Institut für Visualisierung und Interaktive Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3384

Modellierung kognitiver Prozesse in der Visualisierung

Stephan Engelhardt

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Thomas Ertl
Betreuer:	Dipl.-Phys. Michael Raschke
begonnen am:	23.08.2012
beendet am:	22.02.2013
CR-Klassifikation:	I.2.0, I.6.4, I.6.5, J.4

Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit der Erstellung eines kognitiven Modells, um das Lesen von Visualisierungen zu simulieren. Für die Überprüfung des Mehrwerts neuer Visualisierungstypen müssen häufig aufwendige Studien mit Probanden durchgeführt werden. Mithilfe von Eye-Tracking-Systemen zeichnen Forscher die Augenbewegungen (Sakkaden) der Probanden auf und optimieren daraufhin die neuen Visualisierungstypen. Um die Kosten für Studien zu senken und die Optimierungsvorgänge zu beschleunigen, wird in dieser Arbeit ein kognitives Modell vorgestellt, das Augenbewegungen einzelner Probanden einer Visualisierungsstudie simuliert. Hierzu berechnet das Modell Fixationen und Fixationszeiten anhand verschiedener Einflussfaktoren. Das theoretisch erarbeitete Modell wurde in Form eines Prototyps als Computerprogramm implementiert und ist durch eine Studie validiert worden.

Abstract

This work presents a cognitive model to simulate the reading of visualizations. Often there have to be extensive studies with human subjects to evaluate the gain of value of new visualizations. With the aid of eye tracking systems researchers record the subjects' eye movements (saccades) and optimize their visualizations. To reduce the expenses for user studies and to speed up the optimization process, this paper presents a cognitive model, which simulates eye movements of subjects in a visualization user experiment. The model calculates fixations and fixation durations using various factors. It has been partially implemented as a prototype and was validated in a user study.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Im Bereich der Visualisierung entwickeln Forscher ständig neue Visualisierungstechniken. Die Gründe hierfür sind unterschiedlich. Zum einen werden grafische Darstellungen für Daten entwickelt, wofür noch keine Visualisierungen existieren. Zum anderen sollen bestehende Visualisierungen auf ihre kognitive Ergonomie oder für bestimmte Fragestellungen optimiert werden. Auch die Entwicklung neuer Visualisierungstypen, die parallel zu bestehenden Visualisierungen die gleichen Daten repräsentieren, ist ein Aufgabengebiet des Forschungsbereichs Visualisierung.

Um herauszufinden, wie schnell Antworten aus einer Visualisierung gelesen werden können, gibt es bisher nur die Möglichkeit, Benutzerstudien durchzuführen und deren Ergebnisse zu analysieren. Hierzu werden häufig die Augenbewegungen von Probanden mithilfe von Eye-Trackern beim Betrachten von Visualisierungen aufgezeichnet. Die ermittelten Daten werden anschließend auf unterschiedlichste Einflüsse hin analysiert und die untersuchten Visualisierungen optimiert. Durch weitere Benutzerstudien verbessern Forscher die Visualisierungstechniken fortwährend.

Die Aufwände und Kosten für Studien sind enorm. Für die Vorbereitung und Durchführung müssen oft mehrere Wochen investiert werden. Der Preis eines Eye-Trackers beträgt mehrere Tausend Euro und auch die Messergebnisse werden häufig durch unterschiedliche Störfaktoren, wie das Blinzeln eines Probanden, beeinflusst. Kleinste Änderungen in den Visualisierungen müssen erneut durch aufwendige und teure Studien überprüft werden.

Um den Aufwand und die Kosten zu senken, ist es denkbar, die Visualisierungen anstatt durch Studien, mithilfe von mathematischen Modellen zu analysieren. Einen Ansatz hierfür bildet die Modellierung und anschließende Simulation kognitiver Prozesse, die beim Lesen von Visualisierungen im menschlichen Gehirn ablaufen. Die Ergebnisse der Simulationen könnten die Flexibilität und Geschwindigkeit bei der Entwicklung neuer Visualisierungstypen deutlich erhöhen, wobei die Aufwände und Kosten für Studien gleichzeitig zurückgehen. Mithilfe sogenannter „Kognitionsframeworks“ kann bei der Erstellung eines solchen Modells bereits auf viele Erkenntnisse und Theorien kognitiver Prozesse zurückgegriffen werden.

Ziel dieser Arbeit ist die Entwicklung eines kognitiven Modells zur Vorhersage von Fixationen und Zeiten beim Lesen von Visualisierungen. Dieses Modell soll in Form ei-

nes Computerprogramms implementiert und mit der Durchführung einer Benutzerstudie validiert werden.

Die Arbeit gliedert sich in ein Grundlagenkapitel (Kapitel 2), das Basisbegriffe erklärt und beschreibt, wie kognitive Modelle entwickelt werden. Hierzu stellt das Kapitel außerdem zwei existierende Kognitionsframeworks vor. Nach Darstellung existierender Arbeiten (Kapitel 3), welche zum Teil für die Entwicklung des in dieser Arbeit entwickelten Modells verwendet werden, stellt Kapitel 4 nochmals die genaue Aufgabe und einen Lösungsansatz vor. Das darauffolgende Kapitel (Kapitel 5) beschreibt den Aufbau und die Funktion des kognitiven Modells im Detail. Daraufhin wird das implementierte Simulationstool vorgestellt und erklärt, wie dieses erweitert werden kann. Die Arbeit schließt mit einer Studie und deren Analyse zur Validierung des Modells ab (Kapitel 7).

2 Grundlagen der Kognition

Was verbirgt sich hinter dem Begriff „Kognition“ und warum beschäftigen wir uns damit? Bevor die eigentliche Aufgabe, die Modellierung kognitiver Prozesse, angegangen werden kann, müssen zunächst diese und einige weitere Fragen beantwortet werden.

Das Kapitel beschreibt grundlegendes Wissen zur Geschichte und Definition der Kognition (Abschnitte 2.2 und 2.1), zeigt verschiedene Ansichten auf (Abschnitt 2.3), erklärt oft genutzte Vorgehensweisen und stellt anschließend existierende Werkzeuge zur Modellierung kognitiver Prozesse vor (Abschnitt 2.4).

2.1 Was ist Kognition?

Kognition ist das, was im „Geist“ des menschlichen Gehirns abläuft. Sinnesorgane, wie Augen, Ohren oder die Haut nehmen Reize der Umwelt wahr und wandeln diese in elektrische Signale. Dies bezeichnet die Wissenschaft als „Perzeption“. Die Signale werden in unserem Gehirn verarbeitet und mit Erfahrungen und persönlichen Ansichten verknüpft, wodurch mentale Prozesse wie Lernen, Planen und Kreativität möglich werden. Den Vorgang dieser mentalen Prozesse nennt man „Kognition“ (Abbildung 2.1).

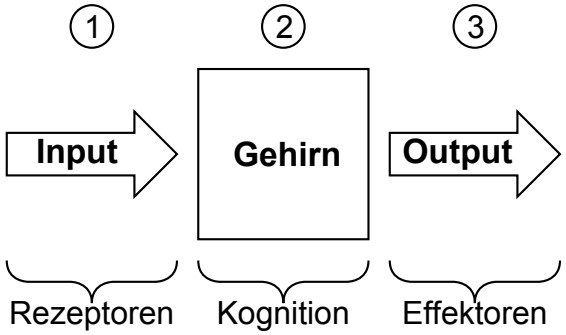


Abbildung 2.1: Definition der Kognition – Die Verarbeitung der Eingangssignale durch Rezeptoren unter Zuhilfenahme innerer Zustände (z. B. Erfahrungen, persönliche Einstellungen usw.).

Die Kognitionswissenschaft gibt es nicht als eigenständige Wissenschaft. Vielmehr ist sie ein Teilgebiet verschiedener Wissenschaften. So beschäftigen sich z. B. Mediziner und Biologen, die im Bereich der Neurowissenschaften forschen, mit den kognitiven Abläufen im Gehirn, indem sie die Funktionsweise einzelner Neuronen und neuronaler Netze untersuchen. Sie gehören damit zu der Gruppe der sogenannten „Konnektionisten“ (siehe Abschnitt 2.3.1). In anderen Wissenschaften, wie z. B. der Psychologie, versucht man häufig auf höherer Ebene zu verstehen, wie Gedanken, Meinungen, Einstellungen, Wünsche oder Absichten in einem Menschen entstehen. Ihnen genügt oft eine gröbere Beschreibung der Abläufe, sodass einzelne Neuronen weniger interessant werden. Die Psychologen sind damit eher der Gruppe der „Symbolisten“ (siehe Abschnitt 2.3.2) zuzuordnen. Wieder andere Wissenschaftler, wie z. B. Informatiker, die im Bereich der künstlichen Intelligenz forschen, entwickeln Algorithmen und Roboter zur Simulation kognitiver Fähigkeiten. Diese Wissenschaftler gehen meist sowohl konnektionistisch als auch symbolistisch vor. Dazu verwenden sie einerseits symbolverarbeitende Algorithmen, andererseits erstellen sie Programme mithilfe von künstlichen neuronalen Netzen, welche eindeutig subsymbolischen Charakter besitzen (siehe Abschnitt 2.3.1).

Genauso wichtig wie der Begriff der Kognition selbst ist seine Entstehungsgeschichte. Im folgenden Abschnitt wird deshalb beschrieben, wie es zum heutigen Verständnis des Kognitivismus kam und wie Meinungen einzelner Forscher zu Konflikten führten, die noch heute zu interessanten Diskussionen anregen können.

2.2 Geschichtliche Entwicklung

„Erkenne dich selbst!“

(Verfasser: unbekannt)

Diese Worte befinden sich an einer Vorsäule des Apollontempels in Delphi und zeigen, dass sich die Menschheit schon seit langer Zeit nicht nur mit der Natur um sich herum, sondern auch mit sich selbst beschäftigt.

Vielmehr als die heutigen Forschungen zielt der Spruch auf die sogenannte „Introspektion“ – die Selbstbeobachtung – ab. Mithilfe dieser Methode war es lange Zeit üblich, herauszufinden, wie mentale Prozesse ablaufen.

Später folgte die Ära des Behaviorismus. Dabei spielten die genauen Prozesse im Gehirn keine Rolle. Innere Zustände wurden abgestritten – es zählten einzig äußerlich messbare Reaktionen des Menschen auf unterschiedliche Reize.

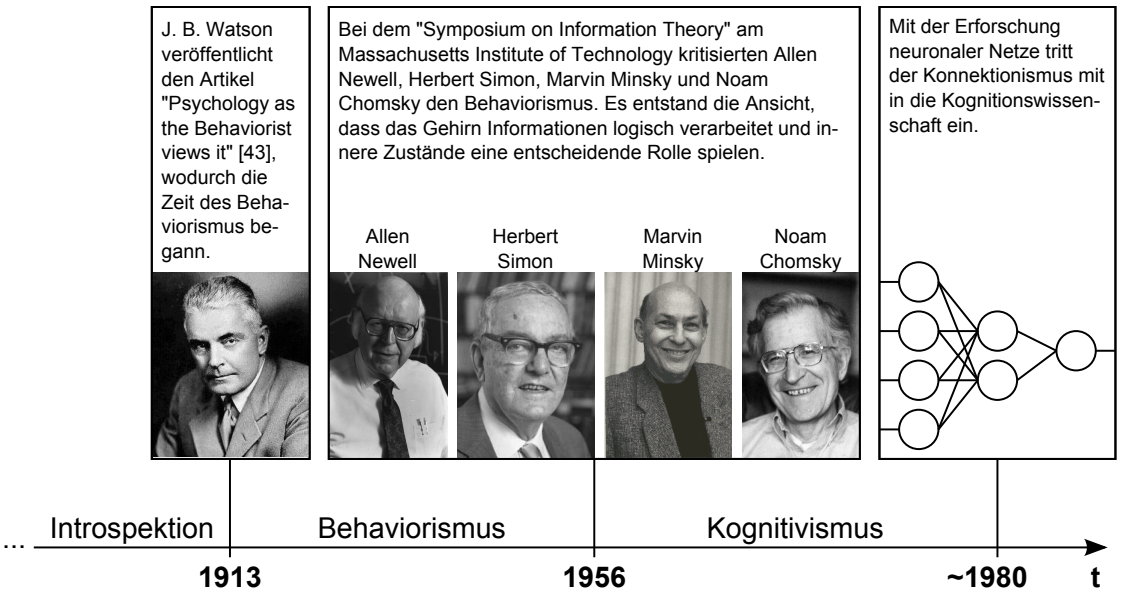


Abbildung 2.2: Geschichtliche Entwicklung der Kognitionswissenschaft – Lange Zeit war die Introspektion die einzig angesehene Wissenschaft, die sich mit Kognition beschäftigte. Erst 1913 kam durch J. B. Watson die Idee des Behaviorismus auf. 1956 wurde dieser bereits von einigen Wissenschaftlern kritisiert – der Kognitivismus entstand. Seit ca. 1980 existiert dieser zusammen mit der Erforschung neuronaler Netze.

Mithilfe moderner Messgeräte ist es teilweise möglich, kognitive Prozesse innerhalb des menschlichen Gehirns zu beobachten und zu analysieren. In dem sogenannten „Kognitivismus“ sind innere Zustände im Gegensatz zum Behaviorismus eine elementare Komponente.

2.2.1 Introspektion

Bei der Introspektion handelt es sich um ein Verfahren, bei dem Probanden bestimmte Aufgaben gestellt bekommen, die sie „sich selbst beobachtend“ lösen müssen. Dazu erklären sie, an was sie bei einem Schritt gerade denken und wie sie für bestimmte Aufgaben vorgehen. Durch geschickte Wahl der Fragen ist es damit unter anderem möglich, einzelne kognitive Prozesse zu identifizieren. Mehr dazu befindet sich im Abschnitt 2.4.3.

Gedanken verändern innere Zustände. Darum ist es fragwürdig, inwieweit die Introspektion bei der wissenschaftlichen Forschung Sinn ergibt, schließlich könnte allein schon der Gedanke daran, die inneren Zustände abzurufen, diese verändern.

Immanuel Kant erkannte bereits vor knapp 250 Jahren, dass

> *„[. . .] selbst die Beobachtung an sich schon den Zustand des beobachteten Gegenstandes alteriert und verstellt. Sie kann daher niemals etwas mehr als eine historische, und, als solche, so viel möglich systematische Naturlehre des inneren Sinnes, d. i. eine Naturbeschreibung der Seele, aber nicht Seelenwissenschaft, ja nicht einmal psychologische Experimentallehre werden“*
>
> (Kant, Immanuel 1786 [24])

1913 kritisierte J. B. Watson in dem Artikel „Psychology as the Behaviorist views it“ [43] die Introspektion und bewarb zugleich die Idee des Behaviorismus.

2.2.2 Behaviorismus

> *„Der Einwand gegen innere Zustände besteht nicht darin, dass sie nicht existierten, sondern darin, dass sie für eine funktionale Analyse nicht relevant sind.“*
>
> (Skinner, Burrhus Frederic 1973 [36])

Der Behaviorismus ignoriert das Geschehen im Gehirn völlig. Ein Behaviorist sieht das Gehirn als eine Art „Black-Box“, deren innere Zustände und Funktionen unbekannt oder uninteressant sind. Es wird nur analysiert, welche Signale und Wahrnehmungen aufgenommen werden und wie der Proband handelt. Diese Methode sollte, im Gegensatz zur Introspektion, objektive Ergebnisse liefern.

Einige Forscher konnten dem Gedanken, innere Zustände zu ignorieren, nichts abgewinnen. In seiner Besprechung „A Review of B. F. Skinner's Verbal Behavior“ [14] kritisierte Noam Chomsky 1959 den Behaviorismus, indem er zeigte, dass dieser nicht, wie Skinner es behauptete, auf das Sprachverhalten von Menschen angewandt werden kann. Der kanadische Psychologe Albert Bandura zeigte 1965 anhand eines Experiments (der Bobo Doll Study [10]) ebenfalls, dass innere Zustände für kognitive Prozesse wichtig sind. Auch der Verhaltensforscher Harry Frederick Harlow stellte fest, dass höhere kognitive Aufgaben nicht mit dem Behaviorismus vereinbar sind [11]. Die Belohnung durch Futter funktionierte in seinen Experimenten bei Ratten und Tauben relativ gut. Bei „höheren“ Lebewesen, wie Affen, konnte er jedoch kein einheitliches Verhalten feststellen. Innere Zustände und Emotionen beeinflussten die Experimente.

Die Zeit, in der der Behaviorismus scharfer Kritik unterlag, wird als „kognitive Wende“ bezeichnet. Sie führte zu einem neuen Denken, dem Kognitivismus.

2.2.3 Kognitivismus

Der Kognitivismus vertritt die Ansicht, die der Definition der Kognition (siehe Abschnitt 2.1) am stärksten ähnelt. Kognitionisten setzen sich, im Gegensatz zu den Behavioristen, speziell damit auseinander, was zwischen „Input“ und „Output“ passiert. Innere Zustände und Emotionen spielen für sie eine sehr entscheidende Rolle bei der Forschung.

Zur Entstehungszeit (um 1965) wurden die ersten programmierbaren Computer benutzt, weshalb man in den Anfängen das Gehirn nur von der symbolischen Seite (siehe Abschnitt 2.3.2) aus betrachtete. Nachdem man sich Mitte der 1980er Jahre mehr mit der Erforschung von neuronalen Netzen beschäftigte, hielten auch diese in einigen Bereichen Einzug in die Kognitionswissenschaft (siehe Abschnitt 2.3.1).

Es gibt noch viele weitere Vorstellungen wie kognitive Prozesse ablaufen, welche im Laufe der letzten Jahrzehnte entstanden. So bestehen beispielsweise Ansichten, wie der Enaktivismus [30], der Dynamizismus [15] oder der Erweiterte Geist [31], welche alle drei in etwa der Auffassung sind, dass Kognition nicht nur im Gehirn, sondern auch in der Umwelt und in der Interaktion mit ihr stattfindet.

Diese Arbeit wird sich ausschließlich auf Ergebnisse des Kognitivismus stützen, also die Konzepte der Konnektionisten und Symbolisten betrachten. Diese beiden Gruppen werden im folgenden Abschnitt näher beschrieben.

2.3 Konnektionisten vs. Symbolisten

Wie in Abschnitt 2.1 angedeutet existieren zwei Gruppen, die sich mit dem Kognitivismus auseinandersetzen. Auf der einen Seite sind die Konnektionisten, die auf die „Kraft“ einzelner Neuronen setzen, indem sie z. B. elektrische Spannungen an den „Eingängen“ von Neuronen anlegen und die Spannungen an deren „Ausgängen“ messen. Auf der anderen Seite befinden sich die Symbolisten, die hauptsächlich Experimente mit Probanden durchführen und dabei z. B. die Aktivitäten der einzelnen Gehirnregionen messen oder Theorien aufstellen, diese modellieren und anschließend validieren.

Aufgrund der unterschiedlichen Denkweisen besteht oft eine Kluft zwischen Konnektionisten und Symbolisten [18]. Die Konnektionisten tasten sich von den einzelnen Neuronen Ebene für Ebene nach oben (siehe Bottom-Up-Ansatz 2.3.1), während sich die Symbolisten bei der Frage beginnend tiefer in das Neuronengeflecht vorarbeiten (siehe Top-Down-Ansatz 2.3.2) (siehe Abbildung 2.3). Beide Vorgehensweisen sind genauso umstritten wie anerkannt, dabei gibt es auch Wissenschaftler, die beide Ansätze verbinden.

Im Folgenden werden beide Gruppen genauer untersucht. Es wird erklärt, wie die einzelnen Wissenschaftler vorgehen und auf Vor- und Nachteile der Ansätze eingegangen.

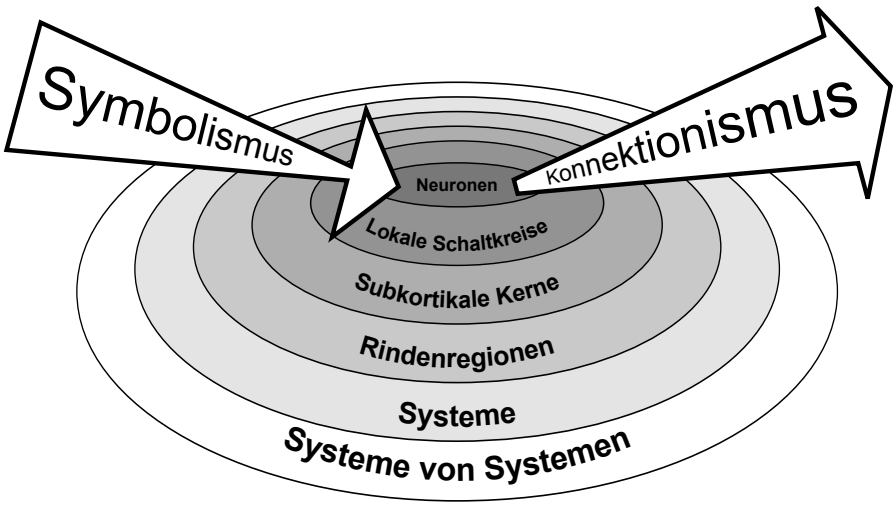


Abbildung 2.3: Die Ebenen der neuronalen Architektur – Konnektionisten tasten sich von den einzelnen Neuronen in höhere Ebenen vor, während Symbolisten kognitive Prozesse in kleinere Prozesse aufteilen und sich in tiefere Ebenen vorarbeiten [17].

Abbildung 2.4: NOT-Gatter (CMOS = **C**omplementary **M**etal **O**xide **S**emiconductor) – Eines der einfachsten Logik-Gatter in der Elektrotechnik. Liegt am Eingang X eine positive Spannung an, kommt an Ausgang Y eine negative Spannung an (Fall A). Liegt am Eingang A eine negative Spannung an, kommt an Ausgang Y eine positive Spannung an (Fall B).

2.3.1 Bottom-Up-Ansatz

Konnektionisten sind Verfechter des Bottom-Up-Ansatzes. Mit diesem Ansatz wird versucht, kognitive Fähigkeiten aus der tiefsten Ebene heraus, der Neuronenebene, zu verstehen. Dieses Kapitel erklärt, wie viele kleine Einheiten zusammen große Entscheidungen treffen können. Daraufhin wird beschrieben, wie das biologische Gehirn arbeitet und künstlich, in Form eines Modells, eingeschränkt nachgebaut werden könnte.

„Vereinte Kraft ist stark.“

(Kaiser Otto III)

Auch wenn das Zitat höchstwahrscheinlich an die Krieger von Kaiser Otto III gerichtet war und dies auf den ersten Blick ein primitives Verständnis wie „viel hilft viel“ vermittelt, so steckt mehr dahinter. Sind zwei Krieger gleich stark und kämpfen getrennt voneinander, ist ihre Gesamtleistung doppelt so hoch, wie die eines einzelnen Kriegers. Kämpfen sie hingegen Seite an Seite, planen gemeinsam Angriffe und warnen sich, wenn ein Gegner den einen Krieger z. B. von hinten angreift, so steigt ihre Gesamtleistung möglicherweise um ein Vielfaches.

Dieses Beispiel kann verwendet werden, um das Vorgehen der Konnektionisten beim Erforschen kognitiver Prozesse zu beschreiben. Die Forscher gehen so weit, dass sie sagen, dass sogar ein Element mit sehr geringer „Intelligenz“ zusammen mit Milliarden anderer solcher Elemente jede beliebige Höhe an Intelligenz aufweisen kann. Was hierbei „hohe Intelligenz“ bedeutet, soll zunächst außen vor gelassen werden. Unter „geringe Intelligenz“ kann man sich jedoch ein System vorstellen, welches so wenig „Sinne“, „Ausgabestellen“ und „Zustände“ wie möglich hat. Um ein solches System an einem praktischen Beispiel zu erklären, kann man ein in der Elektrotechnik oft verwendetes NOT-Gatter mit einem Ein- und einem Ausgang betrachten (Abbildung 2.4). Liegt am Eingang eine positive Spannung an, schaltet das Gatter am Ausgang eine negative Spannung. Liegt dagegen eine negative Spannung am Eingang an, kann am Ausgang eine positive Spannung abgegriffen werden.

In einem Gehirn befinden sich keine Logik-Gatter, wie sie in der Elektrotechnik zu finden sind. Dennoch gibt es dort einen ähnlichen Baustein: das Neuron.

Neuron: Ein Neuron besteht im Wesentlichen aus einem Zellkörper, seinen Dendriten und einem Axon (siehe Abbildung 2.5). Dabei bilden die Dendriten die Eingänge und das Axon den Ausgang des Neurons. Über die Dendriten erhält das Neuron mehrere elektrische Spannungen, welche größtenteils im Axonhügel aufsummiert werden. Überschreitet die Gesamtspannung ein bestimmtes Potenzial, so gibt das Neuron einen elektrischen Impuls an sein Axon weiter (das Neuron „feuert“) – liegt die Gesamtspannung unter diesem Potenzial passiert nichts (Alles-Oder-Nichts-Gesetz [5]).

Kommen nun viele Neuronen zusammen (bei einem Menschen sind das bis zu einer Milliarde [38]), so entsteht ein komplexes Netz – das sogenannte neuronale Netz.

Neuronales Netz: Die Dendriten der Neuronen sind über Synapsen mit den Axonen anderer Neuronen verbunden. Durch die Axone und Dendriten fließen elektrische Ströme, wobei diese durch die Synapsen voneinander getrennt sind (siehe Abbildung 2.5).

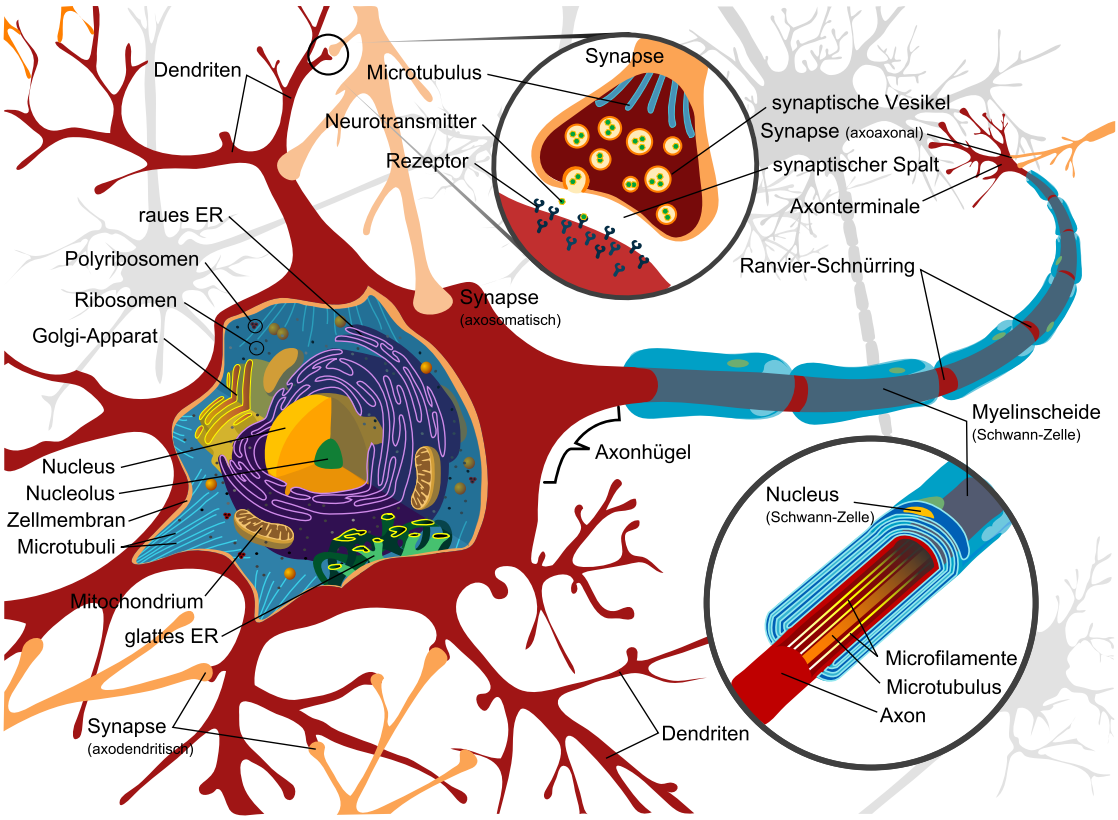


Abbildung 2.5: Der Aufbau eines Neurons [45]

Erst durch einen chemischen Effekt wird dieser Spalt in eine Richtung „überwunden“ und so der Strom weitergetragen. Ist eine Synapse groß, baut sie im Allgemeinen eine hohe elektrische Spannung auf. Ist sie klein, liegt am Dendrit nur eine niedrige Spannung an. Es gibt außerdem sowohl erregende (exzitatorische) als auch hemmende (inhibitorische) Synapsen, welche positive bzw. negative Spannungen aufbauen. Bei einigen Neuronen sind die Dendriten nicht verbunden (primäre Sinneszellen, z. B. Nozizeptoren). Andere Neuronen sind ausschließlich mit sekundären Sinneszellen verbunden (z. B. Fotorezeptoren). Die primären und sekundären Sinneszellen bilden die Eingangsneuronen des neuronalen Netzes. Effektoren (wie Muskeln oder Drüsen) sind ebenfalls mit Axonen verbunden. Sie bilden die Ausgänge des Netzes. Werden einzelne Sinneszellen aktiv, beginnen viele Neuronen zu feuern, andere wiederum bleiben in ihrem Anfangszustand. Maßgeblich ausschlaggebend dafür, welche Effektoren am Ende angesteuert werden, sind die Synapsen. Sie verbinden die Neuronen miteinander und sorgen anhand ihrer Größe für eine unterschiedliche Abschwächung der elektrischen Spannungen (kleine Synapsen = starke Abschwächung, große Synapsen = geringe Abschwächung). Durch ständig wie-

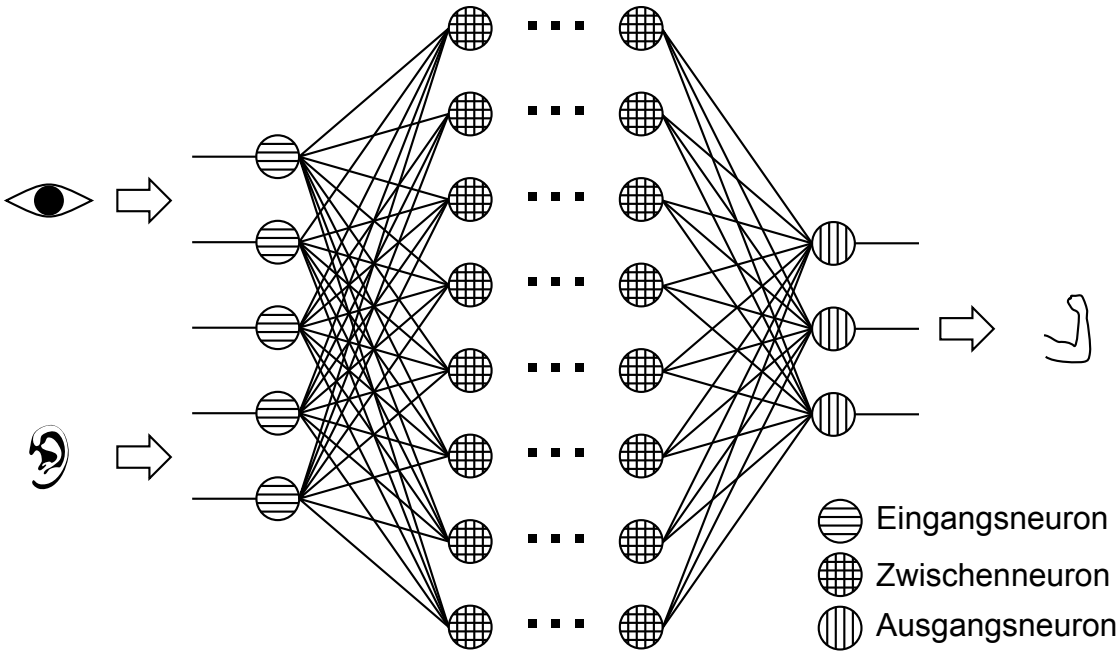


Abbildung 2.6: Künstliches Neuronales Netz – Die Reize der Sinneszellen kommen an den Eingangsneuronen an. Dort werden sie über Zwischenneuronen zu den Ausgangsneuronen bzw. Effektoren weitergeleitet.

derkehrende Reize an einzelnen Neuronen wachsen die Synapsen, während sie an Stellen, die seltener angesteuert werden, schrumpfen. Erst durch diesen Vorgang wird das Lernen möglich (siehe Hebbsche Lernregel [22]).

Konnektionisten arbeiten meist nur mit einem Modell des neuronalen Netzes – dem künstlichen neuronalen Netz (KNN). In diesem werden nahezu alle „Teile“ der natürlichen Neuronen weggelassen. Nur die „Zellkörper“ und „Dendriten“ sind noch vorhanden (Abbildung 2.6). Wie in einem echten neuronalen Netz werden Spannungen an den Eingängen der Eingangsneuronen angelegt. Überschreiten die Spannungen die von den Neuronen abhängigen Schwellenwerte, feuern diese Neuronen an die nächste Neuronenschicht. In dieser Schicht hat jedes Neuron für gewöhnlich mehrere Eingänge, über die die Eingangsspannungen aufsummiert werden. Überschreiten die Summen wiederum bestimmte Werte, feuern diese Neuronen. Der Vorgang zieht sich durch alle Neuronenschichten hindurch, bis die Schicht der Ausgangsneuronen erreicht ist und z. B. Effektoren angesteuert werden.

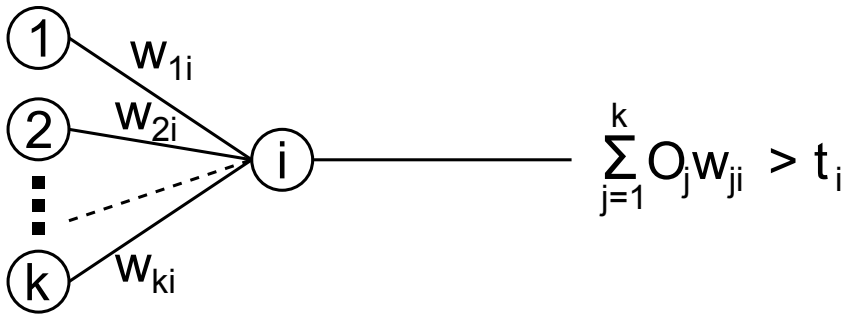


Abbildung 2.7: Berechnung der Aktivierung eines Neurons – Die Produkte der Ausgabe O_j der Neuronen mit der Gewichtung der Kante w_{ij} werden aufsummiert. Überschreitet die Summe einen Schwellwert t_i, feuert das Neuron i.

Die Aktivierung, die an einem Neuron durch die vorgeschalteten Neuronen ankommt, berechnet sich dabei wie folgt, wobei in Abbildung 2.7 dieser Zusammenhang nochmals dargestellt ist:

$$I_i = \sum_{j=1}^{k} O_j * w_{ij} \tag{2.1}$$

Dabei gilt:

- I_i: Aktivierung des Neurons i
- k: Anzahl der vorgeschalteten Neuronen von Neuron i
- O_j: Ausgabe des Neurons j
- w_{ij}: Gewicht (= Größe der Synapse) zwischen Neuron i und Neuron j

Da KNNs meistens in Computerprogrammen simuliert und selten mit elektrischen Bauteilen realisiert werden, wird auf physikalische Einheiten verzichtet.

Die Gewichtungen der Kanten werden dabei, wie beim biologischen Gehirn, durch das Lernen verändert. Dafür gibt es unterschiedliche Verfahren, wie die Hebbsche Lernregel [22] oder den Backpropagation-Algorithmus [35].

Die Konnektionisten sind davon überzeugt, irgendwann alle kognitiven Prozesse verstehen zu können, wenn sie das Gehirn von der Neuronenebene aufbauend analysieren und modellieren. Symbolisten vermuten, dass sie genau mit dem umgekehrten Weg schneller ans Ziel kommen werden: dem Top-Down-Ansatz.

2.3.2 Top-Down-Ansatz

In den Anfängen der Kognitionswissenschaft, als man noch nicht die Möglichkeit hatte einzelne Neuronen zu untersuchen, musste man für die Erforschung des Gehirns sehr einfallsreich sein.

Auch wenn es mit modernen Messgeräten seit einigen Jahrzehnten möglich ist, kleinste Interaktionen zwischen Neuronen zu messen und zu verstehen, ist der Symbolismus nicht ausgestorben. Warum es immer noch Verfechter des Top-Down-Ansatzes gibt und wie interessant auch dieses Vorgehen ist, wird in diesem Abschnitt beschrieben.

> *„Divide et impera – teile und herrsche“*
>
> (Verfasser: unbekannt)

Nach diesem Prinzip gehen viele Forscher an komplexe Aufgaben heran (z. B. Informatiker beim Quicksort-Algorithmus oder bei der schnellen Fourier-Transformation). Auch das Gehirn ist eine komplexe Einheit, die, laut einigen Symbolisten (z. B. Pylyshyn [20], Fodor [19]), nicht im Detail verstanden werden kann. Um dem Verständnis Schritt für Schritt näher zu kommen, können komplexe kognitive Aufgaben in mehrere primitive Aufgaben unterteilt werden. Diese müssen wiederum weiter unterteilt werden – so lange, bis eine Einheit entsteht, die vollkommen verstanden werden kann. Dabei muss die Einheit nicht zwangsweise die Ebene eines einzelnen Neurons sein, sondern darf bereits auf höherer, für einen Menschen logisch nachvollziehbaren, Ebene definiert werden.

Zuerst einmal scheint es naheliegend, das Gehirn von Grund auf verstehen zu wollen (siehe Bottom-Up-Ansatz 2.3.1). Versetzt man sich jedoch z. B. in die Rolle eines Psychologen, stellt man fest, dass der umgekehrte Weg sinnvoll sein kann. Ein Werbepsychologe ist kaum daran interessiert, wie ein Netz aus Neuronen das Bild eines Logos in den Gehirnen einer Zielgruppe abspeichert. Dagegen ist es für den Werbepsychologen elementar zu wissen, welche Assoziationen sie mit dem Logo verknüpfen, welche Gefühle sie beim Anblick haben, oder wie die Leute damit zum Kaufen angeregt werden können. Solche Zusammenhänge können unter anderem mithilfe von Studien ermittelt werden. Für einen Konnektionisten mag das Vorgehen oberflächlich erscheinen, es wird aber in diesem Zusammenhang mit Sicherheit schnellere Ergebnisse liefern können und für einen Werbepsychologen ausreichend sein.

Die Wissensbasis, die für das Verhalten eines Menschen verantwortlich ist, wird oft in Form eines semantischen Netzes dargestellt. Sowohl das beschreibende Wissen über Fakten (das deklarative Wissen) als auch das Wissen darüber, wie Dinge in der Umwelt verändert werden können (das prozedurale Wissen) können darin beschrieben werden.

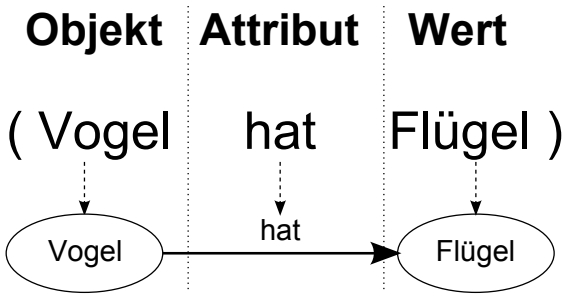


Abbildung 2.8: Objekt-Attribut-Wert-Tripel – Jedes Tripel definiert die Eigenschaft eines bestimmten Objekts.

Semantisches Netz: Ein semantisches Netz besteht aus Datentripeln, die durch einen Graph repräsentiert werden. Dabei bildet der erste und dritte Wert jedes Tripels je einen Knoten und der zweite Wert die Kante zwischen diesen (Abbildung 2.8). Die genaue Bedeutung der Knoten und Kanten kann unterschiedlich definiert werden. Meist gibt es eine zugehörige Ontologie, welche die Bedeutung beschreibt. In Abbildung 2.9 ist beispielhaft ein semantisches Netz abgebildet. Hierbei handelt es sich um ein hypothetisches Netz, das nach Collins und Quillian Wissen in einem Menschen darstellen könnte [16]. Die zugehörige Ontologie definiert z. B. Kanten, welche eine Eltern-/Kind-Relation darstellen, durch die mehrere Ebenen entstehen (Ebene eins bis drei). Dadurch lässt sich im dargestellten Netz beispielsweise ablesen, dass ein Strauß ein Vogel und dieser wiederum ein Tier ist. Besonders die Beziehungen, die sich über mehrere Knoten hinweg erstrecken (z. B. Strauß ist Tier), erfordern spezielle Suchstrategien. Dafür gibt es einerseits Algorithmen der Graphentheorie, wie die Breitensuche, andererseits existieren Algorithmen, die der Ausbreitung in einem neuronalen Netz ähneln. Eine Klasse dieser Algorithmen ist die sogenannte „Aktivierungsausbreitung“ oder „Spreading Activation“.

Spreading Activation: Mithilfe eines Spreading-Activation-Algorithmus lassen sich Assoziationen zu einem oder mehreren gegebenen Startknoten finden. Die ermittelten Assoziationen sind unterschiedlich gewichtet und können dadurch flexibel weiterverwendet werden. Für den Algorithmus werden zu jeder Kante eines semantischen Netzes Gewichtungen definiert. Diese können sowohl positiv als auch negativ sein. Zusätzlich kann jeder Knoten einen bestimmten Schwellenwert, Abschwächungsfaktor und weitere Eigenschaften besitzen. Der Algorithmus startet bei einem oder mehreren gegebenen Knoten, deren Assoziationen gefunden werden sollen. Von dort aus werden alle Nachbarknoten über die entsprechenden Kanten aktiviert. Die Stärke der Aktivierung hängt dabei von der Gewichtung der jeweiligen Kante ab. Dieser Vorgang wird nun rekursiv wiederholt, bis die Aktivierung so weit abgeschwächt ist, dass keine weiteren Knoten mehr aktiviert werden können. Nun können die einzelnen Knoten ausgelesen und beispielsweise die mit

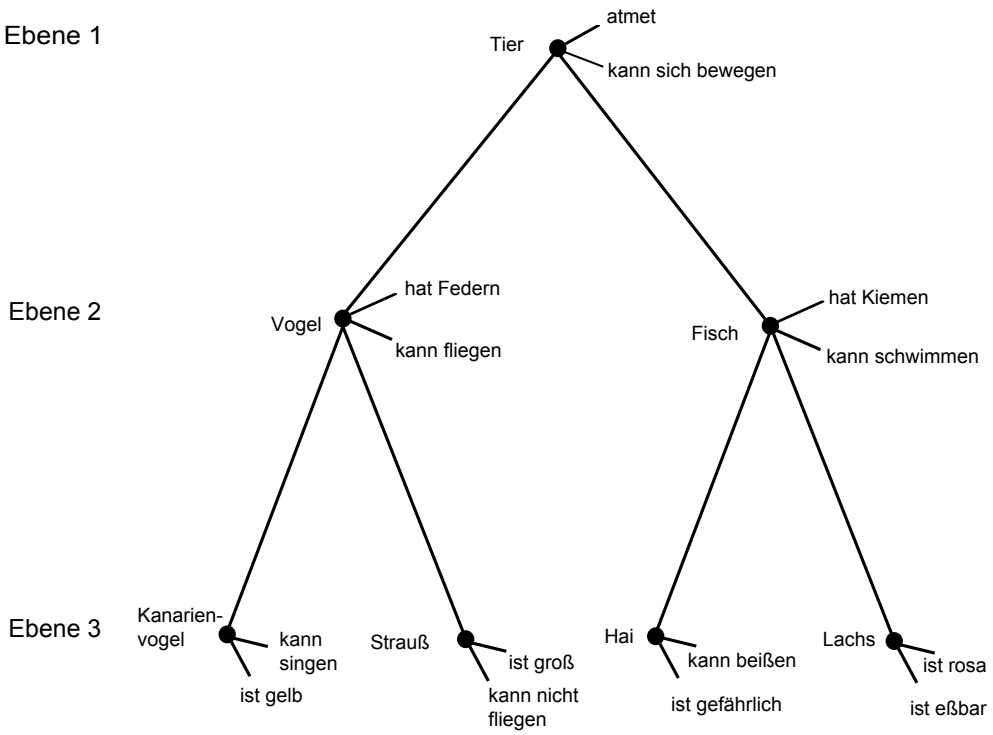


Abbildung 2.9: Semantisches Netz nach Collins und Quillian – Das Netz stellt Wissen über verschiedene Tiere und ihre Beziehungen zueinander dar. [16]

der höchsten Aktivierung weiterverwendet werden. Der genaue Ablauf ist nicht fest vorgegeben, sondern kann frei modifiziert werden. So könnte man beispielsweise die Kanten für eine Eltern-/Kind-Relation anders behandeln, als die Kanten für einfache Attribute. Abbildung 2.10 verdeutlicht einen Spreading-Activation-Algorithmus an einem Beispiel.

Sowohl der Bottom-Up- als auch der Top-Down-Ansatz sind in der heutigen Forschung vertreten. Beide Vorgehen haben ihre Vor- und Nachteile, welche in folgendem Abschnitt erläutert werden.

2.3.3 Vor- und Nachteile der Ansätze

Im vorigen Abschnitt wurden zwei Verfahren vorgestellt, mit denen Wissenschaftler die kognitiven Prozesse im menschlichen Gehirn erforschen. Auf der einen Seite sind die Konnektionisten, die den Bottom-Up-Ansatz (siehe Abschnitt 2.3.1) vertreten, auf der anderen Seite befinden sich die Symbolisten, die Verfechter des Top-Down-Ansatzes (siehe Abschnitt 2.3.2) sind.

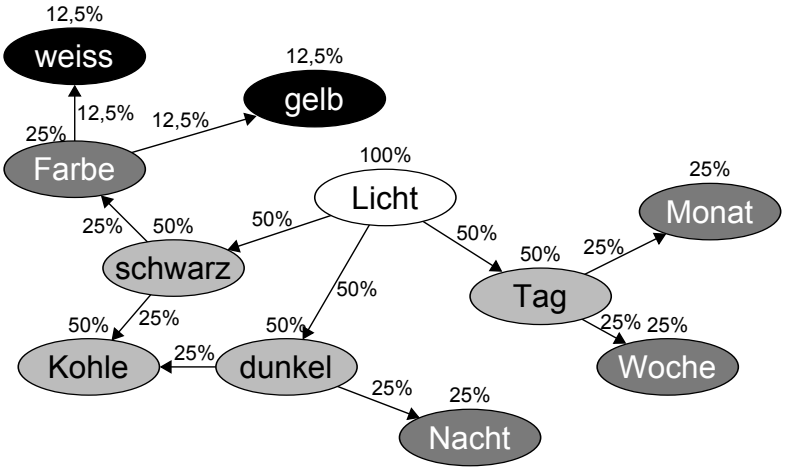


Abbildung 2.10: Spreading-Activation-Algorithmus – In diesem Beispiel hat jede Kante eine Gewichtung von 0,5. Der Knoten „Licht“ wird mit 100% aktiviert. Der Knoten „Kohle“ wird von den beiden Knoten „schwarz“ und „dunkel“ zu je 25% und somit zur Summe von 50% aktiviert.

Beide Herangehensweisen haben ihre Vorzüge, was die folgenden Aufzählungen verdeutlichen sollen.

Die Vorteile des Bottom-Up-Ansatzes sind: [26]

- Es werden kaum Theorien aufgestellt. Stattdessen werden einzelne Neuronen und deren Interaktionen untereinander erforscht. Am Ende soll ein kognitives Modell entstehen, das komplett auf dem Vorbild der Natur beruht.
- Komplexe und für einen Menschen dennoch selbstverständliche kognitive Fähigkeiten, wie Mustererkennung oder Sprache, können durch neuronale Netze bereits sehr gut simuliert werden.
- Durch Redundanz ist es möglich, dass die konnektionistischen Systeme trotz partieller Ausfälle ihre Arbeit weiterhin zufriedenstellend verrichten.

Die Vorteile des Top-Down-Ansatzes sind:[19]

- Theoretisch weiß man zu jeder Zeit wie etwas funktioniert. Da komplexe Probleme fortwährend in kleinere Probleme zerlegt werden, lässt sich das Gesamtsystem immer vollständig verstehen.
- Die meisten komplexen Probleme lassen sich durch kleine, einfache Probleme beschreiben, welche nicht bis auf Neuronenebene heruntergebrochen werden müssen. Dadurch führt der Top-Down-Ansatz oft sehr schnell zum Ziel.

- Bei logischen Aufgaben, bei denen es auf diskrete Ergebnisse ankommt, liefern Algorithmen, die mit dem Top-Down-Ansatz entworfen wurden, sehr gute Werte.
- Durch die meist gröbere Herangehensweise werden weniger Rechenschritte benötigt, als bei konnektionistischen Verfahren, was der begrenzten Rechenleistung, auch von modernen Computern, entgegenkommt.

Die Nachteile der beiden Ansätze ergeben sich jeweils durch die Vorteile des anderen. Top-Down-Systeme, die auf symbolischer Ebene arbeiten sind im Bereich der Mustererkennung ungeeignet und beim Ausfall einzelner Subsysteme nicht mehr funktionstüchtig. Bottom-Up-Systeme, basierend auf subsymbolischen Algorithmen, benötigen dagegen eine hohe Rechenleistung und sind somit für Echtzeitsysteme oft nur schwer einsetzbar.

Nachdem die beiden grundsätzlichen Ansätze zum Modellieren kognitiver Prozesse vorgestellt und miteinander verglichen wurden, soll im Folgenden das genaue Vorgehen für die Modellierung auf symbolischer Ebene näher beschrieben werden.

2.4 Kognitive Modellierung

Dieser Abschnitt befasst sich mit der allgemeinen Herangehensweise zum Aufbau kognitiver Modelle. Nach einer kurzen Definition (Abschnitt 2.4.1) werden die vier grundlegenden Phasen vorgestellt, aus denen ein Modellierungsprozess im Regelfall aufgebaut ist (Abschnitt 2.4.2). Die beiden Phasen „Empirische Untersuchung“ und „Adäquatheitsprüfung“ werden dabei in den Abschnitten 2.4.3 und 2.4.6 näher beleuchtet. Außerdem werden einige existierende Simulationsumgebungen vorgestellt (Abschnitt 2.4.4) und gezeigt, wie Modelle für diese geschrieben werden (Abschnitt 2.4.5).

Bevor jedoch auf die einzelnen Phasen eingegangen wird, soll kurz beschrieben werden, was kognitive Modellierung im Detail bedeutet.

2.4.1 Definition

Als Erklärung, was unter „kognitive Modellierung“ zu verstehen ist, gibt es zwei übliche Definitionen:

> *Versuch, für „ausgewählte kognitive Leistungen Symbolstrukturen (für Daten und Regeln) anzugeben und zu zeigen, dass mit eben diesen Daten und Regeln die zu erklärende kognitive Leistung erbracht werden kann“.*
>
> (Tack, Werner 1995 [37])

> *„Die Spezifikation und Implementation virtueller kognitiver Systeme, bestehend aus rekonstruierten repräsentationalen Strukturen und einer diese enkodierenden und interpretierenden kognitiven Architektur.“*
>
> (Wallach, Dieter 1998 [41][1])

Die Definition von Tack beschränkt sich dabei eindeutig auf das symbolische Vorgehen und lässt offen, was mit dem entwickelten Modell passieren wird. Wallachs Definition ist im Hinblick auf die symbolische oder subsymbolische Herangehensweise offen, geht jedoch einen Schritt weiter als Tack und sagt aus, dass das Modell am Ende mit einer kognitiven Architektur implementiert sein muss.

Diese Arbeit stützt sich im Wesentlichen auf die Definition von Wallach, wobei hauptsächlich auf symbolischer Ebene gearbeitet wird.

Im Folgenden werden nun die vier Phasen beschrieben werden, die die Erstellung eines kognitiven Modells begleiten.

2.4.2 Vier Phasen der Modellierung

Nach der Definition im vorigen Abschnitt werden nun die vier Phasen der kognitiven Modellierung („Aufgabenanalyse“, „Empirische Untersuchung“, „Modellimplementierung“ und „Adäquatheitsprüfung“) nach Wallach [41][2] vorgestellt.

Phase 1: Aufgabenanalyse
In der ersten Phase geht es darum, die zu lösende Aufgabe zu spezifizieren und ein theoretisches Modell dafür zu erstellen. In der Spezifikation muss festgelegt werden, welches Wissen für das Lösen der Aufgabe notwendig ist. Außerdem sind die Beziehungen zwischen den einzelnen Wissenseinheiten zu definieren. Über das Wissensmodell hinaus müssen zudem mögliche kognitive Prozesse, die in Verbindung mit dem Wissensmodell das Lösen der Aufgabe ermöglichen, spezifiziert werden.

Phase 2: Empirische Untersuchung
Je nach Art des Modells, das entworfen werden soll (idiographisches[3], prototypisches[4] oder individuelles Modell[5]), werden in der zweiten Phase empirische Untersuchungen mit einem oder mehreren Probanden durchgeführt. Dadurch können Komponenten des theoretischen Modells überprüft und fehlende Bestandteile ermittelt werden. Einige Möglichkeiten hierfür sind im Abschnitt 2.4.3 beschrieben.

[1] Seite 37
[2] Seite 44
[3] Modell, das anhand *eines* ausgewählten Probanden erstellt wurde und diesen simuliert
[4] Modell, das *alle* Personen einer bestimmten Gruppe simuliert
[5] Modell, das sowohl allgemeine Teile einer bestimmten Personengruppe als auch individuelle Teile einzelner Personen simuliert

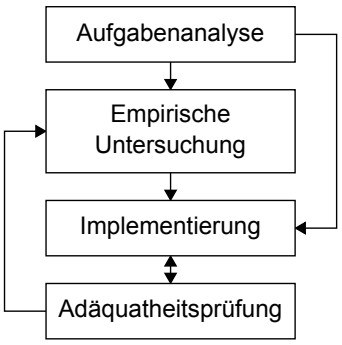


Abbildung 2.11: Vier Phasen der Modellierung – Der in der Praxis übliche Zyklus der einzelnen Phasen bei der Modellierung kognitiver Systeme.

Phase 3: Modellimplementierung
Die dritte Phase umfasst die Programmierung des Modells auf Grundlage der vorigen Phasen. Oft wird hierzu nicht von Grund auf mit der Entwicklung begonnen. Stattdessen bedient man sich sogenannter Kognitionsframeworks, wie sie in Abschnitt 4.2 vorgestellt werden.

Phase 4: Adäquatheitsprüfung
Nach der Implementierung muss das zuvor erstellte Modell auf Gültigkeit geprüft werden. Dazu führt man, wie in der zweiten Phase, Untersuchungen mit Probanden durch und vergleicht deren Handeln mit den Ergebnissen der Simulation. Hierbei kann es vorkommen, dass sich die Simulation bei einigen Tests zwar den Versuchspersonen ähnlich verhält, bei anderen Tests jedoch versagt (Identifikationsproblem). Um die Wahrscheinlichkeit zufällig korrekter Ergebnisse zu minimieren, ist es daher wichtig, viele verschiedene Messverfahren durchzuführen. Einige Verfahren werden im Abschnitt 2.4.6 beschrieben.

Das Durchlaufen der Phasen erfolgt im Regelfall stetig, wobei ein Zurückspringen in vorherige Phasen übliche Praxis ist. Vor allem die Implementierung des Modells in der dritten Phase muss aufgrund der Ergebnisse der Adäquatheitsprüfung ständig optimiert werden (siehe Abbildung 2.11).

Im folgenden Abschnitt sollen einige Verfahren für die empirische Untersuchung in der zweiten Phase der Vier-Phasen-Modellierung vorgestellt werden.

2.4.3 Möglichkeiten empirischer Untersuchungen

Bevor ein Modell entworfen werden kann, muss herausgefunden werden, wie die einzelnen Prozesse bei einem Menschen ablaufen. Die Schwierigkeit hierbei ist, dass das menschliche Gehirn für die Wissenschaft bisher mehr oder weniger eine Black-Box ist und nicht

an beliebigen Stellen gemessen werden kann, was wann passiert. Anderson beschreibt dieses Problem mit einem einfachen Vergleich:

> *„Developing a theory in cognitive psychology is much like developing a model for the working of the engine of a strange new vehicle by driving the vehicle, being unable to open it up to inspect the engine itself.“*
>
> (Anderson, John R. 1980/1988 [8])

Durch Messungen mit unterschiedlichen Ansätzen ist es jedoch möglich, Abläufe in kleinere kognitive Prozesse aufzuteilen. Ziele hierfür können unter anderem sein:

- Extraktion einzelner, unabhängiger kognitiver Prozesse
- Zeitmessungen kognitiver Prozesse
- Wahrscheinlichkeitsverteilungen bei der Entscheidungsfindung

Im Folgenden werden einige Messverfahren vorgestellt:

Selbstauskunft/Introspektion: Die wohl einfachste Methode den Ablauf kognitiver Prozesse zu extrahieren, bietet sich in dem Verfahren der Introspektion. Ein Proband bekommt eine Aufgabe, welche er lösen muss. Dabei soll er in jedem Schritt laut sagen oder aufschreiben, was er gerade denkt und wie er weiter vorgehen möchte.

Dieses Verfahren ist trotz seiner Einfachheit nicht zu unterschätzen. Allerdings gibt es dazu auch harsche Kritik einiger Wissenschaftler (z. B. Wundt [46]), da die Konzentration auf die eigentliche Aufgabe verloren gehen und viel mehr im Bewusstsein stattfinden könnte, als wenn die Aufgabe ohne Selbstauskünfte gelöst werden würde.

RT-Messung: Bei der RT-Messung wird die Zeit gemessen, die ein Proband benötigt, um auf einen Input zu reagieren (Reaktionszeitmessung) bzw. um eine Entscheidung zwischen mehreren Möglichkeiten zu treffen. Dieses Verfahren wird oft angewendet, um genauere Informationen über den Aufbau des semantischen Gedächtnisses einzelner Probanden zu erhalten.

Zwei spezielle RT-Verfahren sind die Subtraktionsmethode und die additive-factors Methode, die im Folgenden beschrieben werden:

Subtraktionsmethode: Die Subtraktionsmethode bietet die Möglichkeit, Zeiten für kleinste kognitive Prozesse zu messen. Dazu werden zwei Aufgaben gestellt, wobei sich diese einzig um den zu messenden Aspekt unterscheiden. Die Differenz der RT-Messungen ergibt die Zeit, die für den entsprechenden Aspekt benötigt wird.

Additive Factor Methode: Mithilfe dieser Methode lässt sich die Unabhängigkeit kognitiver Faktoren feststellen. Ist beispielsweise die RT eines (Sub-)Prozesses bekannt und soll dieser alleine für eine Entscheidungsfindung verantwortlich sein, muss bei mehrfacher Hintereinanderausführung der Aufgabe die RT linear ansteigen.

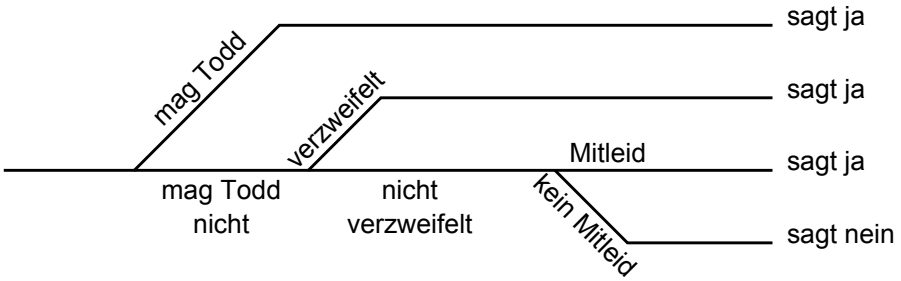
Soll die Aufgabe beispielsweise sein, zwei Zahlen miteinander zu addieren, wird dies bei 100 zufälligen Kombinationen darauf hinauslaufen, dass es 100-mal so lange dauert wie das Lösen einer dieser Aufgaben. Sind die 100 Kombinationen hingegen so aufgebaut, dass die erste Zahl immer eine Fünf ist und die zweite Zahl nach jeder Kombination um eins erhöht wird ($5 + 3$, $5 + 4$, $5 + 5$, $5 + 6$, ...), so wird die Versuchsperson nach einer Weile das Ergebnis der Aufgabe vorhersagen können. Die RT für den Versuch wird dann nicht mehr linear ansteigen – ein weiterer kognitiver Prozess überlagert allmählich den ursprünglichen Prozess.

Dissozationsverfahren: Dieses Verfahren wird hauptsächlich in der Neuropsychologie angewandt. In Verbindung mit bildgebenden Verfahren (PET, MRI, MET, ...) lassen sich damit komplexere mentale Prozesse extrahieren. Grundsätzlich benötigt man mehrere Probanden, die unterschiedliches Wissen über ein bestimmtes System besitzen. Die Probanden bekommen alle die gleiche Aufgabe gestellt, wobei ihre Gehirnaktivität während der Lösung gemessen wird. Die Differenz bzw. Schnittmenge der Aktivitäten zeigt nun, welche kognitiven Fähigkeiten wofür benötigt werden.

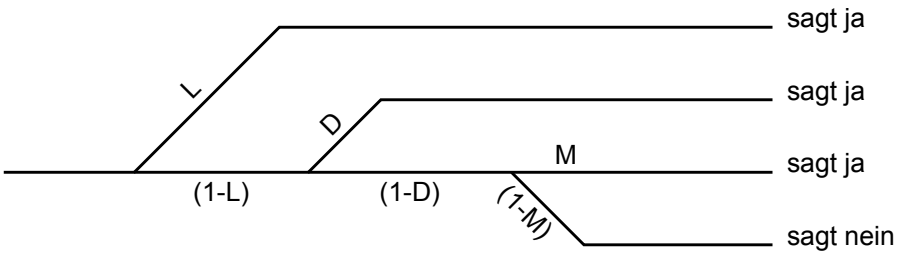
Der Versuch könnte folgendermaßen ablaufen: Es gibt eine komplexe Maschine, die von einem Arbeiter gesteuert werden soll, der kein Wissen über den inneren Aufbau der Maschine besitzt. Daraufhin wird die Maschine von ihrem Ingenieur, der sowohl Wissen über den inneren Aufbau und Ablauf als auch Wissen über die Bedienung besitzt, gesteuert. Die Messung der Gehirnaktivitäten beider Probanden wird sich daraufhin darin unterscheiden, dass der Ingenieur zusätzlich zu den Aktivitäten des Arbeiters Gedächtnisabrufe über innere Zustände der Maschine ausübt.

Multinominale Modellierung: Ziel dieses Verfahrens ist die Messung von Wahrscheinlichkeiten nicht beobachtbarer mentaler Prozesse mithilfe von mehreren beobachtbaren Prozessen. Zum einfacheren Verständnis folgt ein Beispiel [1]:

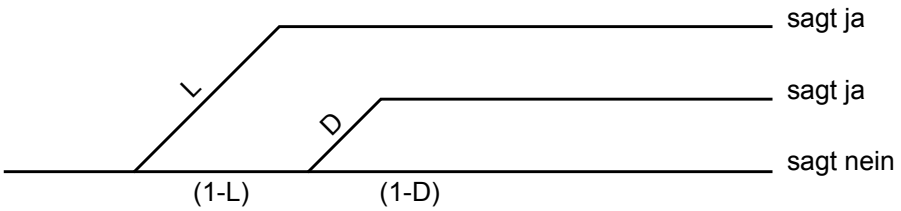
Todd fragt Betty, ob sie mit ihm zum Abschlussball geht. Betty kann nun „Ja“ oder „Nein“ sagen. Ihre Entscheidung hängt von mehreren Faktoren ab. Betty sagt „Ja“, wenn sie Todd mag, sie verzweifelt ist, weil sie bisher noch kein anderer gefragt hat, oder weil sie Mitleid mit ihm hat. In allen anderen Fällen sagt sie „Nein“. Die Möglichkeiten sind in Abbildung 2.12a in Form eines „multinomialen Entscheidungsbaums“ abgebildet.



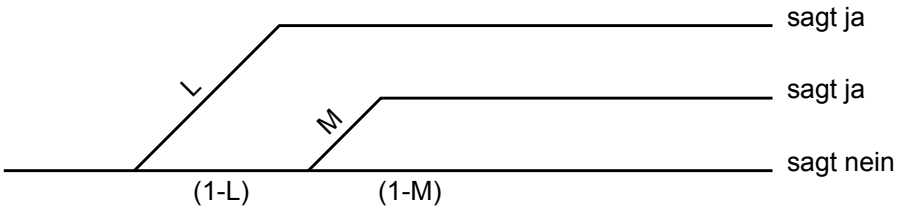
(a) Bettys Möglichkeiten zu entscheiden, ob sie mit Todd zum Abschlussball geht.



(b) Fall 1: Die Wahrscheinlichkeiten für alle Möglichkeiten, die Betty hat (L = „Betty mag Todd“, D = „Betty ist verzweifelt, weil noch kein anderer gefragt hat, ob sie mit ihm auf den Abschlussball geht“, M = „Betty hat Mitleid mit Todd“).



(c) Fall 2: Die Wahrscheinlichkeiten für alle Möglichkeiten, die Betty hat, wenn sie kein Mitleid mit Todd zu haben braucht, weil dieser auch ein anderes Mädchen fragen kann.



(d) Fall 3: Die Wahrscheinlichkeiten für alle Möglichkeiten die Betty hat, wenn sie nicht verzweifelt zu sein braucht, weil sie schon von anderen Jungs gefragt wurde.

Abbildung 2.12: Multinomiale Modellierung – Entscheidungsbäume für die Möglichkeiten für Betty, mit Todd zum Abschlussball zu gehen.

Die Aufgabe ist nun, herauszufinden, mit welcher Wahrscheinlichkeit Betty Todd mag (L) bzw. nicht mag ($1-L$), verzweifelt ist (D) bzw. nicht verzweifelt ist ($1-D$) und Mitleid (M) bzw. kein Mitleid ($1-M$) hat (Abbildung 2.12b).

Angenommen Betty weiß, dass Todd auch andere Mädchen fragen kann und eines dieser sicher zusagen wird. In diesem Fall braucht Betty kein Mitleid haben, wodurch sich der Baum zu dem in Abbildung 2.12c ändert.

Der dritte Fall geht davon aus, dass Betty mit Sicherheit noch von anderen Jungs gefragt werden wird, wodurch der Verzweiflungsast wegfällt (Abbildung 2.12d).

Wenn nun für jeden dieser Fälle mehrere Betty-Todd-Paare befragt werden, könnten z. B. folgende Ergebnisse auftreten:

- Fall 1 – Betty sagt „Ja“: 65%
- Fall 2 – Betty sagt „Ja“: 40%
- Fall 3 – Betty sagt „Ja“: 55%

Mithilfe der Maximum-Likelihood-Methode [6] lässt sich das folgende Gleichungssystem lösen:

$$0.65 = L + (1-L)D + (1-L)(1-D)M \tag{2.2}$$
$$0.4 = L + (1-L)D \tag{2.3}$$
$$0.55 = L + (1-L)M \tag{2.4}$$
$$\tag{2.5}$$

Daraus ergeben sich nun Werte für D, L und M, woraus ein kognitives Modell erstellt werden kann.

Blickbewegungsmessung: Gerade im Bereich der Visualisierung ist die Methode der Blickbewegungsmessung unausweichlich. Mithilfe spezieller Messgeräte, sogenannter „Eye-Tracker“, können Augenbewegungen gemessen und somit Rückschlüsse darauf gezogen werden, wie und in welcher Reihenfolge der Proband eine bestimmte Aufgabe löst.

Nach der Extrahierung der (Sub-)Prozesse folgt die Implementierung des Modells. Damit bei der Entwicklung nicht jedes Mal bei null angefangen werden muss, bedient man sich für gewöhnlich kognitiver Frameworks, welche bereits viele verifizierte Standardmodelle mitliefern. In folgendem Abschnitt werden einige Frameworks vorgestellt und miteinander verglichen.

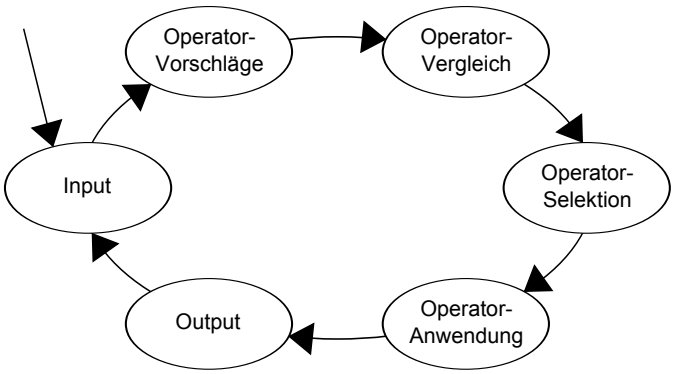


Abbildung 2.13: Zyklus in Soar – Soar durchläuft kontinuierlich einen Zyklus, in dem Produktionsregeln zuerst gefiltert und schließlich eine ausgewählte angewandt wird.

2.4.4 Existierende Simulationsumgebungen

Um für die Implementierung des Modells auf möglichst viele bereits verifizierte Modelle zurückgreifen zu können, bedient man sich häufig sogenannter „Kognitionsframeworks“. Diese beschreiben allgemein mentale Prozesse des Gehirns, wie verschiedene Gedächtnisarten und Problemlösungsmechanismen. Oft bieten diese Frameworks auch spezialisierte Komponenten an, die beispielsweise Lösungsstrategien für die visuelle Wahrnehmung oder für motorische Fähigkeiten bereitstellen. Dieser Abschnitt betrachtet die beiden Kognitionsframeworks Soar und ACT-R. Daraufhin werden die Frameworks miteinander verglichen.

Soar

Soar wurde 1983 von Allen Newell, John Laird und Paul Rosenbloom an der Carnegie Mellon University entwickelt. Es arbeitet ausschließlich auf symbolischer Ebene mithilfe von Produktionsregeln. Eine Produktionsregel ist einer WENN-DANN-Anweisung nachempfunden. Der WENN-Teil besteht dabei aus einer oder mehreren Bedingungen, während der DANN-Teil eine oder mehrere Aktionen beinhaltet. Sowohl die Bedingungen als auch die Aktionen werden dabei ausschließlich durch Tripel beschrieben, welche in ein semantisches Netz überführt werden können (siehe Abschnitt 2.3.2). Mithilfe der Produktionsregeln wird das prozedurale Wissen (= Wissen, wie Dinge zu tun sind) repräsentiert. Der Arbeitsspeicher in Soar enthält das deklarative Wissen (= Faktenwissen), auf das die Produktionsregeln angewandt werden. Die Aktionen einer Regel werden genau dann ausführbar, wenn alle Tripel der Bedingung zum aktuellen Zeitpunkt im Arbeitsspeicher vorhanden sind. Können mehrere Produktionsregeln „feuern“, wählt Soar

die mit der größten Relevanz aus und führt deren Aktionsteil aus. Die Relevanz berechnet Soar dabei durch Ergebnisse von vorherigen Durchläufen oder zufällig – je nachdem, ob eine oder mehrere Regeln mit höchster Relevanz als Kandidat ausstehen. Dies entspricht einem Lernvorgang in Soar. Die Aktionen im Aktionsteil einer Produktionsregel können das semantische Netz im Arbeitsspeicher verändern. Nach der Ausführung kommen andere Produktionsregeln als Kandidat infrage und der Zyklus beginnt von vorne (Abbildung 2.13). Über den genauen Aufbau und die Funktionsweise gibt [27] Aufschluss.

ACT-R

ACT-R ist ebenfalls an der Carnegie Mellon University entstanden. John R. Anderson begann 1983 mit der Entwicklung und beteiligt sich noch heute an der Weiterentwicklung des Modells. Ähnlich wie in Soar, besitzt auch ACT-R Produktionsregeln, die für die Steuerung des Modells verantwortlich sind. Die Regeln verfügen über einen Bedingungs- und einen Aktionsteil, wobei beide auf sogenannten „Chunks“ arbeiten. Ein Chunk beschreibt eine Wissenseinheit mit beliebig vielen Eigenschaften. Oft sind diese Wissenseinheiten Gegenstände aus der realen Umwelt, wie z. B. ein Auto. Die Eigenschaften können einfache Werte wie „Farbe = rot“ oder „Gewicht = 1200 kg“ besitzen. Andererseits kann die Eigenschaft einer Wissenseinheit auch auf eine andere Wissenseinheit verweisen. Ein Auto besitzt beispielsweise einen Motor, der selbst weitere Eigenschaften hat. Durch die Verknüpfungen untereinander bildet das so aufgebaute Wissen ein semantisches Netz (siehe Abschnitt 2.3.2).

ACT-R besteht aus den folgenden drei Hauptkomponenten, welche in Abbildung 2.14 dargestellt sind:

Module: Module beschreiben einzelne Bereiche bzw. Aufgaben des menschlichen Gehirns. So gibt es beispielsweise Module für die visuelle Wahrnehmung, für die auditive Wahrnehmung, für das deklarative und das prozedurale Wissen.

Buffer: Die Kommunikation zwischen den einzelnen Modulen findet durch sogenannte „Buffer“ statt. Jedes Modul besitzt mindestens einen Buffer, über welchen es Eingaben annimmt und Ausgaben produziert.

Mustererkennung: Die Mustererkennung prüft den Produktionsspeicher auf gültige Produktionsregeln und feuert diese bei entsprechender Übereinstimmung.

Produktionsregeln besitzen in ihrem Bedingungsteil beliebig viele Buffer, wodurch sie definieren, dass sich bestimmte Chunks in den entsprechenden Modulen befinden müssen, damit die jeweilige Produktionsregel von der Mustererkennung gefeuert werden kann. In ihrem Aktionsteil besitzen Produktionsregeln ebenfalls Buffer, über die Chunks entfernt, verändert, hinzugefügt oder angefragt werden können.

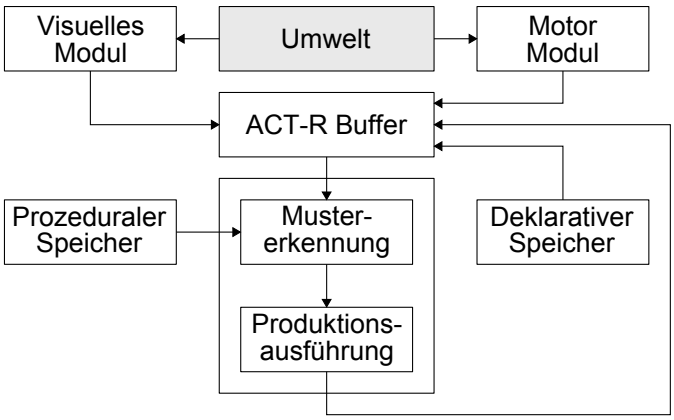


Abbildung 2.14: Architektur von ACT-R – Einzelne Module (in der Abbildung: „Visuelles Modul“, „Motor Modul“, „Prozeduraler Speicher“ und „Deklarativer Speicher“) stellen ihre Ausgaben über Buffer für die Mustererkennung bereit. Diese feuert einzelne Produktionsregeln, welche über die Buffer wiederum Daten in den Modulen manipulieren oder abfragen. [2]

Für diese Arbeit ist das visuelle Modul besonders von Interesse. Das Modul unterscheidet zwischen dem sogenannten „Wo-“ und „Was-Raum“. Ein sichtbares Objekt muss immer in beiden Räumen definiert werden. Der Wo-Raum beinhaltet Informationen über die Position und Größe dieses Objekts, wohingegen im Was-Raum objektrelevante Eigenschaften, wie Farbe oder Text für das Objekt hinterlegt sind. Der Was-Raum ist in ACT-R insofern eingeschränkt, dass dort zu den Ausmaßen eines Objekts nur Höhe und Breite verwendet werden. Dies führt dazu, dass vor allem diagonal liegende Objekte einen sehr viel größeren Bildbereich einnehmen, als sie tatsächlich benötigen. Zusätzlich liegt in dem visuellen Modul die Beschränkung, dass nur zweidimensionale Objekte verarbeitet werden können.

ACT-R modelliert sowohl symbolisches als auch subsymbolisches Verhalten. So ist über bestimmte Parameter einstellbar, dass ACT-R für jede Produktionsregel zusätzliche Eigenschaften wie die Nützlichkeit oder die Kosten bzw. den Aufwand für das Abfeuern der Regeln abspeichert. Die Werte kann ACT-R während der Simulation anhand unterschiedlicher Metriken, wie die Häufigkeit mit der eine Produktionsregel bisher gefeuert hat, dynamisch anpassen, wodurch ein Lerneffekt entsteht. Ein weiterer subsymbolischer Charakter entsteht in ACT-R durch die Abfrage der Chunks aus dem deklarativen Gedächtnis mittels Spreading Activation (siehe Abschnitt 2.3.2) und zusätzlichen durch ACT-R vorgegebenen stochastischen Schwankungen. Chunks, die häufiger abgefragt werden als andere, erhalten eine höhere Aktivierung, wodurch sie schneller und bevorzugt gefunden werden können.

Eine ausführliche Beschreibung von ACT-R sowie Dokumentationen, Beispiele und Referenzen finden sich auf der ACT-R-Webseite: `http://act-r.psy.cmu.edu`.

Fazit

Die beiden Frameworks unterstützen den Entwickler erheblich bei der Implementierung eines kognitiven Modells. Sowohl Soar als auch ACT-R sind Produktionssysteme, was grundsätzlich auf eine symbolische Verarbeitung hindeutet. Auch wenn ACT-R meist als Hybrid bezeichnet wird, ist die subsymbolische Ebene nach Ansicht des Autors dieser Arbeit dennoch nur minimal umgesetzt. Vor allem die zufällige Entscheidung für das Feuern einer Produktionsregel, wenn mehrere Regeln gleich relevant sind, widerstrebt der in Abschnitt 2.3.1 beschriebenen Definition der subsymbolischen Herangehensweise. Dennoch kommt diese Umsetzung dem menschlichen Gehirn weitaus näher, als die ausschließliche Verwendung eines symbolischen Modells, wie es in Soar zu finden ist. Auch die striktere Trennung der einzelnen Module in ACT-R ist positiv hervorzuheben. Dadurch besitzt man eine weitere Ebene, auf welcher effizient gearbeitet werden kann. Im Gegensatz zu Soar ist es mit ACT-R zudem möglich, quantitative Messungen, wie Reaktionszeiten einzelner Abläufe im Gehirn abzurufen.

Im Folgenden wird beispielhaft beschrieben, aus welchen Teilen ein kognitives Modell in ACT-R grundsätzlich besteht.

2.4.5 Implementierung eines Modells

ACT-R wurde in der Programmiersprache Lisp geschrieben. Eine gute Einführung in Common Lisp findet sich in [3]. Listing 2.1 zeigt ein Grundgerüst, anhand dessen der Aufbau eines Modells in ACT-R beschrieben werden soll.

Listing 2.1: Grundgerüst eines ACT-R-Modells

```
1  (clear-all)
2
3  (define-model modellname
4    (sgp :parameter-1 wert-1 :parameter-2 wert-2 ...)
5
6    (chunk-type chunk-type-name slot-name-1 slot-name-2 ...)
7    (chunk-type chunk-type-name slot-name-1 slot-name-2 ...)
8    ...
9
10   (add-dm
11     (chunk-name isa chunk-type-name slot-1-wert slot-2-wert ...)
```

```
12          (chunk-name isa chunk-type-name slot-1-wert slot-2-wert ...)
13          ...
14      )
15
16      (p produktion-name
17          (=|+|?)buffer-name>
18              slot-name slot-wert
19              slot-name slot-wert
20              ...
21          (=|+|?) buffer-name>
22              slot-name slot-wert
23              ...
24          ...
25        ==>
26          (=|+|?) buffer-name>
27              slot-name slot-wert
28              ...
29          ...
30      )
31      (p produktion-name
32        ...
33      )
34      ...
35
36      (goal-focus chunk-name)
37
38      (spp produktion-name :(at|u|reward) wert)
39      (spp produktion-name :(at|u|reward) wert)
40      ...
41  )
```

Das Grundgerüst ist in folgende Teile untergliedert:

Modellname: Mit `define-model` (Zeile 3) wird die Definition des Modells eingeleitet. Der erste Parameter besteht dabei aus einem beliebigen Namen für das zu erstellende Modell. Der zweite Parameter besteht aus dem eigentlichen Modell, welches in den folgenden Punkten näher beschrieben wird.

Modulparameter: Jedes Modul kann seine eigenen Parameter besitzen. Über diese ist das entsprechende Modul konfigurierbar. Das deklarative Modul hat unter anderem einen Parameter für die Stärke des zufälligen Rauschens beim Abfragen deklarativen Wissens (`:ans`) und einige Parameter, die in die Berechnung der Abfragezeit

von Wissen einfließen (`:le`, `:lf`). Jeder Parameter hat einen Standardwert, welcher mit der Funktion `get-parameter-default-value` abgerufen werden kann. Eine Liste mit den Parametern und aktuellen Werten aller Module kann mit `(sgp)` ausgegeben werden. Mit dieser Funktion werden außerdem die entsprechenden Parameter gesetzt (Zeile 4).

Chunk-Types: Mithilfe der Funktion `chunk-type` werden die Chunk-Typen für das Modell definiert (Zeilen 6 bis 8). Der erste Parameter gibt einen eindeutigen Namen für den Typ an. Alle weiteren Parameter definieren beliebig viele Slots, die ein Chunk dieses Typs besitzt.

Deklaratives Gedächtnis: Das deklarative Gedächtnis des Modells kann bereits von Anfang an mit Chunks befüllt werden. Dazu werden mit der Funktion `add-dm` beliebig viele Tupel übergeben, die jeweils einen Chunk repräsentieren (Zeilen 10 bis 14). Der erste Parameter jedes Tupels definiert dabei einen Namen. Über den zweiten und dritten Parameter wird festgelegt, von welchem Chunk-Typ der Chunk abstammt. Mit den folgenden Parametern können den einzelnen Slots Startwerte übergeben werden, wobei diese auch leer bleiben können.

Produktionsregeln: Produktionsregeln sind ähnlich zu WENN-DANN-Anweisungen in üblichen Programmiersprachen (C, C++, C#, Java, ...). Der WENN-Teil besteht aus Abfragen der Modul-Buffer (Zeilen 16 bis 34). Die Funktion `p` erzeugt eine neue Produktionsregel, die den ersten Parameter als Namen hat. Danach folgen Buffer, bei denen jeweils die Inhalte der einzelnen Slots mit den Bedingungen übereinstimmen müssen. Ist die Überprüfung erfolgreich, kann die Produktionsregel feuern und der DANN-Teil ausgeführt werden. Darin können wiederum die Inhalte der Buffer verändert werden.

Ziel: Damit die Simulation des Modells gestartet werden kann, muss ein Ziel definiert sein. Dieses besteht aus einem Chunk des deklarativen Moduls und wird an die Funktion `goal-focus` übergeben (Zeile 36).

Produktionsparameter: Zusätzlich zu den Modulparametern können mit der Funktion `spp` für jede Produktionsregel jeweils drei Parameter festgelegt werden (Zeilen 38 bis 40):

- `at`: Hiermit kann die Zeit festgelegt werden, die die übergebene Produktionsregel zum Feuern benötigt. Wird nichts Spezielles angegeben, wird der Standardwert im Modulparameter `dat` verwendet.
- `u`: Hiermit kann der Nutzen der übergebenen Produktionsregel festgelegt werden. Dies ist dann sinnvoll, wenn der automatische Lernmechanismus deaktiviert ist. Der Standardwert hierfür ist im Modulparameter `iu` festgelegt.
- `reward`: Dieser Wert gibt eine Belohnung an, die die übergebene Produktionsregel dann angerechnet bekommt, wenn sie gefeuert hat. Dadurch steigt im

automatischen Lernmodus der Parameter u automatisch an, wodurch diese Regel mit einer höheren Wahrscheinlichkeit erneut feuert.

2.4.6 Validierung der Modelle

Nach der Implementierung des Modells muss dieses auf Gültigkeit überprüft werden. Dazu werden, wie bei den empirischen Untersuchungen in Phase 2, Studien mit Probanden durchgeführt. Ziel ist es, die vom Modell vorhergesagten mit den empirisch ermittelten Daten zu vergleichen, um daraufhin die Aussage zu erhalten, wie gut das Modell die entsprechende Aufgabe simuliert.

Im Folgenden werden nun einige Verfahren vorgestellt, welche empirische Messungen erlauben und ein Maß für die Gültigkeit des Modells liefern.

Operatorsequenzen: Mithilfe von Operatorsequenzen lässt sich abschätzen, wie gut die Simulation mit dem Messergebnis einer realen Situation übereinstimmt. Als Operator ist in diesem Fall eine atomare Aktion gemeint, die der Simulator bzw. der Proband ausführt. Mehrere dieser Operatoren hintereinander (= Operatorsequenz) bilden eine größere Aktion. Ist es nun möglich, die atomaren Aktionen beim Probanden zu messen (z. B. mit den in Abschnitt 2.4.3 vorgestellten Verfahren), können diese mit den vom modellierten System ausgeführten Aktionen verglichen werden. Die Anzahl der Übereinstimmungen kann dabei als Maß der Korrektheit angesehen werden. Es kann vorkommen, dass der Proband einzelne Aktionen, die das Modell vornimmt, nicht ausführt. In diesem Fall müssen diese „Lücken“ so gefüllt werden, dass das größtmögliche Matching auftritt.

Besteht eine Sequenz, die das Modell abhandelt, beispielsweise aus den Operatoren $A1$, $A2$, $A3$ und $A4$ und wird bei einem Probanden festgestellt, dass dieser den Operator $A2$ nicht abhandelt, sondern von A1 direkt zu $A3$ übergeht, so ergibt sich folgendes Bild:

Modell: $A1 \rightarrow A2 \rightarrow A3 \rightarrow A4$
Proband: $A1 \rightarrow A3 \rightarrow A4$

In diesem Fall wird $A2$ beim Probanden durch einen Platzhalter (z. B. X) ersetzt:

Modell: $A1 \rightarrow A2 \rightarrow A3 \rightarrow A4$
Proband: $A1 \rightarrow X \rightarrow A3 \rightarrow A4$

In dem Beispiel ergibt sich dadurch eine Übereinstimmung von 75%. Card, Moran und Newell haben für dieses Verfahren einen Algorithmus entwickelt [12].

Operator-Latenz-Zeiten: Kann die Zeit zwischen zwei Operatoren bei einem Probanden gemessen werden, kann diese mit den vom Modell vorhergesagten Zeiten verglichen werden. Auch hier lässt sich der prozentuale Unterschied als Maß der Genauigkeit

des Modells verwenden. In [25] wird dieses Verfahren am Beispiel des Spiels „Türme von Hanoi“ angewandt.

Zustandsübergänge: Mithilfe des Modells lassen sich in jedem Zustand die möglichen Folgezustände identifizieren. Wählt ein Proband zu einem bestimmten Zeitpunkt einen Zustand, der laut Modell nicht möglich ist, muss dies ein Indikator für einen schlechten Lösungsweg sein und der Proband sollte irgendwann in einen Zustand gelangen, der auf dem Lösungsweg des Modells liegt. Gelangt er auf einem anderen Weg zum Ziel, muss das Modell ab diesem Zustand korrigiert werden.

Transfereffekte: Transfereffekte treten beim Erlernen neuer Aufgaben auf. Die Lernzeit ist immer abhängig von der Anzahl der Regeln, die hinzugelernt werden müssen [33]. Sind zwei Aufgaben ähnlich zueinander und wird die kleinere von beiden bereits beherrscht, wird nur noch die Zeit für die zusätzlichen Regeln zum Lernen der großen Aufgabe benötigt (common-rules-Hypothese [34]). Mithilfe dieser Annahmen lassen sich besonders gut Vorhersagen nachweisen, aus deren einzelnen Prozessen eine komplexe Aufgabe besteht.

Die perfekte Übereinstimmung der simulierten Daten mit den Probanden ist bei der Überprüfung selten der Fall. Sind die Simulationsergebnisse nicht befriedigend, sollte die erste Phase der Modellierung wiederholt werden und die Erfahrungen der Analyse in die Optimierung einfließen. Das Modell wird dadurch iterativ verbessert.

3 Existierende Arbeiten

Dieses Kapitel gibt Einblick in eine Auswahl wissenschaftlicher Arbeiten, die sich mit den Themenbereichen Kognition, Visualisierung und Modellierung kognitiver Prozesse auseinandersetzen. Es wird kurz beschrieben, was die jeweiligen Ziele der Arbeiten waren, wie sie umgesetzt wurden und inwiefern sich die vorliegende Arbeit von den einzelnen Artikeln abgrenzt.

Im Einzelnen werden nun die drei operator-basierten Modelle „DVM“ (Abschnitt 3.1.2), „GOMS“ [13] und „UCIE“ [29] [28] vorgestellt. Im Anschluss wird im Abschnitt 3.2 ein Modell vorgestellt, das eine semantische Beschreibung von Graphen ermöglichen soll und Ansätze zum Lesen von Graphen beinhaltet. Im letzten Abschnitt dieses Kapitels wird CogTool, ein Tool zum Simulieren von Interaktionszeiten im Bereich der Mensch-Rechner-Interaktion, betrachtet (Abschnitt 3.3).

3.1 Operator-Basierte Modelle

Operator-Basierte Modelle bauen immer auf dem gleichen Grundprinzip auf. Diesem liegt zugrunde, dass die Augenbewegungen beim Lesen einer Visualisierung fest vorgegeben sind und dadurch immer den idealen Scan Path (der kürzeste Weg, den die Augen zurücklegen müssen, um alle zum Lesen einer Visualisierung notwendigen Elemente mindestens ein Mal betrachtet zu haben) eines Experten widerspiegeln. Im Folgenden werden nun drei Modelle vorgestellt, die nach diesem Prinzip arbeiten. Die Beschreibungen sind jeweils in die drei Abschnitte „Aufbau und Funktionsweise“, „Vorgegebene Werte“ und „Bewertung und Abgrenzung“ aufgeteilt.

3.1.1 GOMS

Das GOMS-Modell wurde 1983 von S. Card, T. P. Moran und A. Newell vorgestellt [13]. Es bildet die Grundlage für viele weitere Modelle, wie das Keystroke-Level Model (KLM) oder das UCIE-Model (siehe Abschnitt 3.1.3).

Aufbau und Funktionsweise

GOMS leitet sich aus „**G**oals, **O**perators, **M**ethods and **S**election rules“ ab, wobei die einzelnen Teile folgende Bedeutungen haben:

- **Goals** definieren Ziele, welche selbst wiederum in Unterziele unterteilt sein können.
- **Operators** definieren alle Aktionen, die in einem bestimmten Zustand möglich sind.
- **Methods** sind Gruppen mehrerer, hintereinander ausgeführter Operatoren, die zusammengesetzt zum Erreichen eines Ziels führen.
- **Selection Rules** sind Regeln, die bestimmen, welche Methode wann ausgeführt wird.

Die Erstellung eines speziellen Modells mit dem GOMS-Modell verhält sich ähnlich der Implementierung eines Modells mit ACT-R: Das Hauptziel wird in möglichst kleine Unterziele aufgeteilt. Durch Aktionen werden Übergänge zwischen den Unterzielen definiert.

Vorgegebene Werte

Da das Modell auf sehr abstrakter Ebene definiert ist, gibt es keine Unterteilung in spezielle Operationen, wodurch auch keine festen Zahlenwerte im Modell festgelegt sind.

Bewertung und Abgrenzung

GOMS bildet die Grundlage für weitere kognitive Modelle, wie KLM [13] und CogTool [23]). Da GOMS jedoch auf sehr abstrakter Ebene spezifiziert ist und nur grobe Konzepte zum Modellieren kognitiver Prozesse vorgibt, wird sich diese Arbeit vielmehr auf die beiden folgenden GOMS-basierten Modelle DVM und UCIE stützen.

3.1.2 DVM

Das Diagram Viewing Model (DVM) wurde von Raschke et al. entwickelt. Es orientiert sich an dem von Card, Moran und Newell vorgestellten Keystroke Level Model (KLM) [13]. Durch Hintereinanderausführung elementarer Operationen, wie Mausbewegungen oder Tastatureingaben beim Interagieren mit grafischen Benutzeroberflächen, schätzt es die für eine Aufgabe benötigte Zeit ab. Das DVM wurde dafür entwickelt, Zeiten zum Ablesen von Daten aus Koordinatensystemen modellieren zu können. Richtlinien zum Design von Visualisierungen können zwar bei der Gestaltung eines neuen Diagrammtyps verwendet werden, jedoch lässt sich die Effizienz dieses Diagramms nur durch aufwendige

Benutzerstudien quantisieren. Mithilfe des DVM können Zeitmessungen von verschiedenen Diagrammtypen mit unterschiedlichen Aufgaben schnell berechnet und direkt miteinander verglichen werden.

Aufbau und Funktionsweise

Da das DVM nicht für den HCI-, sondern für den Visualisierungsbereich entwickelt wurde, sind die elementaren Operationen ferner kognitiver als motorischer Art. Folgende Operationen werden im DVM definiert:

- $\mathcal{P}$: Lesen der beiden Koordinatenwerte eines Datenpunktes
- $\mathcal{A}$: Lesen des Wertes eines Punktes auf einer eindimensionalen Koordinatenachse
- $\mathcal{V}$: Erkennen eines Punktes mit einer bestimmten Eigenschaft aus vielen Punkten (z. B. „Welcher Punkt hat den höchsten Wert?“)
- $\mathcal{F}$: Fokussieren einer bestimmten Position
- $\mathcal{M}$: Mentale Aktivität, die während der Wahrnehmung der Visualisierung abläuft

Soll die Aufgabe sein, in einem Diagrammtyp die Koordinaten eines Punktes abzulesen, muss der Pfad, den der Betrachter mit seinen Augen verfolgen soll, festgelegt werden. Daraufhin kann man durch Aufsummierung der entsprechenden Operationen, die diesen Pfad repräsentieren, die Gesamtzeit zur Lösung der Aufgabe berechnen.

Vorgegebene Werte

Eine Vorstudie mit vier Probanden ergab, dass jeder Proband beim Durchführen einer visuellen Aufgabe näherungsweise die gleichen Bereiche in einer Visualisierung fokussierte. Auf Basis dieser Ergebnisse wurde als Grundlage für das DVM die Hypothese formuliert, dass ein idealer Scan Path definiert werden kann. Dadurch konnten für die jeweilige Aufgabe die Operatoren in einer bestimmten Reihenfolge hintereinander angegeben und daraus die Gesamtzeit, die für den kompletten Vorgang benötigt wird, ausgerechnet werden.

Die Zeiten t für die Operationen $\mathcal{F}$ und $\mathcal{M}$ wurden mit Werten aus vorhandenen Modellen als fix angenommen ($t_{\mathcal{F}} = 600ms$, $t_{\mathcal{M}} = 1{,}35s$). $t_{\mathcal{P}}$, $t_{\mathcal{A}}$ und $t_{\mathcal{V}}$ wurden mithilfe einer weiteren Studie mit 27 Probanden und unterschiedlichen Diagrammen sowie verschiedenen visuellen Aufgaben empirisch ermittelt (Tabelle 3.1). Durch eine weitere Studie wurde das DVM validiert. Abbildung 3.1 zeigt die vom DVM vorhergesagten Zeiten im Verhältnis zu den Quartilen, der in der Validierungsstudie gemessenen Werte, bei acht verschiedenen Aufgaben.

Operator	**Zeit [s]**	**Operator**	**Zeit [s]**
$\mathcal{A}_{hns}$	3,6	$\mathcal{P}_{ns}(n)$	$4{,}4n + 3{,}0$
$\mathcal{A}_{hws}$	2,4	$\mathcal{P}_{ws}(n)$	$2{,}8n + 3{,}3$
$\mathcal{A}_{vns}$	3,2	$\mathcal{P}_{g}(n)$	$2{,}7n + 1{,}5$
$\mathcal{A}_{vws}$	1,9	$\mathcal{V}[\text{größtes von n}]_{ns}$	$-0{,}1n + 3{,}3$
$\mathcal{V}[\text{1 von n}]_{hns}$	3,4	$\mathcal{V}[\text{größtes von n}]_{ws}$	$-0{,}05n + 2{,}0$
$\mathcal{V}[\text{1 von n}]_{hws}$	2,2	$\mathcal{V}[\text{größtes von n}]_{wg}$	$-0{,}1n + 2{,}3$
$\mathcal{V}[\text{1 von n}]_{vns}$	3,8		
$\mathcal{V}[\text{1 von n}]_{vws}$	2,8		

Tabelle 3.1: Empirisch ermittelte Zeiten der einzelnen Operatoren beim DVM – Die genaue Bedeutung der einzelnen Operatoren mit ihren Indizes können dem Originaldokument entnommen werden.

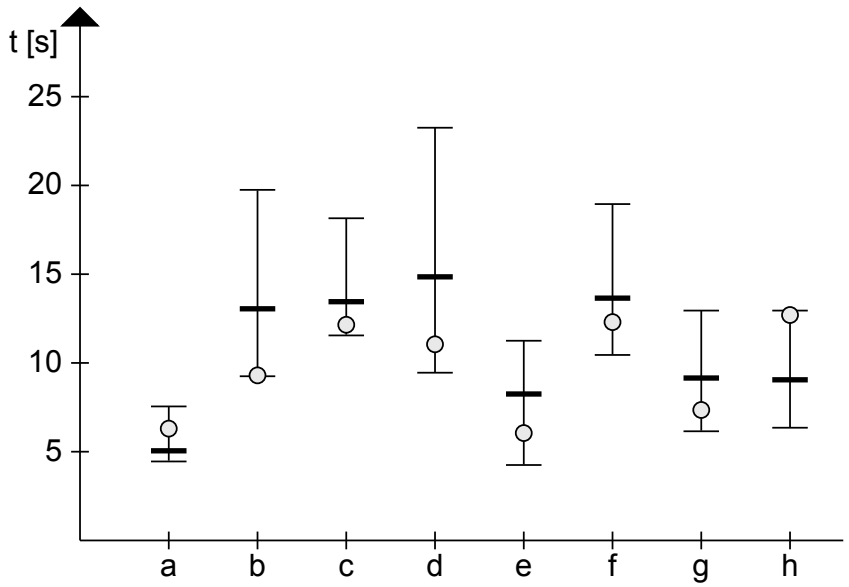


Abbildung 3.1: Vergleich der gemessenen mit den vorhergesagten Zeiten beim DVM – Die dicken Linien geben die Durchschnittswerte der in der Studie gemessenen Werte an. Die dünnen Linien zeigen die 1. und 3. Quartile. Die vom DVM vorhergesagten Zeiten sind als Kreise dargestellt. a–h repräsentieren verschiedene Aufgabentypen.

Bewertung und Abgrenzung

Der Vergleich, der gemessenen mit den vorhergesagten Zeiten zeigt, dass das Modell bei allen Aufgabentypen korrekte Vorhersagen innerhalb der 1. und 3. Quartile liefert. Allerdings lässt es einige (notwendige) Faktoren außen vor, sodass das Modell nur für bestimmte Diagramme mit hohen Einschränkungen verwendet werden kann. Die Größe des Diagramms, die Anzahl der Zahlenstriche pro Einheit, die Abstände der Datenpunkte von den Zahlenstrichen, sowie Farbeigenschaften einzelner Punkte werden nicht berücksichtigt. Ein Lerneffekt, der beim Lesen vieler Diagramme hintereinander auftreten kann, wird ebenfalls nicht modelliert.

Diese Arbeit verfolgt die Grundidee des DVM, sich an dem KLM zu orientieren. Das Aufteilen des Gesamtprozesses und das Aufaddieren der einzelnen Operatorzeiten soll als Vorlage für das zu entwickelnde Modell dienen.

3.1.3 UCIE

UCIE wurde von G. L. Lohse in den beiden Artikeln „A cognitive Model for the Perception and Understanding of Graphs“ [29] und „A Cognitive Model for Understanding Graphical Perception“ [28] 1991 bzw. 1993 vorgestellt. Die Abkürzung „UCIE“ steht für „Understanding Cognitive Information Engineering“ und deutet auf eine allgemeine Theorie zur Verarbeitung kognitiver Prozesse hin. UCIE baut zwar auf dem GOMS-Model (siehe Abschnitt 3.1.1) auf, tatsächlich simuliert es jedoch, ähnlich dem DVM (siehe Abschnitt 3.1.2), lediglich Zeitmessungen für das Ablesen und Vergleichen von Punkten in zweidimensionalen Säulen-, Punkt- und Liniendiagrammen.

Aufbau und Funktionsweise

Damit das Lesen eines Diagramms mit UCIE simuliert werden kann, muss das Diagramm zunächst in einzelne grafische Objekte unterteilt und semantisch annotiert werden. Jeder Koordinatenachse, jedem Datenpunkt und jeder Legende (um nur einige Elemente zu nennen) muss man die in Tabelle 3.2 dargestellten Eigenschaften zuweisen. Das Modell des Diagramms gelangt nun als Eingabe in das UCIE-Modell. Zusätzlich verlangt das UCIE-Modell eine Anfrage, die definiert, was mit welchen Parametern simuliert werden soll. Die Anfrage liegt in Form von Text vor, den das Modell in seine einzelnen Bestandteile zerlegt. Damit kann UCIE herausfinden, um welchen Diagrammtyp und welchen Aufgabentyp (Ablesen eines Wertes, Vergleichen zweier Werte, Abschätzen des Trends mehrerer Werte) es sich handelt. Mithilfe des Graphen-Schemas von S. Pinker (siehe Abschnitt 3.2) ordnet das UCIE-Modell die einzelnen Begriffe der Anfrage den Bezeichnern im Diagramm zu. Daraufhin kann UCIE Simulationsziele definieren und die Augenfixationen ausführen.

Eigenschaft	Gespeicherte Information
Objekt-Nummer	Eindeutige Id: $i = 1$ bis n Objekte
Pixel-Position	(X_1, Y_1, X_2, Y_2) Ecken des eingrenzenden Rechtecks
Klasse	Eine von zwölf Klassen (title, xlabel, yscale, data, ...)
Typ	Text, Linie oder Symbol
Farbe	Eine von 16 Farben
Textur	Eine von neun Texturen
Ordnung	Reihenfolge
Beschriftung	Text im Bild

Tabelle 3.2: Eigenschaften eines Objekts im UCIE-Modell – Jedem Objekt eines Diagramms im UCIE-Modell muss eine in der Spalte „Eigenschaften“ aufgeführte Eigenschaft zugewiesen werden [28].

Die Dauer einer Fixation auf ein grafisches Objekt berechnet das UCIE-Modell anhand folgender Formel:

$$\sum_{i=1}^{n} \tau_{ij} \left(((1 - \text{Diff}_i) + \text{Ovlap}_i + \text{Sim}_i) \left(1 - \frac{1}{9} \text{Prox}_i \right) \right) \tag{3.1}$$

Dabei gilt:

- n: Anzahl der peripheren Objekte, die außer dem zu fixierenden Objekt im aktuellen Blickfeld liegen
- $\tau_{ij}[ms]$: Zeit, die benötigt wird, um zwei grafische Objekte im Arbeitsgedächtnis miteinander zu vergleichen
- Diff_i: Relativer Farbunterschied $(0 - 1)$ bezüglich der Lichtfrequenz zwischen dem zu fixierenden Objekt und dem peripheren Objekt i
- Overlap_i: Relative Pixelüberschneidung $(0 - 1)$ des zu fixierenden Objekts mit dem peripheren Objekt i
- Sim_i: Relative Ähnlichkeit $(0 - 1)$ der geometrischen Form zwischen dem zu fixierenden Objekt und dem peripheren Objekt i
- $\text{Prox}_i[mm]$: Abstand zwischen dem zu fixierenden Objekt und dem peripheren Objekt i

Die genauen Bedeutungen der einzelnen Variablen können der Originalquelle [28] entnommen werden. Das UCIE-Modell bedient sich zusätzlich zu den Fixationszeiten einiger Zeitkonstanten, die im Abschnitt „Vorgegebene Werte“ zu finden sind. Die Gesamtzeit für das Lösen einer Aufgabe in einem Diagramm berechnet sich aus der Summe aller

Fixations- und Augenbewegungszeiten sowie den Zeiten, die für die kognitiven Prozesse beim Lesen einer Visualisierung benötigt werden.

Die Reihenfolge und Anzahl der Augenfixationen richtet sich im UCIE-Modell nicht nur nach den Graphen-Schemata. Das UCIE-Modell modelliert ein Arbeitsgedächtnis und einen visuellen Bildspeicher, welche jeweils auf eine Maximalkapazität von drei (Arbeitsgedächtnis) bzw. neun (visuelles Gedächtnis) Objekten begrenzt sind. Hat eine Legende in einem Diagramm beispielsweise mehr als drei Elemente, kann sich das Modell nur an die drei zuletzt gelesenen Elemente „erinnern“ und muss die anderen zu einem späteren Zeitpunkt „nachlesen“. Dadurch entstehen zusätzliche Augenfixationen, welche die Gesamtzeit beeinflussen.

Vorgegebene Werte

Das UCIE-Model definiert selbst keine neuen Zeitkonstanten. Stattdessen verwendet es bekannte Werte aus der Literatur, wie sie Tabelle 3.3 entnommen werden können.

Bewertung und Abgrenzung

Das UCIE-Modell berücksichtigt einige Faktoren bei der Lösung von Aufgaben. So werden bei Fixationen beispielsweise nicht nur die fokussierten Punkte, sondern auch die entsprechenden Umgebungen mit der Anzahl der Objekte, deren Farben und Formen berücksichtigt. Die Zeitwerte aus Tabelle 3.3 fließen ebenfalls in die Berechnungen ein. Durch das Graphen-Schema von Pinker [32] bietet das UCIE-Modell außerdem eine hohe Flexibilität, was die Verwendung neuer Diagrammarten relativ leicht ermöglicht.

Abbildung 3.2 veranschaulicht das Ergebnis einer Studie mit 28 Probanden, in der verschiedene Diagramme und Diagrammtypen untersucht wurden [29]. Der Vergleich zeigt, dass die vom UCIE-Modell vorhergesagten Bearbeitungszeiten bei den meisten Aufgabentypen zwischen der 1. und 3. Quartile der in der Studie gemessenen Zeiten liegen. Abweichungen zu den Durchschnittswerten befinden sich etwa im Bereich $[0; 1{,}5]s$.

In dieser Arbeit werden einige Elemente des UCIE-Modells verwendet. So sind Zeiten aus Tabelle 3.3 und das Graphen-Schema interessante Ansätze. Das zu entwickelnde Modell wird jedoch auf abstrakterer Ebene arbeiten, sodass nicht nur Diagramme, sondern auch andere Visualisierungen, wie Graphen gelesen werden können.

[1] Die ausführlichen Referenzen finden sich in [32]

Zeit [ms]	Beschreibung	Referenz[1]
372	Eingabe mit einer Taste	Card, Moran, Newell, 1983
1200	Vergleich zweier Einheiten im Gedächtnis	J. R. Olson & G. M. Olson, 1990
1180	Auf einer linearen Skala interpolieren	Boff & Lincoln, 1988
300	Ein Wort mit sechs Buchstaben erkennen	John & Newell, 1990
230	Eine Augenbewegung durchführen	Russo, 1978
70	Einen mentalen Schritt ausführen	J. R. Olson & G. M. Olson, 1990
92	Eine Wahrnehmung beurteilen	Welford, 1973
33	Zwei Zahlen im Arbeitsgedächtnis miteinander vergleichen	Cavanaugh, 1972
38	Zwei Farben im Arbeitsgedächtnis miteinander vergleichen	Cavanaugh, 1972
40	Zwei Buchstaben im Arbeitsgedächtnis miteinander vergleichen	Cavanaugh, 1972
47	Zwei Wörter im Arbeitsgedächtnis miteinander vergleichen	Cavanaugh, 1972
50	Zwei Formen im Arbeitsgedächtnis miteinander vergleichen	Cavanaugh, 1972
68	Zwei Gestalten im Arbeitsgedächtnis miteinander vergleichen	Cavanaugh, 1972
4	Zeit, die für die Augenbewegung von einem Grad benötigt wird	Kosslyn, 1983

Tabelle 3.3: Verwendete Zeitkonstanten des UCIE-Modells – Das Modell nutzt Zeiten, die von unterschiedlichen Autoren ermittelt wurden. [32]

3.2 Theorie über das Verstehen von Diagrammen

In seinem Artikel „A Theory of Graph Comprehension“ [32] von 1993 stellt S. Pinker eine Theorie auf, die beschreibt, wie Diagramme von Menschen gelesen werden. Dabei wird die Simulation zum Lesen eines Diagramms auf sehr abstrakter Ebene betrachtet. So werden beispielsweise keine konkreten Algorithmen und Zeiten definiert. Stattdessen beschreibt Pinker sowohl die Modellierung eines Diagramms als auch dessen Interpretation mithilfe von Graphen.

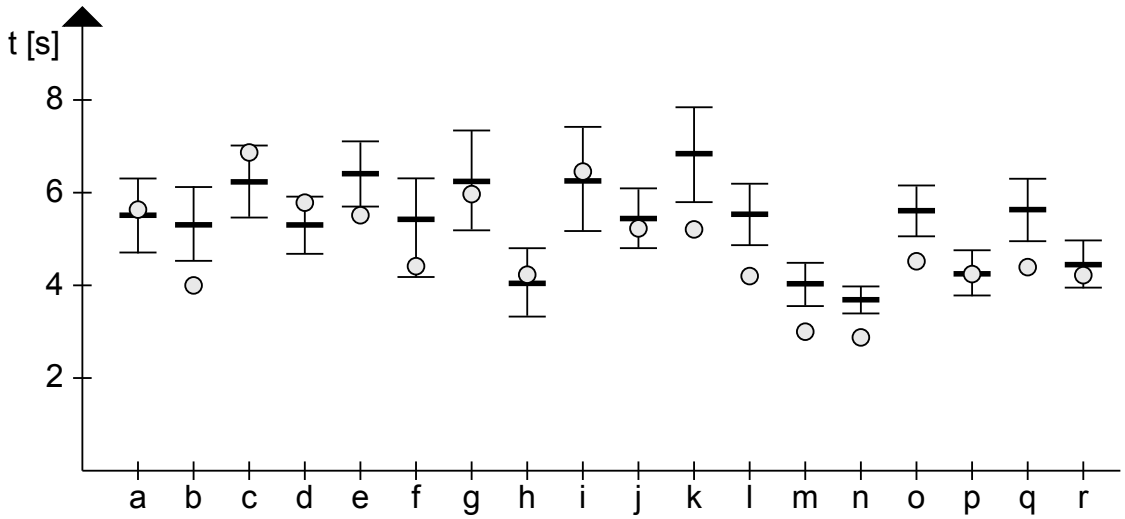


Abbildung 3.2: Vergleich der gemessenen mit den vorhergesagten Zeiten beim UCIE-Modell – Die dicken Linien geben die Durchschnittswerte der in der Studie gemessenen Werte an. Die dünnen Linien zeigen die 1. und 3. Quartile. Die vom UCIE-Modell vorhergesagten Zeiten sind als Kreise dargestellt. a–r repräsentieren verschiedene Aufgabentypen. [32]

3.2.1 Aufbau und Funktionsweise

Zum besseren Verständnis der Theorie werden zunächst einige Begriffe eingeführt, die Pinker in seinem Artikel verwendet.

Visuelles Array[2]: Als visuelles Array werden die „Rohdaten“ bezeichnet, welche direkt von der Retina über den Sehnerv gelangen. Auf einen Rechner bezogen, sind dies die Farbwerte der einzelnen Pixel eines Bildes.

Visuelle Beschreibung[3]: Die Modellierung eines Diagramms und dessen Interpretation werden in sogenannte „Visuelle Beschreibungen“ gefasst. Diese beschreiben Eigenschaften und Relationen zwischen einzelnen Objekten in Diagrammen und bilden damit eine zum visuellen Array übergeordnete Ebene. Unäre und binäre Relationen (Eigenschaften bzw. Relationen zwischen zwei Objekten) können in Form von gerichteten Graphen dargestellt werden, wodurch die Beschreibungen komplexer Szenen an Übersichtlichkeit gewinnen (Abbildung 3.3).

Die visuelle Beschreibung eines kompletten Diagramms soll zwar so umfangreich wie möglich sein, jedoch ist laut Pinker immer nur eine Untermenge an Relatio-

[2] im Original: „Visual Array“
[3] im Original: „Visual Description“

nen bzw. ein Teilgraph zu einem bestimmten Zeitpunkt im Arbeitsgedächtnis eines Menschen. So sind beispielsweise die Koordinaten aller Punkte in einem Punktdiagramm im Graphen enthalten, jedoch können nur wenige Punkte gleichzeitig im Arbeitsgedächtnis gespeichert werden.

Diagramm-Schema[4]: Da jeder Diagrammtyp unterschiedlich modelliert und gelesen werden muss, definiert Pinker sogenannte Diagramm-Schemata. Ein solches Schema definiert, welche grundsätzlichen Beziehungen für einen speziellen Diagrammtyp gelten und wie aus diesem unterschiedliche Informationen gelesen werden können. Aus einem Schema heraus können Instanzen, sogenannte „Diagramm-Modelle", erstellt werden. Ein Diagramm-Modell repräsentiert ein einzelnes Diagramm des Diagrammtyps, welcher vom Schema spezifiziert wird und ähnelt zum größten Teil dem Schema selbst. So werden die fixen Beziehungen und Objekte vom Schema direkt in das Diagramm-Modell übernommen. Objekte, die für jedes Diagramm-Modell unterschiedlich sind (z. B. die Koordinaten eines Punktes), werden als Klassen interpretiert und im Modell entsprechend ihrer Anzahl mehrfach instanziiert.

In Pinkers Graphenschreibweise sind Klassen durch einen Stern (*) im entsprechenden Knoten markiert. Zur Beschreibung des Lesevorgangs enthält das Diagramm-Schema zusätzliche Nachrichten-Flags[5], die an Knoten und Kanten im Graph definiert sein können. Jeder Flag kann mehrere Indizes und Variablen enthalten, die durch die im Folgenden beschriebenen Prozesse durch Werte substituiert werden.

Ein umfangreiches Beispiel eines solchen Schemas findet sich in Abbildung 3.4, in der ein Teil des Schemas eines Säulendiagramms abgebildet ist.

Zum Ablesen eines Wertes in einem Diagramm werden aus den Pixel-Informationen des visuellen Arrays Objekte identifiziert und als Modell in eine visuelle Beschreibung transferiert. Mithilfe der erfassten Objekte sucht der nachfolgend beschriebene MATCH-Prozess ein passendes Schema aus und instanziiert dieses. Daraufhin werden konzeptuelle Nachrichten (im Folgenden beschrieben) generiert, aus denen einfache Fragen beantwortet werden können. Schlussfolgernde Prozesse arbeiten auf einer höheren kognitiven Ebene und verarbeiten die gefundenen Werte weiter, indem z. B. mathematische Operationen wie Additionen oder Multiplikationen angewandt werden. Der komplette Zusammenhang dieser Prozesse ist in Abbildung 3.5 dargestellt und wird im Originaldokument [32] näher beschrieben. Die Aufgaben der einzelnen Prozesse werden nun ausführlicher betrachtet.

MATCH-Prozess

Pinkers Theorie geht davon aus, dass der Mensch mehrere Diagramm-Schemata kennt. Wird ihm nun ein Diagramm vorgelegt, identifiziert er zunächst einzelne Objekte, wie

[4] im Original: „Graph Schemas"
[5] im Original: „Message Flags"

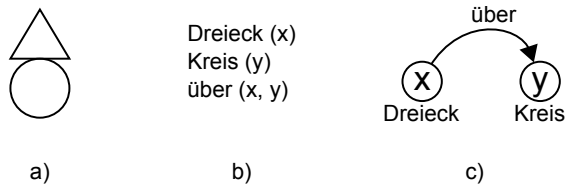


Abbildung 3.3: Objekt-Relationen in einer Szene – Die Szene im visuellen Array a) wird mithilfe von Relationen b) in einen Graph c) überführt [32].

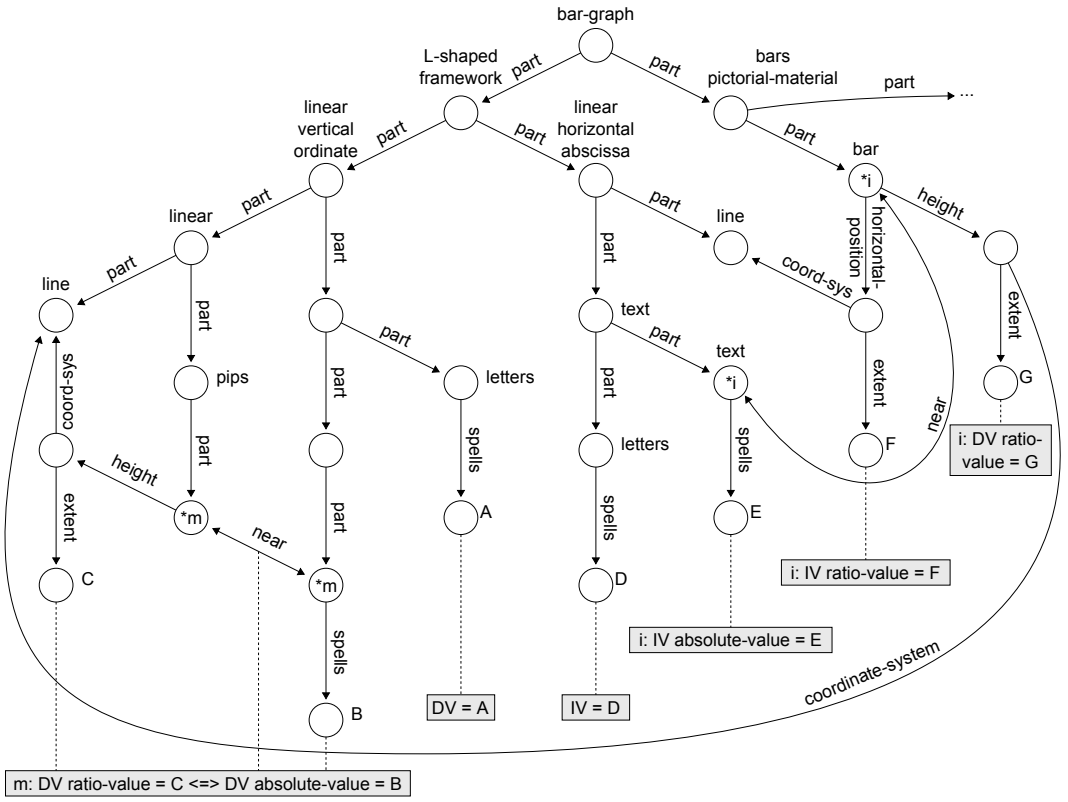


Abbildung 3.4: Schema eines Säulendiagramms nach Pinker – Beschreibung eines Säulendiagramms auf allgemeiner Ebene. A-G sind Platzhalter für entsprechende Werte der instanziierten Diagramme. Die Sterne (*) in einigen Knoten definieren, dass an dieser Stelle beliebig viele Kindknoten existieren können. Nachrichten-Flags (graue Fähnchen) geben die Reihenfolge der Suche nach bestimmten Kriterien vor. [32]

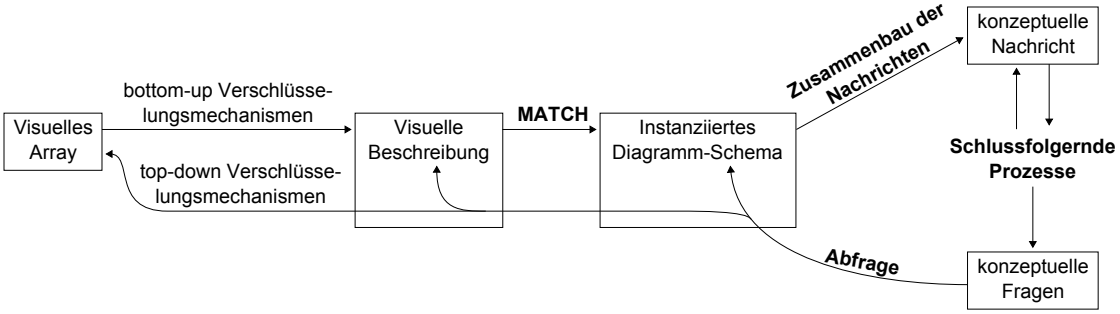


Abbildung 3.5: Prozesse der visuellen Wahrnehmung bei Diagrammen nach Pinker – Die Pixel-Informationen eines Diagramms werden mithilfe einer Art Pipeline durch mehrere Prozesse verarbeitet, bis zuletzt die Antwort der gestellten Anfrage erzeugt wird. [32]

Koordinatenachsen oder Punkte und generiert aus deren Relationen eine visuelle Beschreibung. Dieser Vorgang wird durch den von Anderson und Bower [9] vorgestellten MATCH-Prozess beschrieben.

Dieser gleicht die visuelle Beschreibung mit jedem abgespeicherten Schema ab und wählt das Schema der größten Übereinstimmung mit dem gegebenen Diagramm durch ein Goodness-Of-Fit-Verfahren (z. B. Chi-Quadrat-Test) aus. Daraufhin instanziiert der Prozess das gewählte Schema mit den Werten der visuellen Beschreibung und verwirft evtl. unnötige Teile.

Zusammenbau der Nachrichten

Nachdem durch den MATCH-Prozess das passendste Schema ausgewählt und instanziiert wurde, sorgt dieser Prozess für die Auswertung der Nachrichten-Flags. Die darin befindlichen Variablen werden durch die Werte im instanziierten Schema ersetzt. Die Indizes werden durchnummeriert und für alle Nachrichten-Flags mit demselben Index wird je eine sogenannte „konzeptuelle Nachricht“[6] erzeugt. Aus diesen Nachrichten lassen sich bereits einfache Antworten ablesen, wie für die Frage „Welche Höhe hat Säule X?“.

Abfrage

Jede Simulationsaufgabe wird durch eine Anfrage bestimmt. Der Abfrageprozess durchsucht in einem ersten Schritt die konzeptuellen Nachrichten nach vorhandenen Antworten für die Anfrage. Da, wie erwähnt, nie das komplette Modell des Diagramms im Arbeitsgedächtnis ist, kann es vorkommen, dass erneute Anfragen an die visuelle Beschreibung

[6] im Original: „Conceptual Message“

oder sogar das visuelle Array nötig sind. Erstreckt sich die Aufgabe über mehrere konzeptuelle Nachrichten (z. B. „Wie groß ist der Höhenunterschied zwischen den beiden Säulen A und B (größer, kleiner oder gleich)“), kombiniert sie der Abfrageprozess, indem er gültige Nachrichten-Flags aktiviert (konzeptuelle Nachrichten erzeugt), um daraus weitere gültige Nachrichten-Flags zu finden und letztlich einfache Fragen beantworten zu können. Diese Antworten werden daraufhin kombiniert und die anfangs komplexe Anfrage kann beantwortet werden.

Schlussfolgernde Prozesse

Die schlussfolgernden Prozesse beschreibt Pinker nur kurz, da sie auf abstrakteren Ebenen als dem bloßen Ablesen von Werten in Diagrammen arbeiten. Mit ihnen ist es beispielsweise möglich, Steigungen in Liniendiagrammen oder absolute Höhenunterschiede zwischen zwei Säulen zu „berechnen“. Der Verarbeitungskomplexität sind keine Grenzen gesetzt, sodass hier je nach Wissen des Betrachters und darstellender Information des Diagramms z. B. Gefühle die Interpretationen beeinflussen können.

3.2.2 Bewertung und Abgrenzung

Auch wenn der Artikel von Pinker ausschließlich theoretischer Natur ist und kein Prozess praktisch beschrieben wird, bildet die Theorie dennoch eine gute Grundlage für diese Arbeit. Die Idee der Modellierung von Schemata, mit welchen sich sowohl die Diagramme selbst als auch deren Leseprozess beschreiben lassen, bieten hohes Potenzial. Um hohe Flexibilität bei der Verarbeitung unterschiedlicher Visualisierungstypen zu ermöglichen, soll die Grundidee der Schemata auf alle Visualisierungstypen angepasst und übernommen werden.

3.3 CogTool

CogTool [23] wird seit 2009 an der Carnegie Mellon University entwickelt. Es ist, im Gegensatz zu den zuvor beschriebenen Modellen und Theorien, ein implementiertes, lauffähiges Tool mit grafischer Oberfläche. Mithilfe von CogTool können Interaktionszeiten bei der Bedienung von grafischen Benutzeroberflächen von Programmen und Webseiten vorhergesagt werden. Dadurch ist es z. B. möglich, die Anordnung und Größe einzelner Widgets zu optimieren. Das Tool baut ebenso wie das DVM (siehe Abschnitt 3.1.2) auf dem KLM auf – entwickelt wurde es mithilfe des Kognitionsframeworks ACT-R.

3.3.1 Aufbau und Funktionsweise

Ein CogTool-Projekt besteht aus zwei Teilen: Dem Ablaufplan[7], der die einzelnen Oberflächen des zu überprüfenden Programms definiert, und den Prozessen[8], die mögliche Handlungsabläufe festlegen.

Ablaufplan

Der Ablaufplan beinhaltet ein oder mehrere Bildschirminhalte[9]. Diese wiederum repräsentieren die Benutzeroberfläche des zu untersuchenden Programms in jedem Zustand. Dazu bietet CogTool einige Widgets, wie Buttons, Eingabefelder und Menüs, mit welchen die Oberflächen modelliert werden können. Jedes Widget besitzt als Eigenschaften eine Größe und eine Position auf der Benutzeroberfläche. Zudem können weitere widgetspezifische Eigenschaften festgelegt werden. Hierzu zählt z. B. das Verhalten beim Anklicken eines Buttons mit der Maus oder die Möglichkeit der Eingabe eines Textes in ein Eingabefeld. Auf solch ein Ereignis kann reagiert werden, indem das entsprechende Widget mit einem Übergang[10] zu einem anderen Bildschirminhalt verknüpft wird. Im Ablaufplan werden alle Übergänge zwischen den Bildschirminhalten mit den Verknüpfungen zu den entsprechenden Widgets definiert. Abbildung 3.6 zeigt den Ablaufplan einer App, die für die Orientierung von Touristen in einer Stadt dient, in CogTool.

Alternativ zur manuellen Erstellung des Ablaufplans, bietet CogTool die Funktion, eine Webseite automatisiert in ein Modell zu übersetzen. Hierzu müssen die URL und einige weitere Einstellungen für die zu modellierende Webseite angegeben werden. CogTool lädt daraufhin alle Seiten herunter, erstellt je einen Bildschirminhalt pro Seite und erzeugt Übergänge anhand der HTML-Links.

Prozesse

Die Wege, die ein Benutzer im Ablaufplan wählen kann (z. B. 1: „Drücke Button X“ – 2: „Gebe Text in Textfeld A ein“ – 3: „Drücke Button Y“), werden in Form von Prozessbeschreibungen definiert. Ein Prozess beinhaltet genau ein Skript[11], das diesen Weg beschreibt. Es ist somit möglich, über mehrere Wege vom selben Start- zum selben Ziel-Bildschirminhalt zu gelangen und hierfür pro Weg jeweils ein Skript zu erstellen.

[7] im Original: „Storyboard“
[8] im Original: „Tasks“
[9] im Original: „Frames“
[10] im Original: „Transition“
[11] im Original: „Script“

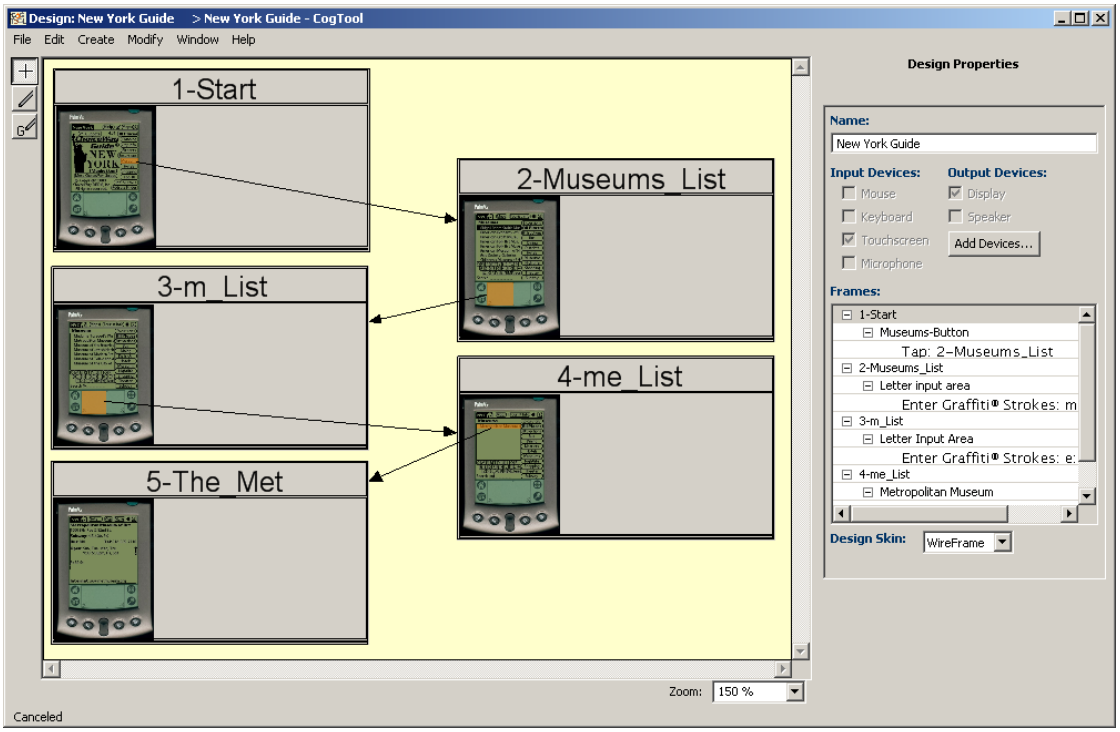


Abbildung 3.6: Beispielhafter Ablaufplan in CogTool – Der Ablaufplan zeigt die Oberfläche einer Smartphone-App zur Orientierungshilfe von Touristen in New York. Die Pfeile stellen Übergänge zwischen den Bildschirminhalten dar.

Funktionsweise

Wurden der Ablaufplan und ein oder mehrere Prozesse erstellt, kann CogTool für jeden Prozess die Zeit berechnen, die ein erfahrener Benutzer für die gleiche Aufgabe benötigt. In diese Berechnung fließen außer den Größen und Positionen der einzelnen Widgets zusätzlich soft- und hardwareabhängige Zeiten, welche in den Übergängen festgelegt werden können, und kognitive Zeiten des Benutzers mit ein. Wurden für eine Aufgabe unterschiedliche Prozesse definiert, können die daraus berechneten Zeiten miteinander verglichen und die Bedienung des untersuchten Programms optimiert werden. Zusätzlich bietet CogTool die Möglichkeit, die berechnete Zeit in einzelne Prozesse unterteilt visuell zu analysieren (siehe Abbildung 3.7).

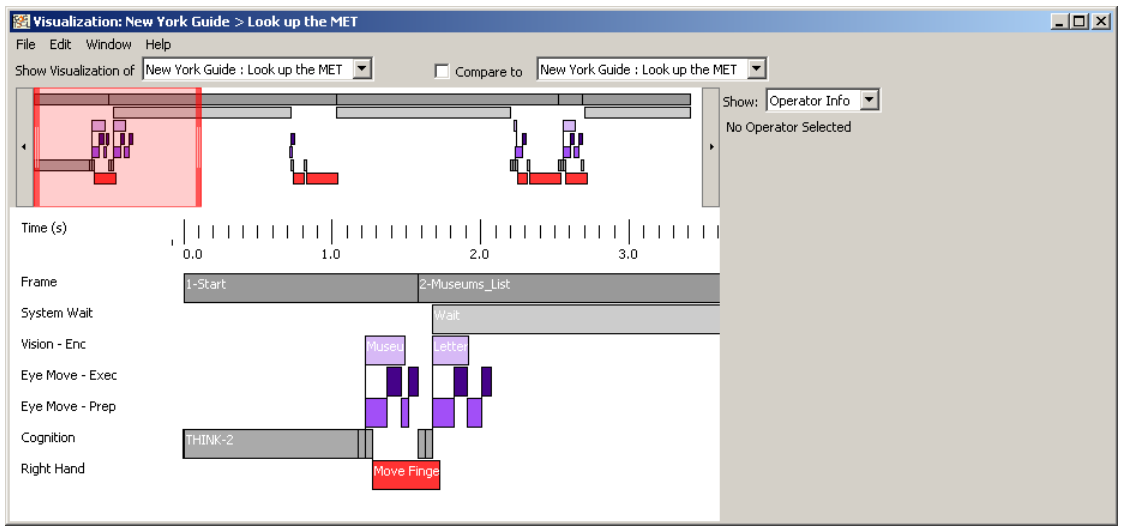


Abbildung 3.7: Beispielhafte Visualisierung der Reaktionszeiten in CogTool – Die Visualisierung zeigt die Zeiten der einzelnen Komponenten beim Abarbeiten eines Prozesses. Die Bezeichnungen der Komponenten stammen vom Kognitionsframework ACT-R [7].

3.3.2 Bewertung und Abgrenzung

CogTool erlaubt es dem Entwickler, seine entworfene grafische Benutzeroberfläche und Bedienung auf einfache Art und Weise quantitativ in Form von Zeit zu evaluieren. Dazu sind kein wissenschaftliches Expertenwissen und keine komplizierten Berechnungen notwendig. Die Idee, das Sichtbare mithilfe einer grafischen Oberfläche zu modellieren und daraufhin auf Bedienzeiten zu evaluieren, soll auch in dieser Arbeit übernommen werden. Da CogTool speziell für den HCI-Bereich entwickelt wurde, fließen keine detailliertere Faktoren, wie die Anzahl benachbarter Widgets um einen zu klickenden Button, in das Modell ein. Gerade bei der Simulation von Visualisierungsprozessen ist es jedoch ein unabdingbares Kriterium, ob sich ein Objekt z. B. auf einer weißen Fläche oder vor einem Schachbretthintergrund befindet. Auch die Lernfähigkeit der Simulation wird bei CogTool vernachlässigt: Ein Benutzer, der eine Oberfläche oder Visualisierung bereits mehrfach verwendet hat, kann diese auf Dauer schneller bedienen und interpretieren.

4 Aufgabe & Lösungsansatz

Das Ziel dieser Arbeit ist die Entwicklung eines Modells, mit dem kognitive Prozesse im menschlichen Gehirn beim Betrachten von Visualisierungen simuliert werden können.

Dieses Kapitel wird zuerst das Ziel bzw. die Aufgabenstellung der Arbeit beschreiben (Abschnitt 4.1). Die in den vorigen Kapiteln vorgestellten Tools, Frameworks und vorhandenen Modelle sollen daraufhin nochmals näher auf ihre Brauchbarkeit für die Umsetzung des Lösungsansatzes untersucht werden (Abschnitt 4.2).

4.1 Aufgabenstellung

Auch wenn nur wenige neue Visualisierungstypen in den (wissenschaftlichen) Alltag gelangen, werden ständig neue Visualisierungen entwickelt. Für die Ermittlung des Mehrwerts neuer Visualisierungen werden gewöhnlich Studien durchgeführt. Diese sind oft sehr aufwendig. Man benötigt meist Probanden aus unterschiedlichen Personenkreisen und Altersklassen. Der Zeitaufwand für die Erstellung und Durchführung der Studie liegt nicht selten bei mehreren Tagen oder Wochen, wodurch auch der finanzielle Aspekt eine große Rolle spielt. Die Preise für einen Eye-Tracker belaufen sich auf mehrere Tausend Euro. Ein weiteres Problem sind die Messfehler, die bei den Aufzeichnungen entstehen können. Hier müssen gut 10-20 % der Daten aufgrund von Blinzeln oder zu schnellen Kopfbewegungen des Probanden verworfen werden. Optimiert man infolge der Studienergebnisse einen Visualisierungstyp, beginnt eine neue, ebenso aufwendige Studie, um die Verbesserungen aufzuzeigen. Dies macht den Prozess sehr unflexibel, vor allem wenn es darum geht, geringe Optimierungsfaktoren zu validieren.

Um dem hohen Aufwand und den Kosten entgegenzuwirken und gleichzeitig die Erforschung neuer Techniken zu beschleunigen, soll das Ziel dieser Arbeit sein, ein Modell zu erstellen, welches auf Basis bereits erforschter kognitiver Prozesse und empirisch ermittelter Daten aus durchgeführten Studien das Verhalten eines Menschen beim Betrachten von Visualisierungen simuliert. Die durch die Simulation erhaltenen Daten können direkt in die Weiterentwicklung von Visualisierungen einfließen und innerhalb weniger Sekunden erneut ermittelt werden – ein iteratives, flexibles Forschen im Bereich der Visualisierung wird möglich.

Als Ausgangsdaten sollen in erster Linie Durchführungszeiten von visuellen Aufgaben betrachtet werden. Diese entscheiden letztlich, ob eine Visualisierung schneller oder langsamer gelesen werden kann als eine andere, und definieren damit eine quantifizierbare Metrik für eine Bewertung. Um festzustellen, warum eine Visualisierung z. B. langsamer lesbar ist, hilft zudem die Simulation der Augenbewegungen und Fixationszeiten des Betrachters. Anhand derer lassen sich unter anderem Abstände, Größen und Farben von Elementen optimieren.

Nachdem ein solches Modell erstellt wurde, soll ein Teil davon (z. B. für einen bestimmten Visualisierungstyp) prototypisch in Form eines Computerprogramms implementiert werden. Zuletzt beinhaltet die Aufgabe die Durchführung einer Studie, in welcher nachgewiesen werden soll, dass das implementierte Modell Antwortzeiten und Augenbewegungen in einem angemessenen Maß korrekt vorhersagen kann.

4.2 Lösungsansatz

Bei der Recherche zu existierenden Arbeiten (siehe Kapitel 3) im Bereich kognitive Modellierung wurde festgestellt, dass das Kognitionsframework ACT-R sehr häufig zur Anwendung kommt. Die fortwährende Weiterentwicklung und der Ansatz sowohl symbolischer als auch subsymbolischer Vorgehensweisen machen ACT-R zu einer vielversprechenden Ausgangsbasis. Da ACT-R außerdem hervorragend dokumentiert ist und viel zusätzliche Literatur existiert, soll das Framework als Grundbaustein zur Lösung der Aufgabe dienen.

Neben einigen Basis-Produktionsregeln und deklarativen Wissens-Chunks sollte das Modell weitgehend flexibel bleiben, sodass jederzeit zusätzliche Visualisierungstypen hinzugefügt werden können. Um dies zu ermöglichen, orientiert sich diese Arbeit an dem Graphen-Modell von S. Pinker (siehe Abschnitt 3.2), indem z. B. die Erstellung eines neuen Schemas den Lesevorgang einer neuen Visualisierung definiert.

Die Beschreibung des Bewegungspfads der Augen soll nicht wie beim DVM oder UCIE-Modell (Abschnitte 3.1.2 und 3.1.3) weitgehend festgelegt sein. Vielmehr soll stattdessen die visuelle Suche in den Vordergrund treten und die Augenbewegungen z. B. von der Umgebung (Anzahl Nachbarelemente, Farben, Textur usw.) der aktuellen Fixation abhängig sein. Für die Berechnung der Fixationszeiten werden das UCIE-Modell und die darin verwendeten Formeln Vorbild sein.

Außer dem Modell selbst ist auch die grafische Benutzeroberfläche für das Erstellen neuer Visualisierungs-Schemata (siehe Abschnitt 3.2) Teil der Aufgabe. CogTool (Abschnitt 3.3) ermöglicht bereits die Erstellung des Modells einer Oberfläche für den HCI-Bereich. Diese Arbeit soll sich an der Benutzer-Interaktion von CogTool orientieren und das Prinzip auf Visualisierungen erweitern.

5 Lösungskonzept

Dieses Kapitel umfasst die Beschreibung des Modells, das sowohl die Augenbewegungen als auch die Fixationszeiten eines Menschen beim Lösen einer Aufgabe mithilfe einer Visualisierung vorhersagen kann. Zur Erstellung des Modells werden die ersten zwei der in Abschnitt 2.4.2 beschriebenen vier Phasen bearbeitet.

Die erste Phase, die Aufgabenanalyse, beinhaltet die genaue Beschreibung der modellrelevanten Eingabedaten, wie die Aufgliederung und die Zusammenhänge des deklarativen Wissens sowie den Ablauf der Simulation durch einzelne Verarbeitungsschritte (Abschnitt 5.1). Die zweite Phase, die Empirische Untersuchung, beschäftigt sich mit einer Studie, um die benötigten kognitiven Prozesse mit den zugehörigen Reaktionszeiten zu identifizieren (Abschnitt 5.2).

5.1 Aufgabenanalyse (= Phase 1)

Der Abschnitt beschreibt den Aufbau und die Funktionsweise des Simulationsmodells im Detail. Abbildung 5.1 zeigt schematisch die Hauptkomponenten des Modells auf, die sich in die beiden Bereiche Eingabe- und Verarbeitungsmodell einordnen lassen.

Das Eingabemodell ist in die Anfrage (Abschnitt 5.1.4) sowie die folgenden drei Ebenen aufgeteilt, wobei deren Zusammenhänge in Abbildung 5.2 dargestellt sind:

Ontologie: Beinhaltet die Grundelemente für alle Visualisierungstypen. (Abschnitt 5.1.1)

Visualisierungsschemata & Visualisierungsschema-Editor: Bilden Teilmengen der Ontologie und definieren Klassen-Verknüpfungen, welche für den jeweiligen Visualisierungstyp möglich bzw. fix sind. Außerdem geht aus ihnen hervor, wie Visualisierungen zu lesen sind. (Abschnitt 5.1.2)

Visualisierungsmodell & Visualisierungsmodell-Editor: Beschreiben die Visualisierungen auf Grundlage der Visualisierungsschemata. (Abschnitt 5.1.3)

Das Verarbeitungsmodell beschreibt die folgenden Prozesse, die das Lesen einer Visualisierung simulieren:

Passendes Schema finden: Wählt für ein gegebenes Visualisierungsmodell das optimale Visualisierungsschema aus. (Abschnitt 5.1.5)

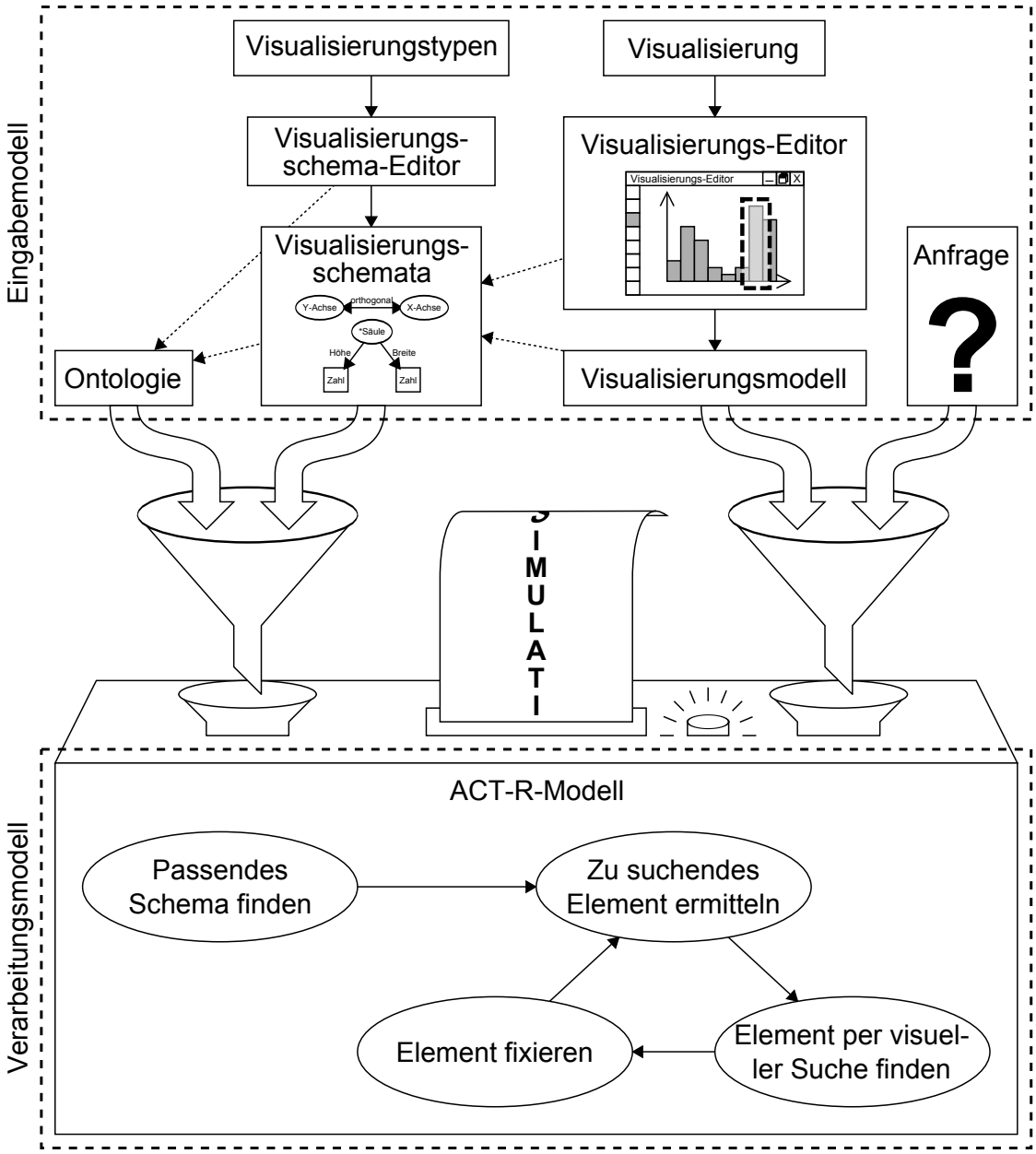


Abbildung 5.1: Lösungskonzept – Das Simulationsmodell kann als eine Art Maschine dargestellt werden, wobei diese als Eingabe eine Ontologie, mehrere Visualisierungsschemata, ein Visualisierungsmodell und eine Anfrage erhält (Eingabemodell – oben). Daraufhin verarbeitet die Maschine die Anfrage, indem sie kognitive Prozesse eines Menschen für das Lesen der Visualisierung simuliert (Verarbeitungsmodell – unten).

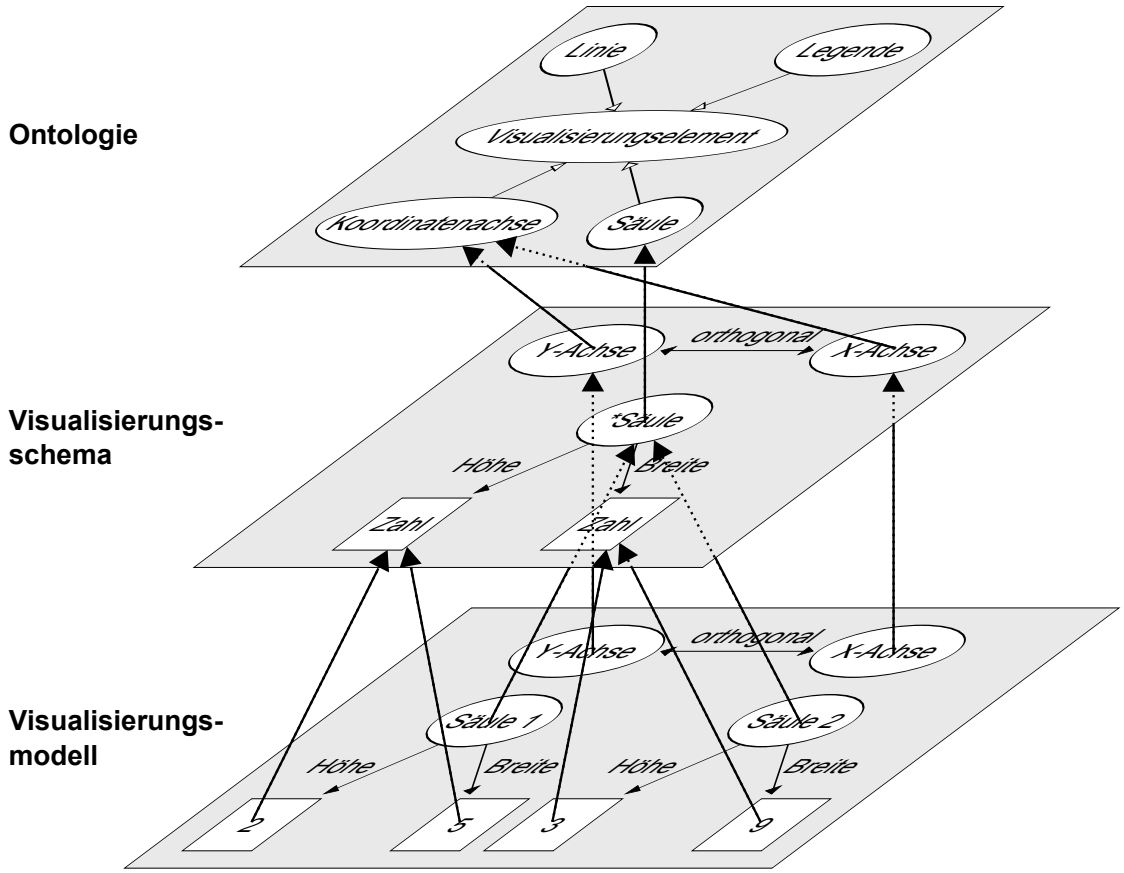


Abbildung 5.2: 3-Ebenen-Eingabemodell – Die Visualisierungsschemata bestehen aus Instanzen einzelner Klassen der Ontologie. Sie definieren außerdem Relationen, die für den entsprechenden Visualisierungstyp und somit für alle Modelle dieses Typs fix sind. Die Visualisierungsmodelle nutzen die Instanzen und Relationen der Visualisierungsschemata wiederum als Klassen und bilden Instanzen davon.

Zu suchendes Element ermitteln: Findet im Visualisierungsmodell das nächste zu suchende Element. (Abschnitt 5.1.6)

Element per visueller Suche suchen: Sucht ein Element im Visualisierungsmodell mithilfe von Regeln der visuellen Suche. (Abschnitt 5.1.7)

Element fixieren: Simuliert die Fixation eines Elements im Modell. (Abschnitt 5.1.8)

5.1.1 Ontologie

Einen Grundbaustein für die Beschreibung von Visualisierungsschemata und damit auch für die Erstellung von Visualisierungsmodellen ist eine zugrunde liegende Ontologie. Diese beschreibt alle vorkommenden Grundelemente, wie Punkte, Koordinatenachsen oder Legenden und die zwischen diesen fix vorliegenden Relationen.

Um Übersichtlichkeit innerhalb der Ontologie zu bewahren, wird sie in drei Stufen aufgeteilt (siehe Abbildung 5.3). Diese sind wie folgt definiert:

1. Stufe: Stufe 1 definiert ausschließlich Vererbungs- und „hat“-Relationen. Dadurch beschreibt die Stufe lediglich, welche Ontologie-Elemente es gibt und aus welchen Unterelementen diese aufgebaut sind.

2. Stufe: In dieser Stufe werden alle zur ersten Stufe zusätzlichen Klassen, Relationen und Literale definiert, die ein Mensch den jeweiligen Elementen innerhalb kürzester Zeit zuordnet. Dies sind hauptsächlich unscharfe Eigenschaften, die sofort oder innerhalb von wenigen Sekunden ins Bewusstsein gelangen (z. B. „Element A liegt rechts von Element B“ oder „Element A hat die Farbe Rot“).

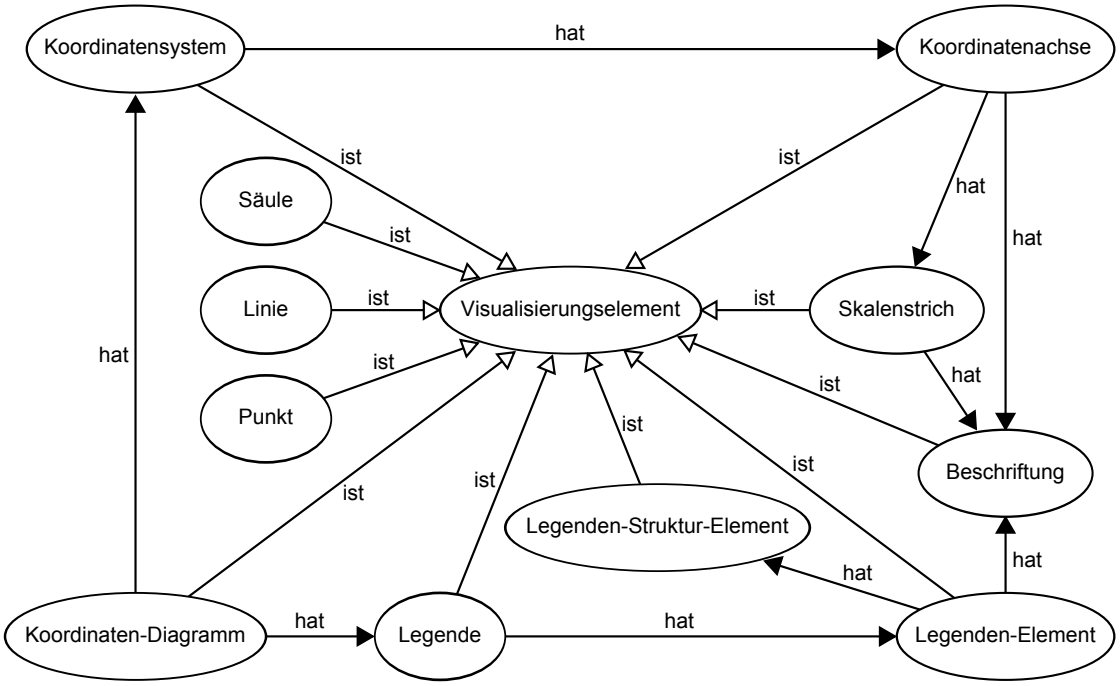
3. Stufe: Alle exakten Werte, wie Größe oder Position sind in der dritten Ontologie-Stufe dargelegt. Die Werte können sowohl relativ (z. B. Werte von 0 bis 1) als auch absolut (z. B. Größe in Pixel) vorliegen. Es ist davon auszugehen, dass alles, was in der dritten Stufe definiert ist, nie präzise von einem Menschen erkannt werden kann. Die Ontologie in dieser Stufe ist somit nicht als kognitiv motiviert zu betrachten, wird jedoch für die späteren Berechnungen der Augenbewegungen unbedingt benötigt.

Die in Abbildung 5.3 dargestellte Ontologie beinhaltet Klassen und Relationen, die für die Erstellung von Punkt-, Linien- und Säulendiagrammen verwendet werden können.

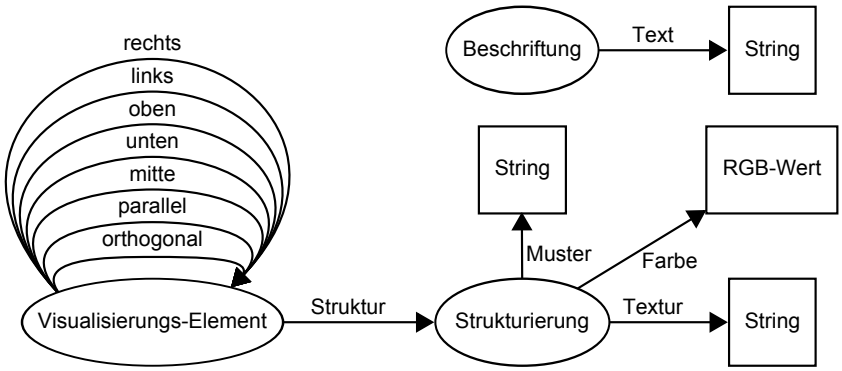
5.1.2 Visualisierungsschemata & Visualisierungsschema-Editor

Jeder Visualisierungstyp besitzt Strukturen, die für alle damit visualisierten Daten immer gleich bleiben. Die im vorigen Abschnitt beschriebene Ontologie gibt Klassen vor, die für die Generierung der Strukturen verwendet werden dürfen. Eine solche Struktur wird im Weiteren „Visualisierungsschema“ genannt und orientiert sich an den Diagramm-Schemata von S. Pinker (siehe Abschnitt 3.2).

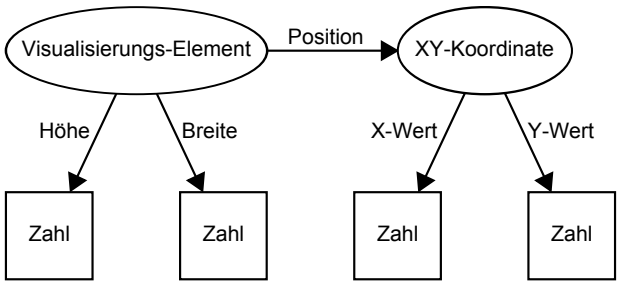
Ein Visualisierungsschema kann einerseits als Instanz der zuvor beschriebenen Ontologie interpretiert werden, andererseits bildet ein Visualisierungsschema eine eigene Ontologie für einen speziellen Visualisierungstyp. Außerdem werden darin Leseregeln für entsprechende Aufgaben definiert.



(a) Ontologie 1. Stufe: Diese Ontologie-Stufe beschreibt, welche Ontologie-Elemente es gibt und aus welchen diese aufgebaut sind.



(b) Ontologie 2. Stufe: In dieser Stufe werden zusätzlich zur ersten Stufe Klassen, Relationen und Literale definiert, welche unscharf und unmittelbar bzw. innerhalb weniger Sekunden wahrgenommen werden können.



(c) Ontologie 3. Stufe: Die dritte Stufe erweitert die Ontologie um „scharfe“ Eigenschaften wie Größe oder Position von Visualisierungselementen.

Abbildung 5.3: 3-Stufen-Ontologie – Die Ontologie des Modells ist in drei Stufen eingeteilt. Mit den dargestellten Klassen und Relationen können Modelle für Punkt-, Linien- und Säulendiagramme erstellt werden.

Die einzelnen Elemente sind folgendermaßen spezifiziert:

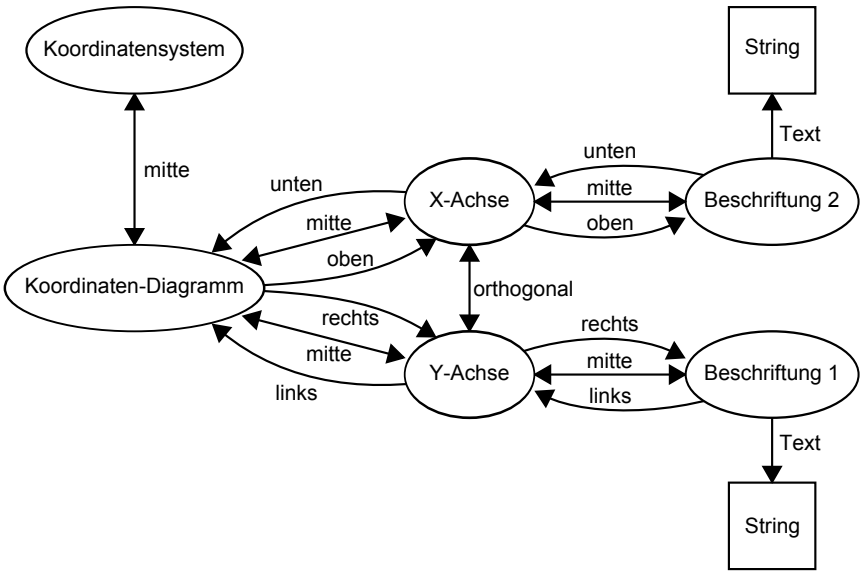
Vordefinierte Elemente und Relationen: Jedes Visualisierungsmodell besitzt Elemente, die für den jeweiligen Visualisierungstyp spezifisch sind. Diese Elemente sind fest vorgegeben und durch fixe Relationen miteinander verbunden. Die Elemente und Relationen liegen jedem Visualisierungsmodell des repräsentierenden Typs als unveränderbares Gerüst zugrunde. Ein Visualisierungsschema beinhaltet somit Instanzen von Klassen der Ontologie und schreibt Relationen zwischen diesen vor. Das Visualisierungsschema eines zweidimensionalen Säulendiagramms besitzt beispielsweise folgende Instanzen, welche wie in Abbildung 5.4a miteinander verbunden sein könnten:

- 1 x Koordinaten-Diagramm
- 1 x Koordinatensystem
- 2 x Koordinatenachse jeweils mit 1 x Beschriftung

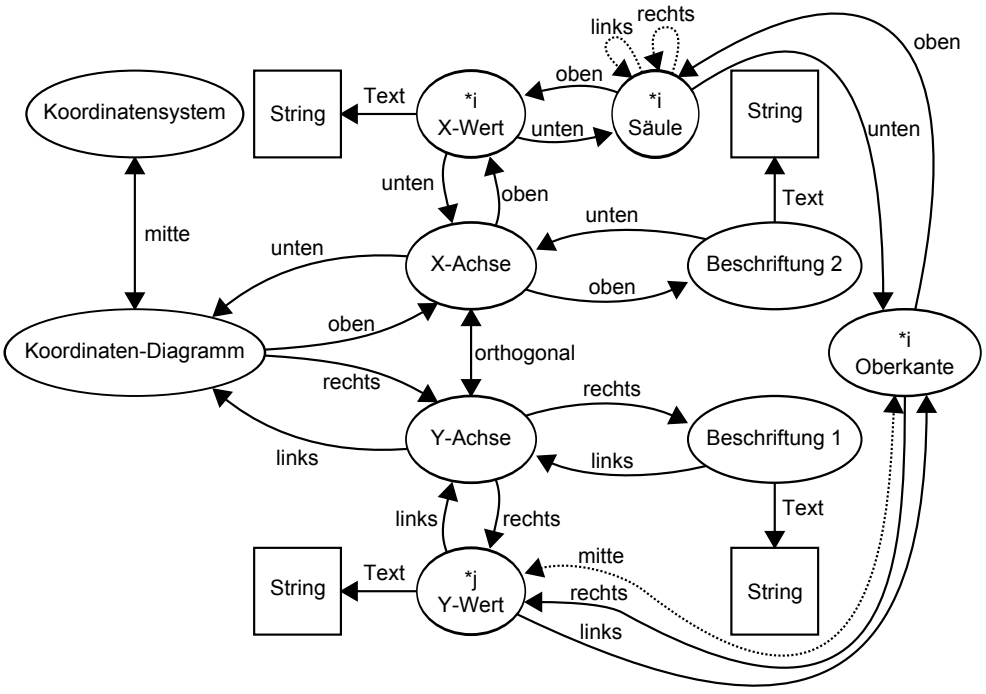
Im Visualisierungsmodell verwendbare Klassen und Relationen: Zusätzlich zu den fix definierten Elementen und Relationen in einem Schema müssen bestimmte Freiheitsgrade für die einzelnen Visualisierungsmodelle existieren. Mögliche Freiheitsgrade sind:

- Anzahl der Instanzen bestimmter Klassen
- Mögliche Relationen zwischen bestimmten Klassen

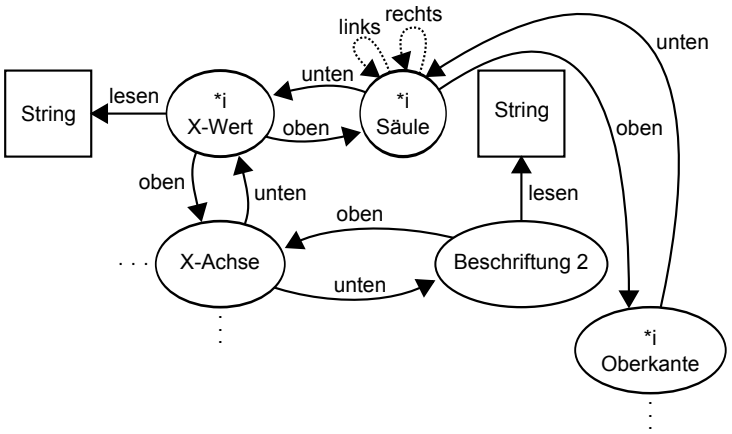
Einzig diese Freiheitsgrade sollen für die Visualisierungsmodelle zur Verfügung stehen. Um Konsistenz zu bewahren, dürfen keine weiteren Klassen oder Relationen der Ontologie verwendet werden, die nicht durch das Schema definiert sind. Abbildung 5.4b zeigt das erweiterte Schema am Beispiel des Säulendiagramms.



(a) Die vordefinierten Elemente und Relationen eines Visualisierungsschemas für zweidimensionale Säulendiagramme.



(b) Zusätzlich zu den fixen Elementen und Relationen aus obiger Abbildung definiert das Schema Freiheitsgrade mithilfe eines Sterns (*). „*i Säule“ bedeutet beispielsweise, dass ein Visualisierungsmodell beliebig viele Instanzen der Klasse „Säule“ besitzen kann. Die gepunkteten Relationen deuten mögliche Relationen in den Visualisierungsmodellen an.



(c) Um das Visualisierungsschema mit den Leseregeln zu vervollständigen, werden zusätzliche Relationen spezifiziert, die Übergänge zwischen den einzelnen Elementen im Visualisierungsmodell definieren (siehe auch Abbildung 5.5).

Abbildung 5.4: Visualisierungsschema eines zweidimensionalen Säulendiagramms

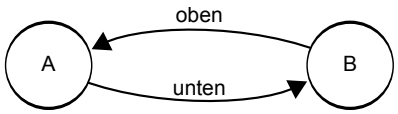


Abbildung 5.5: Visualisierungsschema – Leseregeln – Element A liegt über Element B. Befindet sich die aktuelle Fixation auf Element A und Element B soll betrachtet werden, muss die Relation bzw. der Übergang „unten“ gewählt werden. Dies führt dazu, dass sich die Augen nach unten bewegen müssen. Liegt die aktuelle Fixation auf Element B und Element A soll fixiert werden, müssen sich die Augen nach „oben“ bewegen.

Leseregeln für den Visualisierungstyp: Ein weiterer Zweck der Diagrammschemata ist die Möglichkeit, daraus Regeln für das Ablesen bestimmter Eigenschaften unterschiedlicher Visualisierungstypen zu entnehmen. Das heißt, dass in einem Visualisierungsschema für den Visualisierungstyp „zweidimensionales Säulendiagramm“ z. B. spezifiziert sein muss, welche visuellen Elemente in welcher Reihenfolge von der Simulation fixiert werden müssen, um die Höhe einer Säule abzulesen. Diese Leseregeln sind in Form zusätzlicher Relationen spezifiziert. Es ist zu beachten, dass die Leseregel-Relationen von den vorausgegangenen Orientierungsrelationen unterschieden werden. Im Gegenteil zu diesen definieren die Leseregel-Relationen einen **Übergang** von einem Element A zu einem Element B, wobei der Übergang stattfindet, indem sich die Augen in die durch die Relation vorgegebene Richtung bewegen müssen (siehe Abbildung 5.5). Die Leseregeln beschreiben somit

eine Transformation des Was-Raumes in den Wo-Raum (siehe „visuelles Modul“ – ACT-R Abschnitt 2.4.4). Abbildung 5.4c zeigt eine Teilmenge der Leseregeln für das Beispiel des zweidimensionalen Säulendiagramms. Die zuvor definierten Orientierungsrelationen sind hier nicht dargestellt, um Übersichtlichkeit zu wahren.

Je genauer ein Visualisierungsschema definiert ist bzw. je mehr vorgeschriebene Elemente und Relationen vorgegeben werden, desto differenzierter kann in einem späteren Prozess (siehe Abschnitt 5.1.5) ein zu einem gegebenen Visualisierungsmodell passendes Visualisierungsschema gefunden werden.

Für das Erstellen von Visualisierungsschemata wird ein Tool benötigt, mit welchem Klassen und Relationen aus einer Ontologie ausgewählt und miteinander verbunden werden können. Dieser Vorgang kann komplett mithilfe einer grafischen Benutzeroberfläche umgesetzt werden.

5.1.3 Visualisierungsmodell & Visualisierungsmodell-Editor

Das Visualisierungsmodell beschreibt die zu simulierende Visualisierung mit all ihren Elementen und Relationen, die Instanzen der Klassen des Visualisierungsschemas sind.

Um ein Modell aus einer Visualisierung zu erstellen, wird ein Visualisierungsmodell-Editor benötigt. Hierfür wird das Tool „cEdit“[1] verwendet. Mit diesem können Bereiche in einem Bild, das eine Visualisierung darstellt, markiert und als bestimmte Klassen der Ontologie definiert werden. Außerdem bietet das Tool die Möglichkeit, diesen Elementen zusätzliche Parameter wie Text oder Farbe des zugehörigen Visualisierungselements anzugeben. Abbildung 5.6 zeigt die Skizze des Visualisierungsmodell-Editors.

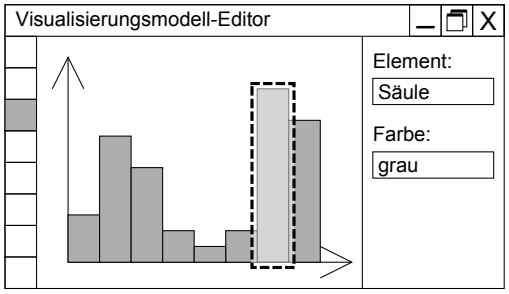


Abbildung 5.6: Visualisierungsmodell-Editor – Mit diesem Tool können Bildelemente markiert und als Visualisierungselemente definiert werden. Jedes Visualisierungselement hat kann Eigenschaften besitzen, die in der Ontologie festgelegt sind (in der Abbildung: „Element = Säule“, „Farbe = grau“).

[1] cEdit ist Teil der „cVIS“-Suite von Raschke et al.

5.1.4 Anfrage

Damit das Eingabemodell komplett ist und simuliert werden kann, fehlt zuletzt die Angabe darüber, welche Eigenschaft aus dem Visualisierungsmodell ausgelesen werden soll. Diese Anfrage muss in Form von Tupeln vorliegen, was im Folgenden anhand eines Beispiels beschrieben werden soll:

Angenommen es liegen die in Abbildung 5.3 beschriebene Ontologie, das in Abbildung 5.4 beschriebene Visualisierungsschema und ein beliebiges Visualisierungsmodell eines zweidimensionalen Säulendiagramms vor. Die Fragestellung für das Säulendiagramm in Abbildung 5.7 könnte folgendermaßen lauten:

> *Welche Höhe hat die Säule mit der Beschriftung „C“ der Achse mit der Beschriftung „Zeit“?*[2]

Daraus lassen sich nun die in Listing 5.1 aufgeführten Tupel bilden.

Listing 5.1: Tupel der Anfrage „Welche Höhe hat die Säule mit der Beschriftung 'C' der Achse mit der Beschriftung 'Zeit'?“

```
1   (Beschriftung1, Beschriftung)
2   (Beschriftung2, Beschriftung)
3   (Beschriftung3, Beschriftung)
4   (Achse1,        Achse)
5   (Säule1,        Säule)
6   (Oberkante1,    Oberkante)
7
8   (Beschriftung1, Text="Zeit")
9   (Beschriftung1, Achse1)
10  (Achse1,        Beschriftung2)
11  (Beschriftung2, Text="C")
12  (Beschriftung2, Säule1)
13  (Säule1,        Oberkante1)
14  (Oberkante1,    Beschriftung3)
15  (Beschriftung3, Text=?)
```

Die Tupel 1 bis 6 definieren alle in der Frage verwendeten Variablen mit ihrer zugehörigen Klasse in der Ontologie. So gibt es beispielsweise die drei Beschriftungen „Höhe“, „Beschriftung 'C'“ und „Beschriftung 'Zeit'“. Zwischen diesen Variablen werden in den weiteren Tupeln Beziehungen hergestellt, welche aus der Fragestellung folgern.

Die Tupel 8 bis 15 beschreiben Beziehungen zwischen den in der Fragestellung vorkommenden Variablen. Eine Ausnahme bilden in dem Beispiel die beiden Tupel 13 und

[2] Die Frage mag im ersten Moment unnatürlich klingen, muss jedoch so exakt formuliert werden um Mehrdeutigkeiten zu verhindern.

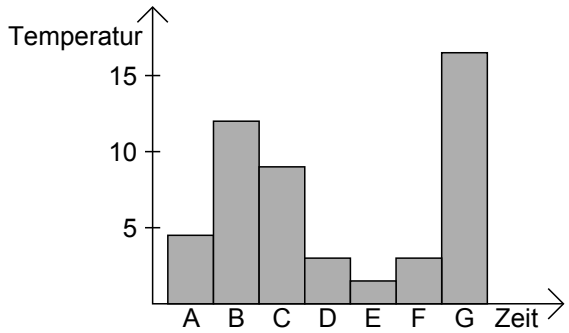


Abbildung 5.7: Beispieldiagramm für eine Anfrage

14. Da die Oberkante einer Säule nicht in der Frage auftaucht, müssen durch einen zusätzlichen Prozess schemabedingte Tupel ersetzt werden. So wurden in dem Beispiel aus `(Säule1, Beschriftung3)` die beiden Tupel `(Säule1, Oberkante1)` und `(Oberkante1, Beschriftung3)`.

Ein Tupel kann im Grunde als binäre Relation zwischen zwei Visualisierungselementen interpretiert werden, wobei der Typ der Relation nicht bekannt ist. Dieser wird erst während der Simulation im Verarbeitungsmodell ermittelt (siehe Abschnitt 5.1.6).

Mit einem „=“ in einem der beiden Elemente eines Tupels werden Literale mit einem bestimmten Inhalt definiert. Soll genau dieser Inhalt ermittelt werden, muss dem „=“ ein „?“ folgen. Im Beispiel soll der Text der Beschriftung „Beschriftung3“ gefunden werden.

Die komplette Anfrage, d. h. die Auflistung aller Tupel, wird nun zusammen mit den zuvor definierten Teilen des Eingabemodells in die Simulation gegeben. Die Simulation beginnt daraufhin, die im Folgenden beschriebenen Prozesse abzuarbeiten.

5.1.5 Passendes Schema finden

Nachdem dem Simulationsmodell die Ontologie, mehrere Visualisierungsschemata, ein Visualisierungsmodell und eine Anfrage übergeben wurden, beginnt die Simulation im ersten Schritt damit, zu erkennen, um was für einen Visualisierungstyp es sich bei dem gegebenen Visualisierungsmodell handelt. Hierzu muss das zu dem Visualisierungsmodell passendste Visualisierungsschema gefunden werden. Die Simulation wählt hierzu immer das Schema aus, welches die meisten mit dem Visualisierungsmodell übereinstimmenden Klassen und Relationen besitzt.

In Abbildung 5.8 sind beispielhaft ein Schema für 3D-Punktdiagramme und ein Schema für 2D-Säulendiagramme sowie ein Visualisierungsmodell eines 2D-Säulendiagramms abgebildet. Die Übereinstimmung eines Visualisierungsmodells mit einem Visualisierungsschema errechnet sich durch das Verhältnis der jeweils übereinstimmenden zu allen

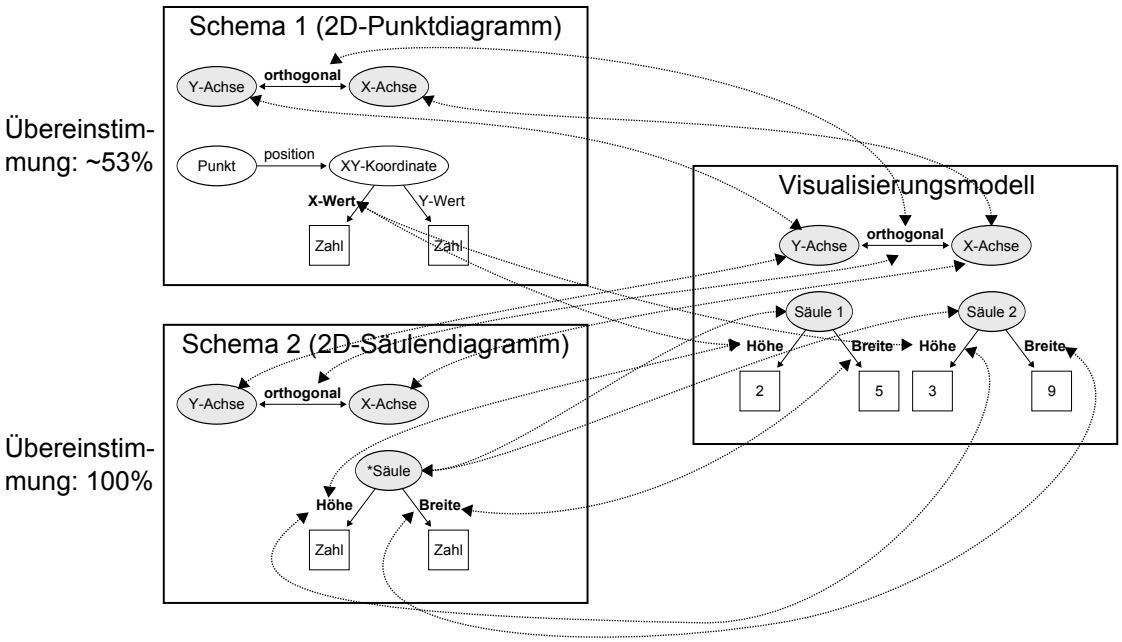


Abbildung 5.8: Finden des passenden Visualisierungsschemas – Die Übereinstimmung des gegebenen Visualisierungsmodells mit dem Schema eines 2D-Säulendiagramms ist höher (100%) als die Übereinstimmung mit dem Schema eines 3D-Punktdiagramms ($\sim 53\%$), wonach die Simulation das Modell als 3D-Säulendiagramm interpretiert.

vorhandenen Klassen und Relationen. Im Beispiel der Abbildung ist die Übereinstimmung für das 2D-Säulendiagramm höher als die für das 3D-Punktdiagramm, weshalb der Algorithmus der Simulation entscheidet, dass die gegebene Visualisierung als 2D-Säulendiagramm interpretiert wird.

Im nächsten Schritt wird mithilfe des nun gefundenen Visualisierungsschemas ermittelt, welche Visualisierungselemente fixiert werden müssen.

5.1.6 Zu suchendes Element ermitteln

Dieser Prozess nutzt die Tupel der Anfrage und das im Vorigen gefundene Visualisierungsschema, um damit herauszufinden, welche Elemente für die Simulation in welcher Reihenfolge fixiert werden müssen. Bei jedem Aufruf gibt der Prozess das nächste zu suchende Element und den räumlichen Bereich, in dem im Visualisierungsmodell gesucht werden soll, an den nächsten Prozess (siehe Abschnitt 5.1.7) weiter.

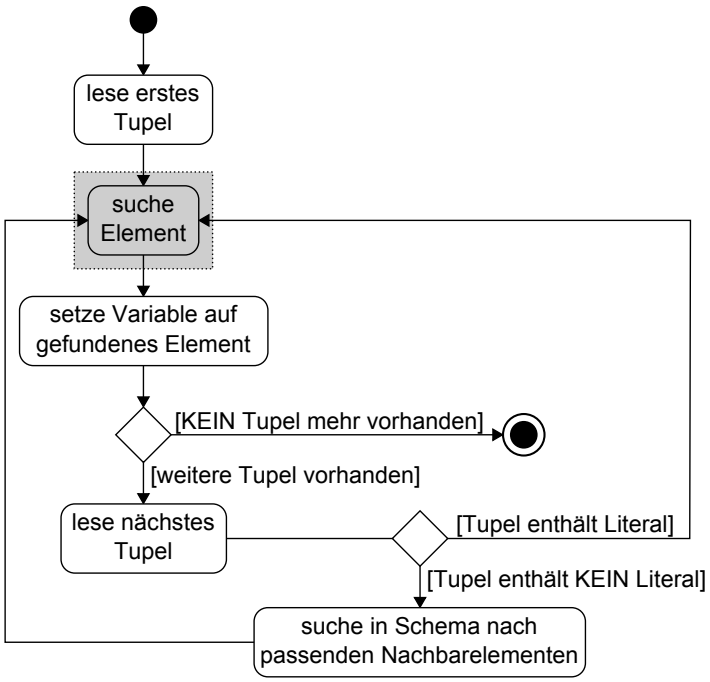


Abbildung 5.9: Aktivitätsdiagramm – Zu suchendes Element ermitteln – Das Diagramm beschreibt die einzelnen Schritte, die abgehandelt werden müssen, um die für die Simulation zu suchenden Elemente zu ermitteln. Der graue Bereich beinhaltet den Suchvorgang, welcher im nächsten Prozess (Abschnitt 5.1.7) genauer beschrieben wird.

Das Aktivitätsdiagramm in Abbildung 5.9 stellt den im Folgenden beschriebenen Ablauf dar. Abbildung 5.10 zeigt für die ersten vier Tupel der Anfrage aus Listing 5.1 und dem Visualisierungsschema aus Abbildung 5.4c beispielhaft die relevanten Elemente und Leseregeln in ihrer zu ermittelnden Reihenfolge.

Um herauszufinden, welche Elemente gesucht werden müssen, wird die Liste mit den Anfrage-Tupeln in der gegebenen Reihenfolge abgearbeitet. Aus der Anfrage im Listing 5.1 folgt beispielsweise, dass zuerst ein Element der Klasse `Beschriftung` mit dem Text „Zeit“ gesucht werden muss. Wurde durch die nachfolgenden Prozesse dieses Element gefunden und fixiert, erfolgt wiederum ein Aufruf des Prozesses, der das nächste zu suchende Element ermittelt. Da die Beschriftung gefunden wurde, wird die Variable `Beschriftung1` auf dieses Element festgelegt (im Falle des Visualisierungsschemas aus Abbildung 5.4c das Element `Beschriftung 2`).

Aus dem zweiten Tupel der Beispielanfrage resultiert, dass als Nächstes ein Element der Klasse `Achse` gesucht werden muss. Da der Typ der Visualisierung und damit die

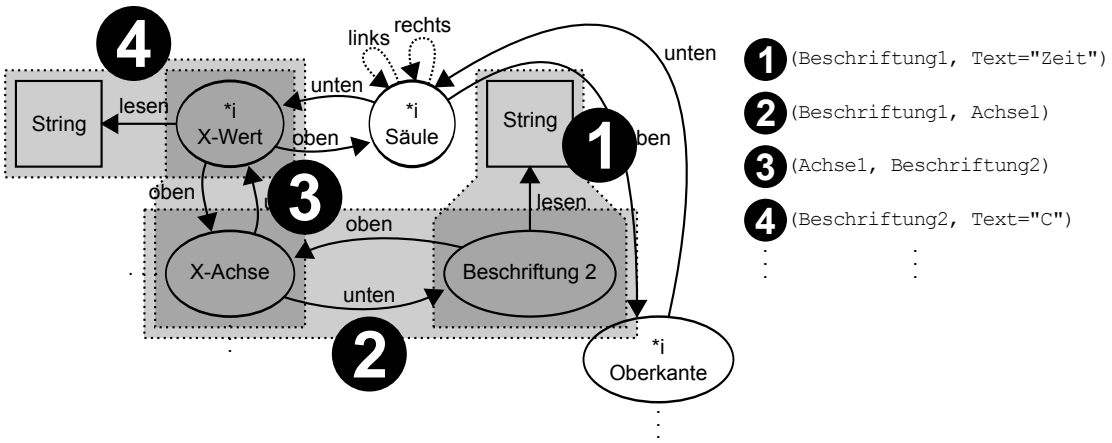


Abbildung 5.10: Zu suchendes Element ermitteln – Die Tupel der Anfrage auf der rechten Seite werden nacheinander abgearbeitet. Im Visualisierungsschema links sind die ermittelten Elemente den einzelnen Tupeln zugeordnet. Die Klassen dieser Elemente werden zusammen mit den verbindenden Leseregeln an den nächsten Prozess weitergereicht, der mithilfe visueller Suche die Elemente im Visualisierungsmodell sucht.

ungefähre Lage einzelner Elemente zueinander bereits bekannt ist, sucht der Prozess im Visualisierungsschema zur Beschriftung das Nachbarelement, welches eine Instanz der Klasse `Achse` ist. Wurde beispielsweise das Schema aus Abbildung 5.4c gewählt, findet der Prozess das Element `X-Achse`. Um dem nachfolgenden Prozess (siehe Abschnitt 5.1.7) eine ungefähre Suchrichtung vorzugeben, wird zusammen mit der zu suchenden Klasse (`Achse`) die Relation zwischen `Beschriftung 2` und `X-Achse` in Form einer Leseregel („Bewege die Augen nach **oben**“) an den nächsten Prozess weitergereicht.

Nachdem das Element gefunden und fixiert wurde, wird der Prozess ein weiteres Mal aufgerufen. Der Variable `Achse1` wird das Element `X-Achse` zugewiesen und mithilfe des nächsten Anfrage-Tupels das nächste zu suchende Element ermittelt. Der zuvor beschriebene Vorgang wiederholt sich, bis alle Anfrage-Tupel abgearbeitet wurden.

Der nächste Abschnitt beschreibt den Prozess, der mithilfe der bisher ermittelten Informationen (Visualisierungstyp/-schema, zu suchendes Element, Suchrichtung) ein Element im Visualisierungsmodell sucht.

5.1.7 Element per visueller Suche suchen

Aufgrund einiger Eigenschaften wie Anordnung, Größe oder Form eines Elements im Visualisierungsmodell wird die Aufmerksamkeit der Augen auf verschiedene Bereiche in

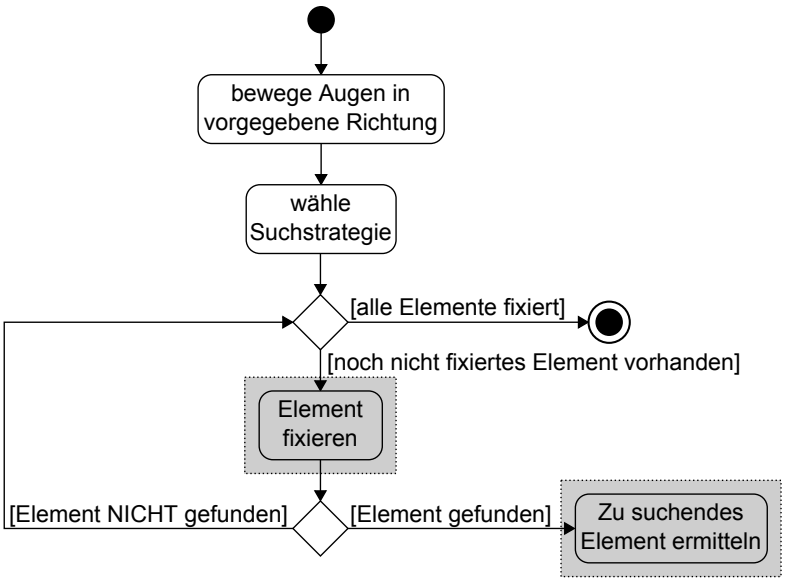


Abbildung 5.11: Aktivitätsdiagramm – Element per visueller Suche suchen – Das Diagramm beschreibt die einzelnen Schritte, die abgehandelt werden müssen, um ein zu suchendes Element im Visualisierungsmodell zu finden. Die grauen Bereiche beinhalten den Fixationsprozess, welcher in Abschnitt 5.1.8 näher betrachtet wird und den bereits beschriebenen Prozess, um im Visualisierungsmodell zu suchende Elemente zu ermitteln (siehe Abschnitt 5.1.6).

der Visualisierung gelenkt. Gerade beim Suchen eines Elements mit einer bestimmten Eigenschaft ist die Suchstrategie hierfür entscheidend über Dauer und Anzahl der Fixationen. Dieser Abschnitt beschreibt den Prozess des Suchens eines zu findenden Elements im Visualisierungsmodell sowie beispielhaft dafür mögliche Suchstrategien.

Der Prozess erhält von den vorigen Prozessen (siehe Abschnitte 5.1.6 und 5.1.5) das zum Visualisierungsmodell passende Visualisierungsschema, ein im Visualisierungsmodell zu suchendes Element und eine Richtung, in die sich die Augen vom aktuellen Punkt aus bewegen müssen, um das Element zu finden.

Das Aktivitätsdiagramm in Abbildung 5.11 verdeutlicht den Prozess, wie das zu suchende Element im Visualisierungsmodell gesucht wird.

Der erste Schritt der Simulation beginnt damit, die Augen in die vom vorigen Prozess übergebene Richtung zu bewegen (sogenannte „Sakkade“). Da die Angabe nicht nummerisch exakt, sondern nominal (z. B. rechts, links, oben, unten) vorliegt, wird dem nächstgelegenen Element in der gegebenen Richtung von der aktuellen Fixation die Aufmerksamkeit der Augen zuteil. Die Dauer der Sakkade ist dabei höher, je größer die

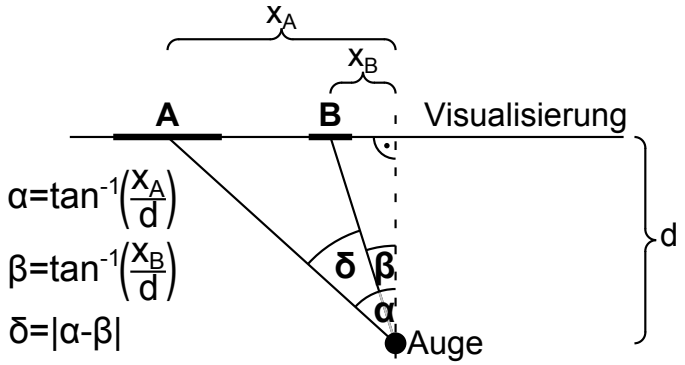


Abbildung 5.12: Umrechnung von Koordinaten in Winkel – Für die Ermittlung der benötigten Zeit für eine Sakkade wird der Winkel benötigt, den die Augen vom aktuell fixierten zum zu fixierenden Element zurücklegen. Mithilfe von Winkelfunktionen, dem Abstand d zwischen den Augen und der Visualisierung lassen sich die Längeninformationen der Visualisierung in Winkel umrechnen.

Distanz des zu fixierenden Punktes von dem aktuell fixierten Punkt ist. Je näher sich der Punkt an der aktuellen Fixation befindet, desto kürzer ist die Bewegungszeit der Augen. Das Modell verwendet hierfür die in [21] beschriebenen Zeiten:

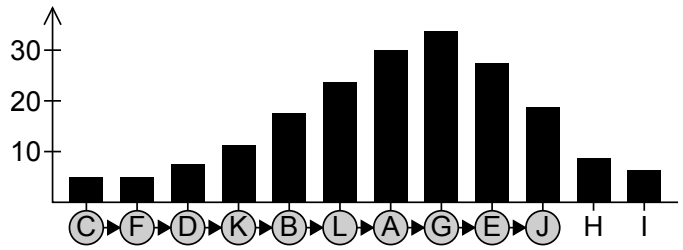
- $150ms$ für die Vorbereitung der Augenbewegung
- $50ms$ für die Festlegung der Muskelbewegungen
- $20ms$ für die Sakkade
- $2ms$ pro Grad, das sich die Augen bewegen müssen

Die Zeit, die eine Sakkade benötigt, ermittelt die Simulation mit folgender Formel, wobei die einzelnen Parameter und die Herleitung aus Abbildung 5.12 ersichtlich sind:

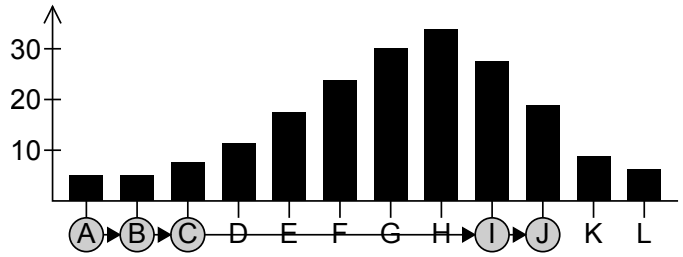
$$t = 220ms + \left|\tan^{-1}\frac{x_A}{d} - \tan^{-1}\frac{x_B}{d}\right| * 2ms \tag{5.1}$$

Dabei gilt:

- $t[ms]$: Zeit zum Ausführen einer Sakkade
- $x_A[cm]$: Abstand des aktuellen Fixationspunkts vom Augenmittelpunkt auf der Visualisierung
- $x_B[cm]$: Abstand des zu fixierenden Fixationspunkts vom Augenmittelpunkt auf der Visualisierung
- $d[cm]$: Abstand des Betrachters von der Visualisierung



(a) Unsortiert – Bei der Suche nach der Säule „J“ beginnt die Simulation die ersten Beschriftungen der horizontalen Achse zu lesen. Da die Beschriftungen offenbar keiner bekannten Folge entsprechen, überprüft die Simulation jede einzelne Beschriftung bis Säule „J“ gefunden wird.



(b) Alphabetisch sortiert – Im sortierten Fall liest die Simulation ebenfalls die ersten Beschriftungen der horizontalen Achse. Nach der dritten Beschriftung tritt die Vermutung auf, dass es sich um alphabetisch sortierte Beschriftungen handelt. Da nach Säule „J“ gesucht werden muss, können dadurch Beschriftungen schätzungsweise übersprungen und Säule „J“ schneller gefunden werden.

Abbildung 5.13: Visuelle Suche bei einem Säulendiagramm – Die Reihenfolge und somit die Suchstrategie der betrachteten Beschriftungen ist abhängig von ihrer Sortierung.

Nachdem die Augen ein Startelement fixiert haben, beginnt die eigentliche Suche nach dem aufzufindenden Element. Hierfür ist vor allem die Reihenfolge, in der die Elemente im Visualisierungsmodell überprüft werden, entscheidend. Es ist äußerst unwahrscheinlich, dass die Elemente in einer zufälligen Reihenfolge kontrolliert werden. Stattdessen werden beispielsweise mehrere nah beieinanderliegende Elemente als eine Gruppe wahrgenommen [44] und innerhalb dieser mit einer bestimmten Suchstrategie gesucht. Soll jedoch z. B. nach einem Element mit der Farbe Rot gesucht werden und alle bis auf ein Element im Visualisierungsmodell sind grün, so kann das rote Element fixiert werden, ohne zuvor andere Elemente überprüft zu haben (präattentive Wahrnehmung).

Der Start jeder Suche im Visualisierungsmodell beginnt mit der Wahl einer geeigneten Suchstrategie. Am Beispiel eines Säulendiagramms werden hier zwei Strategien für das Lesen von Beschriftungen vorgestellt. Abbildung 5.13a zeigt ein Säulendiagramm, bei dem die Beschriftungen der horizontalen Achse keinerlei Ordnung folgen. Die Simulation

würde hier beim Suchen der Säule „J“ alle Beschriftungen von links nach rechts (kulturell bedingte Leserichtung) überprüfen, bis die Beschriftung mit dem Text „J“ gefunden wurde. Das Säulendiagramm in Abbildung 5.13b besitzt alphabetisch sortierte Beschriftungen auf der horizontalen Achse. Die Simulation würde hier ebenfalls von links nach rechts die einzelnen Beschriftungen lesen. Nach einer bestimmten Anzahl (in der Abbildung nach der dritten Beschriftung) überspringt die Simulation einige Beschriftungen, da vermutet wird, dass die Beschriftungen alphabetisch sortiert sind und sich das „J“ somit weiter rechts befindet.

Je nach Anordnung und weiteren visuellen Eigenschaften der Elemente (z. B. Größe, Farbe...) im Visualisierungsmodell wählt die Simulation bei der Suche nach dem aufzufindenden Element eine passende Suchstrategie aus. Die Strategien entsprechen dabei allgemein den Prinzipien der visuellen Suche [40] und sind nicht vom Visualisierungstyp abhängig. Zusätzliche Strategien können problemlos in das Modell eingebaut werden.

Nachdem eine Suchstrategie gewählt wurde, werden die Augenbewegungen mithilfe der Formel 5.1 nacheinander zu den entsprechenden Elementen bewegt und für jedes Element jeweils der nächste Prozess (Abschnitt 5.1.8) abgearbeitet. In diesem werden die Elemente abhängig von ihren Eigenschaften eine bestimmte Zeit lang fixiert und auf Übereinstimmung mit dem zu suchenden Element überprüft (siehe Abschnitt 5.1.8).

5.1.8 Element fixieren

Der letzte Prozess, der für die Simulation benötigt wird, behandelt die eigentliche Fixation des zu fixierenden Elements im Visualisierungsmodell, das durch die vorigen Prozesse ermittelt wurde. Im Einzelnen beinhaltet der Prozess zwei Schritte: Im ersten Schritt wird die Dauer ermittelt, die das zu fixierende Element betrachtet wird. Der zweite Schritt, welcher zeitgleich mit dem ersten Schritt einhergeht, untersucht, ob es sich bei dem fixierten Element um das zu suchende Element handelt.

Die Zeit, die für die Fixation benötigt wird, ermittelt die Simulation mithilfe der Formel 3.1 des UCIE-Modells (siehe Abschnitt 3.1.3).

Das Modell legt zugrunde, dass das Erkennen, um was für ein Element es sich bei dem momentan fixierten Element handelt, kürzer dauert als die Fixation. Da die Erkennung parallel zu der Fixation abläuft, ist die simulierte Zeit der Fixation ausschlaggebend für die Dauer des aktuellen Prozesses. Das Aktivitätsdiagramm in Abbildung 5.14 verdeutlicht den Ablauf des Fixationsprozesses.

Mit den Beschreibungen der einzelnen Komponenten des Eingabe- und Verarbeitungsmodells ist das Simulationsmodell komplett spezifiziert und die erste Phase abgeschlossen. Die zweite Phase beschäftigt sich damit, die Bestandteile des konzeptionellen Modells durch empirische Untersuchungen nachzuweisen.

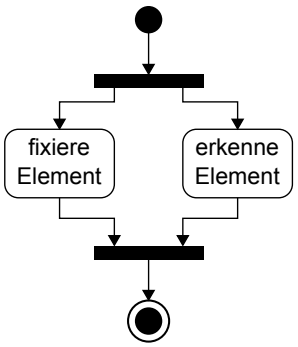


Abbildung 5.14: Aktivitätsdiagramm – Element fixieren – Das Diagramm beschreibt die beiden Schritte, die den Prozess des Fixierens beschreiben.

5.2 Empirische Untersuchung (= Phase 2)

Nach Abschluss der ersten Phase beginnt der Nachweis der Konzepte im Simulationsmodell mithilfe von empirischen Untersuchungen. Hierzu werden die einzelnen Prozesse des erstellten Verarbeitungsmodells betrachtet und Studienvorschläge erbracht, um die Prozesse um noch fehlende Teile zu ergänzen und auf Validität zu überprüfen.

5.2.1 Passendes Schema finden

Der erste Prozess im Simulationsmodell befasst sich damit, ein zu einem gegebenen Visualisierungsmodell passendes Visualisierungsschema zu finden. Hierzu werden alle gespeicherten Schemata nacheinander mit dem Visualisierungsmodell verglichen und jeweils die Anzahl der übereinstimmenden Elemente ermittelt. Zuletzt gibt der Prozess das Visualisierungsschema mit den meisten Übereinstimmungen an den nächsten Prozess weiter (siehe Abschnitt 5.1.5).

Mithilfe eines Experiments muss herausgefunden werden, wie lange ein solcher Suchprozess dauert. Als Stimuli sollten einfache Visualisierungen verschiedener Visualisierungstypen, wie Säulen-, Punkt- oder Liniendiagramme gewählt werden, da hier die einzelnen Visualisierungselemente schnell und objektiv wahrnehmbar sind. Für jeden Visualisierungstyp muss ein Visualisierungsschema vorliegen. Diese Schemata sollten so viele Klassen und Relationen wie möglich enthalten, um den jeweiligen Visualisierungstyp klar von den anderen Typen unterscheiden zu können.

Eine mögliche Studie könnte folgenden Aufbau haben: Die Probanden bekommen nacheinander unterschiedliche Stimuli vorgelegt, worauf sie so schnell wie möglich antworten müssen, um welchen Visualisierungstyp es sich jeweils handelt. Dabei muss den

Probanden im Voraus bekannt sein, welche unterschiedlichen Visualisierungstypen gezeigt werden können. Die Zeit vom Beginn des Vorlegens der Visualisierung bis zum Beginn der Antwort der Probanden wird gemessen und notiert.

Es wird vermutet, dass die Ähnlichkeit zweier Visualisierungstypen Einfluss auf die Antwortzeit hat. Die Ähnlichkeit zweier Visualisierungsschemata wird definiert als die Anzahl übereinstimmender Klassen und Relationen zwischen den Schemata im Verhältnis zur Summe der Anzahl der Klassen und Relationen der Visualisierung und dem jeweiligen Schema. Beruhend auf dieser Annahme könnte z. B. eine Zunahme der Antwortzeit aufgrund starker Ähnlichkeiten zwischen dem korrekten und anderen Visualisierungsschemata feststellbar sein. Mithilfe einer Regressionsanalyse über die Durchführungszeiten aller Probanden und einer quantitativ definierten Ähnlichkeit zwischen den Schemata kann dadurch eine Funktion ermittelt werden, die die Antwortzeit in Abhängigkeit des Visualisierungsschemas mit den meisten Übereinstimmungen mit dem Visualisierungsmodell berechnet.

5.2.2 Zu suchendes Element ermitteln

Der nächste Prozess im Simulationsmodell ermittelt die im Visualisierungsmodell zu suchenden Elemente. Mithilfe der Anfrage (siehe Abschnitt 5.1.4) wird das zuvor ausgewählte Visualisierungsschema mit der in Abschnitt 5.1.6 beschriebenen Methode traversiert und die entsprechenden Klassen an den nächsten Prozess weitergereicht.

Die Feststellung, dass ein Mensch die einzelnen Elemente, die in der Anfrage enthalten sind, in der Visualisierung fixiert und daraus die Frage beantwortet, ist plausibel und kann durch einfache Eye-Tracking-Studien nachgewiesen werden (siehe z. B. Abschnitt 3.1.2). Viel interessanter ist die konkrete Reihenfolge, in der die Klassen traversiert werden. Daraus ergeben sich die Leseregeln in einem Visualisierungsschema (siehe Abbildung 5.10). Um die Leseregeln für ein bestimmtes Schema zu ermitteln, können Untersuchungen mit Probanden durchgeführt werden.

Zur Vorbereitung der Studie gehören mehrere Stimuli des zu untersuchenden Visualisierungstyps sowie unterschiedliche Fragestellungen, die die Probanden aus den Visualisierungen ablesen und beantworten müssen. Mithilfe eines Eye-Trackers werden während den Lesevorgängen die Augenbewegungen der Probanden aufgenommen und auf die einzelnen Elemente des Visualisierungsmodells abgebildet (Transformation des Wo-Raums in den Was-Raum). Die daraus resultierenden Reihenfolgen der Fixationen können nun in Form von Leseregeln in das entsprechende Visualisierungsschema übertragen werden.

5.2.3 Element per visueller Suche suchen

Die Simulation der Suche nach dem zu suchenden Element auf dem Visualisierungsmodell erfolgt mithilfe vorgegebener Strategien und der Simulation von Augenbewegungen, wie es in Abschnitt 5.1.7 beschrieben wurde. Ziele der empirischen Untersuchung bei diesem Teil des Simulationsmodells sind einerseits die Ermittlung der verschiedenen Strategien, andererseits die Verifizierung der Funktion für die Berechnung der Zeitdauer einer Sakkade (siehe Formel 5.1).

Ermittlung von Suchstrategien

Um herauszufinden, welche Strategien ein Mensch bei der Suche nach einem bestimmten Element in einem bestimmten Kontext verfolgt, müssen Studien durchgeführt werden, die unterschiedlichste Visualisierungen mit verschiedenen Fragestellungen untersuchen. Auch hier muss, wie in Abschnitt 5.2.2, ein Eye-Tracker zur Messung der Augenfixationen verwendet werden.

Die Auswertung der Messergebnisse und die Erstellung von Suchstrategien werden sich als schwierig gestalten, da vor allem der Grund für die Wahl einer bestimmten Suchstrategie nicht aus den Fixationen abgeleitet werden kann. Vielmehr müssen bereits bei der Erstellung der Stimuli Vermutungen vorliegen, wonach für eine Visualisierung eine bestimmte Suchstrategie gewählt werden wird. Die Untersuchung wird die Vermutung daraufhin lediglich tendenziell bestätigen oder widerlegen. Eine andere Möglichkeit ist die zur Augenbewegungsmessung zusätzliche Befragung der Probanden mittels Introspektion. Dieses Vorgehen ist jedoch, wie in Abschnitt 2.2.1 beschrieben, umstritten.

Verifizierung der Funktion für die Berechnung der Zeitdauer einer Sakkade

Mithilfe eines Eye-Trackers lässt sich messen, wie lange es dauert, die Augen von einem Punkt zu einem anderen Punkt zu bewegen. Den Probanden einer Studie werden mehrere Punkte mit unterschiedlichen Abständen zueinander und unterschiedlichen Abständen der Augen zur jeweiligen Visualisierung gezeigt. Die Aufgabe der Probanden ist es, die einzelnen Punkte nacheinander zu betrachten. Mit den Messdaten des Eye-Trackers und den vorgelegten Visualisierungen lässt sich die Genauigkeit der Formel 5.1 abschätzen.

5.2.4 Element fixieren

Der letzte zu untersuchende Prozess im Simulationsmodell beschreibt die Fixation eines vorgegebenen Elements im Visualisierungsmodell. Die Fixation besteht hierbei aus zwei Schritten: dem Erkennen des Elements und der Fixation selbst. Beide Schritte werden

gleichzeitig abgehandelt, wobei bereits angenommen wurde, dass der Erkennungsvorgang maximal so lange dauert wie die Fixation (siehe Abschnitt 5.1.8). Herauszufinden gilt es einerseits, wie lange das Erkennen eines Elements tatsächlich dauert, andererseits muss die Funktion für die Berechnung der Fixationszeit (siehe Formel 3.1) auf Gültigkeit überprüft werden.

Bereits die Grundannahme, dass die Erkennung und die Fixation parallel ablaufen, macht die Messung der einzelnen Schritte im entsprechenden Kontext nahezu unmöglich. Stattdessen ist es mit einer Studie relativ einfach realisierbar, die Gesamtzeit für den Prozess zu messen und die Ergebnisse mit der Formel 3.1 zu vergleichen. Hierzu wird, wie bei den vorigen Untersuchungen, ein Eye-Tracker verwendet, wobei den Probanden mehrere Visualisierungen vorgezeigt werden. Diese Visualisierungen beinhalten verschiedene Elemente, die unterschiedliche Ausprägungen in den in der Formel 3.1 verwendeten Variablen aufweisen. Mithilfe der gemessenen Augenbewegungen und den Ausprägungen der Parameter in den Visualisierungen lassen sich Fixationszeiten auf den einzelnen Elementen messen. Die Ergebnisse können daraufhin mit den Vorhersagen der Formel 3.1 verglichen und somit verifiziert werden.

Mit dem Abschluss der empirischen Untersuchungen ist die zweite Phase beendet und die dritte Phase, die Modellimplementierung, wird eingeleitet. Darin wird anhand einzelner Modellkomponenten gezeigt, dass das theoretisch entwickelte Modell in ein lauffähiges Computerprogramm übertragen werden kann.

6 Modellimplementierung (= Phase 3)

Nachdem der Aufbau des Modells durch die erste Phase der Modellierung festgelegt und einzelne Komponenten mithilfe von empirischen Untersuchungen validiert wurden, beginnt die dritte Phase, die Implementierung des Modells in Form eines Prototyps.

Das Kapitel gliedert sich in die Abschnitte „Architektur“ (Abschnitt 6.1), in welchem die einzelnen Komponenten der Implementierung beschrieben werden und „Beschreibung des Simulationstools“ (Abschnitt 6.2), der die grafische Benutzeroberfläche und einzelne Funktionen des Simulationstools vorstellt.

6.1 Architektur

Dieser Abschnitt beschreibt die Architektur des entwickelten Simulationstools mit seinen Kernkomponenten. Nach einer kurzen Übersicht über die wichtigsten Komponenten in Abschnitt 6.1.1 wird in den Abschnitten 6.1.2 und 6.1.3 vorgestellt, wie neue Visualisierungselemente und Simulationen für das Simulationstool entwickelt werden können.

6.1.1 Übersicht

Das Simulationstool ist eine WPF-Anwendung und wurde in C# entwickelt. Bei der Implementierung wurde großer Wert auf die Erweiterbarkeit gelegt, weshalb eine Plugin-Architektur mit MEF[1] angestrebt wurde. Die im Lösungskonzept beschriebenen Visualisierungsschemata werden dabei ebenso durch Plugins (`VisualizationSimulation-Plugins` – siehe Abschnitt 6.1.3) umgesetzt wie die existierenden Visualisierungselemente in der Ontologie (`VisualObject-Plugins` – siehe Abschnitt 6.1.2). Abbildung 6.1 verdeutlicht den Zusammenhang der Hauptkomponenten des Simulationstools.

6.1.2 VisualObject-Plugins

Jedes `VisualObject-Plugin` definiert ein Visualisierungselement und besteht jeweils aus einer Klasse, welche die abstrakte Klasse `VisualObject` des `Plugin-Interfaces` erwei-

[1] MEF = Managed Extensibility Framework – Framework zur Erstellung von erweiterbaren Anwendungen. MEF ist eine Komponente des .NET Frameworks 4.0.

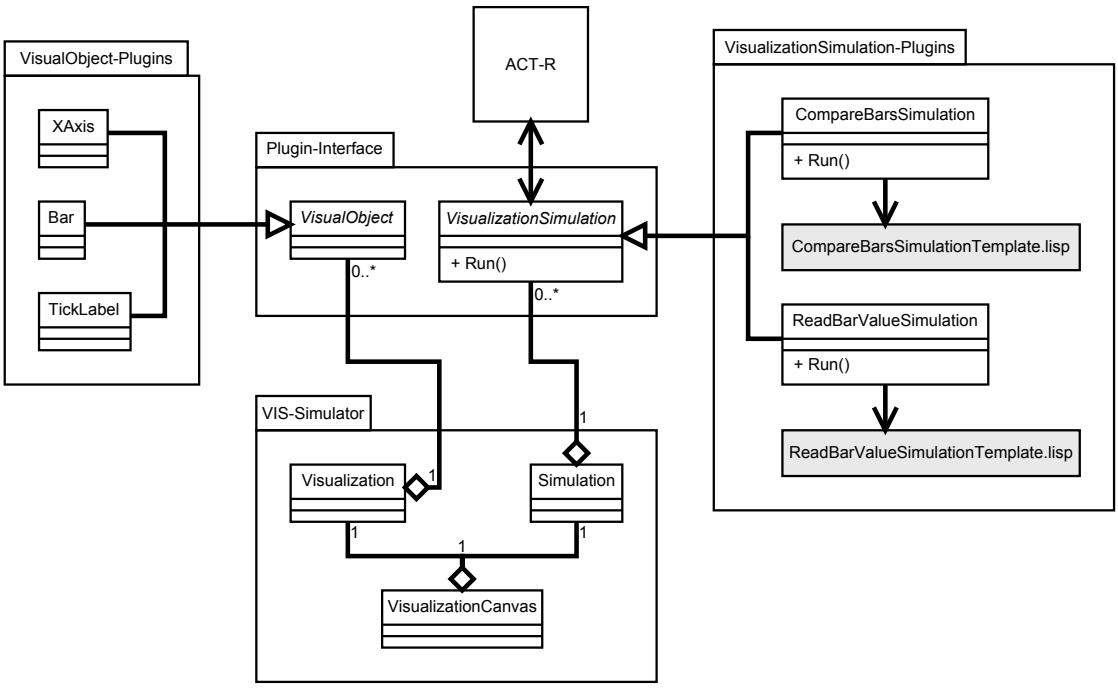


Abbildung 6.1: Architektur des Simulationstools – Das Simulationstool ist in vier Hauptkomponenten aufgeteilt. Die `VisualObject-Plugins` beschreiben die Visualisierungselemente, die in einem Visualisierungsmodell vorkommen können, und bilden somit einen Teil der Ontologie. Die `VisualizationSimulation-Plugins` entsprechen den Visualisierungsschemata des Modells. Für die Verwaltung der einzelnen Plugins, der Anzeige und Benutzerinteraktion dienen die beiden Komponenten `VIS-Simulator` und `Plugin-Interface`.

tert. In Abbildung 6.2 sind die Funktionen aufgelistet, welche von der `VisualObject-Plugin`-Klasse implementiert werden müssen.

Im Folgenden werden die einzelnen Funktionen und Properties beschrieben:

OntologyId: Diese Property muss den eindeutigen Namen des Visualisierungselements aus der zuvor definierten Ontologie zurückgeben (z. B. „Säule“, „X-Achse“).

Create(): Mithilfe dieser Funktion werden neue Visualisierungselemente erstellt. Dies ist nicht über den Konstruktor der `VisualObject-Plugin`-Klasse möglich, da durch EMF bereits beim Laden des Plugins eine Instanz dieser Klasse erzeugt wird.

Die ersten vier Parameter (`x1`, `y1`, `x2`, `y2`) beschreiben die Position und die Ausmaße des Visualisierungselements. Über den Parameter `color` wird die Farbe des

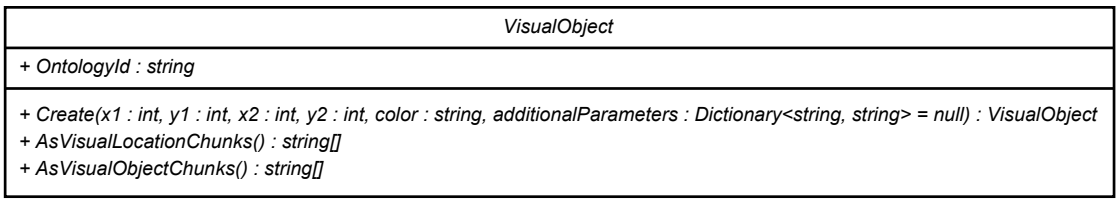


Abbildung 6.2: Zu implementierende Funktionen und Properties für ein `VisualObject-Plugin` – Eine `VisualObject-Plugin`-Klasse muss die abstrakte Klasse `VisualObject` erweitern und die abgebildeten Funktionen und Properties implementieren.

Elements in einen für ACT-R interpretierbaren String (z. B. „black“, „red“) festgelegt. Weitere elementspezifische Parameter können mit einer Name-Wert-Liste über den Parameter `additionalParameters` angegeben werden.

AsVisualLocationChunks(): Diese Funktion gibt die für ACT-R benötigten VisualLocation-Chunks für das Visualisierungselement in Form von Strings zurück. Für eine Säule in einem Säulendiagramm, könnte die Funktion beispielsweise die beiden folgenden Chunks zurückgeben:

> „(ISA visual-location screen-x 93 screen-y 410 kind bar value bar width 55 height 184 color black)“
>
> „(ISA visual-location screen-x 93 screen-y 318 kind bar-top value bar-top width 55 height 1 color black)“

AsVisualObjectChunks(): Die Funktion wird aufgerufen, um die für ACT-R benötigten VisualObject-Chunks für das Visualisierungselement in Form von Strings zu erhalten. Für eine Säule in einem Säulendiagramm könnte die Funktion beispielsweise die beiden folgenden Chunks zurückgeben:

> „(ISA bar value ’Säule’ width 55 height 184 color black screen-x 93 screen-y 410)“
>
> „(ISA bar-top value ’Säulenoberkante’ width 55 height 1 color black screen-x 93 screen-y 318)“

Die Klasse `VisualObject` beinhaltet außerdem Hilfs-Properties, welche z. B. aus der Position und Größe (Höhe und Breite) eines Elements dessen Mittelpunkt berechnen und zurückgeben.

6.1.3 VisualizationSimulation-Plugins

Für jede Kombination aus Visualisierungstyp und -simulation muss je ein `VisualizationSimulation-Plugin` erstellt werden. Dieses generiert für gegebene Visualisierungselemente ACT-R-Code, welcher daraufhin ausgeführt wird. Ein `VisualizationSimulation-Plugin` enthält immer eine Klasse, welche die abstrakte Klasse `VisualizationSimulation` im `Plugin-Interface` erweitert. Abbildung 6.3 zeigt die zu implementierenden Funktionen und Properties der abstrakten Klasse.

Im Folgenden werden die einzelnen Funktionen und Properties beschrieben:

VisualizationName: Diese Property gibt den Namen des Visualisierungstyps zurück (z. B. „Säulendiagramm“). Der Name erscheint in der Benutzeroberfläche des Simulationstools zusammen mit den Bezeichnungen der anderen geladenen Visualisierungstypen zur Auswahl in einer Drop-down-Liste (siehe Abbildung 6.6 (3)).

SimulationName: Diese Property gibt den Namen des Simulationstyps bzw. des Aufgabentyps zurück (z. B. „Säulen vergleichen“, „Wert ablesen“). Der Name erscheint zusammen mit den Namen der anderen Simulationstypen des gleichen Visualisierungstyps in einer Drop-down-Liste (siehe Abbildung 6.6 (3)).

Control: Um für den Simulationstyp zusätzliche Parameter angeben zu können (z. B. welche Säule in einem Säulendiagramm gesucht werden soll), gibt diese Property ein `UserControl` zurück, in welches der Benutzer die entsprechenden Werte eingeben kann (siehe Abbildung 6.6 (3)). Die Property ist nicht abstrakt und muss bei nicht benötigten Parametern nicht überschrieben werden. Im nicht implementierten Fall werden keine zusätzlichen Parameter in der Benutzeroberfläche angezeigt.

buildLispCode(): Mithilfe dieser Funktion wird der zum Simulationstyp gehörende ACT-R-Code erzeugt. Da dieser bis auf die Definitionen der VisualLocation-Chunks und VisualObject-Chunks für den Simulationstyp immer gleich bleibt, eignet es sich, hierfür den ACT-R-Code mit allen benötigten Produktionsregeln als Vorlage (Template) zu erstellen. Die an die Funktion übergebenen Chunks werden beim Generieren des ACT-R-Codes in die entsprechenden Stellen im Template eingefügt und der so erzeugte Code zurückgegeben (siehe auch „Produktionsregeln“ im anschließenden Beispiel-Template). Am Ende dieses Abschnitts wird anhand eines Beispiels gezeigt, wie ein solches ACT-R-Template aufgebaut sein kann.

Das Sequenzdiagramm in Abbildung 6.4 verdeutlicht, wie eine Simulation auf einem Visualisierungsmodell abläuft. Der Aufruf der Funktion `Run()` in einer `VisualizationSimulation`-Instanz führt dazu, dass zuerst mithilfe der übergebenen Chunks ein entsprechender ACT-R-Code erzeugt wird. Daraufhin startet das Simulationstool einen Thread, welcher das ACT-R-Programm im Hintergrund startet und dessen Ausgaben abfragt. Immer wenn aus diesen Ausgaben Augenbewegungen hervorgehen, werden die-

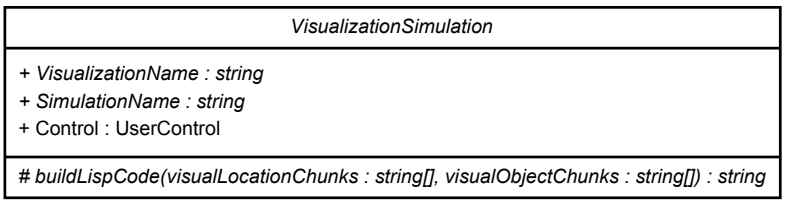


Abbildung 6.3: Zu implementierende Funktionen und Properties für ein `Visualization-Simulation-Plugin` – Eine `VisualizationSimulation-Plugin`-Klasse muss die Klasse `VisualizationSimulation` erweitern und die abgebildeten Funktionen und Properties implementieren.

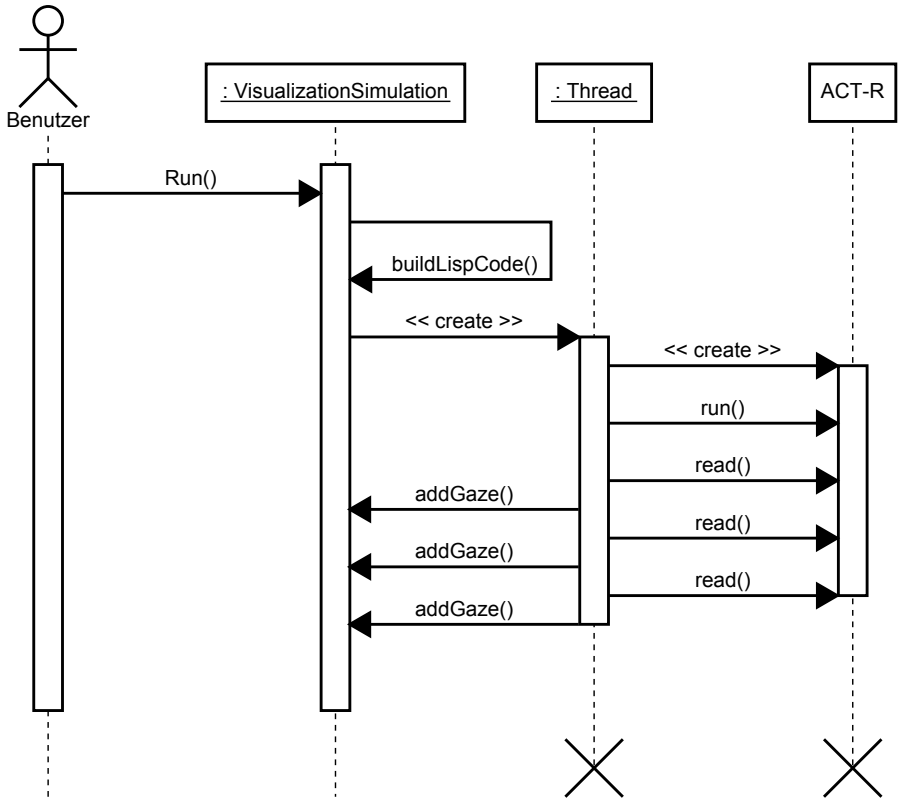


Abbildung 6.4: Ablauf für die Simulation – Das Sequenzdiagramm verdeutlicht den Ablauf in einer `VisualizationSimulation` während einer Simulation. Nachdem der entsprechende ACT-R-Code erzeugt wurde (`buildLispCode()`), startet ein Thread, welcher den entsprechenden Code ausführt und die Kommunikation mit ACT-R sicherstellt.

se an die Simulation weitergeleitet, wo Sakkaden erstellt und dem Benutzer in Form von Linien grafisch dargestellt werden (siehe auch Abbildung 6.6 (6)).

Anhand eines Beispiels wird im Folgenden beschrieben, wie das ACT-R-Template für das Ablesen der Höhe einer Säule in einem Säulendiagramm umgesetzt wurde.

ACT-R-Template – Beispiel (Höhe einer Säule ablesen)

Dieser Abschnitt soll beispielhaft zeigen, wie der ACT-R-Code für die Simulation zum Ablesen der Höhe einer Säule in einem Säulendiagramm aufgebaut sein kann. Dazu werden Platzhalter verwendet, welche beim Starten einer Simulation im Simulationstool durch bestimmte Teile ersetzt werden.

Das Grundgerüst eines jeden ACT-R-Programms wurde bereits in Abschnitt 2.4.5 beschrieben. Die dort vorgestellten Einzelteile werden im Folgenden mit beispielhaften Inhalten gefüllt.

Modellname: Der Name des Modells kann frei gewählt werden und hat keinen Einfluss auf die Gesamtfunktion. Im Beispiel soll das Modell „read-bar-value“ heißen.

Modulparameter: Die meisten Parameter können auf den Standardwerten belassen werden. Die beiden Parameter `:viewing-distance` und `:pixels-per-inch` werden im Modell festgelegt:

Listing 6.1: Modulparameter des Beispielmodells

```
1 (sgp :viewing-distance {VIEWING_DISTANCE} :pixels-per-inch {
    PIXELS_PER_INCH})
```

`:viewing-distance` gibt den Abstand in Zoll an, den der Betrachter von der Visualisierung – in dem Fall vom Bildschirm – hat. `:pixels-per-inch` definiert die Auflösung des Bildschirms in Pixel pro Zoll. Im Template sind die Werte mit den beiden Platzhaltern „{VIEWING_DISTANCE}“ und „{PIXELS_PER_INCH}“ versehen. Durch Ersetzen der Platzhalter ist es möglich, die Werte vom Simulationstool aus zu setzen und somit variabel zu halten.

Chunk-Types: Als Chunk-Types müssen alle in der Simulation und somit in den Produktionsregeln verwendeten Visualisierungselemente mit ihren individuellen Eigenschaften (in ACT-R: „Slots“) vorhanden sein. Für das Lesen der Höhe einer Säule in einem Säulendiagramm sind folgende Chunk-Types notwendig:

Listing 6.2: Chunk-Types des Beispielmodells

```
1  (chunk-type (axis (:include visual-object)) type)
2  (chunk-type (tick-label (:include text)) axis)
3  (chunk-type (bar (:include visual-object)) screen-x screen-y)
4  (chunk-type (bar-top (:include visual-object)) screen-x
       screen-y)
5  (chunk-type goal state abscissa x-axis-screen-y)
```

Prinzipiell können die Chunk-Types als Teil des Visualisierungsschemas betrachtet werden. Sie legen die Teilmenge der Visualisierungselemente aus der Ontologie fest, die das Visualisierungsmodell verwendet.

Der Chunk-Type `goal` enthält die drei Slots `state`, `abscissa` und `x-axis-screen-y`, in welchen während der Simulation Wissen gespeichert wird. Der Slot `state` enthält dabei immer den aktuellen Schritt, der auszuführen ist, und gibt damit den grundsätzlichen Simulationsablauf vor. `abscissa` speichert die zu suchende Skalenbeschriftung unter der horizontalen Achse des Säulendiagramms. `x-axis-screen-y` wird in der Simulation in einem der ersten Schritte gesetzt. In diesem Slot wird die Y-Koordinate der horizontalen Achse des Säulendiagramms gespeichert. Diese Information beschleunigt das Auffinden der Skalenbeschriftungen.

Deklaratives Gedächtnis: Das deklarative Gedächtnis enthält im Beispielmodell alle Zwischenziele, die zum Lesen der Höhe einer Säule notwendig sind. Dies sind einfache Chunks, welche keine zusätzlichen Slots besitzen. Für die Unterscheidung der X- und Y-Achse werden außerdem zwei weitere Chunks definiert, die als Werte für den Slot `type` im Chunk-Type `axis` dienen. In folgendem Listing sind die verwendeten Chunks für das Modell aufgelistet:

Listing 6.3: Chunks des deklarativen Gedächtnisses im Beispielmodell

```
1  (find-x-axis          ISA chunk)
2  (find-abscissa        ISA chunk)
3  (find-next-abscissa   ISA chunk)
4  (find-bar             ISA chunk)
5  (find-bar-top         ISA chunk)
6  (find-ordinate-label  ISA chunk)
7  (read-value           ISA chunk)
8  (x-axis               ISA chunk)
9  (y-axis               ISA chunk)
```

Außerdem werden im deklarativen Gedächtnis alle VisualLocation- und VisualObject-Chunks des Visualisierungsmodells abgelegt. Dies geschieht durch folgenden Code:

Listing 6.4: VisualLocation- und VisualObject-Chunks

```
1  (define-chunks {VISUAL_LOCATION_CHUNKS})
2  (define-chunks {VISUAL_OBJECT_CHUNKS})
```

Die beiden Platzhalter „{VISUAL_LOCATION_CHUNKS}“ und „{VISUAL_OBJECT-_CHUNKS}“ werden im Simulationstool durch die im geladenen Visualisierungsmodell enthaltenen Visualisierungselemente ersetzt.

Um ein initiales Ziel setzen zu können, muss dieses ebenfalls zuvor als Chunk im deklarativen Gedächtnis erstellt werden:

Listing 6.5: Ziel-Chunk des Beispielmodells

```
1  (g1 ISA goal state find-x-axis abscissa "{X_VALUE}")
```

Das erste Ziel ist somit das Auffinden der horizontalen Achse (`x-axis`). Im Ziel-Chunk ist außerdem die zu suchende Skalenbeschriftung gespeichert. Dies wird mithilfe eines Platzhalters („{X_VALUE}“) umgesetzt, damit der Wert durch das Simulationstool dynamisch vorgegeben werden kann.

Produktionsregeln: Die Produktionsregeln bilden den Hauptteil des ACT-R-Templates. Sie steuern die Augenbewegungen und können somit als einen weiteren Teil des Visualisierungsschemas (siehe „Leseregeln“, Abschnitt 5.1.2) betrachtet werden. Da genau ein Template für jede Simulation in einem Visualisierungsschema existiert, sind die Produktionsregeln fix und nicht, wie z. B. die Modulparameter oder VisualLocation- bzw. VisualObject-Chunks, durch Platzhalter angegeben.

Abbildung 6.5 zeigt die einzelnen Produktionsregeln und deren Übergänge in Form eines Aktivitätsdiagramms.

Ziel: Das Ziel wurde bereits bei der Beschreibung des deklarativen Gedächtnisses erstellt. Mit folgendem Code wird das Ziel in den `goal-buffer` geladen:

Listing 6.6: Ziel des Beispielmodells

```
1  (goal-focus g1)
```

Produktionsparameter: Das Modell verwendet die vorgegebenen Parameter bei allen Produktionen. Dadurch entfällt die Definition dieser Parameter beim Beispielmodell.

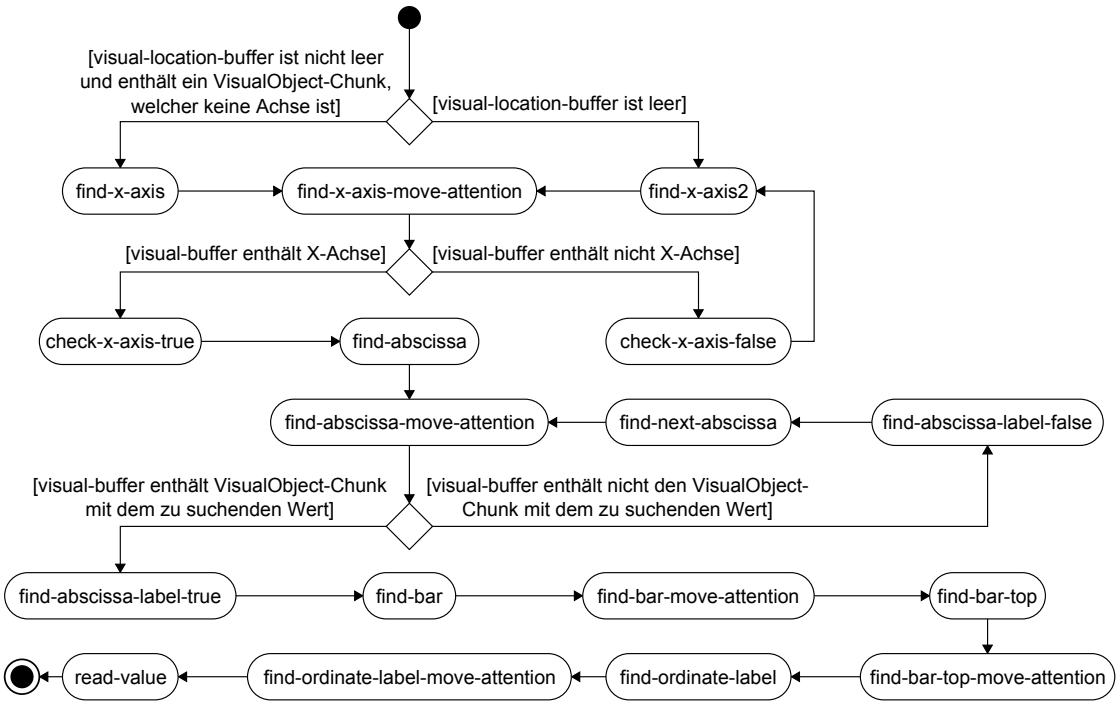


Abbildung 6.5: Produktionsregeln für das Bestimmen der Höhe einer Säule – Das Aktivitätsdiagramm zeigt die einzelnen Produktionsregeln sowie deren Übergänge, die benötigt werden, um die Höhe einer Säule in einem Säulendiagramm zu bestimmen.

6.2 Beschreibung des Simulationstools

Dieser Abschnitt beschreibt die Benutzeroberfläche und die einzelnen Funktionen des entwickelten Prototyps für die Simulation des Modells, das in den vorigen Abschnitten vorgestellten wurde.

Abbildung 6.6 zeigt die grafische Benutzeroberfläche des Simulationstools. Im Folgenden werden die einzelnen Komponenten detailliert beschrieben:

1 – Hauptmenü: Über den Menüpunkt „Motif-Datei laden“ kann eine Motif-Datei geladen werden. Eine Motif-Datei enthält ein Bild, mehrere MOs[2], welche als Visualisierungselemente interpretiert werden und AOIs[3], die z. B. für die Weiterverarbeitung der simulierten Daten verwendet werden können. Eine solche Datei kann mit dem Tool cEdit erstellt werden.

[2] MO = Semantisches Objekt
[3] AOI = Area of Interest

Beim Klick auf den Menüpunkt „Einstellungen“ öffnet sich ein Fenster, über das sich unter anderem Parameter für ACT-R einstellen lassen (siehe Abbildung 6.7).

2 – Ansicht: Mithilfe der Checkbox „Objektrahmen anzeigen“ ist es möglich, die in der Motif-Datei annotierten MOs grafisch hervorzuheben.

3 – Simulationsparameter: Über die Drop-down-Listen „Visualisierungstyp“ und „Simulationsart“ lässt sich festlegen, was auf der aktiven Visualisierung simuliert werden soll. In der Abbildung soll die Simulation beispielsweise die Visualisierung als Säulendiagramm interpretieren und zwei Säulen miteinander vergleichen.

Je nach Auswahl der beiden Drop-down-Listen werden unterschiedliche Parameter angezeigt. Im Beispiel der Abbildung müssen zwei Skalenbeschriftungen der X-Achse angegeben werden. In der Simulation werden diese später gesucht und die entsprechenden Säulen miteinander verglichen.

4 – Steuerbuttons: Beim Klick auf den Start-Button wird die Simulation mit den eingegebenen Parametern auf der geladenen Visualisierung ausgeführt.

Nachdem die Simulation durchgelaufen ist, können die ermittelten Fixationen mithilfe des Buttons „Exportieren“ abgespeichert werden. Hierbei öffnet sich ein vom Betriebssystem abhängiger Speichern-Dialog, in dem zwischen CSV[4]- und Bild-Export (im PNG-Format) gewählt werden kann. In der erstellten CSV-Datei sind zu jedem berechneten Zeitpunkt die Augenposition in Pixel sowie die an dieser Stelle befindliche MO und AOI angegeben. Die Bilddatei enthält das in der Motif-Datei definierte Bild mit den eingezeichneten Fixationen und Sakkaden, das mit dem im Simulationstool angezeigten Bild identisch ist (siehe Abbildung 6.6 (6)).

5 – Log: Hier werden alle ACT-R-Ausgaben angezeigt. Aus diesen ist unter anderem ablesbar, welche Produktionsregeln wann ausgeführt wurden und wie lange die Simulation gedauert hat (im Beispiel der Abbildung 3,313 Sekunden).

6 – Live-Anzeige der Simulation: Dieser Bereich zeigt die geladene Visualisierung an. Während des Simulationsablaufs werden dort die einzelnen Fixationen und Sakkaden gezeichnet. Ist die Checkbox „Objektrahmen anzeigen“ aktiviert, sind zudem die Bereiche der MOs grafisch hervorgehoben.

Das Einstellungsfenster, welches beim Klicken auf den Menübutton „Einstellungen“ erscheint, ist in Abbildung 6.7 dargestellt. Die einzelnen Teile der Benutzeroberfläche werden im Folgenden beschrieben:

1 – Pfade: In den beiden Eingabefeldern „Lisp-Compiler“ und „ACT-R“ müssen die Pfade für den installierten Lisp-Compiler (im Beispiel der Abbildung wurde Clozure CL verwendet) und die ACT-R-Installation angegeben werden.

[4]CSV = Comma-separated values: Dateiformat zur Abspeicherung tabellarischer Daten

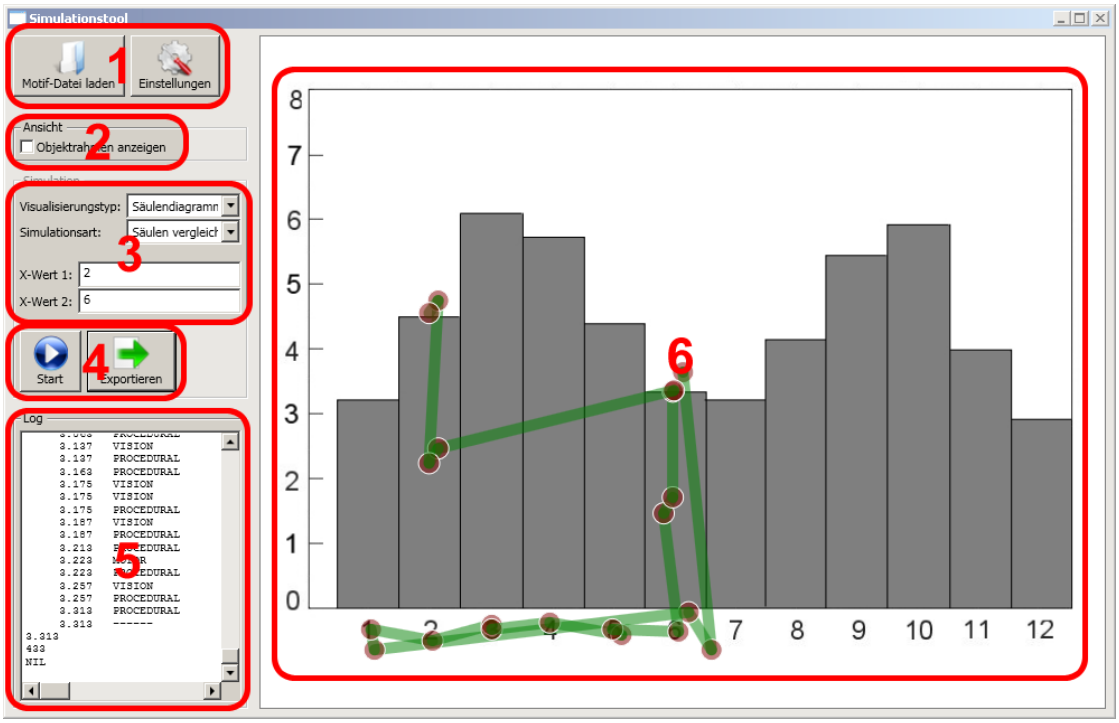


Abbildung 6.6: Aufbau der Benutzeroberfläche des Simulationstools – Die Oberfläche besteht aus den Bereichen „Hauptmenü“ (1), „Ansicht“ (2), „Simulationsparameter“ (3), „Steuerbuttons“ (4), „Log“ (5) und „Live-Anzeige der Simulation“ (6).

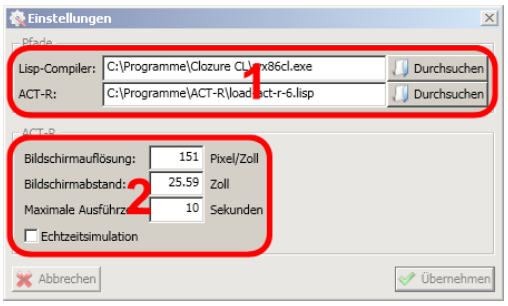


Abbildung 6.7: Einstellungsfenster des Simulationstools – Das Einstellungsfenster besteht aus den beiden Teilen „Pfade“ (1) und „ACT-R“ (2), über die die Grundeinstellungen des Tools vorgenommen werden können.

2 – ACT-R-Einstellungen: Mithilfe der hier eingegebenen Daten werden einzelne Platzhalter in den Templates für die ACT-R-Simulationen ersetzt und Startparameter für die Simulation an ACT-R übergeben. In dem Eingabefeld „Bildschirmauflösung“ wird die Auflösung des anzeigenden Bildschirms in Pixel pro Zoll eingegeben (Im Beispiel der Abbildung 151 Pixel/Zoll[5]). Der Bildschirmabstand gibt den Abstand des Betrachters zum Bildschirm in Zoll an (im Beispiel der Abbildung 25,59 Zoll $\approx$ 65 cm).

Da ACT-R beim Ausführen der Produktionsregeln theoretisch in eine Endlosschleife gelangen kann (siehe Halteproblem [39]), wird die Angabe einer maximalen Ausführungszeit in Sekunden benötigt. Diese sollte grundsätzlich 5-10 Sekunden höher gewählt werden, als die Simulation voraussichtlich dauern wird. Kann dies nicht abgeschätzt werden, muss die Maximalzeit schrittweise erhöht werden, bis die Simulationszeit darunter liegt.

Wird die Checkbox „Echtzeitsimulation“ aktiviert, führt ACT-R die Simulation in Echtzeit aus. Die einzelnen Fixationen dauern in der Live-Anzeige dann genau so lange, wie sie bei einem Menschen dauern würden. Ist die Checkbox nicht aktiviert, simuliert ACT-R das Modell in hardwareabhängiger Minimalzeit. Dies hat jedoch keinen Einfluss auf die vorhergesagte Zeit im Log-Bereich.

Die Bedienung des Simulationstools erfolgt grundsätzlich in fünf Schritten:

1. Eine z. B. mit cEdit erstellte Motif-Datei wird geladen.
2. Der Typ der Visualisierung und die Simulationsart werden ausgewählt.
3. Parameter, die für die entsprechende Kombination aus Visualisierung und Simulationsart notwendig sind, werden angegeben.
4. Die Simulation wird mit einem Klick auf „Start“ ausgeführt.
5. Die ermittelten Fixationen werden über den Button „Exportieren“ abgespeichert und können daraufhin z. B. mit anderen Tools weiterverarbeitet werden.

Da ACT-R während der Simulation stochastische Parameter einfließen lässt, können die Fixationen und Zeiten bei jeder Simulationsausführung abhängig vom Modell um einige Millisekunden variieren. Für eine genaue Untersuchung bietet es sich deshalb an, eine Simulation mehrfach auszuführen und z. B. Mittelwerte weiter zu verwenden.

Im nächsten Kapitel wird mithilfe einer Studie die letzte Phase der Modellierung abgeschlossen: Es soll untersucht werden, inwiefern die durch das Simulationstool vorhergesagten Fixationen und Zeiten mit tatsächlich gemessenen Werten übereinstimmen.

[5]Hierbei handelt es sich um ein 24”-Display mit einer Pixel-Anzahl von 1920 x 1200 woraus durch Rechnung eine Auflösung von 151 Pixel/Zoll resultiert.

7 Adäquatheitsprüfung (= Phase 4)

Nachdem die ersten drei Phasen abgeschlossen sind, wird die letzte Phase der Modellierung, die Adäquatheitsprüfung, bearbeitet. Dazu wurde eine Studie durchgeführt, um die gemessenen Werte mit den vom entwickelten Simulationstool vorhergesagten Zeiten und Fixationen zu vergleichen.

Nach einer kurzen Einleitung in Abschnitt 7.1 wird die Studie mit ihrem Aufbau und der Durchführung beschrieben (siehe Abschnitt 7.2). Am Ende des Kapitels werden die Ergebnisse vorgestellt, mit den simulierten Daten verglichen (siehe Abschnitt 7.3) und analysiert (siehe Abschnitt 7.4).

7.1 Einleitung

Mithilfe einer Studie soll herausgefunden werden, ob das entwickelte Simulationstool korrekte Vorhersagen für das Lesen von Visualisierungen macht. Hierzu stehen zwei zu Fragen im Vordergrund:

- Stimmen die simulierten Fixationspunkte bzw. Sakkaden für einzelne Visualisierungen mit realen Augenbewegungen überein?
- Stimmen die vorhergesagten Gesamtzeiten zum Beantworten von Aufgaben mit den realen Zeitmessungen der Studie überein?

Zur Beantwortung dieser Fragen wurde eine Pilot-Studie mit sieben Probanden durchgeführt. Um starke Abweichungen zwischen den Messungen der Probanden ausreichend zu berücksichtigen, muss eine größere Studie durchgeführt werden. Der Ablauf kann sich jedoch an der hier vorgestellten Studie orientieren.

7.2 Studie

Bei der Studie handelt es sich um ein Eye-Tracking-Experiment, bei dem mehreren Probanden unterschiedliche Fragen zu einzelnen Visualisierungen gestellt werden. Die hierbei aufgezeichneten Augenbewegungen sollen Aufschluss darüber geben, ob das Simulationstool korrekte Vorhersagen macht.

	Proband 1	Proband 2	Proband 3	Proband 4
Geschlecht	männlich	männlich	männlich	männlich
Alter $[a]$	26	26	23	24
Deutschsprachig	ja	ja	ja	ja
Letzte Mathematiknote in der Schule	3	1	2	4
Aktuelle Mathematiknote	3	-	3	3
Wöchentliche Computerspielzeit $[h]$	4	8	2	10
Wöchentliche Computerarbeitszeit $[h]$	20	40	25	56
Sehhilfe	nein	nein	Brille	nein
Abschluss	Diplom	Diplom	allg. Hochschulreife	allg. Hochschulreife
Studienfach	Softwaretechnik	Softwaretechnik	Informatik	Automotive Systems Engineering
Nebenfächer/Anwendungsfach	Verkehrstechnik	Automatisierung	Computerlinguistik	-
Angestrebter Abschluss	-	-	Diplom	Bachelor
Vorkenntnisse im Bereich der Visualisierung	ja	ja	ja	nein

Tabelle 7.1: Probanden 1–4 der Pilot-Studie

Dieser Abschnitt wird zuerst die Gruppe der einzelnen Probanden vorstellen (siehe Abschnitt 7.2.1). Daraufhin werden der genaue Versuchsaufbau (siehe Abschnitt 7.2.2) und die unterschiedlichen Aufgaben (siehe Abschnitt 7.2.3) beschrieben. Zum Schluss wird die Durchführung der Studie geschildert (siehe Abschnitt 7.2.4).

7.2.1 Probanden

Bei der Studie haben eine weibliche und sechs männliche Probanden im Alter zwischen 23 und 26 Jahren teilgenommen. Alle Probanden waren deutschsprachig und sind Studenten oder haben einen studentischen Abschluss. Die genauen Informationen über die sieben Probanden sind den Tabellen 7.1 und 7.2 entnehmbar.

	Proband 5	Proband 6	Proband 7
Geschlecht	männlich	weiblich	männlich
Alter $[a]$	24	25	26
Deutschsprachig	ja	ja	ja
Letzte Mathematiknote in der Schule	3	4–5	3
Aktuelle Mathematiknote	2	4–5	2
Wöchentliche Computerspielzeit $[h]$	18	0	2
Wöchentliche Computerarbeitszeit $[h]$	25	2-3	50
Sehhilfe	nein	Brille	nein
Abschluss	allg. Hochschulreife	Staatsexamen	Diplom
Studienfach	Informatik	Deutsch/Geographie	Immobilientechnik & Immobilienwirtschaft
Nebenfächer/Anwendungsfach	Techn. Kybernetik	-	Finanzwirtschaft
Angestrebter Abschluss	Diplom	-	-
Vorkenntnisse im Bereich der Visualisierung	ja	nein	nein

Tabelle 7.2: Probanden 5–7 der Pilot-Studie

7.2.2 Aufbau

Für die Durchführung der Studie wurde der Eye-Tracker Tobii T60XL verwendet. Dieser hat eine Bildschirmdiagonale von $24''$ und eine Auflösung von 1920 x 1200 Pixel. Die maximale Abtastrate zur Messung der Augenpositionen beträgt $60Hz$, sodass etwa alle $17ms$ eine Messung stattfindet. Der Abstand der Probanden zum Bildschirm des Eye-Trackers wurde mithilfe einer Kopfstütze konstant gehalten und betrug ca. $65cm$ (siehe Abbildung 7.1).

7.2.3 Aufgaben

Jeder Proband bekam in der Studie neun verschiedene Säulendiagramme gezeigt. Zu jedem dieser Diagramme gab es jeweils die folgenden drei Aufgabenstellungen, wodurch jeder Proband 27 Fragen zu beantworten hatte (siehe Tabelle 7.3):

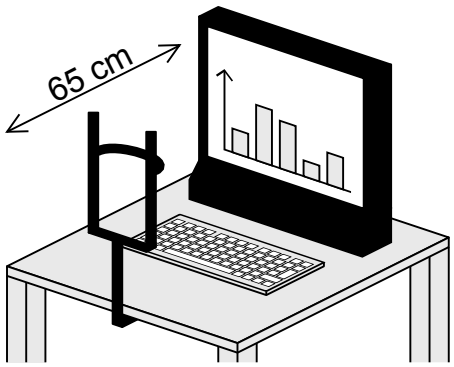


Abbildung 7.1: Studienaufbau – Der Eye-Tracker Tobii T60XL und eine Kopfstütze in einem Abstand von 65cm zueinander.

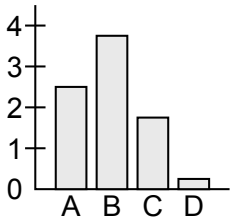


Abbildung 7.2: Studienaufgabe – Höhe einer Säule bestimmen – Die Probanden durften nur Werte angeben, die auf der vertikalen Achse des Säulendiagramms vorhanden waren (eine Interpolation zwischen zwei Werten war nicht verlangt). Die Höhen der einzelnen Säulen in der Abbildung sind: $A = 2$ oder 3, $B = 4$, $C = 2$, $D = 0$

Welche Höhe hat Säule . . . ? Bei dieser Frage musste der Proband auf der horizontalen Achse nach dem gegebenen Namen suchen und daraufhin die Höhe der entsprechenden Säule auf der vertikalen Achse ablesen. Hierbei war zu beachten, dass nur Beschriftungen der vertikalen Achse genannt werden dürfen, auch wenn die Höhe der Säule zwischen zwei Werten lag (siehe Abbildung 7.2).

Welche Säule ist höher: . . . oder . . . ? Die Antwort der Probanden musste bei dieser Frage der Name der entsprechenden Säule sein (die zugehörige Beschriftung unter der horizontalen Achse).

Welches ist die höchste Säule und wie hoch ist diese? Wenn diese Frage gestellt wurde, mussten die Probanden den Namen der entsprechenden Säule und deren Höhe nennen. Dabei war, wie bei der Frage nach der höchsten Säule, zu beachten, dass nur Höhen genannt werden durften, die auf der vertikalen Achse des Säulendiagramms vorhanden waren.

Nr.	Stimulus	Aufgabentyp	Aufgabe
a	8	1	Welche Höhe hat Säule „Region D“?
b	4	3	Welches ist die höchste Säule und wie hoch ist diese?
c	3	1	Welche Höhe hat Säule „C“?
d	7	3	Welches ist die höchste Säule und wie hoch ist diese?
e	6	3	Welches ist die höchste Säule und wie hoch ist diese?
f	1	3	Welches ist die höchste Säule und wie hoch ist diese?
g	4	2	Welche Säule ist höher? „25 bis 30“ oder „-5 bis 0“?
h	3	3	Welches ist die höchste Säule und wie hoch ist diese?
i	8	3	Welches ist die höchste Säule und wie hoch ist diese?
j	6	2	Welche Säule ist höher? „X-Box“ oder „Playstation“?
k	2	1	Welche Höhe hat Säule „6“?
l	9	1	Welche Höhe hat Säule „Cannabis“?
m	5	1	Welche Höhe hat Säule „Nintendo“?
n	3	2	Welche Säule ist höher? „B“ oder „C“
o	2	3	Welches ist die höchste Säule und wie hoch ist diese?
p	5	2	Welche Säule ist höher? „Playstation“ oder „X-Box“?
q	9	3	Welches ist die höchste Säule und wie hoch ist diese?
r	7	2	Welche Säule ist höher? „3“ oder „5“?
s	5	3	Welches ist die höchste Säule und wie hoch ist diese?
t	8	2	Welche Säule ist höher? „Region D“ oder „Region E“?
u	2	2	Welche Säule ist höher? „8“ oder „1“
v	7	1	Welche Höhe hat Säule „4“?
w	4	1	Welche Höhe hat Säule „130 bis 135“?
x	1	2	Welche Säule ist höher? „3“ oder „4“?
y	9	2	Welche Säule ist höher? „Heroin“ oder „Zigaretten“?
u	1	1	Welche Höhe hat Säule „3“?
aa	6	1	Welche Höhe hat Säule „Playstation“?

Tabelle 7.3: Bei der Studie gestellte Fragen – Diese Fragen wurden bei der durchgeführten Studie den einzelnen Probanden in der vorgegebenen Reihenfolge gestellt.

7.2.4 Durchführung

Um mögliche Störfaktoren zu minimieren, musste zu Beginn jeder Proband, falls vorhanden, sein Handy ausschalten. Daraufhin unterschrieben die Probanden eine Aufklärung, dass alle Daten anonym gesammelt werden und jederzeit die Möglichkeit besteht, die Studie abzubrechen oder eine Pause einzulegen. Die Anonymität wurde dadurch hergestellt, dass eine zufällige Identifikationsnummer für jeden Probanden generiert und diesem zugeteilt wurde. Im Anschluss füllten die Probanden einen Fragebogen aus, in dem sie die Informationen aus den Tabellen 7.1 und 7.2 angaben.

Bevor der Hauptteil der Studie durchgeführt wurde, musste jeder Proband einen Sehtest ausführen. Dieser bestand aus zwei Teilen. Beim ersten Teil mussten die Buchstaben aus Abbildung 7.3a mit einer Größe von $18cm$ x $29cm$ im Abstand von $3m$ laut vorgelesen werden. Wenn ein Proband eine der obersten vier Zeilen nicht erkennen konnte, wurden die aufgezeichneten Daten dieses Probands nicht ausgewertet. Der zweite Teil des Sehtests bestand aus der Aufgabe, die in Abbildung 7.3b dargestellten Ishihara Farbtafeln abzulesen. Auch hier wurden nur die Daten der Probanden ausgewertet, die alle Zahlen erkennen konnten.

Nachdem die Probanden die Beschreibung der Studie (siehe Anhang) gelesen und gegebenenfalls Fragen dazu gestellt hatten, wurde das eigentliche Experiment durchgeführt. Vor jeder der 27 Aufgaben wurde ein weißes Bild mit einem roten Kreuz in der Mitte eingeblendet (siehe Abbildung 7.4). Um sicherzustellen, dass die Probanden beim Beginn jeder Aufgabe die gleiche Stelle auf dem Bildschirm fixierten, sollten sie sich zuvor auf das Kreuz fixieren. Währenddessen wurde eine der in Abschnitt 7.2.3 beschriebenen Fragen gestellt. Sobald der Proband sich sicher war, die Frage richtig verstanden zu haben, drückte er auf einer Tastatur die Leertaste, woraufhin sich das zu der entsprechenden Frage gehörende Säulendiagramm einblendete. Auf diesem musste der Proband so schnell wie möglich die Lösung ablesen und die Leertaste ein weiteres Mal drücken. Das nächste rote Kreuz wurde eingeblendet und der Proband beantwortete die zuvor gestellte Frage. Gemessen wurden dabei die Augenbewegungen mithilfe des Eye-Trackers und die Zeit, zwischen den zwei Tastendrücken, welche der Anzeigedauer des jeweiligen Säulendiagramms entspricht und als Zeit zum Beantworten der Frage betrachtet wurde.

7.3 Ergebnisse der Studie

Abbildung 7.5 zeigt die Fixationspunkte und Sakkaden, die bei der Beantwortung einzelner Fragen aufgezeichnet bzw. simuliert wurden. In den einzelnen Abbildungen sind jeweils grün die Sakkaden eines Probanden und blau die vorhergesagten Sakkaden des Simulationstools eingezeichnet. Die eingekreisten Zahlen zeigen die Reihenfolge der Fi-

P C S F

B C H E U

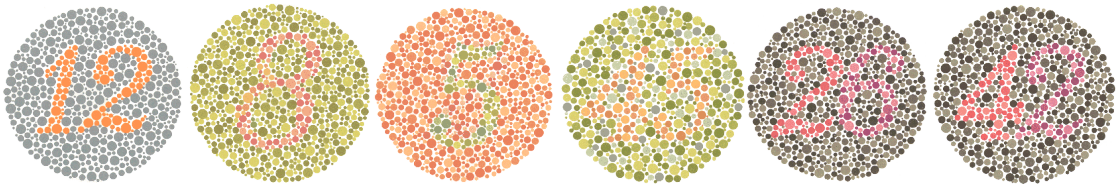
N D G P B H E

U T F Z D E C

P D C Z U E G H E

H F D N U S P G E Z

(a) Um sicherzustellen, dass kein Proband eine zu starke Sehschwäche hatte, mussten die Buchstaben mit einer Größe von 18cm x 29cm im Abstand von 3m laut vorgelesen werden. Konnte ein Proband eine der obersten vier Zeilen nicht lesen, wurden die gemessenen Daten des Probands nicht ausgewertet.



(b) Um Farbsehschwächen bei den Probanden auszuschließen, mussten alle Probanden die abgebildeten sogenannten Ishihara Farbtafeln laut vorlesen. Wurde eine der Farbtafeln nicht richtig erkannt, galt der Test als nicht bestanden und die Messdaten des Probands wurden nicht ausgewertet. [4]

Abbildung 7.3: Sehtests zum Studienbeginn – Um sicherzustellen, dass die Probanden gleiche Voraussetzungen haben was die Sehschärfe und Farberkennung angeht, wurden vor Beginn der Studie zwei Sehtests durchgeführt. Hat ein Proband einen der Tests nicht bestanden, wurde der Datensatz dieses Probands nicht ausgewertet.

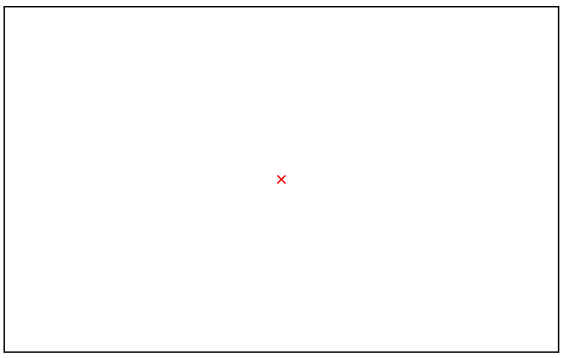


Abbildung 7.4: Orientierungsbild vor jeder Aufgabe – Damit die Fixation bei jedem Probanden und bei jeder Aufgabe an der gleichen Stelle auf dem Bildschirm beginnt, wurde bei der Studie vor jedem Säulendiagramm dieses bildschirmfüllende Bild (in der Studie: 1920 x 1200 Pixel) eingeblendet. Die Probanden sollten dabei das rote Kreuz fixieren.

xationen des jeweiligen Probanden bzw. der Simulation. Aus Tabelle 7.3 sind die Fragen für die gezeigten Stimuli abzulesen.

Das Diagramm in Abbildung 7.6 beschreibt die vom Simulationstool vorhergesagten und die bei der Studie gemessenen Zeiten. Die Medianwerte der gemessenen Zeiten, die die Probanden für das Lösen der Aufgaben benötigt haben, sind durch dicke Linien dargestellt. Das Diagramm beinhaltet zudem jeweils die 1. und 3. Quartile für die einzelnen Messungen (dünne Linien). Die grauen Kreise bezeichnen die Mittelwerte der vorhergesagten Zeiten des Simulationstools[1].

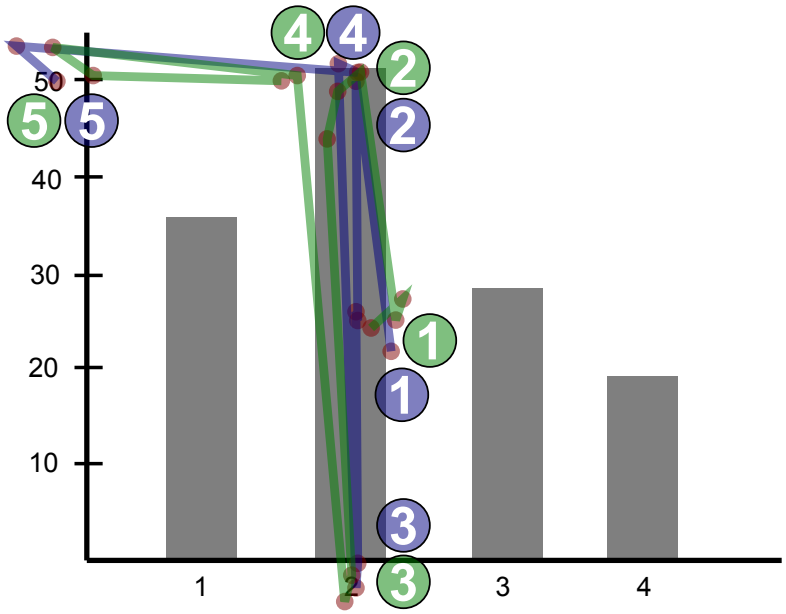
7.4 Analyse und Diskussion

Anhand der beiden Fragen, die im Abschnitt 7.1 beschriebenen wurden, wird im Folgenden untersucht, ob die Messdaten der Studie mit den simulierten Fixationspunkten und Zeiten übereinstimmen.

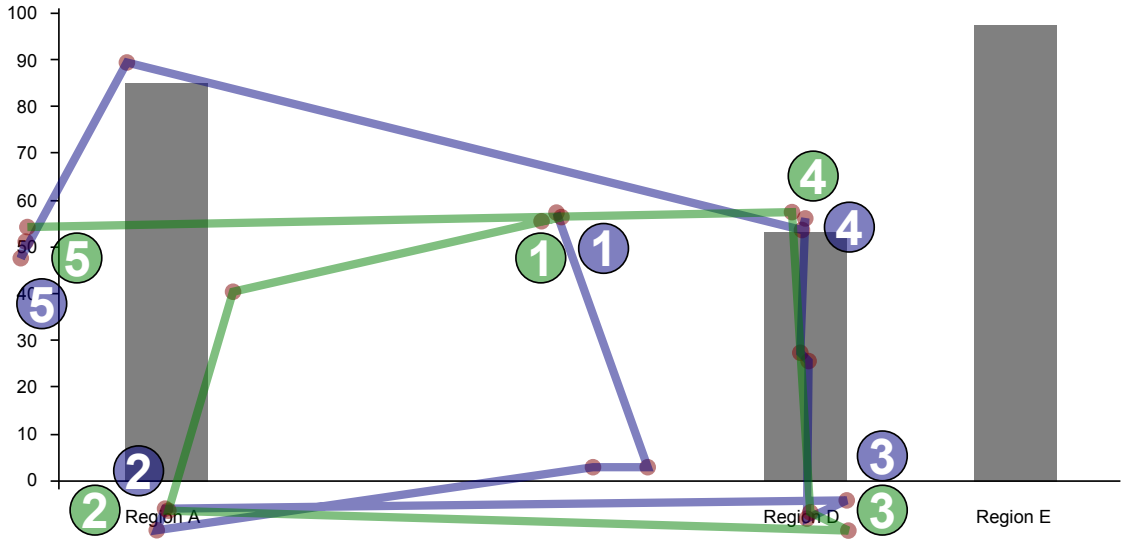
7.4.1 Fixationspunkte

Die Diagramme mit den eingezeichneten Sakkaden der Probanden und der Simulationen aus den Abbildungen 7.5a, 7.5b, 7.5c und 7.5d zeigen tendenziell eine gute Vorhersage

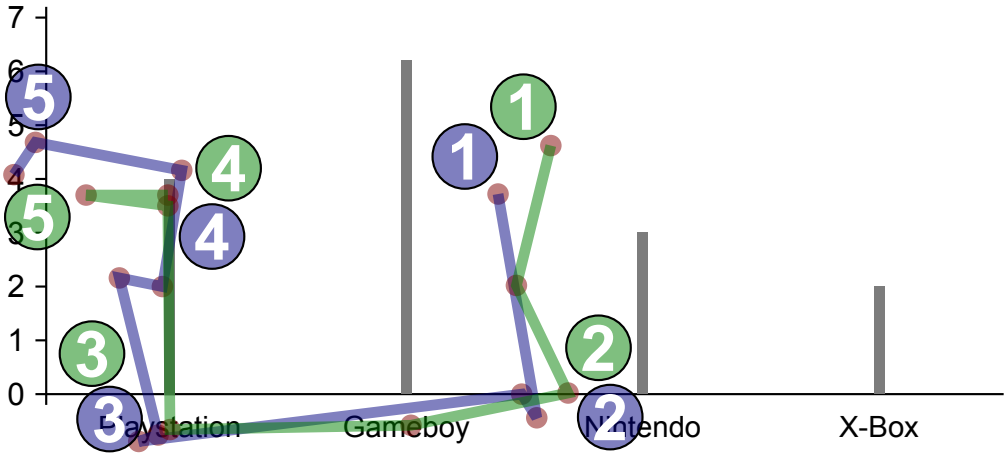
[1]Da in das Simulationsmodell stochastische Parameter einfließen, wurden die Mittelwerte der Simulation auf Grundlage von vier Simulationsdurchgängen gebildet. Ab dieser Anzahl änderte sich das arithmetische Mittel der Zeiten nur noch im Mikrosekundenbereich, was als ausreichend betrachtet wurde.



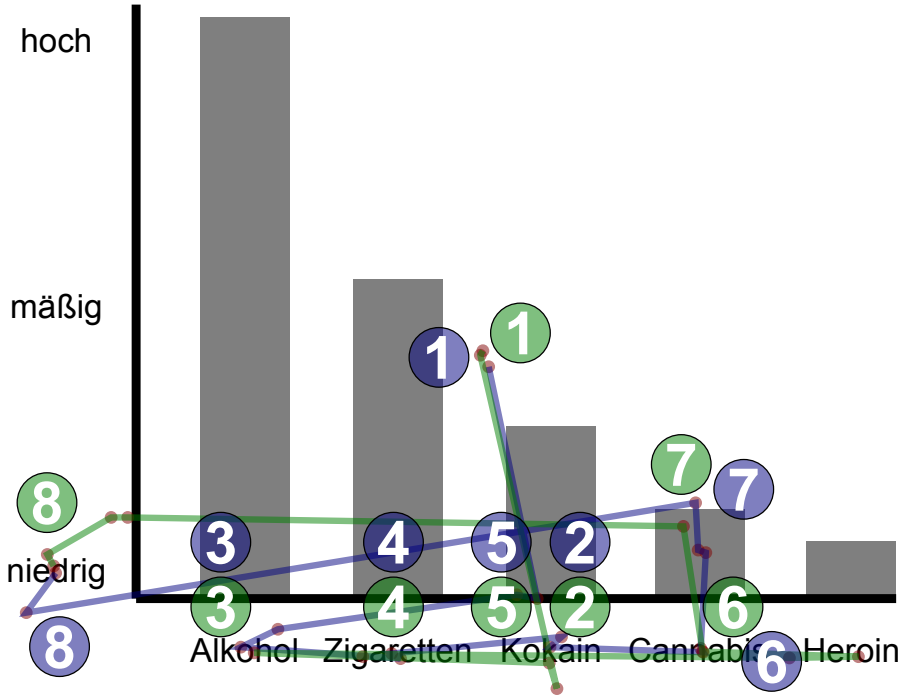
(a) Aufgabe f – „Welches ist die höchste Säule und wie hoch ist diese?“



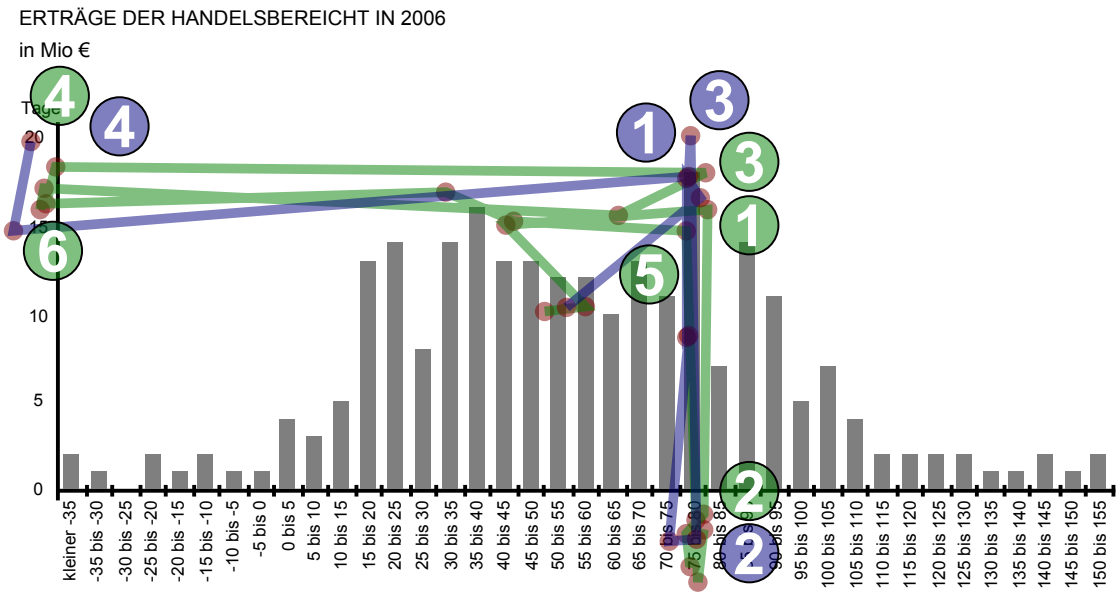
(b) Aufgabe a – „Welche Höhe hat Säule ’Region D’?“



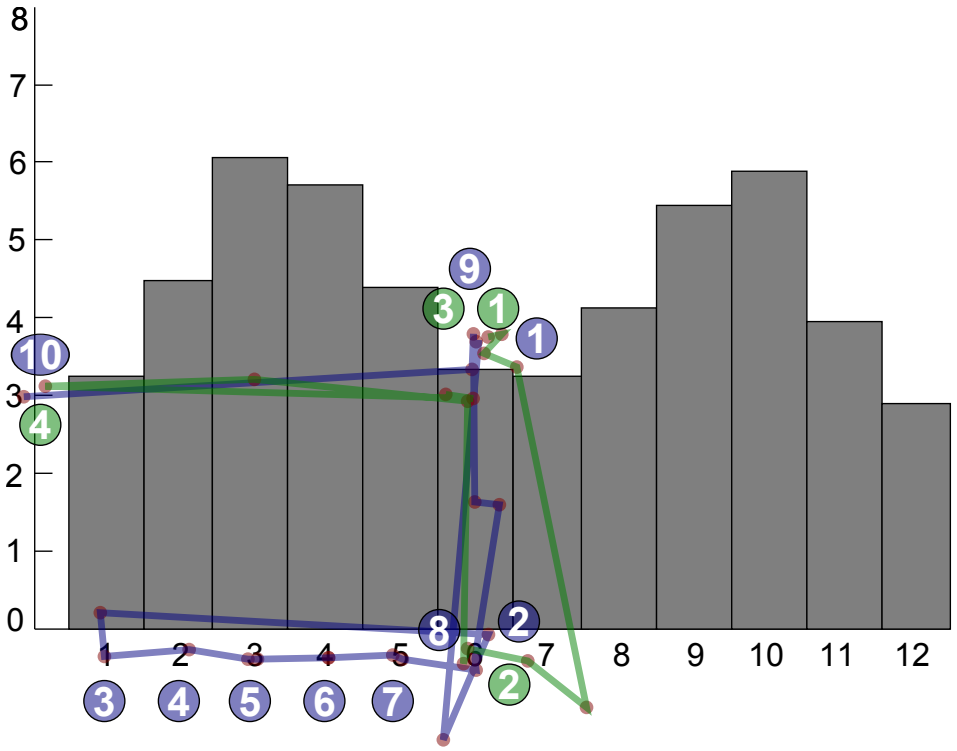
(c) Aufgabe aa – „Welche Höhe hat Säule 'Playstation'?“



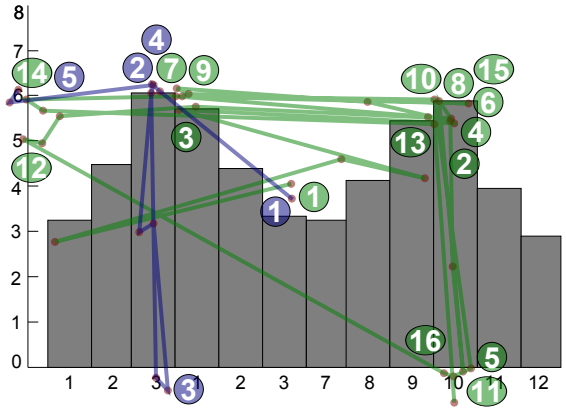
(d) Aufgabe l – „Welche Höhe hat Säule 'Cannabis'?“



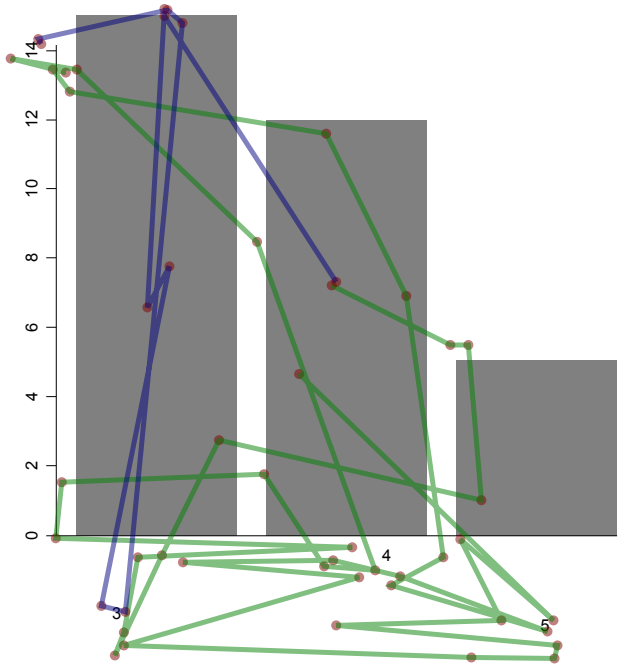
(e) Aufgabe b – „Welches ist die höchste Säule und wie hoch ist diese?“



(f) Aufgabe k – „Welche Höhe hat Säule ’6’“?



(g) Aufgabe o – „Welches ist die höchste Säule und wie hoch ist diese?“



(h) Aufgabe d – „Welches ist die höchste Säule und wie hoch ist diese?“ (Aufgrund der Komplexität der Sakkaden wurde auf die Beschriftungen der Reihenfolge der Fixationen verzichtet.)

Abbildung 7.5: In der Studie gemessene und vorhergesagte Sakkaden – Die Bilder zeigen Sakkaden einzelner Probanden (grün) im Vergleich zu den vorhergesagten Sakkaden des Simulationstools (blau). Die eingekreisten Zahlen beschreiben die Reihenfolge der Fixationen des jeweiligen Probands bzw. der Simulation. Zu jedem Diagramm ist die Aufgabe mit angegeben, die ausgeführt werden sollte (siehe hierzu Tabelle 7.3).

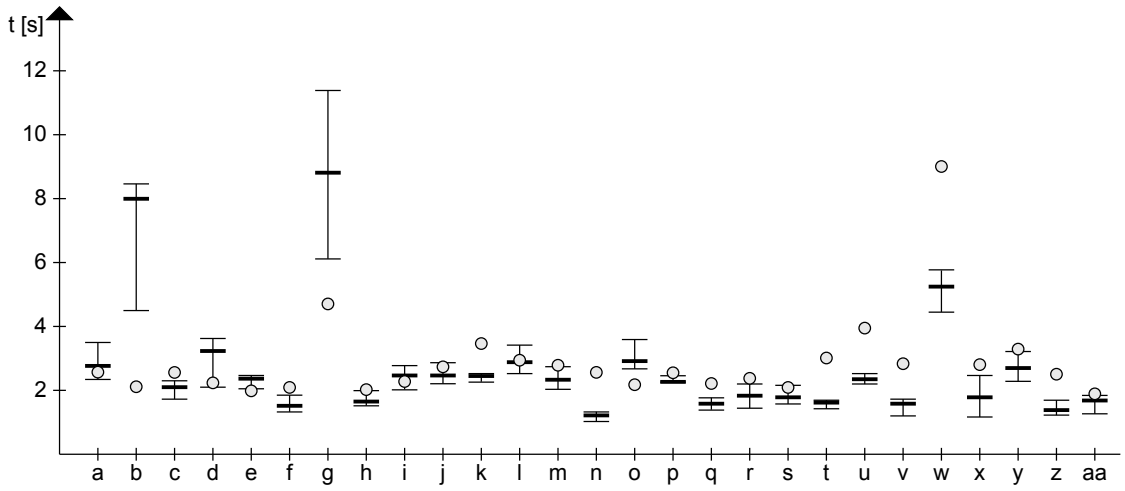


Abbildung 7.6: Vergleich der gemessenen mit den vorhergesagten Zeiten bei der Studie – Die dicken Linien geben die Medianwerte der in der Studie gemessenen Werte an. Die dünnen Linien zeigen die 1. und 3. Quartile. Die vom Simulationstool vorhergesagten Zeiten sind als Kreise dargestellt. a–aa repräsentieren die verschiedenen Aufgaben, welche in Tabelle 7.3 aufgeführt sind.

des Modells, was im Folgenden näher untersucht wird:

Abbildung 7.5a: Hier war die Aufgabe das Suchen der höchsten Säule und das Nennen ihrer Bezeichnung und Höhe. Die ersten Sakkaden führen zur Oberkante der höchsten Säule (Säule „2“). Daraufhin lesen der Proband und die Simulation den Namen der Säule unter der horizontalen Achse ab. Nach einer erneuten Fixation zur Säulenoberkante führen die Blicke nach links zu der Höhenbeschriftung der Säule (Höhe „50“).

Sowohl die vorhergesagten als auch gemessenen Fixationen des Probanden deuten darauf hin, dass das eigentliche Finden der Säule im peripheren Gesichtsfeld wahrgenommen und Top-Down-gesteuert[2] wird. Ein aktives Suchen dieser Säule, indem beispielsweise alle Säulen nacheinander fixiert werden müssen, ist nicht nötig.

Abbildung 7.5b: In dieser Aufgabe sollten der Proband und die Simulation die Säule „Region D“ im Diagramm suchen und deren Höhe ablesen. Die ersten gemessenen unterscheiden sich hier von den vorhergesagten Sakkaden. Das erste fixierte Element ist dennoch sowohl bei der Simulation als auch beim Proband die erste

[2]Mit dem Begriff „Top-Down-gesteuert“ ist nicht der im Abschnitt 2.3.2 beschriebene Top-Down-Ansatz gemeint. Stattdessen ist hier die bewusst gesteuerte Fixation auf ein bestimmtes Grafikobjekt abhängig von der Fragestellung gemeint (siehe auch [42]).

Beschriftung der horizontalen Achse („Region A“). Daraufhin wird die nächste Beschriftung rechts daneben fixiert und als die zu suchende Beschriftung identifiziert („Region D“). Nun führen die Fixationen zur Säulenoberkante und von dort horizontal zu den Beschriftungen der vertikalen Achse, um dort die Höhe abzulesen.

Abbildung 7.5c: Bei dieser Aufgabe sollte die Höhe einer Säule (Säule „Playstation“) abgelesen werden. Die erste Säulenbeschriftung wird gelesen und als die zu suchende identifiziert, worauf deren Höhe mithilfe der Säulenoberkante und der horizontal dazu liegenden Höhenbeschriftung abgelesen wird. Hier ist nahezu eine völlige Übereinstimmung des Modells mit dem Probanden gegeben.

Abbildung 7.5d: Wie bei den letzten beiden Aufgaben sollte auch hier die Höhe einer Säule (Säule „Cannabis“) ermittelt werden. Sowohl die simulierten als auch die in der Studie gemessenen Fixationen beginnen von links ausgehend, die einzelnen Säulenbeschriftungen zu fixieren. Nachdem die zu suchende Beschriftung („Cannabis“) gefunden wurde, bewegt sich der Blick auf die entsprechende Säulenoberkante und von dort zur Höhenbeschriftung der Säule („niedrig“), worauf diese Beschriftung abgelesen wird.

Hier ist zu erkennen, dass die Säulenbeschriftungen nacheinander von links nach rechts gelesen werden, wodurch die Korrektheit des Modells im Bereich der visuellen Suche in diesem Beispiel gegeben ist.

Entgegen der guten Vorhersagen des Simulationstools, deuten die Aufzeichnungen aus den Abbildungen 7.5e, 7.5f, und 7.5g auf Fehler im Modell hin:

Abbildung 7.5e: Die Aufgabe war das Auffinden der höchsten Säule und das Nennen deren Höhe. Die Fixationen des Probanden zeigen, dass z. B. die Höhe der Säule nicht, wie es die Simulation durchführt, mit einer einzigen Sakkade in Richtung der vertikalen Achse abgelesen wird. Stattdessen wechselt der Blick des Probanden drei mal zwischen der Höhenbeschriftung und der höchsten Säule hin und her, bis der Wert genannt wird.

Abbildung 7.5f: Bei diesem Stimulus mussten die Probanden und die Simulation die Höhe der Säule „6“ ablesen. Das Modell läuft den bekannten Weg wie in Abbildung 7.5d ab, indem die Simulation beginnt, die Säulenbeschriftungen von links nach rechts zu lesen, bis die Beschriftung „6“ gefunden wird. Der Proband hingegen fixiert die richtige Beschriftung in einem einzigen Schritt. Die übrigen Sakkaden zum Ablesen der Höhe sind bei der Simulation und beim Probanden identisch.

Abbildung 7.5g: Beim gleichen Stimulus wie zuvor sollten die Probanden und das Simulationstool nun die höchste Säule im Diagramm suchen und deren Höhe nennen. Das Modell findet diese Säule in einem Schritt und liest den Wert ab. Der Proband bewegt seinen Blick wiederholend zwischen den Säulen „3“ und „10“ hin und her, bevor er die zweithöchste Säule als die höchste identifiziert.

Nr.	Stimulus	Aufgabentyp	Aufgabe	Probanden
a	8	1	Welche Höhe hat Säule „Region D“?	2,4
c	3	1	Welche Höhe hat Säule „C“?	1,7
d	7	3	Welches ist die höchste Säule und wie hoch ist diese?	2
e	6	3	Welches ist die höchste Säule und wie hoch ist diese?	1
f	1	3	Welches ist die höchste Säule und wie hoch ist diese?	1
j	6	2	Welche Säule ist höher? „X-Box“ oder „Playstation“?	1
l	9	1	Welche Höhe hat Säule „Cannabis“?	4
y	9	2	Welche Säule ist höher? „Heroin“ oder „Zigaretten“?	3
aa	6	1	Welche Höhe hat Säule „Playstation“?	2,3,6

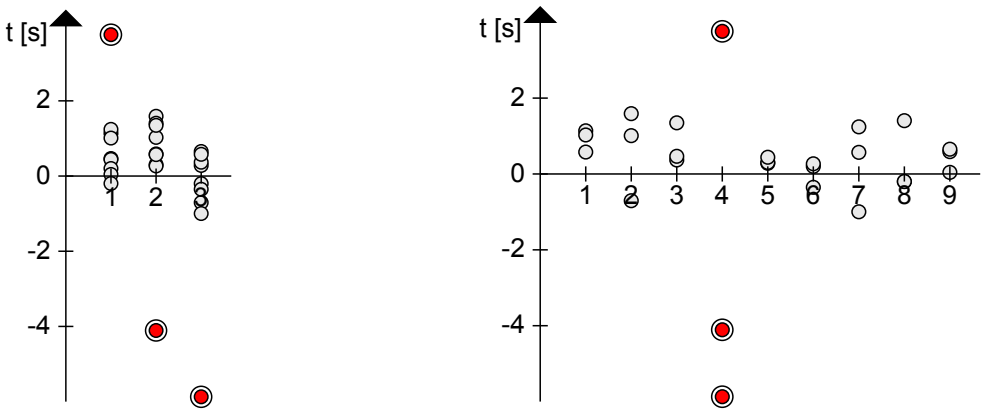
Tabelle 7.4: Übereinstimmende Fixationen einzelner Probanden mit der Vorhersage des Simulationstools

Außer den bisher gut analysierbaren Messungen ergab die Studie auch Bewegungspfade, die keinen Vergleich mit den vorhergesagten Werten erlauben. Als Beispiel hierfür ist das Ergebnis aus Abbildung 7.5h zu nennen. Die Aufgabe war das Finden der höchsten Säule und das Nennen deren Höhe. Der Proband vollzog scheinbar unkontrollierte Augenbewegungen, vor allem im unteren Bereich des Diagramms. Die Simulation hingegen führte die bekannten Fixationen zum Finden der Säule und Ablesen deren Höhe aus.

Allgemein gab es bei 13 von insgesamt 189 Fällen eine Übereinstimmung der vom Simulationstool vorhergesagten mit den in der Studie gemessenen Fixationen. Diese Fälle sind in Tabelle 7.4 aufgelistet. Somit gab es zu jedem Aufgabentyp mindestens zwei mit dem Simulationsmodell übereinstimmende Messungen.

7.4.2 Zeiten

Abbildung 7.6 zeigte bereits die in der Studie gemessenen und die vom Simulationstool vorhergesagten Zeiten sowie die 1. und 3. Quartile der Messungen. Das Diagramm lässt erkennen, dass die vorhergesagten Zeiten in der Größenordnung der Zeiten liegen, die bei der Studie ermittelt wurden: Sechs der 27 vorhergesagten Werte liegen innerhalb der 1.



(a) Abweichungen in Abhängigkeit des Aufgabentyps (1–3)

(b) Abweichungen in Abhängigkeit des Diagramms (1–9)

Abbildung 7.7: Abweichungen der gemessenen von den vorhergesagten Zeiten – Die horizontalen Achsen geben den Aufgabentyp bzw. das angezeigte Diagramm an. Die vertikalen Achsen definieren die zeitliche Abweichung der gemessenen von den vorhergesagten Werten (positive Zeiten bedeuten, dass das Modell eine zu hohe Zeit vorhergesagt hat, negative Zeiten bedeuten eine zu gering eingeschätzte Zeit des Modells). Die markierten Messpunkte zeigen besonders hohe Abweichungen ($> 3{,}5s$).

und 3. Quartile. 18 weitere Werte befinden sich im Bereich von $\pm 1{,}5s$ um die gemessenen Medianwerte der einzelnen Aufgaben. Die drei übrigen vorhergesagten Zeiten liegen mit einem Abstand von $> 3{,}5s$ von den gemessenen Werten entfernt.

Abbildung 7.7 zeigt, dass hohe Abweichungen der Zeiten zwar bei allen drei Aufgabentypen, jedoch nur bei einem bestimmten Säulendiagramm auftreten (Diagramm 4). Aufgrund dieser Abweichung wird das Diagramm für die weitere Auswertung der Daten ausgenommen. Da die Messung der Zeiten nur in den Fällen korrekt sein kann, in denen zumindest die Fixationspunkte mit den modellierten Fixationen übereinstimmen, werden für eine weitere Auswertung außerdem auch alle Messungen entfernt, bei denen diese Fixationen nicht mit dem Modell übereinstimmen. Für die weitere Analyse werden dadurch ausschließlich die in Tabelle 7.4 aufgelisteten Messungen betrachtet.

Werden die Messergebnisse in Abhängigkeit der vorhergesagten Zeiten aufgetragen, erhält man durch eine Regressionsanalyse die in Abbildung 7.8 dargestellte Regressionsgerade ($t_p(t_s) = 1{,}081t_s - 504{,}15$). Punkte auf der im Diagramm zusätzlich eingezeichneten gestrichelten Geraden definieren den optimalen Fall, dass die jeweils gemessene Zeit für den gegebenen Fall exakt der vorhergesagten Zeit entspricht. Die Abweichung der Steigung der Regressionsgeraden zur Steigung der Winkelhalbierenden ($m = 1$) entspricht

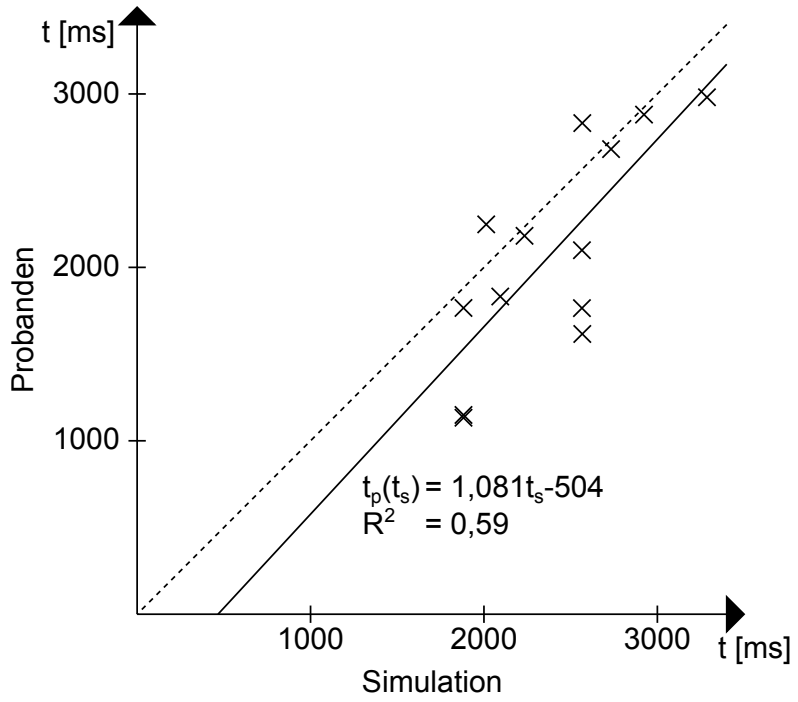


Abbildung 7.8: Benötigte Durchführungszeit der Probanden in Abhängigkeit zur Vorhersagezeit – Die Punkte stellen die in Tabelle 7.4 aufgelisteten Messwerte in Abhängigkeit der von der Simulation vorhergesagten Zeiten dar. Die Regressionsgerade (durchgezogene Linie) liegt annähernd parallel zur Winkelhalbierenden (gestrichelte Linie), die eine perfekte Übereinstimmung der Messerergebnisse zu den vorhergesagten Werten darstellt.

etwa 8,1%. Für niedrige Zeiten, die in der Studie gemessen wurden, kann die Regressionsgerade näherungsweise als Parallele zur Winkelhalbierenden angenommen werden. Die vorhergesagten Zeiten sind also grundsätzlich ca. $500ms$ zu hoch angesetzt. Aus dem Bestimmtheitsmaß der Regression von $R^2 = 0{,}59$ ergibt sich eine Wahrscheinlichkeit von ca $p = 0{,}001$, dass die Nullhypothese wahr ist. Die vorhergesagten und gemessenen Zeiten sind also mit einer hohen Wahrscheinlichkeit linear abhängig, was dafür spricht, dass das Modell keine zufälligen Werte liefert.

8 Zusammenfassung

Ziel dieser Arbeit war es, ein Modell von kognitiven Abläufen zu entwerfen, die beim Lesen von Visualisierungen im menschlichen Gehirn stattfinden. Die auf dem Modell beruhende Simulation soll es ermöglichen, Visualisierungen auf ihre kognitive Ergonomie hin zu untersuchen, ohne aufwendige und kostenintensive Studien mit Probanden durchführen zu müssen. Damit einhergehend würden die schneller verfügbaren Ergebnisse eine iterative Optimierung der untersuchten Visualisierungen ermöglichen.

Für die Entwicklung des Modells wurden bereits existierende Modelle und Frameworks für die Simulation kognitiver Prozesse untersucht. Einige Grundideen, wie die Definition von Wissen über bestimmte Visualisierungstypen mithilfe von Graphen-Schemata, wurden für das entworfene Modell übernommen und angepasst.

Das entwickelte Konzept sieht vor, die Simulation in zwei Hauptkomponenten, das Eingabe- und das Verarbeitungsmodell, aufzuteilen. Das Eingabemodell besteht aus einer dreistufigen Ontologie, mehreren Visualisierungsschemata, die das Grundwissen über einzelne Visualisierungstypen und die darin geltenden Leseregeln enthalten, die zu simulierende Visualisierung in Form eines Visualisierungsmodells und der Aufgabenstellung. Das Verarbeitungsmodell besteht aus den vier Hauptprozessen „Passendes Schema finden“, „Zu suchendes Element ermitteln“, „Element per visueller Suche suchen“ und „Element fixieren“, die selbst in mehrere kleine Prozesse aufgeteilt werden. Durch Aufruf der Prozesse im Verarbeitungsmodell kann die Simulation in der gegebenen Visualisierung Vorhersagen über die Reihenfolge der fixierten Elemente und die für die Fixationen benötigten Zeiten berechnen. Die zum Beantworten der Anfrage ermittelte Gesamtzeit kann nun als Maß für die Leistung eines bestimmten Visualisierungstyps verwendet werden und damit einen Vergleich mit anderen Visualisierungstypen ermöglichen.

Um die Simulationsergebnisse zu verifizieren, wurde eine Studie mit sieben Probanden durchgeführt. Einige Messungen in dieser Studie haben gute Übereinstimmungen mit den vorhergesagten Zeiten und Fixationen des entworfenen Simulationstools. Es wurde jedoch festgestellt, dass vor allem die visuelle Suche mit den einzelnen Suchstrategien erweitert werden sollte.

8.1 Diskussion

Auch wenn die vorhergesagten Sakkaden des entwickelten Simulationstools häufig nicht perfekt den realen Sakkaden eines Menschen beim Lesen einer Visualisierung entsprechen, liefern die ermittelten Zeiten bei den korrespondierten Sakkaden dennoch eine gute Vorhersage für die Messergebnissen der Studie. Vor allem bei komplexen Diagrammen stimmen die Vorhersagen der Sakkaden und dadurch auch die vorhergesagten Zeiten nur bei einer kleinen Anzahl der Messungen überein.

Bei der Analyse der Fixationspunkte hat die Simulation folgende Einschränkungen gezeigt: So wurde beispielsweise festgestellt, dass die einzelnen Säulenbeschriftungen häufig nicht von links nach rechts fixiert werden, wenn nach einer bestimmten Säulenbeschriftung gesucht werden soll. Stattdessen nutzte die visuelle Suche der Probanden hier oft Abkürzungen, wodurch die vorhergesagten Sakkaden nicht realistisch wurden. Dies traf insbesondere bei sortierten Säulenbeschriftungen zu, was bereits im Lösungskonzept vermutet wurde. Die Implementierung des Simulationstools bildet jedoch nur die einfache Suchstrategie, ein bestimmtes visuelles Element zu suchen, indem im Visualisierungsmodell alle Elemente von links nach rechts fixiert werden, ab. Es konnte somit nicht nachgewiesen werden, ob das entworfene Modell bessere Vorhersagen macht als das implementierte Simulationstool.

Die Analyse der Studie zeigte auch, dass Menschen selten nach einmaliger Fixation aller zum Lösen einer Frage benötigten Elemente, die vollkommene Sicherheit haben, eine korrekte Antwort zu geben. Einzelne visuelle Elemente wurden wiederholt von den Probanden fixiert und mit anderen Elementen verglichen. Die Probanden fixierten beispielsweise häufiger die zwei höchsten Säulen eines Diagramms, die fast gleich hoch waren, um zu entscheiden, welche die höchste ist. Das Simulationstool sah auch für diesen Fall keine Implementierung vor. Im entwickelten Modell würde dieser Vorgang jedoch durch passende Suchstrategien umgesetzt werden können.

Eine weitere Inkonsistenz ergibt sich bei der falschen Antwort eines Menschen. Dieses Verhalten ist weder im Simulationstool noch im entwickelten Modell berücksichtigt worden. Jedoch sind hier die Gründe für eine falsche Antwort genauer zu untersuchen. Wenn der Proband während dem Lesen einer Visualisierung durch andere Gedanken oder seine Umwelt abgelenkt wird, ist dies kein Indiz für eine schlechte Visualisierung. Hingegen deuten z. B. kaum sichtbare Eigenschaften einer Visualisierung auf eine schlecht lesbare Visualisierung hin.

Bei den Vorhersagen der benötigten Zeiten ermittelte das Simulationstool gute Werte, sofern nur die Messungen der Probanden betrachtet wurden, bei denen die Sakkaden mit den vom Simulationstool vorhergesagten Sakkaden übereinstimmten. Grundsätzlich schätzte die Simulation eine zu hohe Zeit für das Lesen einer Visualisierung ab. Hierüber lässt sich viel spekulieren, worauf jedoch verzichtet wird, da die Anzahl der Messungen

mit korrekten Sakkaden (13 von 189) als zu gering erachtet wird. Die Wahrscheinlichkeit einer zufällig korrekten Vorhersage der Zeiten wird als zu hoch erachtet.

Die Studie hatte eher einen prototypischen Charakter, sodass daraus keine repräsentativen Daten entstanden. Die Anzahl von sieben Probanden lässt kaum verlässliche Aussagen für die untersuchten Fragen zu. Zudem konnte in dieser Arbeit nur das Simulationstool, nicht das entwickelte Modell, mit den Studienergebnissen verglichen werden. Die für die Studie gewählten Stimuli waren nicht optimal für die Validierung des Simulationstools geeignet. Stattdessen wären z. B. ausschließlich Diagramme mit unsortierten Säulenbeschriftungen und eindeutig unterscheidbaren Höhen zu wählen gewesen.

Allgemein ist davon auszugehen, dass der Ansatz des Simulationstools vielversprechend ist und darauf weiter aufgebaut werden kann. Die genannten Kritikpunkte stützen sich bis auf die falschen Antworten von Menschen ausschließlich auf das prototypisch umgesetzte Simulationstool, das in erster Linie nur eine einzige Suchstrategie bei der visuellen Suche implementiert.

8.2 Ausblick

Da das Modell zwar detailliert entworfen, jedoch nur ein Teil implementiert wurde, liegt die Aufgabe im Weiteren darin, alle beschriebenen Prozesse und Prozessabläufe in einem Computerprogramm zu implementieren. Insbesondere sind umfangreiche Suchstrategien im Prozess der visuellen Suche umzusetzen.

Des Weiteren ist denkbar, das Modell bzw. die Implementierung auf 3D-Visualisierungen zu erweitern. Hierzu muss das visuelle Modul in ACT-R umgeschrieben werden, da dies, wie im Grundlagenkapitel (siehe Abschnitt 2.4.4) beschrieben, nur für zweidimensionale Visualisierungselemente ausgelegt ist. Auch die Einschränkung, Ausmaße von Elementen ausschließlich durch achsenparallele Rechtecke definieren zu können, könnte dabei gelöst werden. Erst damit wäre es z. B. möglich, das Lesen von Liniendiagrammen zu modellieren, da insbesondere hier lange, diagonal liegende Elemente vorliegen, für die die Beschreibung der Ausmaße durch achsenparallele Rechtecke nicht optimal ist.

Um die Vorhersagen des Modells zu optimieren, müssen in weiteren Untersuchungen die in dieser Arbeit beschriebenen kognitiven Prozesse detaillierter betrachtet werden. Einflussfaktoren, wie Farben und Texturen bis hin zu Schriftarten und -größen einzelner Visualisierungselemente, spielen vor allem bei der visuellen Suche eine große Rolle und bewirken je nach Detailgrad unterschiedlichste Sakkaden und Fixationszeiten.

Eine umfassende Erweiterungsmöglichkeit besteht in der Definition und Verarbeitung der Visualisierungsschemata. In erster Linie wäre eine baumartige Struktur innerhalb der Schemata denkbar. Dadurch können redundante Klassen- und Relationsdefinitionen minimiert und Leseregeln generalisiert werden. Verbindet man die Schemata nicht

baumartig, sondern in Form eines gerichteten Graphen, könnte sogar eine Art Mehrfachvererbung, wie sie einige objektorientierte Programmiersprachen (z. B. C++, Python) ermöglicht, erzeugt werden. Liegen z. B. ein Schema für Diagramme mit logarithmischen Achsen und ein Schema für zweidimensionale Punktdiagramme vor, könnte aus Verbindung dieser beiden Schemata mit wenig Mehraufwand ein Schema für zweidimensionale logarithmische Punktdiagramme erzeugt werden. Durch eine Erweiterung des Prozesses, der das Finden des optimalen Schemas für ein gegebenes Visualisierungsmodell ermöglicht, wäre das Modell in der Lage, zu einem noch unbekannten Visualisierungstyp mehrere Visualisierungsschemata auszuwählen. Dadurch könnten Leseaufgaben auf neuen Visualisierungstypen simuliert werden. An dieser Stelle könnte ein Lernprozess in das Modell eingearbeitet werden, der automatisiert neue Visualisierungsschemata generiert.

Trotz allem zeigt die durchgeführte Studie, dass jeder Mensch teilweise unterschiedliche Strategien beim Lesen von Visualisierungen anwendet und das kognitive Modell somit nur Spezialfälle simuliert. In einem abschließenden Satz soll hierzu der Mathematiker und Physiker Georg Lichtenberg zitiert werden:

> *„Eine völlige Gleichheit der Menschen lässt sich gar nicht denken.“*
>
> (Lichtenberg, Georg Christoph 1742 – 1799)

Anhang

Studiendurchführung

Das Ziel dieser Studie ist es, für verschiedene Säulendiagramme die Augenbewegungen und Durchführungszeiten der Probanden aufzuzeichnen.

Die Studie wird komplett am Eye-Tracker ablaufen. Der Eye-Tracker zeichnet bei den einzelnen Aufgaben jeweils die Augenbewegung auf. Diese Daten werden später ausgewertet, um die Bewegung der Augen zu analysieren und Durchführungszeit für die einzelnen Aufgaben zu messen.

Jede Aufgabe wird am Anfang durch die Kalibrierung der Augen am Eye-Tracker begonnen. Es wird insgesamt 3 Aufgabentypen geben:

1. Ablesen der Höhe einer bestimmten Säule
2. Vergleichen der Höhen zweier Säulen
3. Bestimmen der höchsten Säule und Ablesen der Höhe

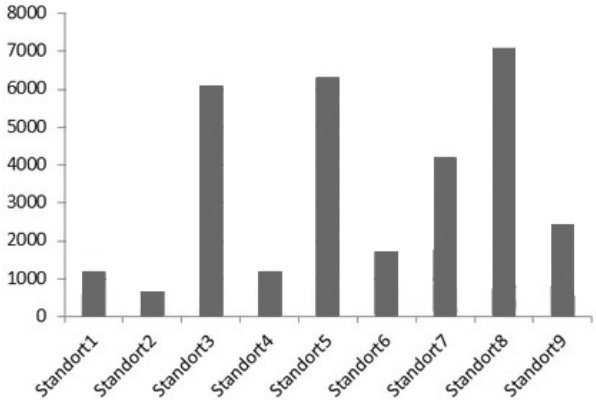
Es gibt insgesamt 9 verschiedene Säulendiagramme, wobei im Laufe der Studie für jedes Diagramm alle 3 Aufgaben in zufälliger Reihenfolge abgefragt werden.

Nach der Kalibrierung beginnt die eigentliche Aufgabe. Bei jedem Bildwechsel wird zuerst ein rotes Kreuz in der Bildschirmmitte angezeigt. Das rote Kreuz dient als Ausgangspunkt der Augen für das nächste Bild. Die Augen müssen immer auf das rote Kreuz gerichtet werden, bevor das nächste Bild eingeblendet wird.

Aufgabentyp 1: Ablesen der Höhe einer bestimmten Säule

Beim ersten Aufgabentyp wird das Ablesen der Höhe einer vorgegebenen Säule untersucht. Die vorgegebene Säule ist dabei anhand ihrer Bezeichnung auf der „X-Achse“ definiert. Die Höhe einer Säule ist auf- bzw. abgerundet zu den Beschriftungen der „Y-Achse“ zu betrachten. Beim Ablesen der Höhe sollte also immer die am nächsten gelegene Beschriftung genannt werden.

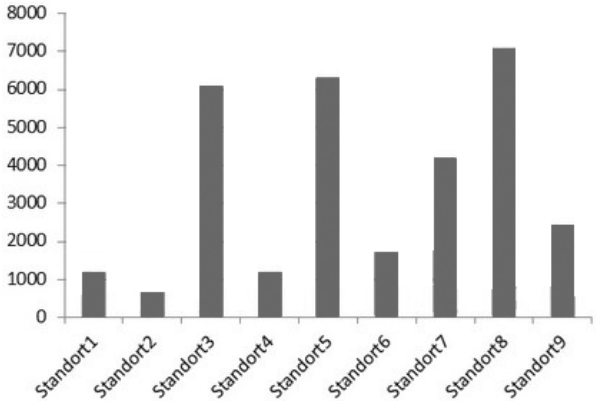
Beispiel: Wie hoch ist Säule für Standort 6? (Antwort: 2000 – Achtung: NICHT Wert zwischen 1000 und 2000 angeben!)



Aufgabentyp 2: Vergleichen der Höhen zweier Säulen

Beim zweiten Aufgabentyp wird das Vergleichen der Höhen zweier Säulen untersucht. Hierbei werden zwei Säulen wie in Aufgabentyp 1 anhand ihrer „X-Achsen-Beschriftung“ genannt. Aufgabe ist es, schnellst möglich zu antworten, welche der beiden Säulen höher ist.

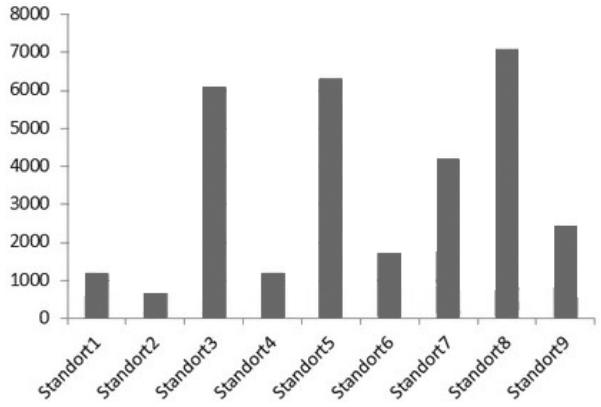
Beispiel: Welche Säule ist höher? Die Säule für Standort 5 oder die für Standort 2? (Antwort: Standort 5)



Aufgabentyp 3: Bestimmen der höchsten Säule und Ablesen der Höhe

Beim dritten Aufgabentyp wird das Bestimmen der höchsten Säule eines Diagramms sowie deren Höhe abgefragt. Zu nennen ist dabei zuerst der Name der höchsten Säule und darauffolgend deren Höhe in gerundeter Weise entsprechend der Beschriftungen (siehe Aufgabentyp 1).

Beispiel: Welches ist die höchste Säule und wie hoch ist diese? (Antwort: Standort 8 - 7000 – Achtung: NICHT Wert zwischen 6000 und 7000 angeben!)



Literaturverzeichnis

[1] http://www.uark.edu/misc/lampinen/tutorials/multinomial.htm

[2] http://act-r.psy.cmu.edu/about

[3] *The Common Lisp Cookbook*. http://cl-cookbook.sourceforge.net/. Version: Oktober 2012

[4] *Ishihara Farbtafeln*. http://www.biologiedidaktik.salzburg.at/Humanbiologie/Dateien/ishihara_farbtafeln.pdf. Version: Februar 2013

[5] Adrian, E. D.: The all-or-none principle in nerve. In: *Journal of Physiology* 47 (1914), S. 460–474

[6] Aldrich, J.: R. A. Fisher and the Making of Maximum Likelihood 1912-1922. In: *Statistical Science* 12 (1997), Nr. 3, S. 162–176. – ISSN 08834237

[7] Anderson, J. R.: ACT: A simple theory of complex cognition. In: *American Psychologist* 51 (1996), Nr. 4, S. 355

[8] Anderson, John R.: *Cognitive Psychology and its Implications*. Seventh Edition. Worth Publishers, 2009. – ISBN 9781429219488

[9] Anderson, John R. ; Bower, G. H.: *Human Associative Memory*. 3 Revised. Psychology Press, 1973. – 101–125 S. – ISBN 9780898591088

[10] Bandura, A.: Influence of models reinforcement contingencies on the acquisition of imitative response. In: *Journal of Personality and Social Psychology* 1 (1965), Nr. 6, S. 589–595

[11] Blum, Deborah: *Love at Goon Park: Harry Harlow and the Science of Affection*. 0002. Basic Books, 2011. – ISBN 978–0465026012

[12] *Kapitel* Matching Operator Sequences. In: Card, S. K. ; Moran, T. P. ; Newell, A.: *The Psychology of Human-Computer Interaction*. 1. Lawrence Erlbaum Associates, 1983. – ISBN 0898592437, S. 190 f.

[13] Card, S. K. ; Moran, T. P. ; Newell, A.: *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, 1983. – ISBN 0898592437

[14] Chomsky, N.: A Review of B. F. Skinner's Verbal Behavior. In: *Language* Bd. 35, Linguistic Society of America, 1959, S. 26–58

[15] Clark, A.: The Dynamical Challenge. In: *Cognitive Science* 21 (1997), Nr. 4, S. 461–481

[16] Collins, A. M. ; Quillian, M. R.: Retrieval Time from Semantic Memory. In: *Journal of verbal learning and verbal behavior* 8 (1969), Nr. 2, S. 240–247

[17] In: Damasio, A. R.: *Descartes' Irrtum: Fühlen, Denken und das menschliche Gehirn*. 1. List Taschenbuch, 2004. – ISBN 978–3548604435, S. 60

[18] Eliasmith, C. ; Bechtel, W.: Symbolic versus subsymbolic. In: *Encyclopedia of cognitive science* (2003)

[19] Fodor, J. A. ; McLaughlin, B. P.: Connectionism and the problem of systematicity: why Smolensky's solution still doesn't work. In: *Cognition* 35 (1990), Nr. 2, S. 183–204

[20] Fodor, J. A. ; Pylyshyn, Z. W.: Connections and symbols. Version: 1988. `http://dl.acm.org/citation.cfm?id=58066.58067`. Cambridge, MA, USA : MIT Press, 1988. – ISBN 0–262–66064–4, Kapitel Connectionism and cognitive architecture: a critical analysis, 3–71

[21] Fuchs, A. F.: The saccadic system. In: Rita, P. Bachy (Hrsg.) ; Collins, C.C. (Hrsg.): *The Control of Eye Movements*. Academic Press Inc, 1971. – ISBN 9780120710508, S. 343–362

[22] Hebb, D. O.: *The Organization of Behavior: A Neuropsychological Theory*. Parental Adviso. Lawrence Erlbaum Assoc Inc, 2002. – ISBN 0805843000

[23] John, B. E.: CogTool: Predictive Human Perfomance Modeling by Demonstration. In: *Proceedings of the 19th Conference on Behaviour Representation in Modeling and Simulation*, 2010

[24] Kant, I.: *Metaphysische Anfangsgründe der Naturwissenschaft*. Riga, 1786

[25] Karat, J.: A model of problem solving with incomplete constraint knowledge. In: *Cognitive Psychology* 14 (1982), Nr. 4, S. 538–559

[26] Legendre, G. ; Smolensky, P. ; Smolensky, P. ; Miyata, Y. ; Miyata, Y.: Principles for an Integrated Connectionist/Symbolic Theory of Higher Cognition. 1992. – Forschungsbericht

[27] Lehman, J. F. ; Laird, J. ; Rosenbloom, P. u. a.: A gentle introduction to Soar, an architecture for human cognition. In: *Invitation to Cognitive Science: Methods, Models, and Conceptual Issues* (2006), S. 211

[28] Lohse, G. L.: A Cognitive Model for Understanding Graphical Perception. In: *Human-Computer Interaction* 8 (1993), Nr. 4, S. 353–388

[29] Lohse, J.: A Cognitive Model for the Perception and Understanding of Graphs. In: *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, ACM New York, 1991, S. 137–144

[30] Menary, R.: *Radical enactivism: Intentionality, phenomenology, and narrative: Focus on the philosophy of Daniel D. Hutto*. Bd. 2. John Benjamins Publishing Company, 2006

[31] Menary, R.: *The Extended Mind*. MIT Press, 2010

[32] *Kapitel* A Theory of Graph Comprehension. In: Pinker, S.: *Artificial Intelligence and the Future of Testing*. Lawrence Erlbaum Assoc Inc, 1990. – ISBN 0805801170, S. 73–126

[33] Polson, P. G. ; Kieras, D. E.: A quantitative model of the learning and performance of text editing knowledge. In: *ACM SIGCHI Bulletin* 16 (1985), Nr. 4, S. 207–212

[34] Polson, P. G. ; Muncher, E. ; Engelbeck, G.: A test of a common elements theory of transfer. In: *ACM SIGCHI Bulletin* Bd. 17, ACM New York, 1986, S. 78–83

[35] Rumelhart, D. E. ; Hinton, G. E. ; Williams, R. J.: Learning representations by back-propagating errors. In: *Nature* 323 (1986), Nr. 6088, S. 533–536

[36] Skinner, B. F.: Wissenschaft und menschliches Verhalten, Kindler Verlag, 1973, S. 41

[37] Tack, W. H.: Wege zu einer differentiellen kognitiven Psychologie. In: *Bericht über den 39. Kongreß der Deutschen Gesellschaft für Psychologie in Hamburg*, K. Pawlik, 1994, S. 117

[38] Thompson, R.: Das Gehirn: Von der Nervenzelle zur Verhaltenssteuerung, Spektrum Akademischer Verlag, 2001

[39] Turing, A.: On computable numbers, with an application to the Entscheidungsproblem. In: *B. Jack Copeland* (2004), S. 58

[40] Veksler, B.: *Modeling the visual search process*, Rensselaer Polytechnic Institute, Diss., 2011

[41] Wallach, D.: *Komplexe Regelungsprozesse: Eine kognitionswissenschaftliche Analyse*. Deutscher Universitäts-Verlag, 1998. – ISBN 3–8244–4309–0

[42] *Kapitel* Visual Queries. In: Ware, C.: *Visual Thinking for Design*. Denise E. M. Penrose, 2008. – ISBN 978–0–12–370896–0, S. 12–14

[43] Watson, J. B.: Psychology as the Behaviorist Views it. In: *Psychological Review* 20 (1913), Nr. 2, S. 158–277

[44] Wertheimer, M.: Untersuchungen zur Lehre von der Gestalt. II. In: *Psychologische Forschung* 4 (1923), 301-350. http://dx.doi.org/10.1007/BF00410640. – DOI 10.1007/BF00410640. – ISSN 0033–3026

[45] Wikipedia: *Nervenzelle — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Nervenzelle&oldid=105498919. Version: Juli 2012

[46] Wundt, W. M.: Kritische Nachlese zur Ausfragemethode. In: *Archiv für die gesamte Psychologie* Bd. 12, 1908, S. 445–459

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Stephan Engelhardt)