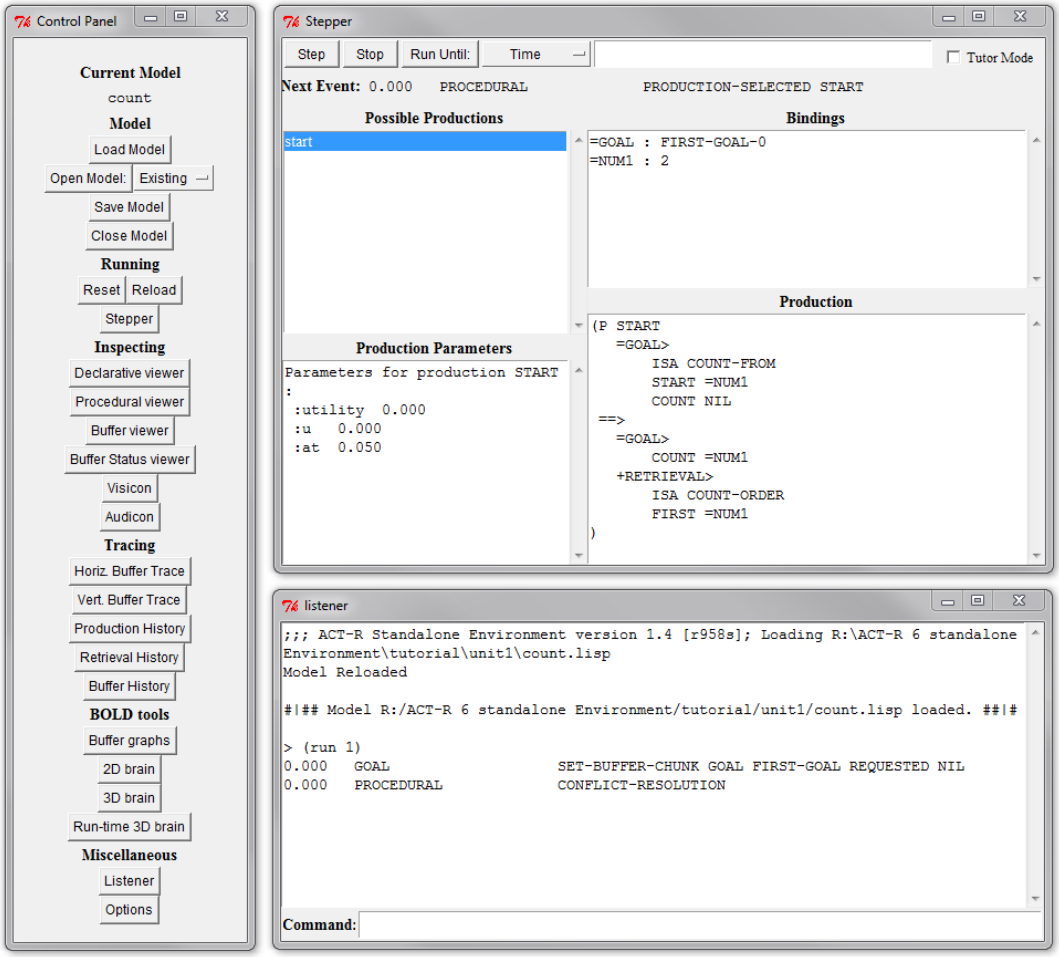


Abbildung 33: Hauptfenster von ACT-R 6: Control Panel (links), Stepper (rechts oben), Kommandozeilenein- und -ausgabe (rechts unten).

Procedural Memories, sowie aller vorhandenen Puffer anzeigen. Im Abschnitt *Tracing* lassen sich über Buffer Traces graphische Darstellungen zum Ablauf anzeigen und über diverse History-Buttons können Pufferbelegungen zu einzelnen Zeitabschnitten während der Ausführung abgerufen werden. Die BOLD Tools (Blood Oxygen Level Dependent) stellen eine graphische Repräsentation von BOLD Response Prediction Daten dar, wobei die einzelnen Puffer visuell auf Gehirnbereiche abgebildet werden und die Aktivität darin anzeigen (Abbildung 34).

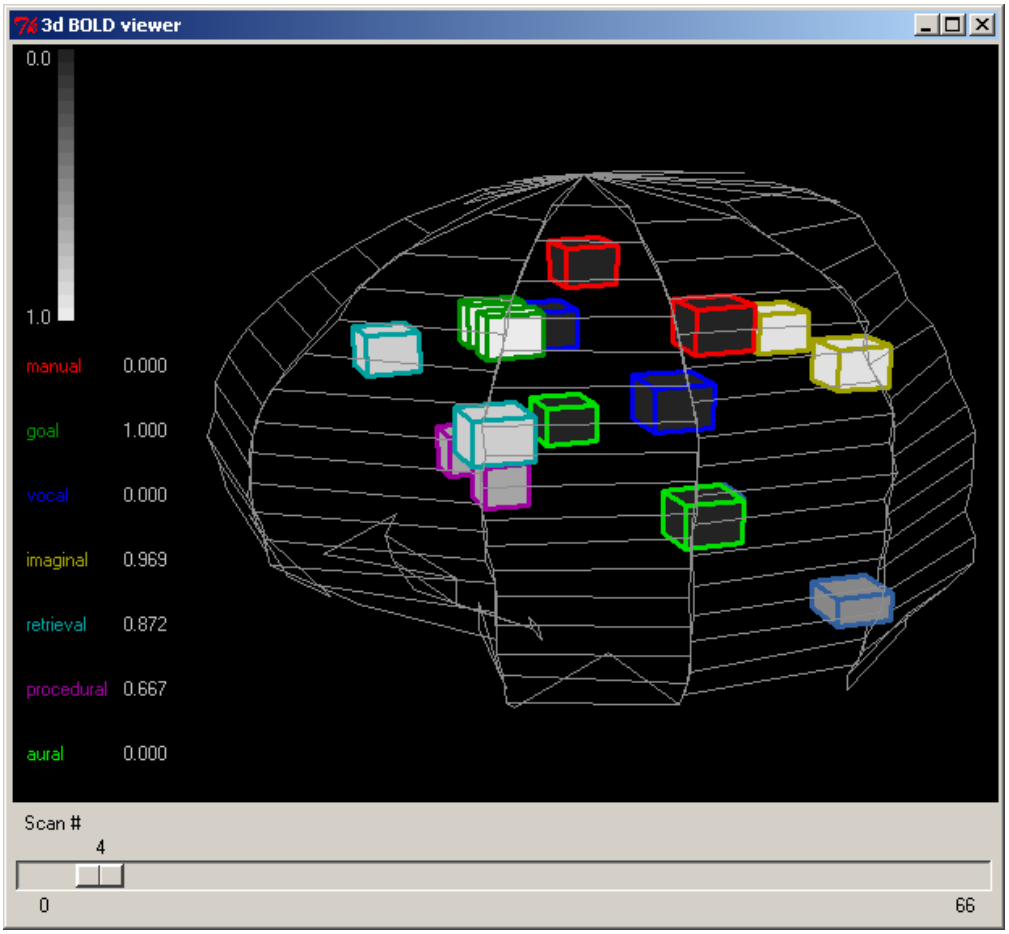


Abbildung 34: BOLD Viewer aus ACT-R: zeigt die Gehirnregionen der einzelnen Puffer an und deren Aktivität während der Ausführung eines Modells.

3.2.3. Beispiel-Modell

Im Folgenden wird ein Beispielmodell vorgestellt, welches in ACT-R geladen und ausgeführt werden kann. Es ist den Tutorials von ACT-R entnommen, welche bereits Teil des Standalone-Archivs sind (tutorial/unit1/count.lisp). Dieses Modell zählt von 2 bis 4 und gibt dabei eine Einführung in grundlegende Konstrukte wie Chunks und Produktionen. Hierfür werden die beiden Puffer Retrieval und Goal verwendet.

ACT-R unterteilt die Wissenrepräsentation in zwei Typen: Deklaratives Wissen stellt Faktenwissen dar, wie z.B. 2 ist der Nachfolger von 1, welches über Chunks repräsentiert wird. Prozedurales Wissen dagegen wirkt sich auf das Verhalten aus und wird nicht bewusst wahrgenommen, wie z.B. die Syntax von natürlichen Sprachen. Der zweite Typ

an Wissen wird in ACT-R durch Produktionsregeln dargestellt.

Chunks
Chunks stellen Elemente des deklarativen Wissens dar und bestehen aus einem Chunk-type und mehreren Slots. Ein Chunk-type kann als eine Kategorie angesehen werden und Slots stellen die Attribute einer Kategorie dar. Die Deklaration eines Chunk-Types sieht wie folgt aus:

```
(chunk-type count-order first second)
```

Da ACT-R Lisp für die Beschreibung des Modells verwendet, müssen sämtliche Befehle in Klammern gesetzt werden. Das erste Argument *chunk-type* gibt an, um welchen Befehl es sich handelt, gefolgt von einer Anzahl an Parametern. Der erste Parameter *count-order* gibt dem Chunk-Type einen Namen, alle danach folgenden Parameter stellen Slots dar, welche mit genau einem Wert belegt werden können.

Um nun Wissen in ACT-R zu hinterlegen, muss der Chunk-Type durch eine Menge von Chunks instanziiert und dem Declarative Memory über den Befehl *add-dm* hinzugefügt werden:

```
(add-dm
   (b ISA count-order first 1 second 2)
   (c ISA count-order first 2 second 3)
   ... )
```

Jeder Chunk erhält einen eindeutigen Namen - hier *b* und *c*. Auf den Namen folgt eine Liste aus Paaren von Slotname und initialer Wert. Der erste Slot *ISA* ist ein spezieller Slot, den jeder Chunk besitzt und welcher den Typ eines Chunks angibt. Dieser kann nicht mehr verändert werden. Alle weiteren können in beliebiger Reihenfolge spezifiziert und mit Anfangswerten versehen werden. Wird einem Slot kein Wert zugewiesen, so bleibt dieser leer und wird durch das Lisp Symbol *nil* dargestellt.

Produktionsregeln
Eine Produktionsregel kann als eine IF-THEN Anweisung gesehen werden. Hierbei stellt die Bedingung die aktuelle Belegung von bestimmten Puffern dar. Wenn die Bedingung erfüllt ist, dann kann die Produktion feuern und Änderungen an den Puffern durchführen. Die Bedingungen werden auch als left-hand side (LHS) und die Aktionen als right-hand side (RHS) bezeichnet. Eine Produktionsregel in ACT-R könnte wie folgt aussehen:

```
(p start
  =goal>                              ;; LHS
     ISA          count-from
     start        =num1
     count        nil
==>
  =goal>                              ;; RHS
```

```
    count          =num1
  +retrieval>
    ISA            count-order
    first          =num1
)
```

Jede Produktionsregel beginnt mit dem Befehl *p*, gefolgt von einem Namen der Produktionsregel - hier *start*. Weiterhin unterteilt sich der Befehl in einen LHS- und einen RHS-Bereich, wobei beide Teile durch ein `==>`-Zeichen voneinander getrennt werden. Wie bereits erwähnt steht der LHS-Bereich für die Bedingung, welche erfüllt sein muss, damit eine Produktion ausgewählt werden kann. Diese enthält eine Liste von Puffern mit jeweils einem vorgegebenen Muster, welches gegen die aktuellen Pufferbelegungen getestet wird. Stimmen die Werte aller spezifizieren Puffer mit den vorgegebenen Mustern überein, so ist die Produktionsregel passend und kann ausgewählt werden. Da es jedoch vorkommen kann, dass mehrere Produktionsregeln auf einen bestimmten Zustand passend sind, wird zunächst eine Konfliktauflösung durchgeführt, welche genau eine Regel auswählt (siehe *Procedural Memory Module* in 3.2.1).

In der oben dargestellten Produktionsregel werden nur Bedingungen an den Goal-Puffer gestellt, alle anderen können einen beliebigen Inhalt aufweisen, um diese Produktion zu aktivieren. Ein Chunk im Goal-Puffer muss hierbei vom Chunk-Type *count-from* sein, welches durch den ISA-Slot angegeben wird. Weiterhin muss im Start-Slot ein Wert stehen, welcher innerhalb der Produktionsregel über die Variable *num1* angesprochen werden kann. Der Slot Count muss leer sein.

Variablen werden durch das Präfix „=“ dargestellt, wobei diese sich für folgende zwei Möglichkeiten einsetzen lassen. Zum einen können diese in der Bedingung verwendet werden, um die Werte von verschiedenen Slots zu vergleichen, ohne alle möglichen Kombinationen wissen zu müssen. Außerdem lassen sie sich einsetzen, um Werte in Slots des Aktionenbereichs zu kopieren. Eine Variable ist jedoch nur innerhalb des Produktionsbefehls gültig.

Um einen Puffer zu spezifizieren, wird die selbe Syntax wie bei Variablen verwendet, wie z.B. `=goal`. Hierbei handelt es sich genauso um Variablen, welche den Chunk repräsentieren, der im entsprechenden Puffer steht. Diese können, wie jede andere Variable auch, verwendet werden, um bestimmte Werte zu testen oder um den Chunk innerhalb einer Aktion zu kopieren.

Der Aktionsbereich (RHS) einer Produktionsregel ist ähnlich aufgebaut wie die Bedingung. Es werden ebenfalls die Puffer angegeben sowie die Slots, auf welchen Änderungen durchzuführen sind. Hier stehen drei Typen von Aktionen zur Verfügung, welche sich pro Puffer durchführen lassen:

- **Buffer Modification:** Direkte Änderung des Chunks, welcher sich momentan im

Puffer befindet. Der Name des Puffers erhält hierbei das Präfix „=“.

- **Buffer Request:** Stellt eine Anfrage an das entsprechende Modul, einen Chunk mit entsprechenden Eigenschaften im Puffer bereitzustellen. Der Puffername erhält hier das Präfix „+“. Im obigen Beispiel wird eine Anfrage an das Declarative Module gestellt, einen Chunk zu finden, welcher vom Typ *count-order* ist und im Slot „First“ den Wert stehen hat, welcher zuvor im Goal-Puffer im Slot „Start“ stand.

- **Buffer Clearing:** Löscht den aktuellen Chunk aus dem angegebenen Puffer, indem der Puffername das Präfix „-“ erhält.

Das vollständige Modell count.lisp zählt von 1 bis 4 und sieht wie folgt aus (aus den ACT-R Tutorials entnommen):

```
(clear-all)

(define-model count

(sgp :esc t :lf .05 :trace-detail high)


(chunk-type count-order first second)
(chunk-type count-from start end count)

(add-dm
 (b ISA count-order first 1 second 2)
 (c ISA count-order first 2 second 3)
 (d ISA count-order first 3 second 4)
 (e ISA count-order first 4 second 5)
 (f ISA count-order first 5 second 6)
 (first-goal ISA count-from start 2 end 4)
 )

(p start
   =goal>
      ISA         count-from
      start       =num1
      count       nil
 ==>
   =goal>
      count       =num1
   +retrieval>
      ISA         count-order
      first       =num1
)

(P increment
   =goal>
      ISA         count-from
      count       =num1
    - end         =num1
   =retrieval>
      ISA         count-order
      first       =num1
      second      =num2
 ==>
   =goal>
```

```
      count          =num2
   +retrieval>
      ISA            count-order
      first          =num2
   !output!          (=num1)
)

(P stop
   =goal>
      ISA            count-from
      count          =num
      end            =num
 ==>
   -goal>
   !output!          (=num)

)

(goal-focus first-goal)
)
```

Listing 1: Vollständiges Modell count.lisp

Nachdem das Model in ACT-R geladen wurde, lässt es sich durch Eingabe des Befehls `(run 1)` im Fenster *listener* ausführen. Der Parameter `1` gibt an, wie lange das Modell maximal ausgeführt werden soll; hier also maximal eine Sekunde. Ist der Stepper geöffnet und der Befehl `run` wird eingegeben, so lässt sich das Modell Schritt-für-Schritt ausführen, wobei die Belegungen der Puffer sowie die möglichen Produktionsregeln zu einem Zeitpunkt beobachtet werden können.

Die Ausgabe des Modells sieht wie folgt aus:

```
> (run 1)
     0.000   GOAL                 SET-BUFFER-CHUNK GOAL FIRST-GOAL REQUESTED NIL
     0.000   PROCEDURAL           CONFLICT-RESOLUTION
     0.000   PROCEDURAL           PRODUCTION-SELECTED START
     0.000   PROCEDURAL           BUFFER-READ-ACTION GOAL
     0.050   PROCEDURAL           PRODUCTION-FIRED START
     0.050   PROCEDURAL           MOD-BUFFER-CHUNK GOAL
     0.050   PROCEDURAL           MODULE-REQUEST RETRIEVAL
     0.050   PROCEDURAL           CLEAR-BUFFER RETRIEVAL
     0.050   DECLARATIVE          START-RETRIEVAL
     0.050   PROCEDURAL           CONFLICT-RESOLUTION
     0.100   DECLARATIVE          RETRIEVED-CHUNK C
     0.100   DECLARATIVE          SET-BUFFER-CHUNK RETRIEVAL C
     0.100   PROCEDURAL           CONFLICT-RESOLUTION
     0.100   PROCEDURAL           PRODUCTION-SELECTED INCREMENT
     0.100   PROCEDURAL           BUFFER-READ-ACTION GOAL
     0.100   PROCEDURAL           BUFFER-READ-ACTION RETRIEVAL
     0.150   PROCEDURAL           PRODUCTION-FIRED INCREMENT
 2
     0.150   PROCEDURAL           MOD-BUFFER-CHUNK GOAL
     0.150   PROCEDURAL           MODULE-REQUEST RETRIEVAL
     0.150   PROCEDURAL           CLEAR-BUFFER RETRIEVAL
     0.150   DECLARATIVE          START-RETRIEVAL
     0.150   PROCEDURAL           CONFLICT-RESOLUTION
     0.200   DECLARATIVE          RETRIEVED-CHUNK D
     0.200   DECLARATIVE          SET-BUFFER-CHUNK RETRIEVAL D
```

```
     0.200   PROCEDURAL              CONFLICT-RESOLUTION
     0.200   PROCEDURAL              PRODUCTION-SELECTED INCREMENT
     0.200   PROCEDURAL              BUFFER-READ-ACTION GOAL
     0.200   PROCEDURAL              BUFFER-READ-ACTION RETRIEVAL
     0.250   PROCEDURAL              PRODUCTION-FIRED INCREMENT
3
     0.250   PROCEDURAL              MOD-BUFFER-CHUNK GOAL
     0.250   PROCEDURAL              MODULE-REQUEST RETRIEVAL
     0.250   PROCEDURAL              CLEAR-BUFFER RETRIEVAL
     0.250   DECLARATIVE             START-RETRIEVAL
     0.250   PROCEDURAL              CONFLICT-RESOLUTION
     0.250   PROCEDURAL              PRODUCTION-SELECTED STOP
     0.250   PROCEDURAL              BUFFER-READ-ACTION GOAL
     0.300   DECLARATIVE             RETRIEVED-CHUNK E
     0.300   DECLARATIVE             SET-BUFFER-CHUNK RETRIEVAL E
     0.300   PROCEDURAL              PRODUCTION-FIRED STOP
4
     0.300   PROCEDURAL              CLEAR-BUFFER GOAL
     0.300   PROCEDURAL              CONFLICT-RESOLUTION
     0.300   ------                  Stopped because no events left to process
```

Listing 2: ACT-R Ausgabe des Modells count.lisp

Die Ausgabe ist dabei in drei Spalten aufgeteilt. Zuerst wird die Zeit in Sekunden angezeigt, zu welcher ein entsprechendes Ereignis aufgetreten ist, gefolgt von dem Modul, welches das Ereignis auslöste und anschließend noch Details. Zwischen mehreren Ereignissen sind die Ergebnisse zu sehen, wie ACT-R von 2 bis 4 zählt.

In der ersten Ausgabezeile ist zu sehen, dass das initiale Ziel in den Goal-Puffer geschrieben wird (`SET-BUFFER-CHUNK GOAL FIRST-GOAL REQUESTED NIL`). Dies wird durch den Befehl `goal-focus` in der Modellbeschreibung ausgelöst, welche den entsprechenden Chunk in den Puffer schreibt. Anschließend wird eine Konfliktauflösung durchgeführt (`CONFLICT-RESOLUTION`), um eine passende Produktion auszuwählen. Hier ist aktuell nur die Regel *start* passend, welche, wie in der nächsten Zeile zu sehen ist, ausgewählt wurde (`PRODUCTION-SELECTED START`). Diese testet zunächst, ob ihre Bedingung erfüllt ist, indem sie den aktuellen Chunk im Goal-Puffer überprüft (`BUFFER-READ-ACTION GOAL`). Danach kann die Produktion gefeuert werden (`PRODUCTION-FIRED START`), wobei im Procedural System ein Parameter hinterlegt ist, welcher angibt, dass hierbei genau 50 ms vom Auswählen bis zum Feuern einer Produktion vergehen. Die nächsten beiden Zeilen zeigen die durchgeführten Aktionen der Produktion an. Diese definiert im RHS-Bereich den Count-Slot des Goal-Puffers zu aktualisieren (`MOD-BUFFER-CHUNK GOAL`) und stellt eine Anfrage an den Retrieval-Puffer einen entsprechenden Chunk bereitzustellen (`MODULE-REQUEST RETRIEVAL`). Nachdem das Declarative-Module die Anfrage bearbeitet hat, wird erneut eine Konfliktauflösung durchgeführt, wobei nun die *increment*-Produktionsregel ausgewählt wird (`PRODUCTION-SELECTED INCREMENT`), welche den eigentlichen Zählschritt durchführt, das Ziel anpasst und schließlich den aktuellen Wert schreibt. Anschließend beginnt dies erneut für die nächste Zahl, jedoch ohne die *start*-Produktion, da die *increment*-Regel bereits das Ziel für den nächsten Schritt festlegt.

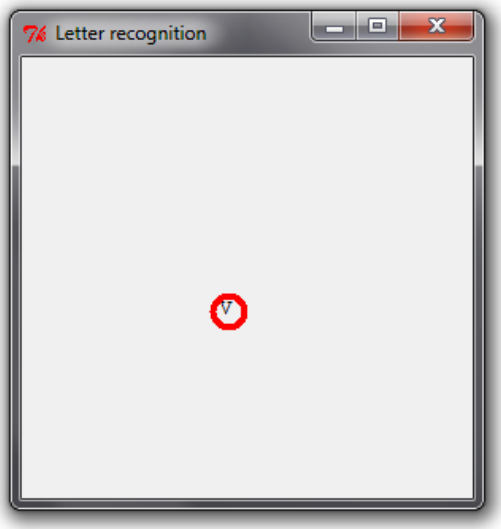


Abbildung 35: Erkennung eines Buchstabens von ACT-R. Der rote Kreis zeigt den Bereich an, auf welchen das Modell momentan seine Aufmerksamkeit richtet.

Weiterführendes Modell: Interaktion mit der Umgebung

Im Folgenden wird ein Modell vorgestellt, in welchem ein Buchstabe auf dem Bildschirm angezeigt wird, dieser vom Benutzer erkannt und anschließend über die Tastatur eingegeben wird (siehe Abbildung 35). Das Model ist zu finden unter `tutorial/unit2/demo2.lisp`. Das Experiment kann entweder von einem Menschen durchgeführt werden (indem dieser den Buchstaben selbst eingibt), oder ACT-R kann die Durchführung des Experiments simulieren (hierfür muss im Modell die Zeile `(setf *actr-enabled-p* nil)` nach `(setf *actr-enabled-p* t)` abgeändert werden). Das Experiment lässt sich mit dem Befehl

```
(do-experiment)
```

ausführen.

Die Erkennung des angezeigten Buchstabens erfolgt über das *Visual*-Modul (auch Vision-Modul genannt). Dieses enthält bereits Mechanismen, um Text zu parsen, oder einfache visuelle Objekte auf einem Fenster zu erkennen. Hierbei werden die Objekte durch ein oder mehrere Features repräsentiert, aus welchen anschließend, durch das Visual-Modul Chunks in deklarativer Form erstellt werden. Für die Erkennung von Buchstaben stehen mehrere Möglichkeiten zur Verfügung. Die Standardvariante ist die Darstellung von Buchstaben entsprechend einer LED-Anzeige. Buchstaben werden dabei in Linien zerlegt, sodass die Features eines Buchstabens den dafür benötigten LED-Segmenten ent-

sprechen. Im folgenden Beispiel ist auf der linken Seite eine LED-Anzeige mit Kodierung der Liniensegmente in Zahlen dargestellt. Während auf der rechten Seite die Features für den Buchstaben `E` ausgewählt wurden und dessen Kodierung als Chunk gezeigt wird [1]:

```
 - -          1 2          - -
|\|/|      3 4 5 6 7       |        (letter-e
 - -          8 9          - -         isa abstract-letter
|/|\|    10 11 12 13 14     |         value "E"
 - -         15 16          - -         line-pos (1 2 3 8 9 10 15 16))
```

Zusätzlich wird ein Interface bereitgestellt, um die Fähigkeiten des Moduls nach eigenen Bedürfnissen zu erweitern. Das Vision-Modul besitzt zwei Puffer: Zum einen den *Visual* Puffer, der einen Chunk hält, um das aktuelle Objekt zu repräsentieren und zum anderen den *Visual-Location*-Puffer, welcher über einen Chunk die Position des Objekts im visuellen Sichtfeld (das Fenster) speichert.

Um nun ein Objekt zu erkennen, wird zunächst eine Anfrage in den Visual-Location-Puffer geschrieben, welcher das Vision-Modul dazu veranlasst, die Position eines Objekts im visuellen Feld zu ermitteln. Falls das Modul ein entsprechendes Objekt findet, erzeugt es für dessen Position einen Chunk und schreibt ihn zurück in den Puffer. Folgender Chunk wird hierfür beim Ausführen des Modells erzeugt:

```
VISUAL-LOCATION0-0-1
   ISA  VISUAL-LOCATION
   SCREEN-X   130
   SCREEN-Y   160
   DISTANCE   15.0
   KIND   TEXT
   COLOR   BLACK
   VALUE   TEXT
   HEIGHT   10
   WIDTH   7
   SIZE   0.19999999
```

Anschließend wird eine Anfrage im Visual Puffer platziert, welche das Visual Modul anweist seine Aufmerksamkeit auf einen bestimmten Punkt zu richten. Hierfür werden die Bildschirmkoordinaten des Objekts, welche im Chunk des Visual-Location-Puffers gespeichert sind, verwendet. Das Visual Modul erzeugt nun einen Chunk aus dem Objekt, welches es an der entsprechenden Position vorfindet und schreibt diesen zurück in den Visual Puffer.

Nachdem nun der angezeigte Buchstabe im Fenster erkannt wurde, muss noch das Drücken der entsprechenden Taste auf der Tastatur simuliert werden. In ACT-R wird dies durch das *Motor*-Modul übernommen. Standardmäßig werden hier nur Handbewegungen unterstützt, um z.B. Tasten zu drücken oder eine Maus zu bewegen. Jedoch ist es auch hier wieder möglich eigene Erweiterungen einzubauen. Das Motor-Modul wird über den *Manual*-Puffer angesprochen, wobei hier keine Chunks durch das Modul erzeugt werden, sondern der Puffer nur verwendet wird, um Befehle an das Motor-Modul zu senden.

Um einen Tastendruck auszulösen, wird ein Chunk vom Typ `press-key` übermittelt, welcher im Slot `key` die zu drückende Taste enthält. Im Modell wird davon ausgegangen, dass sich die Hände in der Grundstellung über der Tastatur befinden, die Taste drücken und danach wieder zur Ausgangsstellung zurückkehren.

Übersicht über Beispielmodelle in ACT-R 6

Tabelle 1 gibt eine kurze Übersicht zu den Modellen, welche ACT-R 6 in der Standalone-Variante bereits mitliefert. Die entsprechenden Dateien sind im *tutorial*-Ordner des zip-Archivs zu finden, worunter sich auch jeweils detaillierte Beschreibungen zu den Modellen befinden.

Lerneinheit	**Beschreibung**
unit 1	Bietet eine Einführung in Chunks und Produktionsregeln. Enthält u.a. Modelle fürs Zählen und Addieren.
unit 2	Beschreibt wie die Interaktion mit der realen Welt stattfindet. Der Standardmechanismus erlaubt es mit dem Computer zu interagieren, wie z.B. visuelle Objekte wahrnehmen, Tastendrücke zu simulieren oder Mausbewegungen durchzuführen.
unit 3	Weiterführendes zur Funktionsweise der visuellen Aufmerksamkeit.
unit 4	Behandelt die Aktivierung von Chunks sowie Lernmechanismen.
unit 5	Abrufen von Informationen aus dem deklarativen Speicher.
unit 6	Auswählen von Produktionsregeln anhand errechneter Utility-Werte.
unit 7	Erlernen von neuen Produktionsregeln.

Tabelle 1: Überblick über Beispielmodelle aus ACT-R

3.2.4. jACT-R

jACT-R [15] ist eine Implementierung von ACT-R in Java. Es baut auf der Eclipse Rich Client Plattform auf, ist also eine Sammlung von Eclipse-Plugins. Zusätzlich zu der Implementierung des ACT-R bietet jACT-R auch eine Entwicklungsumgebung für Modelle an, ebenfalls auf Basis von Eclipse. Die Modelle werden in jACT-R im Gegensatz zum original ACT-R in XML geschrieben. Hierbei gibt es für alle Elemente aus der

LISP-Beschreibung eine XML-Entsprechung. jACT-R bietet auch einen Übersetzer, der zwischen beiden Sprachen konvertieren kann. Eine XML Übersetzung zum Beispielmodell count.lisp aus 3.2.3 ist im Anhang B zu finden.

jACT-R bietet vier verschiedene Schnittstellen zur Simulation.

Module: Module erweitern den Funktionsumfang des Modells in einer theoretisch untermauerten Weise. Das Perceptual-Motor-Modul ist ein solches Modul.

Erweiterung: Erweiterungen tragen zu einem Modell bei, sind aber nicht theoretisch untermauert. Erweiterungen werden für Integrations- oder Berechnungszwecke verwendet, zum Beispiel zur Verbesserung der Performance.

Instrumentierungen: Mit Instrumentierungen lässt sich der Status des Modells während der Simulation überwachen und abfragen.

Sensoren: Sensoren bieten die Möglichkeit jACT-R-Modelle in andere System einzubetten.

Der Autor Anthony Harrison möchte, dass jACT-R weitestgehend theoretisch kompatibel mit ACT-R bleibt, also die zugrunde liegenden Gleichungen und Schlüsselverhalten erhalten bleiben. Trotzdem unterscheidet sich jACT-R in einigen Implementierungsdetails von ACT-R [16], was aber nach Aussagen Harrisons keinen Einfluss auf den Ausgang der Simulation hat. Aufgrund dieser Unterschiede sieht auch das oben gezeigte Beispiel an einigen Stellen anders gegenüber dem original ACT-R Beispiel aus.

Die IDE bietet unter anderem die Möglichkeit, während der Laufzeit das Log und die Pufferzustände zu inspizieren.

3.2.5. Fazit

ACT-R hat gezeigt, dass es sich für ein großes Spektrum von Problemen einsetzen lässt. Zudem stellt das Perceptual-Motor System einen interessanten Ansatz in Bezug auf GUIs und Visualisierungen dar, worüber ACT-R bereits visuelle Informationen wahrnehmen, verarbeiten und anschließend über das Manual Modul manipulieren kann. Dem gegenüber stehen jedoch auch einige Hindernisse: So steht beim Erstellen eines neuen Modells ein recht hoher Einarbeitungsaufwand bevor, um sich mit allen Konzepten von ACT-R vertraut zu machen. Zudem sind in der derzeitigen Version meist nur einfache Mechanismen implementiert, wie z.B. das Erkennen von Buchstaben oder einfachen visuellen Features. ACT-R bietet hier zwar stets die Möglichkeit an, die Module mit eigenen Implementierungen zu erweitern, jedoch muss überlegt werden, ob hierfür der Aufwand für die Einarbeitung in die Sprache Lisp erbracht werden will. Eine mögliche Alternative zu Lisp stellt jACT-R dar, wobei hier modernere Sprachen wie Java und XML verwendet

werden. Allerdings befindet sich jACT-R noch in einer recht frühen Entwicklungsphase.

3.3. CogTool

CogTool ist eine Anwendung zum Erstellen von UI-Prototypen, mit der Besonderheit, das Abläufe innerhalb der Prototypen definiert werden können und die Anwendung automatisch quantitative Aussagen über den Ablauf generiert. Genauer: Die Anwendung simuliert die Durchführung vorgegebener Abläufe durch einen Benutzer und liefert Aussagen über die Dauer einzelner Schritte und in der Summe über den gesamten Ablauf ([3], Kapitel 1.1). Dabei wird das kognitive, perzeptuelle und motorische Verhalten des Benutzers mit Hilfe des ACT-R Frameworks simuliert.

Dieses Kapitel bietet einen groben Überblick zu CogTool und geht anschließend genauer auf die Zusammenarbeit zwischen CogTool und ACT-R ein.

3.3.1. Überblick

Ein konkreter Ablauf in CogTool basiert auf einem sogenannten „Design“. Ein Design besteht aus einer Menge von „Frames“, sowie Übergängen zwischen den Frames („Transitions“).

Ein Frame stellt eine Ansicht einer Anwendung dar. Dies kann eine, mittels den in CogTool eingebauten Standard-Widgets erstellte Oberfläche sein. Alternativ genügt CogTool aber auch ein Screenshot einer Oberfläche, auf den lediglich auf alle Objekte, die für den Interaktionsablauf relevant sind, Widgets gesetzt werden. Transitions repräsentieren Benutzeraktionen, die zum Wechsel der Ansicht führen. Die Standardübergänge sind hierbei Tastatureingaben und Mausaktionen (Klicks, sowie Drag & Drop). Eine Transition besteht somit aus drei Elementen: Dem Widget von dem sie ausgeht (z.B. Klick auf einen konkreten Button, Texteingabe in ein Feld), der Art der Interaktion (Maus, Tastatur) und dem Frame, zu dessen Wechsel die Interaktion hinführt. Dies kann auch der Frame sein, von dem die Transition ausgeht („Self-Transitions“). Somit ist es auch möglich, mehrere sequentielle Aktionen auf denselben Frame zu definieren.

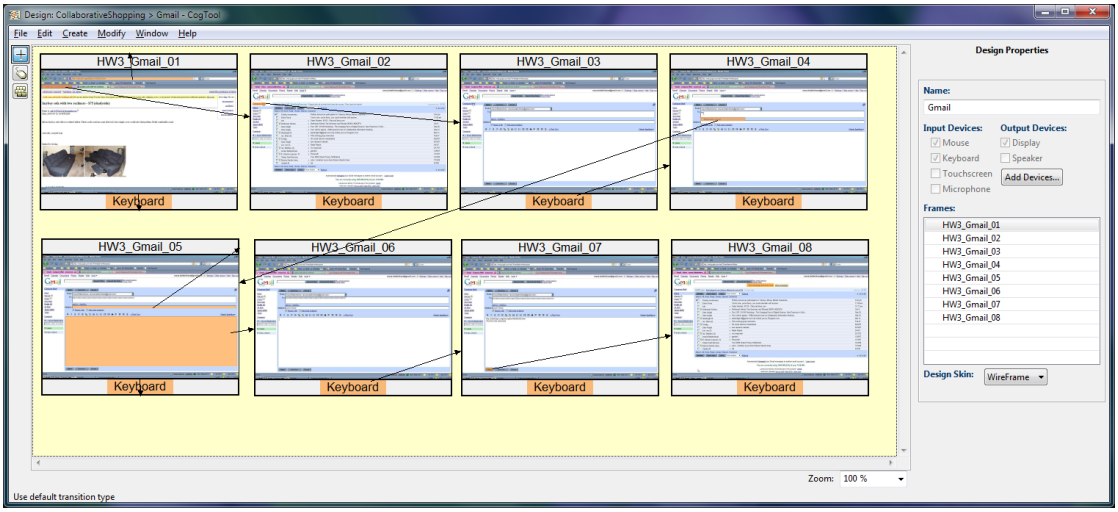


Abbildung 36: Ein CogTool Design mit mehreren Frames und Transitions

Ein Design stellt in CogTool die Basis eines konkreten Ablaufs dar. Vervollständigt wird der Ablauf durch Definition eines „Tasks“. Innerhalb eines Tasks wird ein Design durch eine „Demonstration“ komplettiert. Hierbei durchläuft der Benutzer das mit dem Task assoziierte Design und legt dabei z.B. die konkrete Reihenfolge von Aktionen innerhalb eines Frames fest. Diese sind fest an die Transitions des Designs gebunden. Der Benutzer hat die Möglichkeit, zusätzliche Sonderaktionen, wie das Betrachten von Widgets oder „Denken“, im Ablauf festzuhalten. Dies bedeutet in der Berechnung nichts anderes, als eine Pause mit einer vom Benutzer vordefinierten Dauer einzulegen.

Das Resultat einer Demonstration ist ein automatisch generiertes ACT-R Modell des Tasks, welches wiederum als Eingabe für das ACT-R Framework dient. Die quantitative Vorhersage, wie lange ein Benutzer für die Durchführung der vordefinierten Aufgabe benötigt, basiert nun ausschließlich auf dem generierten Script, das ACT-R verwendet. Im Folgenden wird vorgestellt, wie CogTool den definierten Task in das ACT-R Modell umsetzt.

3.3.2. Erzeugung des ACT-R Modell

Den Task, der auf einem Design ausgeführt wird, setzt CogTool in einem Verfahren ähnlich dem „Keystroke-Level-Model“ um, welches wiederum durch ein ACT-R Modell repräsentiert wird. Konkret ist ein ACT-R Modell eine Lisp-Datei mit Produktionsregeln, die durch das ACT-R Framework ausgeführt werden kann und als Ergebnis die geschätzte Dauer des im Modell beschriebenen Vorgangs liefert. Im Folgenden werden zunächst die Grundlagen zu KLM und EMMA erläutert und anschließend beschrieben, wie diese von

CogTool verwendet werden.

Keystroke-Level-Model

Das Keystroke-Level-Model (KLM) [10] wurde 1983 von Card, Moran und Newell entwickelt [8]. Es soll die Dauer der Interaktion zur Erledigung einer Aufgabe mit einem Informationssystem auf dem „Keystroke-Level“ beschreiben. Dies bedeutet, dass alle Interaktionen in etwa auf der Abstraktionsebene mit dem Drücken einer Taste vergleichbar sein sollen. Demnach wäre z.B. „Im System einloggen“ eine zu grobe Interaktion. Das KLM definiert folgende Interaktionen, auch „Operatoren“ genannt:

- K - Drücken einer Taste auf der Tastatur (0.28 Sekunden)
- T(n) - Eingabe von n Zeichen. T(n) = K * n
- P - Den Mauszeiger auf einen Punkt auf dem Bildschirm führen (1.10 Sekunden)
- B - Drücken der Maustaste oder Loslassen der Taste (0.10 Sekunden)
- BB - Ein Mausklick. Drücken und Loslassen der Maustaste (0.20 Sekunden)
- H - Hand wechselt zwischen Tastatur und Maus (0.40 Sekunden)
- M - Mentale Vorbereitung (1.20 Sekunden)
- W(t) - t Sekunden auf Berechnungen des Systems warten

Die Zeitangaben beruhen auf experimentellen Werten [8] und stellen die durchschnittliche Dauer der genannten Aktionen dar.

Zur manuellen Anwendung des Modells auf eine Aufgabe müsste diese nun so lange auf feinere Stufen aufgelöst und geordnet werden, bis sie durch eine Sequenz der Operatoren ausgedrückt werden kann. Eine besondere Herausforderung ist dabei die Platzierung des „Mentalen Operators“ bzw. „mentale Vorbereitung“ in der Operatorenauflistung. Dieser sollte immer dann in die Sequenz eingesetzt werden, wenn der Proband während der Durchführung der Aufgabe einen Moment lang nachdenken müsste, z.B. zum Orientieren innerhalb der Programmoberfläche.

EMMA

Alle Informationen dieses Abschnitts sind [9] entnommen.

EMMA ist ein Modell, das Augenbewegungen und visuelle Aufmerksamkeit beschreibt. Es ist in die kognitive Architektur ACT-R als Teil der perzeptuell-motorischen Komponenten integriert und wird von CogTool dafür verwendet, um die Bewegungen der Augen beim Erledigen eines Tasks zu beschreiben. Da das KLM für CogTool insbesondere um diese Aspekte erweitert werden muss, soll an dieser Stelle zunächst EMMA erläutert werden, bevor das für CogTool erweiterte KLM vorgestellt wird.

Visuelle Aufmerksamkeit besteht zunächst aus dem Befehl, die Aufmerksamkeit auf ein bestimmtes visuelles Objekt zu lenken. Zeitgleich beginnt der „Codierungsprozess“ des visuellen Objekts: Der Prozess, der die visuelle Repräsentation des Objekts erkennt und in eine abstrakte Form des deklarativen Gedächtnisses überführt (Chunk). Die Dauer des Codierungsprozesses hängt davon ab, wie oft das Objekt im Sichtfeld vorkommt. Auch die Entfernung des Fokuspunktes der Augen zum Objekt, in Bezug auf den Sehwinkel, wird einbezogen.

Augenbewegungen werden durch das Verschieben der visuellen Aufmerksamkeit initiiert. Dabei werden zwei Phasen unterschieden: Vorbereitung und Ausführung. Die Unterscheidung resultiert aus der Tatsache, dass Menschen trotz einer Verschiebung der Aufmerksamkeit die Augenbewegung zum neuen Aufmerksamkeitspunkt je nach Situation abbrechen, falls die Aufmerksamkeit in der Zwischenzeit nochmals umgelenkt wurde. Die Dauer der Augenbewegung errechnet sich aus einem fixen und einem dynamischen Anteil, in Abhängigkeit des zu überbrückenden Sehwinkels.

Die Berechnung der visuellen Aufmerksamkeit und der Augenbewegung wird zusätzlich mit einer „Unschärfe“ ausgestattet, indem die statischen Anteile der Dauer mit Hilfe eines Mittelwerts und einer Standard-Abweichung variiert werden. Zusätzlich wird der eigentliche Ziel-Fokuspunkt des Auges mittels einer Gauss-Verteilung um das eigentliche Ziel gestreut, um ein realitätsnäheres Verhalten des virtuellen Auges zu erzielen.

Der Kontrollfluss des EMMA-Modells lässt sich durch 4 Bereiche beschreiben:

- Kognition: Steuert die Verschiebung der Aufmerksamkeit
- Sicht: Verschiebt die Aufmerksamkeit und codiert visuelle Objekte
- Augenbewegung - Vorbereitung: Bereitet eine Augenbewegung vor
- Augenbewegung - Ausführung: Motorische Ausführung der Bewegung

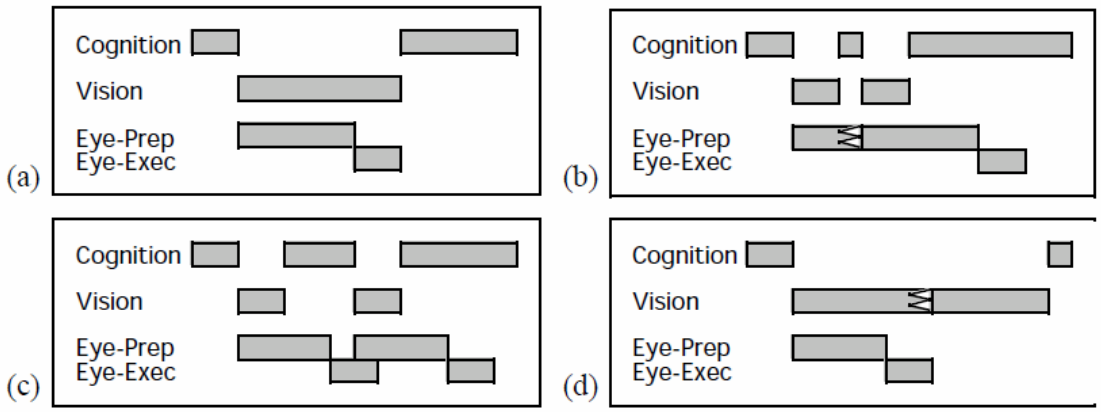


Abbildung 37: Visualisierung des (zum Teil) parallelen Ablaufs der 4 Bereiche ([9], S.4). Siehe Text im Kapitel EMMA.

Unterschiedliche Fälle führen nun zu unterschiedlichen Abläufen (Abbildung 37). Dauert die Codierung (Linie „Vision“) genauso lange wie die Augenbewegung, läuft beides exakt parallel (Fall a). Sendet die Kognition während der Codierung den „Befehl“, wieder die Aufmerksamkeit zu verschieben, wird die Augenbewegung unterbrochen und der neue Vorgang angestoßen (Fall b). Ist die Augenbewegung im gleichen Falle bereits eingeleitet, wird die Bewegung allerdings nicht abgebrochen, trotzdem wird ein neuer Vorbereitungs- und Codierungsprozess parallel eingeleitet (Fall c). Dauert der Codierungsprozess länger als die Augenbewegung, wird der Codierungsprozess nach der Bewegung erneut angestoßen (Fall d). Dieser wird nun aber schneller Ablaufen, da das Auge das zu codierende Objekt exakt fokussiert (es liegt also eine minimale Sehwinkeldifferenz zwischen dem Fokuspunkt der Augen und dem Fokus der Aufmerksamkeit).

KLM, CogTool und ACT-R, in Bezug aufeinander

CogTool verwendet ein Modell, das dem KLM ähnelt. Es definiert folgende Operationen ([3], Kapitel 5.5):

- Eye Movement Preparation - Vorbereitung der Augen, sich auf ein Objekt auf dem Bildschirm zu fixieren (siehe EMMA)
- Eye Movement Execution - Die Augen fixieren ein Objekt auf dem Bildschirm (siehe EMMA)
- Vision Encoding - Repräsentiert das „Codieren“ wahrgenommener visueller Objekte in eine mentale Repräsentation (siehe EMMA)

- System Wait - Entspricht W in KLM
- Cognition - Entspricht M in KLM
- Press Key - Entspricht K in KLM
- Move Cursor - Entspricht P in KLM
- Click Mouse - Entspricht B bzw. BB in KLM
- Hand to Home / Hand to Mouse - Entspricht H in KLM

CogTool erweitert das KLM um mehrere Aspekte. Zum einen wird das Spektrum der Operationen auf die Bewegung der Augen erweitert, z.B. das Fixieren eines Widgets vor der Interaktion mit diesem. Dies erfordert die Einführung von parallelen Operationen. Während im KLM alle Aktionen inhärent sequentiell sind, wäre es unrealistisch, dass Aktionen der Augen und der Hände nur nacheinander ablaufen. Zusätzlich verwendet CogTool im Gegensatz zum KLM keine pauschalen Zeitangaben für Operationen. Die Berechnung der Zeiten ist leider nur über den Quellcode nachvollziehbar.

Beim Berechnen eines Tasks werden die Frames und Transitions des Designs nun automatisch in eine Sequenz der oben genannten Operationen überführt, wobei nun Operationen auch parallel verlaufen können. Insbesondere die Platzierung des „Cognition“ Operators stellt dabei eine Herausforderung dar. Für diese wurde eine eigene Heuristik in CogTool entwickelt, siehe [4], Abschnitt „Rules for placing mental operators“. CogTool bietet die Möglichkeit, die Übersetzung des Tasks in Operatoren zu visualisieren (siehe Abbildung 38):

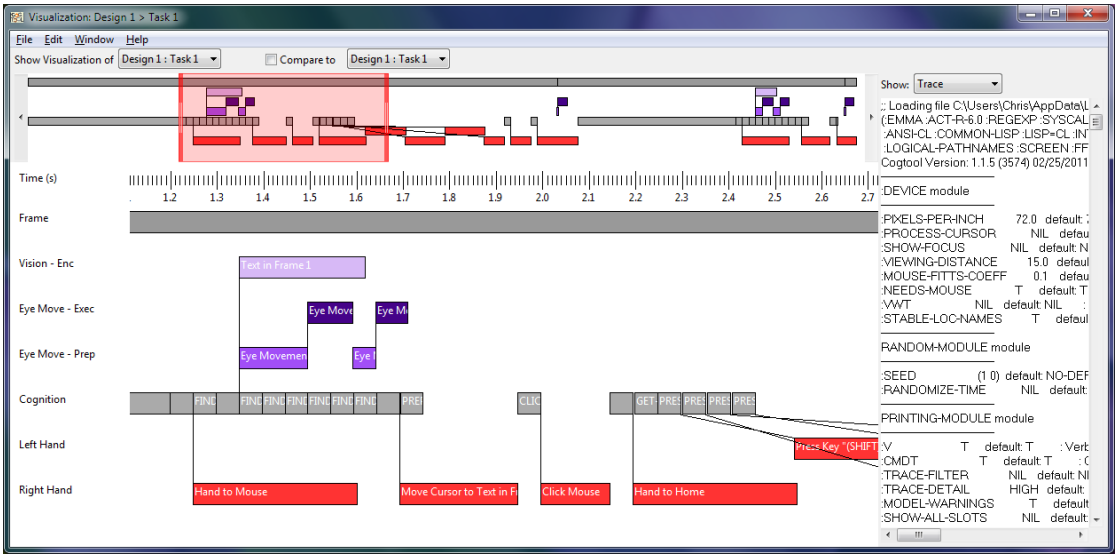


Abbildung 38: Visualisierung der Operatoren eines Tasks in CogTool. Einzelne Blöcke stehen für einzelne Operationen, entsprechend der obigen Kategorisierung. Die Linien zwischen einzelnen Blöcken drücken Anfang-Ende-Abhängigkeiten zwischen den Operationen aus. Dabei laufen mechanische Aktionen der Hände („Left Hand“, „Right Hand“), sowie die Aktionen der Augen („Eye Move - Exec“, „Eye Move - Prep“) jeweils innerhalb der eigenen Zeitlinien sequentiell ab. Gegeneinander und gegenüber „Cognition“ gibt es jedoch Überlappungen, also parallele Abläufe.

Während die Platzierung der Operatoren von CogTool vorgenommen wird, ist die Dauer der Berechnung der einzelnen Operationen nun die Aufgabe von ACT-R. Jede Operation wird in eine Reihe von Produktionsregeln für ACT-R übersetzt. Hier hebt sich das Modell von KLM ab, da für die Berechnung nun keine pauschalen Zeitangaben verwendet werden. Durch den Einsatz des Zwischenspiels aus motorischen, perzeptuellen und kognitiven Modulen in ACT-R „erbt“ CogTool die Validität des ACT-R Modells (vgl. [4], Abschnitt „Tools for easy predictive modeling“). Beispielsweise greift ACT-R für Bewegungen auf Fitts Law zurück und besitzt eine „Preparation Theory“ separat von den Ausführungen, was z.B. dazu führt, dass ein wiederholtes Drücken einer Taste schneller geschieht als ein erstmaliges Drücken, da die Aktion nicht neu vorbereitet werden muss. Auf die Lernfähigkeiten von ACT-R wird durch CogTool jedoch nicht zurückgegriffen, was sich jedoch im Einklang damit befindet, dass CogTool einen erfahrenen Benutzer simulieren soll, der durch die Benutzung der simulierten Anwendung keine Effizienzsteigerung erfährt.

3.3.3. Erweiterbarkeit

Zu CogTool existiert bereits eine Erweiterung, die die Einbindung von UI Prototypen unterstützt, die nicht in CogTool selbst erstellt worden sind [7]. Die Beschreibung der Erweiterung offenbart hierbei, das im Endeffekt immer eine Abbildung der externen GUI bzw. Widgets auf den vordefinierten Satz an CogTool-internen Widgets vorgenommen werden muss. Es ist somit nicht möglich, Widgets zu definieren die nicht bereits in CogTool vorhanden sind, außer sie ließen sich als Zusammensetzung der CogTool-Widgets beschreiben. Da die „atomaren“ Objekte von CogTool aber z.B. Elemente wie „Button“, „Label“ oder „Menü“ sind, lassen sich keine ausgefallenen Elemente beschreiben, erst recht keine Visualisierungen.

3.3.4. Fazit

Bezüglich unserer Fragestellung nach einer automatischen qualitativen Analyse von GUIs und Visualisierungen sind die Ergebnisse aus CogTool ernüchternd. Das einzige Maß dafür, wie gut eine GUI laut CogTool ist, ist die errechnete Durchlaufzeit einer vordefinierten Aufgabenstellung. Das Design der GUI ist ausschließlich als Hilfsmittel zur Definition der Aufgabe genutzt. Wie der Benutzer eine Aufgabe erledigt, und somit auch, wie er mit der GUI interagiert, muss bereits detailliert und komplett manuell in der Aufgabendefinition festgelegt werden. Wie der Benutzer die GUI wahrnimmt, wird bei der Analyse nicht tiefgreifend in Betracht gezogen. So ist z.B. das Ausfüllen eines Formulars aus Sicht von CogTool lediglich eine vordefinierte Abfolge von Maus und Tastatureingaben, wobei das einzige qualitative Maß der Widgets ihre Größe und Entfernung von der Mausposition ist. Es hat somit zwar Auswirkungen, wenn Widgets sehr weit voneinander entfernt sind. Ob die GUI aber mit zu vielen Widgets überfrachtet ist, ob die Widgets mit Ausnahme der Entfernung voneinander sinnvoll platziert sind, ob die GUI durch schlechte Farbwahl schwer zu bedienen ist und schlecht wahrgenommen wird, sind Aspekte, die CogTool nicht analysieren kann. Da die Aufgabendefinition in CogTool bereits durch den „Weg zum Ziel“ vorgegeben werden muss, wird auch außer acht gelassen, wie und ob der Benutzer diesen Weg in der GUI überhaupt intuitiv erkennen könnte.

Im Bezug auf die Analyse einer Visualisierung stellt CogTool folglich auch keine wirkliche Ausgangsbasis dar, da Visualisierungen eben nicht aus einer kleinen Anzahl stark standardisierter Widgets bestehen, die durch eine Größe und Position ausreichend beschrieben sind.

Was CogTool maßgeblich fehlt, in Bezug auf unsere Fragestellung, sind Wahrnehmungsaspekte bezüglich den Widgets selbst. Unter der Annahme, dass eine Visualisierung mit komplett individuell erstellten Darstellungselementen arbeitet, müsste ein entsprechendes Framework zur qualitativen Analyse eine Möglichkeit bieten, ein Modell der angewandten Darstellungselemente nachzubilden. Aufgrund der vielen unterschiedlichen

Arten von Visualisierungen müssten diese Elemente sehr generisch sein, was wiederum die Komplexität beim Abbilden der Widgets oder anderer Elemente drastisch erhöhen würde. Gleichzeitig stellt sich die Frage, wie viele solcher Basiselement nötig wären. Vergleicht man z.B. eine 3D-Volumen-Visualisierung mit einem 2D-Kuchendiagramm, scheint eine Beschreibung beider Visualisierungen mit gleichen „Mitteln“ fragwürdig. Verwirft man den Anspruch, möglichst alles beschreiben zu können, wäre es eventuell realistischer eine Hierarchie bzw. Gruppierung von ausgewählten Visualisierungsarten zu erstellen, welche unter dem Aspekt der Wahrnehmung möglichst nah miteinander verwandt sind und für die sich pro Gruppe eine Reihe von Basiselementen zur einheitlichen Beschreibung finden lässt.

3.4. Tools für Visualisierungen

Das bisher vorgestellte CogTool wurde zur Evaluierung von GUIs entwickelt und es hat sich gezeigt, dass die dort verwendeten Prinzipien sich nur sehr schwierig auf Visualisierungen übertragen lassen. Im Folgenden wird eine kognitive Architektur vorgestellt, die Visualisierungsanwendungen auswerten kann.

3.4.1. CAEVA

CAEVA [19] ist eine kognitive Architektur zur Auswertung von Visualisierungsanwendungen. CAEVA simuliert einen Benutzer einer Visualisierungsanwendung mittels eines kognitiven Modells. CAEVA besteht aus einem kognitiven Modell und einem Interoperabilitätsmodell zur Verbindung mit der Visualisierungsanwendung. Über diese Verbindung ist CAEVA sowohl in der Lage die Anwendung zu manipulieren, also Benutzeraktionen zu simulieren, als auch den Status der Anwendung zu erfassen.

Kognitives Modell

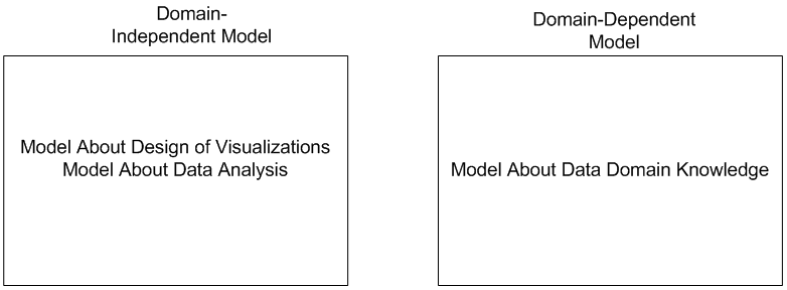


Abbildung 39: Aufteilung des Kognitiven Modells von CAEVA in einen domain-abhängigen und einen domain-unabhängigen Teil für die einfachere Wiederverwendbarkeit des deklarativen und prozeduralen Wissens.

CAEVA benutzt ACT-R zur Implementierung des kognitiven Modells. Dabei besteht das kognitive Modell aus einem domain-unabhängigen Teil und einem domain-abhängigen Teil.

Der domain-abhängige Teil enthält Regeln über den Aufbau von Visualisierungen. Dazu gehören auch Heuristiken, die bei der Erstellung von Visualisierungen zum Einsatz kommen, zum Beispiel für die Auswahl der Visualisierungsart. Der domain-unabhängige Teil beinhaltet Regeln darüber, wie Anwender Daten analysieren, zum Beispiel Art und Umfang der Aggregation der Daten. Für das domain-unabhängige Modell versucht CAEVA eine umfassende Implementierung zu bieten, die für alle Anwendungen gültig ist und im Idealfall nicht angepasst werden muss.

Der domain-abhängige Teil des kognitiven Modells enthält Regeln darüber, wie Menschen das Domainwissen einsetzen, um die Analyse durchzuführen, zum Beispiel welche der Datensätze relevant für die Beantwortung einer Frage sind. Dieser Teil muss an die jeweilige Anwendung angepasst werden.

Interoperabilitätsmodell

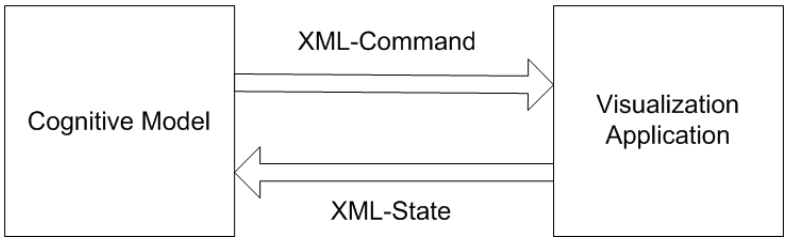


Abbildung 40: Das kognitive Modell und die Visualisierungsanwendung kommunizieren in CAEVA mittels XML-Nachrichten. Dafür muss die Visualisierungsanwendung einen Command-Server implementieren, der die XML-Nachrichten verarbeiten kann.

Die Kommunikation zwischen dem kognitiven Modell und der Visualisierungsanwendung geschieht durch XML-Nachrichten, die in der Richtung vom Modell zur Anwendung die verschiedenen Benutzerinteraktionen kodieren und in der Richtung von der Anwendung zum Modell den aktuellen Anwendungsstatus an das Modell übermitteln. Dafür muss die Visualisierungsanwendung einen Command-Server implementieren, der die XML-Nachrichten verarbeiten kann.

Von CAEVA wurde zwar ein Prototyp entwickelt, allerdings ist dieser nicht verfügbar.

4. Kognitive Simulation von GUIs und Visualisierungen

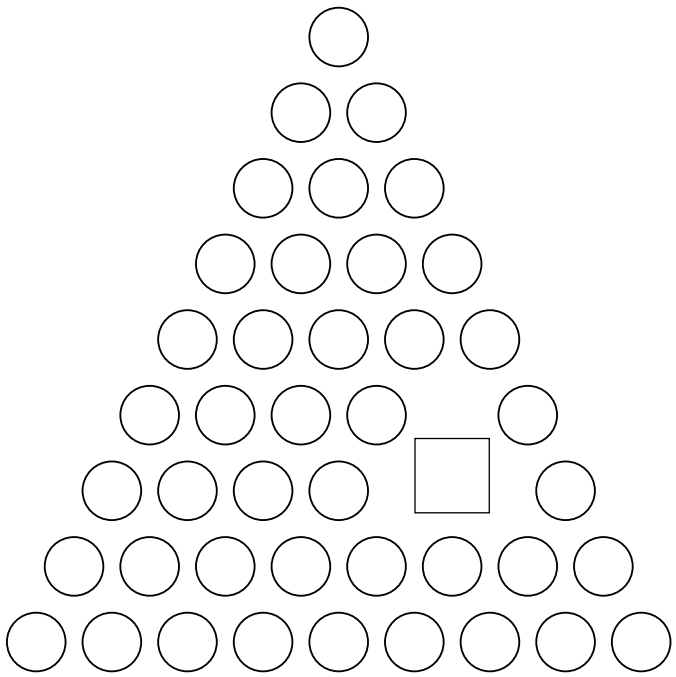


Abbildung 41: 42 Kreise und ein Quadrat. Für einen Betrachter dauert es länger die Kreise zu zählen, als zu bestimmen, ob ein Quadrat zu sehen ist.

Unter kognitiver Simulation von GUIs und Visualisierungen verstehen wir den Einsatz von kognitiven Architekturen zur Evaluation der Effizienz der grafischen Darstellung zur Erreichung eines bestimmten Ziels. Dabei ist zunächst erst einmal die Definition des Ziels wichtig. Dieses kann zum Beispiel sein, in den visualisierten Daten neue Zusammenhänge zu erkennen oder eine vorgegebene These zu bestätigen. Ein Simulationsdurchlauf mit einer kognitiven Architektur wie ACT-R kann nun bestimmen, wie lange ein Mensch benötigen würde, um das vorgegebene Ziel zu erreichen. Je schneller das Ziel erreicht wird, desto effizienter ist die Visualisierung zur Erreichung des Ziels. Wichtig ist dabei, dass diese Aussage nur bezüglich des festgelegten Ziels gültig ist. Dass diese Einschränkung nötig ist, lässt sich mit folgendem Beispiel nachvollziehen: Angenommen wir haben eine Visualisierung, die sehr viele (o.B.d.A. 42) Kreise und ein Quadrat zeigt und wir müssten zur Erreichung des Ziels die Anzahl der Kreise bestimmen, dann würde dies länger dauern, als wenn es ausreichen würde zur Erreichung des Ziels zu bestimmen, ob in der Visualisierung ein Quadrat vorhanden ist (Abbildung 41).

4.1. Was ist nötig, um GUIs / Visualisierungen kognitiv simulieren zu können ?

Zunächst benötigt die Simulation eine Repräsentation der GUI oder der Visualisierung. Diese kann als Bild oder als Beschreibung in maschinenlesbarer Form vorliegen. Dann benötigt die Simulation Wissen über Visualisierungen. Sie muss wissen, welches Element welche Bedeutung hat, bzw. haben könnte und wie diese miteinander zu der Visualisierung verknüpft werden. Außerdem muss das Wissen so umfangreich sein, dass es für die Simulation möglich ist festzustellen, ob und wie das aktuelle Ziel mit der Visualisierung erreicht werden kann. Diese Bedingungen lassen sich in folgender Architektur nach Raschke [25] zusammenfassen:

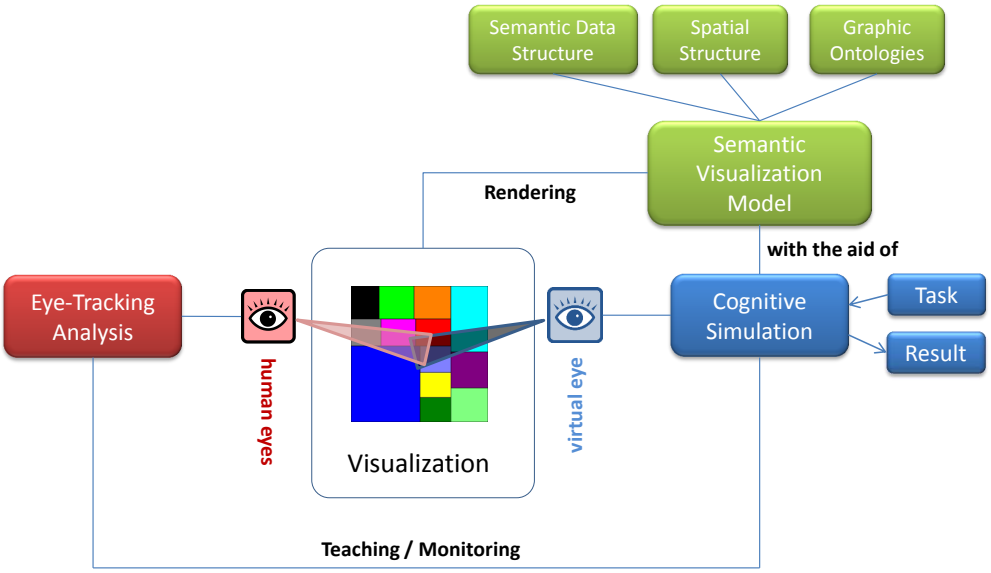


Abbildung 42: Architektur eines Systems zur kognitiven Simulation von Visualisierungen nach Raschke. Ein kognitives Modell dient zur Effizienzbewertung einer Visualisierung. Das Modell wird anhand von Eye-Tracking-Daten verifiziert.

Die Architektur abstrahiert von der eigentlichen Visualisierung, indem diese nur als Modell hinterlegt wird. Die Visualisierung wird also in einer Beschreibungssprache erstellt und aus vordefinierten Elementen zusammengesetzt. Solche Elemente sind zum Beispiel Achsen. Ein Renderer erzeugt aus diesem Modell dann ein Abbild für ein virtuelles Auge. Dieses Abbild muss nicht unbedingt ein gerastertes 2D-Bild sein. Weitere Möglichkeiten werden im Abschnitt 4.3.3 gezeigt. Das virtuelle Auge dient als Schnittstelle für das Kognitionsframework, um die Visualisierung betrachten zu können. In die kognitive Simulation geht auch das mit der Visualisierung verfolgte Ziel ein, damit eine Bewertung

der Effizienz bezüglich diesem möglich wird. Das kognitive Modell muss so viel Wissen enthalten, dass es mit der Visualisierung, die das virtuelle Auge sieht, umgehen kann. Es muss also Wissen über Visualisierungen im Allgemeinen, aber auch Wissen über die Domain der Daten enthalten.

Die Güte und Plausibilität der kognitiven Simulation wird anhand von Vergleichen der Daten aus den Simulationen mit Eye-Tracking-Daten ermittelt.

4.2. Visualisierungstechniken, die sich gut simulieren lassen

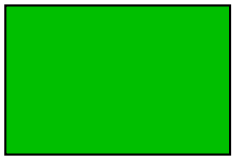


Abbildung 43: Ein kleines rotes Viereck verdeckt durch ein großes grünes Viereck. Das rote Viereck ist für einen menschlichen Betrachter nicht sichtbar. Ein virtuelles Auge, das die Überdeckung nicht beachtet, würde das rote Viereck sehen.

Cog-Tool hat gezeigt, dass sich Abläufe innerhalb von Benutzeroberflächen kognitiv simulieren lassen. Cog-Tool greift dabei auf die Tatsache zurück, dass die zur Erstellung benutzten Widgets klar abgegrenzte Entitäten, sowohl auf dem Bildschirm, als auch in der Beschreibung der Benutzeroberfläche sind. Es besteht also im Allgemeinen bei Benutzeroberflächen keine Diskrepanz zwischen beschriebenen und dargestellten Elementen. Dies ist bei Visualisierungen unter Umständen nicht gegeben, wenn sich zum Beispiel beschriebene Elemente überdecken können. Deshalb lassen sich Visualisierungstechniken, die mit klar abgegrenzten Entitäten arbeiten und diese überdeckungsfrei anordnen, besonders einfach kognitiv simulieren, weil hier die Beschreibung der Visualisierung und die Darstellung der Visualisierung keine Diskrepanz aufweisen. Visualisierungen müssen aber nicht unbedingt überdeckungsfrei sein, um kognitiv simuliert werden zu können. Es ist ausreichend, wenn sich Überdeckungen bestimmen lassen und diese auch in die Simulation einfließen. Betrachten wir dazu folgendes Beispiel: Wir haben ein großes grünes Viereck, das ein kleines rotes Viereck vollständig überdeckt. Beide Vierecke sind jeweils durch die Koordinaten ihrer Eckpunkte gegeben. Wird jetzt in der Simulation davon ausgegangen, dass der Benutzer beide Vierecke gleichermaßen sieht, dann führt diese zu falschen Ergebnissen. Zum Beispiel, dass der Benutzer erkennt, dass ein rotes Viereck vorhanden ist, obwohl die real betrachtete Darstellung diesen Schluss gar nicht zulässt (siehe Abbildung 43). Da wir aber die Koordinaten der Vierecke kennen und die Erkennung einer Überdeckung bei Vierecken durch diese Koordinaten möglich ist, können wir

das Ergebnis verbessern, indem wir diese Erkenntnis in die Simulation einfließen lassen.

4.3. Maschinelle Zugänglichkeit

Im Folgenden werden einzelne Aspekte der unter 4.1 vorgestellten Architektur auf Umsetzungsmöglichkeiten näher untersucht.

4.3.1. Vorbild Semantic Web?

Das Semantic Web ist ein semantisches Netz für das Internet. Ein semantisches Netz ist ein formales Modell von Begriffen und ihren Beziehungen [41]. Dieses Modell kann zur Wissensrepräsentation genutzt werden. Das semantische Netz wird auch als Graph interpretiert, in dem die Begriffe Knoten und die Relationen Kanten sind. Für das Semantic Web steht als Notation das Resource Description Framework (RDF) zur Verfügung. Das RDF definiert Relationen zwischen Ressourcen mittels Tripeln nach dem Subjekt-Prädikat-Objekt-Muster. Eine Ressource im Semantic Web ist alles, was durch eine URI eindeutig bezeichnet werden kann. Diese muss dabei nicht zwangsläufig im Internet erreichbar sein. Außerdem gibt es Ontologie-Beschreibungssprachen, mit denen auf den Ressourcen und Relationen des Semantic Web Ontologien definiert werden können.

Annotationen nach dem Vorbild des Semantic Web sind aus unserer Sicht für die Annotation von Visualisierungen geeignet. Man müsste nur eine geeignete Ontologie erstellen, da es unseres Wissens bisher noch keine Ontologie für Visualisierungen im Semantic Web gibt. Eine Entscheidung für das Semantic Web hätte darüber hinaus auch den Vorteil, dass man bestehende Ontologien mit der Ontologie für Visualisierungen verknüpfen könnte. Es geht also nicht darum, in welcher Sprache man das Wissen aufschreibt, sondern wie man das Wissen in das kognitive Modell bekommt. Man müsste also für jede Sprache, für die man sich entscheidet, eine Art Übersetzer bauen.

Eine direkte Übersetzung der Relationen in einem semantischen Netz in Chunks für ACT-R könnte zum Beispiel wie folgt aussehen:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description
      rdf:about="http://de.wikipedia.org/wiki/Resource_Description_Framework">
    <dc:title>Resource Description Framework</dc:title>
    <dc:publisher>Wikipedia - Die freie Enzyklopädie</dc:publisher>
  </rdf:Description>
</rdf:RDF>
```

Listing 3: Relationen in RDF, Beispiel aus [38]

```
(chunk-type semantic-web-relation subject predicate object)
(add-dm
  (title ISA semantic-web-relation
    subject "http://de.wikipedia.org/wiki/Resource_Description_Framework"
    predicate "http://purl.org/dc/elements/1.1/title"
    object "Resource Description Framework")
  (publisher ISA semantic-web-relation
    subject "http://de.wikipedia.org/wiki/Resource_Description_Framework"
    predicate "http://purl.org/dc/elements/1.1/publisher"
    object "Wikipedia - Die freie Enzyklopädie"))
```

Listing 4: Abbildung der Relationen in Chunks

Das hier gezeigte Beispiel ist stark vereinfacht und müsste für den Einsatz sicher noch erweitert werden, zum Beispiel muss die Simulation noch Regeln enthalten, dass Subjekte mit der gleichen URI auch das gleiche Objekt referenzieren. Ebenso fehlen die Aussagen über die zugrundeliegene Ontologie, die sich auf ähnliche Weise wie Relationen abbilden lassen, da sie ebenfalls in einem XML-Dialekt geschrieben sind. Unklar ist außerdem noch, ob sich eine solche direkte Abbildung überhaupt für eine kognitive Simulation eignet.

Der erfolgreiche Einsatz von ACT-R in CAEVA hat gezeigt, dass es möglich ist, das Wissen über Visualisierungen im deklarativen Gedächtnis abzulegen und für die Simulation zugänglich zu machen, allerdings beschreibt [19] leider nicht wie dies erreicht wurde und welchen Umfang dieses Wissen hatte.

Die semantische Repräsentation einer Visualisierung dient in bisherigen Anwendungen dazu, die Darstellung der Visualisierung von ihrer Beschreibung zu trennen ([27], [26]), ähnlich wie bei HTML und CSS. Dadurch wird es möglich die Darstellung der Visualisierung anzupassen, ohne die Visualisierung verändern zu müssen.

4.3.2. Zugänglichkeit von GUIs

GUIs, die auf Widget-Toolkits basieren, können in der Regel ohne Anpassungen von Maschinen verarbeitet werden. Diese GUIs können als externe beschreibende Dateien, z.B. XAML für WPF, oder direkt als Quellcode in einer Programmiersprache, z.B. Java für SWT, vorliegen. In beiden Fällen liegt es in der Natur der Sache, dass die GUIs aus diesen Beschreibungen erzeugt werden. Das bedeutet also, dass wir durch die Benutzung eines Widget-Toolkits den Bereichen in der Benutzeroberfläche schon eine semantische Bedeutung in Form eines Widgets zuweisen, die wir auch in der Simulation nutzen können. In der Regel werden für die jeweiligen Widgets in der Beschreibung auch Interaktionsmöglichkeiten und Reaktionen definiert, die aber nur begrenzt automatisch ausgewertet werden können, da hier beliebige Seiteneffekte implementiert werden können. Das heißt, an dieser Stelle muss für die vollautomatische Auswertung noch ein wenig Arbeit hineingesteckt werden. Dies kann mit einem externen Werkzeug, wie zum Beispiel Cog-Tool

oder als Annotation direkt in der Beschreibung der GUI geschehen.

4.3.3. Zugänglichkeit von Visualisierungen

Möchte man den Ansatz für die maschinelle Zugänglichkeit von GUIs direkt auf Visualisierungen übertragen, dann muss man sich im klaren darüber sein, dass eine Annotation von Visualisierungen immer auch eine Implikation über den Benutzer enthält, dass er die annotierten Elemente auch so erkennt, wie sie die Annotation beschreibt. Wir können zum Beispiel eine Kugel mit „Erde“ annotieren und der Simulation vorgeben, dass der Benutzer jetzt die Erde auf dem Bildschirm sieht, bekommen aber Probleme, wenn ein realer Benutzer bei dem selben Bild nur eine Kugel erkennen würde, da hier in der Annotation zu viele Informationen impliziert werden. Dieses Problem tritt bei GUIs nicht auf, da es dort allgemein anerkannte Standard-Widgets gibt, die auf jedem Betriebssystem und in jedem Widget-Toolkit ähnlich aussehen und die gleichen Funktionalität haben, solange man sich auf die Standard-Widgets beschränkt und deren Aussehen nicht zu stark verändert. Bei Visualisierungen kann zum Beispiel eine Linie eine Koordinatenachse sein, oder aber auch die Verbindung zwischen zwei Punkten. Sie sehen auf dem Bildschirm für einen realen Nutzer gleich aus und es kann daher nicht automatisch davon ausgegangen werden, dass der Benutzer diese korrekt unterscheidet.

Insgesamt haben wir für die maschinelle Zugänglichkeit von Visualisierungen folgende Möglichkeiten:

Computer Vision: Die Simulation sieht genau das gleiche Bild wie der Benutzer. Die ganze Perzeption muss in der Simulation implementiert sein. Diese Variante wird im Allgemeinen nicht nötig sein, da für die Perzeption nachgewiesene Gesetzmäßigkeiten existieren (siehe Kapitel 2)

Annotation geometrischer Objekte: Das Bild wird für die Simulation aufbereitet, indem aus den annotierten geometrischen Objekten die Features für die Weiterverarbeitung extrahiert werden.

Annotation von Fachobjekten: Die Annotationen geben ähnlich den Widgets für GUIs typische Fachobjekte vor. Diese können dann in der Simulation als erkannt vorausgesetzt werden. Als Fachobjekte verstehen wir Teile der Visualisierung, die benannt werden können und denen eine Bedeutung zugewiesen werden kann, zum Beispiel Achsen oder Datenpunkte.

Der Renderer aus der Architektur von Raschke (Abbildung 42) könnte nun diese maschinell zugängliche Visualisierung mit Koordinaten verknüpfen und dem virtuellen Auge zur Verfügung stellen. Im Falle der Computer Vision erzeugt der Renderer direkt ein 2D-Bild, anstatt die Annotation um Koordinaten zu erweitern. Wenn das virtuelle Auge nicht auf Computer Vision basiert, dann muss der Renderer auch dafür sorgen, dass für

das virtuelle Auge erkennbar ist, welche Elemente verdeckt sind.

4.4. Einschränkungen von symbolischen Kognitionsframeworks

ACT-R ist darauf ausgelegt möglichst umfassend die Entstehung von Intelligenz im menschlichen Gehirn zu simulieren. Die Simulation in ACT-R nutzt als Grundlage den symbolischen Ansatz der künstlichen Intelligenz und erweitert ihn um einige konnektionistische Techniken. Allerdings gibt es mit dem symbolischen Ansatz einige Probleme, die sich zunächst im Gebiet der künstlichen Intelligenz für Roboter gezeigt haben. Das erste Problem ist das sogenannte Symbol Grounding Problem. Dabei steht die Überlegung im Raum, dass der symbolische Ansatz zwar Symbole (Chunks in ACT-R) und Produktionen definiert, mit denen die Symbole verknüpft werden können, aber diese Umformung ohne Wissen über die eigentliche Bedeutung der Symbole und Regeln vorgenommen werden können. Vereinfacht ausgedrückt würde das bedeuten, dass wir eine Aufgabe durch Anwendung von definierten Regeln lösen könnten, ohne genau zu verstehen, was wir eigentlich gemacht haben. Computerprogramme und kognitive Simulationen nach dem symbolischen Ansatz gehen ähnlich vor. Sie wenden auf Daten immer wieder vordefinierte Regeln an und lösen damit Probleme. Dabei müssen sie die Regeln und die Bedeutung der Daten nicht verstehen, sondern die Regeln nur anwenden und die Struktur der Daten analysieren können.

Das zweite Problem ist, dass symbolische kognitive Simulationen nicht mehr richtig funktionieren, wenn etwas unvorhergesehenes passiert und dafür keine Regel vorhanden ist. Das menschliche Gehirn kann allerdings solche Varianz relativ gut berücksichtigen. Das gleiche gilt für Ausfälle eines Teilsystems. Beispielsweise kann ein Computer ein Programm nicht ausführen, wenn eine Bibliothek nicht geladen werden konnte, selbst wenn sie für die gerade gewünschte Funktionalität irrelevant wäre. Ein Mensch dagegen kann sich sehr wohl noch bewegen, auch wenn die Region im Gehirn, die für die Sprache zuständig ist, defekt ist.

Als drittes Problem beschreibt das sogenannte Roboter-Dilemma [24], dass die naive Umsetzung einer kognitiven Simulation nicht echtzeitfähig ist. Eine solche naive Umsetzung würde zum Beispiel zunächst alle Chunks und Produktionen bewerten und dann eine Entscheidung treffen. Das ist im Allgemeinen sehr umfangreich und dauert wegen der großen Wissensbasis lange. Das heißt, eine kognitive Simulation, die echtzeitfähig sein soll, muss die Relevanz von Chunks wesentlich schneller bewerten können, als heute existierende Systeme. Die Berechnungsdauer durch immer schnellere Rechner auf Echtzeit zu senken ist nur für einfache Probleme möglich und daher nicht praxistauglich. Als weiterer Aspekt der Ressourcenbegrenzung kommt noch hinzu, dass man nicht beliebig viel Wissen speichern kann, also das deklarative Gedächtnis wegen begrenzter Speicherkapazität nicht beliebig viele Chunks enthalten kann.

5. Fazit

ACT-R bietet aufgrund der langen Zeit, die es sich schon in Entwicklung befindet, und seiner aktiven und großen Community einen guten Einstiegspunkt für kognitive Simulationen. Außerdem ist ACT-R bei Bedarf erweiterbar, sodass auch eine Nutzung über die derzeitigen Einsatzszenarien hinaus möglich ist. Mit den vorgestellten Techniken ist ein erster Schritt in Richtung der Nutzung von kognitiven Simulationen zur Effizienzbewertung von Visualisierungen getan. Als nächsten Schritt müssen diese Techniken im Praxiseinsatz erprobt werden.

Die vorgestellten Einschränkungen sind für die kognitiven Simulationen zur Effizienzbewertung von Visualisierungen aus unserer Sicht zunächst weniger relevant. Das Symbol Grounding Problem spielt keine Rolle, weil die Simulation die Visualisierung nicht verstehen muss, sondern nur das Verständnis eines Menschen nachbilden muss. Dies ist aus unserer Sicht mit einem kognitiven Modell hinreichend abbildbar. Der Umgang mit Unschärfe und vagen Informationen lässt sich mit einem gewissen Grad an Genauigkeit auf Kosten der Geschwindigkeit aus unserer Sicht auch in einem kognitiven Modell emulieren. Die Ressourcenbeschränkungen sollten für erste Versuche keine Rolle spielen, da es sich um ein zwar großes, aber immer noch beschränktes Wissensgebiet handelt und eine Simulation in Echtzeit zunächst nicht im Vordergrund steht.

Die größte Lücke, die uns während unserer Recherchen aufgefallen ist, besteht darin, wie ein entsprechendes kognitives Modell aussehen soll. Dazu gehört:

- Die Abbildung von formal definierten Wissenstrukturen in Chunks für das deklarative Gedächtnis (4.3.1), sodass diese auch in der Simulation sinnvoll verwendet werden können.
- Erstellung eines Modells der Datendomain.
- Je nach Eingabe für das virtuelle Auge, Abbildung der Perzeption und Aggregation von Objekten zu einem für die Simulation verständlichen Modell der Visualisierung (4.3.3).

In Zukunft wird auch die Echtzeitfähigkeit der Simulation eine zunehmende Rolle spielen, sodass sich damit beschäftigt werden muss, wie die Performance der kognitiven Simulation verbessert werden kann. Ein weiteres Feld wäre neben der Analyse der Perzeption von statischen Visualisierungen die Analyse der Interaktion mit Visualisierungen.

A. Glossar

Chunk Elementare Informationseinheit des Gedächtnis. Wird in mentalen Prozessen als einzelne Entität gesehen. Ein Wort kann z.B. ein Chunk sein.

Fitts Law Model menschlicher Bewegungen, das die benötigte Zeit um möglichst schnell auf eine Zielfläche zu zeigen, als Funktion der Distanz zum Zielbereich, sowie der Größe des Zielbereichs ausdrückt.

Framework Programmiergerüst, das Entwicklern Fähigkeiten anbietet, die sie selbst wiederum beim Entwickeln von Anwendungen einsetzen können.

GUI Graphical User Interface. Die graphische Oberfläche einer Anwendung.

kognitive Architektur, Kognitionsframework Kann zur Implementierung von kognitiven Modellen auf Computern benutzt werden. Kognitive Architekturen können auf anderen kognitiven Architekturen aufbauen.

Kognitives Modell Beschreibt die menschliche Kognition in einem Modell. Ein kognitives Modell basiert auf einer kognitiven Theorie.

Kognitive Theorie Erklärt die menschliche Kognition mittels einer Theorie.

Millersche Zahl 7 +- 2. Größenordnung, wie viele Chunks gleichzeitig im Kurzzeitgedächtnis gehalten werden können.

Nominale Daten Daten, die keine natürliche Ordnung besitzen. Es kann lediglich entschieden werden, ob ein Datum zu einer bestimmten Kategorie gehört oder nicht.

Ontologie Formal geordnete Darstellungen einer Menge von Begrifflichkeiten und der zwischen ihnen bestehenden Beziehungen in einem bestimmten Gegenstandsbereich.

Ordinale Daten Daten, die eine inhärente Ordnung besitzen.

Perzeption Alle Vorgänge, die mit der Wahrnehmung assoziiert sind.

UI Abkürzung für User Interface. Die Benutzerschnittstelle über die der Benutzer mit einer Anwendung interagiert.

Visual Query Eine „Anfrage“ der Kognition an die Perzeption. Es soll ein sichtbarer Gegenstand durch die Perzeption verarbeitet werden, um durch die Kognition in eine mentale Repräsentation überführt zu werden.

Visualisierung Die Überführung von (abstrakten) Daten in eine visuelle Form. Bezeichnet sowohl den Prozess, als auch das Ergebnis.

Visuelles Sichtfeld Alles was durch das Auge zu einem bestimmten Moment wahrgenommen werden kann, begrenzt durch einen horizontalen und vertikalen Winkelbereich.

Volume Rendering Darstellung von volumetrischen Daten mit speziell dafür entwickelten Techniken.

Widget Komponente eines grafischen Fenstersystems

B. jACT-R Beispiel count.xml

```
<actr>
  <model name="Unit1 Count">
<!--
    the modules section allows you to specify what modules you
    want included in this model. Modules not included aren't even
    loaded, the fewer the modules, the faster the model will execute.
    Each module has a set of parameters which are excluded here for
    brevity. The first two are the bare minimum, but you'll be hard
    pressed to write a model without goals or retrievals.

    The latencyFactor below has been set to 0.05 to be consistent
    with the Lisp tutorial. It just makes the retrievals complete
    really fast.
    -->
    <modules>
      <module class="org.jactr.core.module.declarative.six.DefaultDeclarativeModule6"/>
      <module class="org.jactr.core.module.procedural.six.DefaultProceduralModule6"/>
      <module class="org.jactr.core.module.goal.six.DefaultGoalModule6"/>
      <module class="org.jactr.core.module.retrieval.six.DefaultRetrievalModule6">
        <parameters>
          <parameter name="LatencyFactor" value="0.05"/>
        </parameters>
      </module>
    </modules>

   <!--
     the declarative memory section contains chunk and chunktype definitions.
    -->
    <declarative-memory>
    <!--
      count order supports counting on your fingers
      lisp : (chunk-type count-order first second)
      -->
      <chunk-type name="count-order">
        <slot name="first" equals="nil"/>
        <slot name="second" equals="nil"/>
      </chunk-type>

   <!--
      count-from is our basic goal.
      lisp : (chunk-type count-from start end count)
      -->
      <chunk-type name="count-from">
        <slot name="start" equals="nil"/>
        <slot name="end" equals="nil"/>
        <slot name="count" equals="nil"/>
      </chunk-type>

    <!--
      add all the chunks we'll need
      lisp :
      (add-dm
 (b ISA count-order first 1 second 2)
 (c ISA count-order first 2 second 3)
 (d ISA count-order first 3 second 4)
```

```
(e ISA count-order first 4 second 5)
(f ISA count-order first 5 second 6)
(first-goal ISA count-from start 2 end 4)
)
     -->
     <chunk type="count-order" name="b">
       <slot name="first" equals="1"/>
       <slot name="second" equals="2"/>
     </chunk>

     <chunk type="count-order" name="c">
       <slot name="first" equals="2"/>
       <slot name="second" equals="3"/>
     </chunk>

     <chunk type="count-order" name="d">
       <slot name="first" equals="3"/>
       <slot name="second" equals="4"/>
     </chunk>

     <chunk type="count-order" name="e">
       <slot name="first" equals="4"/>
       <slot name="second" equals="5"/>
     </chunk>

     <chunk type="count-order" name="f">
       <slot name="first" equals="5"/>
       <slot name="second" equals="6"/>
     </chunk>

     <chunk type="count-from" name="first-goal">
       <slot name="start" equals="2"/>
       <slot name="end" equals="4"/>
     </chunk>
   </declarative-memory>

  <!--
     procedural memory contains all the productions
    -->
    <procedural-memory>



    <!--
      lisp :
(p start
   =goal>
      ISA         count-from
      start       =num1
      count       nil
 ==>
   =goal>
      count       =num1
   +retrieval>
      ISA         count-order
      first       =num1
)
      -->
      <production name="start">
```

```
        <conditions>
          <match buffer="goal" type="count-from">
            <slot name="start" equals="=num1"/>
            <slot name="count" equals="nil"/>
          </match>
          <!-- for completeness, I am checking that retrieval is free unlike the lisp -->
          <query buffer="retrieval">
            <slot name="state" equals="free"/>
          </query>
        </conditions>
        <actions>
          <modify buffer="goal">
            <slot name="count" equals="=num1"/>
          </modify>
          <add buffer="retrieval" type="count-order">
            <slot name="first" equals="=num1"/>
          </add>
          <output>"Searching for something starting at =num1"</output>
        </actions>
      </production>

 <!-- failed isn't included in the lisp, but its always good
      to model defensively in case of retrieval failures
      -->
      <production name="failed">
        <conditions>
          <match buffer="goal" type="count-from">
            <slot name="start" equals="=num"/>
            <slot name="count" equals="=num"/>
          </match>
          <query buffer="retrieval">
            <slot name="state" equals="error"/>
          </query>
        </conditions>
        <actions>
          <remove buffer="goal"/>
          <remove buffer="retrieval"/>
          <output> "Awh crap, I can't retrieve anything starting with =num "</output>
        </actions>
      </production>
<!--
 lisp :
 (P increment
   =goal>
      ISA         count-from
      count       =num1
    - end         =num1
   =retrieval>
      ISA         count-order
      first       =num1
      second      =num2
 ==>
   =goal>
      count       =num2
   +retrieval>
      ISA         count-order
      first       =num2
   !output!       (=num1)
)
```

```
      -->
      <production name="increment">
        <conditions>
          <match buffer="goal" type="count-from">
            <slot name="count" equals="=num1"/>
            <slot name="end" not="=num1"/>
          </match>
          <match buffer="retrieval" type="count-order">
            <slot name="first" equals="=num1"/>
            <slot name="second" equals="=num2"/>
          </match>
        </conditions>
        <actions>
          <modify buffer="goal">
            <slot name="count" equals="=num2"/>
          </modify>
          <add buffer="retrieval" type="count-order">
            <slot name="first" equals="=num2"/>
          </add>
          <output>"=num1"</output>
        </actions>
      </production>

<!--
  lisp :
(P stop
   =goal>
      ISA         count-from
      count       =num
      end         =num
 ==>
   -goal>
   !output!       (=num)

)
     -->
      <production name="stop">
        <conditions>
          <match buffer="goal" type="count-from">
            <slot name="end" equals="=num"/>
            <slot name="count" equals="=num"/>
          </match>
        </conditions>
        <actions>
          <remove buffer="goal"/>
          <output>"Answer =num"</output>
        </actions>
      </production>



    </procedural-memory>

 <!--
    outside of the procedural memory section, we have the buffers
    where you can set buffer parameters or contents.
    -->
    <buffer name="goal" chunk="first-goal"/>
```

```
    <!--
    these are model parameters. cycle skipping allows the model to
    fast-forward through time if it can't fire a production to the
    next event that may permit production firing.

    peristent execution controls whether or not the model will quit
    once the goal buffer is empty. It's much easier to set this to
    true than to artificially force the model to keep running by
    queueing spurious events
    -->
    <parameters>
      <parameter name="EnableUnusedCycleSkipping" value="true"/>
      <parameter name="EnablePersistentExecution" value="false"/>
    </parameters>
  </model>
</actr>
```

Literatur

[1] ACT-R. Act-r vision module guide. http://chil.rice.edu/projects/RPM/docs/vision_module.html.

[2] ACT-R Research Group. http://act-r.psy.cmu.edu/ (05.04.2011).

[3] Bonnie E. John. Cogtool user guide. http://cogtool.hcii.cs.cmu.edu/sites/default/files/CogToolUserGuide_1_1.pdf.

[4] Bonnie E. John. Predictive human performance modeling made easy. http://cogtool.hcii.cs.cmu.edu/sites/default/files/p455-john.pdf, 2004.

[5] Brath. Concept demonstration metrics for effective information visualization. In *IEEE Symposium on Information Visualization* (1997).

[6] Brath, R. K. Effective information visualization. Master's thesis, University of Toronto, 1999.

[7] Brett Harris, Bonnie E. John, Jonathan Brezin. Human performance modeling for all: Importing ui prototypes into cogtool, 2010.

[8] Card, S., Moran, T., and Newell, A. *The psychology of human-computer interaction*. CRC, 1983.

[9] Dario D. Salvucci. A model of eye movements and visual attention. http://act-r.psy.cmu.edu/papers/299/dds_2000_a.pdf, 2000.

[10] David Kieras. Using the keystroke-level model to estimate execution times. http://www.eecs.umich.edu/people/kieras/GOMS/KLM.pdf, 2001.

[11] Ertl, T. *Vorlesung Visualization*. Universität Stuttgart, 2009.

[12] Foster, D. H., and Ward, P. A. Asymmetries in oriented-line detection indicate two orthogonal filters in early vision. In *Proceedings: Biological Sciences* (1991).

[13] Foster, D. H., and Ward, P. A. Horizontal-vertical filters in early vision predict anomalous line-orientation identification frequencies. In *Proceedings: Biological Sciences* (1991).

[14] Google Maps. http://maps.google.de (03.2011).

[15] Harrison, A. jact-r. http://jactr.org/.

[16] Harrison, A. jact-r divergences. http://jactr.org/node/34.

[17] Hermann, F. *Vorlesung Mensch-Rechner-Interaktion 2.* Fraunhofer IAO, 07 2009.

[18] John R. Anderson, D. B. An integrated theory of the mind. *American Psychological Association 111* (2004), 1036–1060.

[19] Juarez-Espinosa, O. Caeva: Cognitive architecture to evaluate visualization applications. *Seventh International Conference on Information Visualisation* (2003), 589.

[20] Kieras, D. E. Epic: A cognitive architecture for computational modeling of human performance. `http://ai.eecs.umich.edu/people/kieras/epic.html`.

[21] Ledermann, F. parvis. `http://www.mediavirus.org/parvis/`.

[22] Norman, D. *The Design of Everyday Things.* Doubleday, 1990.

[23] Posner, M. I. Orienting of attention. *Quaterly Journal of Experimental Psychology 32* (1980), 3–25.

[24] Pylyshyn, Z. W. Computation and cognition: Issues in the foundation of cognitive science. *Behavioral and Brain Sciences 3* (1980), 111–32.

[25] Raschke, M., and Ertl, T. An interdisciplinary approach for the study of cognitive aspects in visualization and human-computer-interaction. In *Proceedings of the Interact 2011 Conference* (2011).

[26] Roth, S. F., and Mattis, J. Automating the presentation of information. In *Proceedings of the IEEE Conference on Artificial Intelligence Applications* (1991).

[27] SAGE. Papers. `http://www.cs.cmu.edu/~sage/papers.html` (08.06.2011).

[28] Schumann, H., and Müller, W. *Visualisierung - Grundlagen und allgemeine Methoden.* Springer-Verlag, 2000.

[29] Stuart K. Card, Thomas P. Moran, A. N. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, 1983.

[30] Treisman, A., and Sato, S. Conjunction search revisited. *Journal of Experimental Psycology: Human Perception and Performance 16* (1990), 459–478.

[31] Ware, C. *Information Visualization - Perception for Design.* Morgan Kaufmann, 2004.

[32] Weiskopf, D. *Vorlesung Grundlagen der Interaktiven Systeme.* Universitï¿½t Stuttgart, 2008.

[33] Wikipedia. Altersstruktur. http://de.wikipedia.org/wiki/Altersstruktur (22.03.2011).

[34] Wikipedia. Basalganglien. http://de.wikipedia.org/wiki/Basalganglien (11.04.2011).

[35] Wikipedia. Brodmann areas. http://en.wikipedia.org/wiki/Brodmann_areas (11.04.2011).

[36] Wikipedia. Informationsvisualisierung. http://de.wikipedia.org/wiki/Informationsvisualisierung (26.02.2011).

[37] Wikipedia. Japanese writing system. http://en.wikipedia.org/wiki/Japanese_writing_system (22.03.2011).

[38] Wikipedia. Resource description framework. http://de.wikipedia.org/wiki/Resource_Description_Framework (07.06.2011).

[39] Wikipedia. Scatter plot. http://en.wikipedia.org/wiki/Scatter_plot.

[40] Wikipedia. Scientific visualization. http://en.wikipedia.org/wiki/Scientific_visualization (26.02.2011).

[41] Wikipedia. Semantisches netz. http://de.wikipedia.org/wiki/Semantisches_Netz (30.05.2011).

[42] Wikipedia. Türme von hanoi. http://de.wikipedia.org/wiki/T%C3%BCrme_von_Hanoi.

[43] Wikipedia. Two stream hypothesis. http://en.wikipedia.org/wiki/Two_Streams_hypothesis.

[44] Wikipedia. Volume rendering. http://de.wikipedia.org/wiki/Volume_rendering (22.03.2011).

[45] Wilson, R. A., and Keil, F. C. *The MIT Encyclopedia of the Cognitive Sciences*. The MIT Press, 1999.

[46] WinDirStat. Windirstat. http://windirstat.info/.

[47] Wolfe, J. M. Guided search 2.0: A revised model of visual search. *Psyconomic Bulletin & Review 1* (1994), 202–238.

[48] Wolfe, J. M., Friedmann-Hill, S. R., Stewart, M. I., and O'Connell, K. M. The role of categorization in visual search for orientation. *Journal of Expe-*

rimental Psycology: Human Perception and Performance 18 (1992), 34–49.

Erklärung

Hiermit versichern wir, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Stuttgart, ...

Christoph Bergmann

Stuttgart, ...

Christian Dittrich

Stuttgart, ...

Michael Kircher