Universität Bielefeld Technische Fakultät

Diplomarbeit von
Markus von der Heyde
im Fach
Naturwissenschaftliche Informatik

Konzeption und Implementierung von Gedächtnisleistungen

Betreuer:
Dipl. Ing. Nils Jungclaus
Prof. Dr. Ing. Gerhard Sagerer

25. Juli 1997

SFB 360 *Situierte Künstliche Kommunikatoren*

Vorwort

Gedächtnisleistungen sind eigentlich viel zu komplex, um in einer solchen Diplomarbeit angemessen behandelt zu werden. Die vorliegende Arbeit versucht dennoch, einen kleinen Einblick in die Möglichkeiten zu eröffnen, Informationen in Anlehnung an biologische Vorbilder zu verarbeiten. Dabei lege ich besonderen Wert auf die biologische Motivation der einzelnen Verarbeitungsschritte und Verarbeitungsprinzipien.

Meine Arbeit bietet einen Rahmen, in dem weitere Verarbeitungsschritte im Kontext des assoziativen Speicherns von Informationen implementiert und getestet werden können. Die entworfene Architektur ist meines Wissens neu und es wurde bisher noch nicht versucht, mit einer solchen, den biologischen Systemen ähnlichen Struktur Informationen vernetzt zu speichern. Ich habe ein Grundgerüst erarbeitet, in das auch andere Szenarien als die visuelle Informationsverarbeitung des Sonderforschungsbereichs 360 “Situierte Künstliche Kommunikatoren” (SFB) hineinpassen. Durch die Einbettung in den SFB eröffnet sich ein breites Testfeld für die Leistungsfähigkeit des Konzeptes und des Programms. Es soll hier jedoch ein universeller Ansatz zur assoziativen Verarbeitung und Speicherung komplexer, vernetzter Information vorgestellt werden.

Besonderen Dank möchte ich an dieser Stelle meinen Betreuern Nils Jungclaus und Gerhard Sagerer aussprechen, ohne deren Bereitschaft zu vielen Stunden der Diskussion und Beratung die Arbeit erheblich schwieriger gewesen wäre. Weiter möchte ich meinem Kommilitonen Steffen Neumann für sein offenes Ohr bei Fragen zum Betriebssystem und zur Verständlichkeit meiner Ideen danken. Meine Frau Claudia half der Orthographie und manchen Formulierungen der Arbeit auf die Sprünge und deshalb möchte ich ihr an dieser Stelle ganz besonders danken. Der Druck und die Ausführung erfolgte in LaTeXund mit dem Leitfaden und vielen Tips aus [Kop91] und [GMS94].

Hiermit erkläre ich, daß die vorliegende Arbeit von mir selbständig und nur unter Verwendung der erlaubten und aufgeführten Hilfsmittel erstellt wurde.

Markus von der Heyde, Juli 1997

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

1.1 Motivation

In der Informatik werden an vielen Stellen Algorithmen verwendet, die aus einem Satz von Eingabedaten ein Ergebnis berechnen. Es ist ebenfalls Standard, daß diese Berechnungen für verschiedene Eingabesätze häufig wiederholt werden. Die meisten Algorithmen nutzen die Informationen, die sie aus den vorherigen evtl. ähnlichen Berechnungen gewinnen konnten, nicht, obwohl dadurch bei der erneuten Berechnung Zeit gespart werden könnte. Biologische Systeme haben demgegenüber den Vorteil, aus mehrmals verrichteten Vorgängen zu lernen und sich bei Ähnlichkeiten innerhalb der Vorgänge "Arbeit" zu ersparen. Sie erkennen die Ähnlichkeiten mit Hilfe ihres Gedächtnisses. Es ist also sinnvoll, eine Art Gedächtnis für Algorithmen zu entwickeln. Viele Berechnungen lassen sich intern durch eine dem biologischen Gedächtnis ähnliche Form nicht beschleunigen. Dennoch ist ein Effizienzgewinn möglich, bei dem der Algorithmus selbst nicht geändert zu werden braucht, wenn zuvor die Entscheidung getroffen werden kann, ob die Berechnung aus diesen Daten überhaupt ausgeführt werden muß. Eine Leistungssteigerung ist nur möglich, wenn die Entscheidung weniger aufwendig ist als die Berechnung selbst.

In komplexen Bildverarbeitungsverfahren ist eine Informationsreduktion häufig erwünscht, um den Arbeitsaufwand zu minimieren. Ein Gedächtnis für visuelle Eindrücke könnte diese Verfahren in Anlehnung an das biologische Vorbild entlasten und überflüssige, wiederholte Berechnungen vermeiden. In einem Teil des Sonderforschungsbereich 360 "Situierte Künstliche Kommunikatoren" der Universität Bielefeld (kurz SFB) wird eine Kaskade aus komplexen Einzelberechnungen benutzt, um aus digital aufgenommenen Kamerabildern einer Szene mit Baufixteilen eine vollständige 3D-Rekonstruktion zu berechnen. Auf diesem Verarbeitungsweg von der Kamera bis zur Repräsentation im Computer könnten an mehreren Stellen Gedächtnismodule den Informationsfluß reduzieren und stabilisieren. Den einzelnen Bildverarbeitungsmodulen könnte Arbeit erspart und die Gesamtperformance des Systems gesteigert werden.

Im Rahmen dieser Diplomarbeit habe ich ein Konzept erstellt und implementiert, das Gedächtnisleistungen für visuelle Eindrücke ermöglicht. Die vorliegende Dokumentation beschreibt die biologischen Grundlagen, auf denen die Architektur basiert und zeigt die Realisierung des Konzeptes. Der einführende Teil geht auf die Besonderheiten im SFB ein, beleuchtet die Voraussetzungen der Architektur und definiert einige grundlegende Begriffe. Im Kapitel 2 stelle ich die

entworfene Architektur dar und erläutere in den einzelnen Punkten die biologischen Vorbilder und ihre Umsetzung in das technische System. Das Kapitel 3 über die Implementierung beschreibt einige “Highlights” aus der Umsetzung des Konzeptes. In einer Zusammenfassung der Leistungen des implementierten Programms erläutere ich in Kapitel 4 einige grundsätzliche Phänomene, die das Gedächtnis als solches auszeichnen. Danach schließe ich die Arbeit mit einem Ausblick (Kapitel 5) auf mögliche Weiterentwicklungen des Konzeptes und der Implementierung. Im Anhang fasse ich meine Programmiertechniken zusammen, um einen Leitfaden für die Erweiterung zu geben, und lege die im SFB verwendeten Datenstrukturen dar.

1.2 Kontext im SFB

Gemeinsames Ziel des SFBs ist die Schaffung einer Mensch-Maschine-Schnittstelle bei Konstruktionsaufgaben. Über eine Spracheingabe soll eine Konstruktion eines Flugzeugs oder ähnlichen Spielzeugs aus Baufixteilen beschrieben werden können, die parallel dazu von einem Robotersystem ausgeführt wird. Die Szene wird von mehreren Kameras aufgenommen und kontinuierlich vom SFB-System ausgewertet, um eine integrierte Sprach- und Bilderkennung durchführen zu können. Auf dieser Analyse baut die Sprachgenerierung und später auch die Steuerung von Roboterarmen auf. Es sind eine Vielzahl von Aufgaben aus den Gebieten der Robotik, Bildverarbeitung und Sprachverarbeitung zu lösen. Der SFB besteht aus Teilprojekten, die jeweils ein Problemfeld bearbeiten; die daraus entstehende gemeinsame Lösung setzt sich wie in Abbildung 1.1 dargestellt zum vollständigen SFB-System zusammen. Die Abbildung soll den Überblick ermöglichen, muß aber nicht im Detail verstanden werden, um den weiteren Ausführungen folgen zu können.

Bildverarbeitung im SFB

Meine Arbeit soll in dem bildverarbeitenden Teil des SFBs eingesetzt werden. Innerhalb der Bildverarbeitung gliedert sich die Aufgabe von der Kamera bis zur vollständigen 3D-Rekonstruktion in folgende Schritte:

Eine Szene mit Baufixteilen wird von mehreren Kameras aufgenommen und als Folge von Stereobildern in Farbe und Schwarz/Weiß zur Verfügung gestellt. Abbildung 1.2(a) zeigt einen Ausschnitt aus einer Szene; es werden hier vier verschiedene Baufixteile dargestellt, die in den folgenden Verarbeitungsschritten als Beispiel dienen sollen. Diese Bilder werden von Kantendetektoren und Farbsegmentierern weiterverarbeitet. Abbildung 1.2(b) zeigt die gefundenen Kanten im Beispielbild; Abbildung 1.2(c) zeigt die gefundenen Farbregionen aus dem entsprechenden Farbbild. Die Regionen werden mit den Ergebnissen einer holistischen Objekterkennung zu einer Hypothese des “Gesehenen” als 2D-Objekte verarbeitet. In Abbildung 1.2(d) sind die erkannten 2D-Objekte im Bild markiert. Die Informationen werden kombiniert und in einer 3D-Rekonstruktion in eine virtuelle Szene übersetzt, die dem Original möglichst ähnlich sein sollte.

Datenstrukturen

Zwischen den einzelnen Verarbeitungsschritten fließen verschiedene Informationen auf verschiedenen Abstraktionsebenen. Die Daten werden im SFB in einem systemweit vereinbarten Format

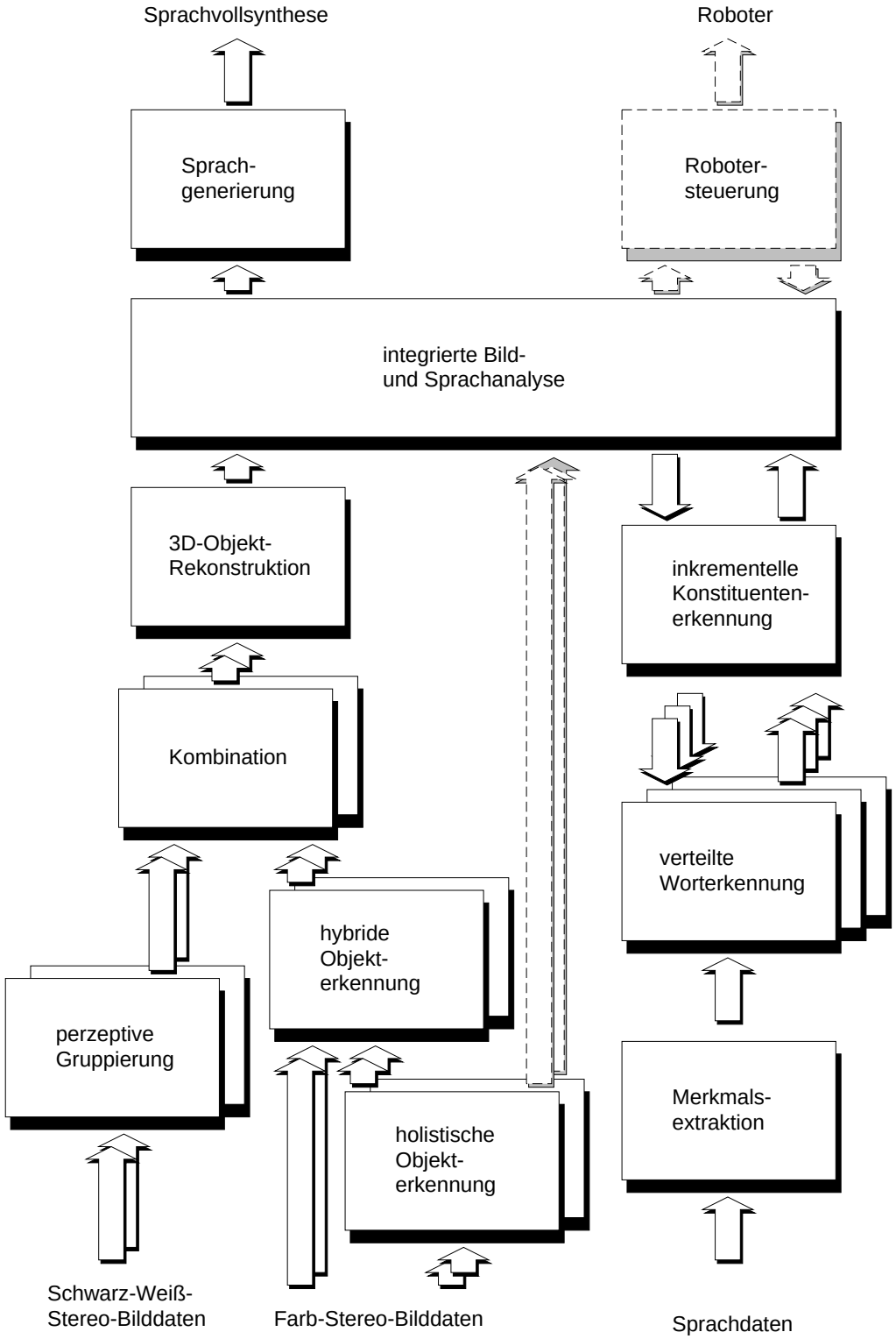
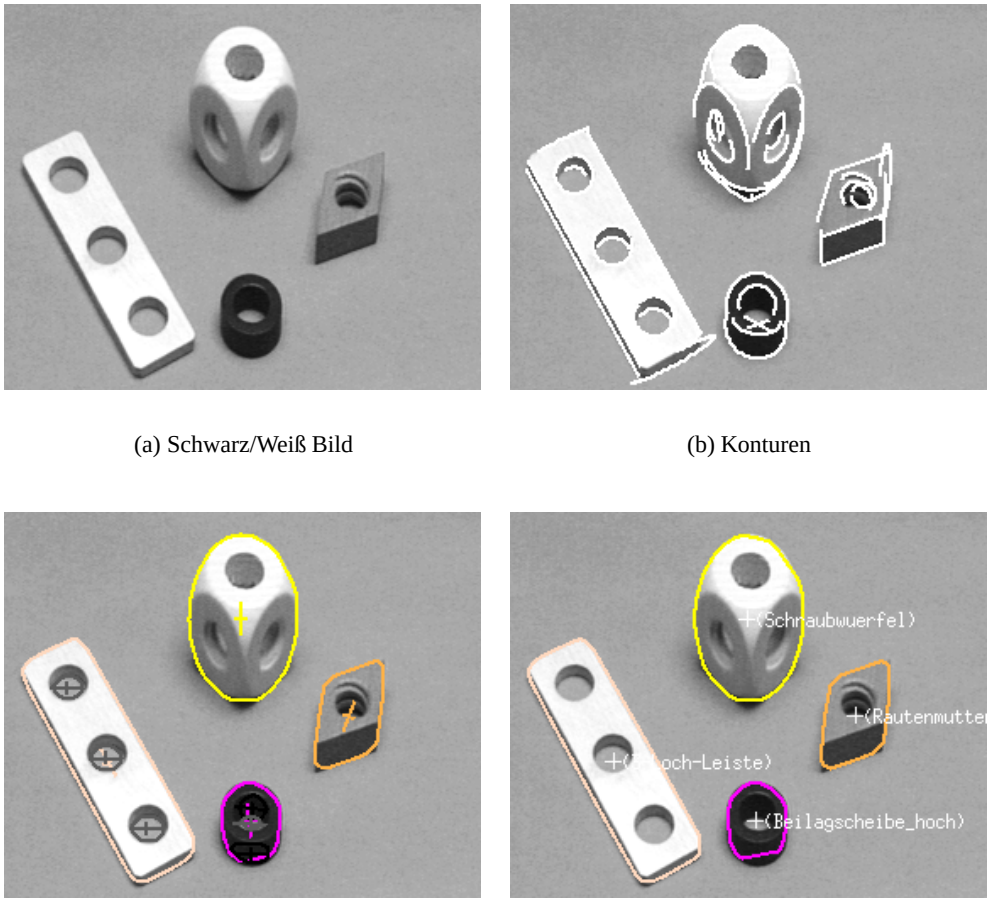


Abbildung 1.1: SFB Architektur von Bild- und Sprachverarbeitung



(a) Schwarz/Weiß Bild

(b) Konturen

(c) Regionen

(d) 2D-Objekte

Abbildung 1.2: SFB-Datenformate

in den verschiedenen Applikationen verarbeitet. Die Applikationen benutzen dazu die Datenstrukturen, die im Anhang B definiert werden, mit denen komplexe SFB-Objekte aufgebaut werden können. Einige der SFB-Objekte ermöglichen rekursive Bezüge dieser dynamisch allozierbaren Datenstrukturen.

Um einen kleinen Einblick zu ermöglichen, ohne die C-Definitionen genau wiederzugeben, fasse ich kurz die wichtigsten SFB-Objekte zusammen. Es gibt neben den reinen Bilddaten in Farbe und Schwarz/Weiß folgende Strukturen:

- **Kontur:** ein Ellipsenbogen mit Start- und Endpunkt, Mitte und Radien (siehe Beispiel in Abbildung 1.2(b)).
- **Region:** ein umschließender Polygonzug (umschreibende Farblinien in Abbildung 1.2(c)) und mehrere beschreibende Größen wie Hauptachse, Farbe, Umfang und Exzentrizität.
- **2D-Objekt:** besteht aus Baufix-Teilbezeichner, Farbe, Region und Sub-2D-Objekten (siehe Beschriftung in Abbildung 1.2(d)).
- **3D-Objekt:** besteht aus Baufix-Teilbezeichner, Farbe, Positionsbeschreibung und Sub-3D-Objekten.

Die Datenstrukturen werden in jedem Verarbeitungsschritt im SFB verschieden gefüllt bzw. in andere Strukturen umgewandelt. Die C-Strukturen werden nicht in jedem Schritt vollständig ausgenutzt. Zwischen den SFB-Applikationen wird der Typ des Informationsflusses zuvor definiert und ein Name für den Datenstrom vereinbart. Die Verarbeitungseinheiten werden zwar im Rechnerverbund verteilt, doch ist bekannt, unter welchem Namen die Komponenten im DACS angesprochen werden können. Jeder Datenstrom (Demand Stream) hat systemweit einen eindeutigen Namen, damit die Daten beim vereinbarten Empfänger ankommen.

Einsatzort Gedächtnis

Das Gedächtnis soll zwischen verschiedene Verarbeitungsschritte (Applikationen) geschaltet werden, um den Informationsfluß zu glätten und die Informationen von mehreren Bildern (bzw. den Verarbeitungszustand von Bildern zu verschiedener Zeitpunkten) verarbeiten zu können. Dabei stehen dem Gedächtnis an den einzelnen Punkten im Verarbeitungsfluß des SFB verschiedenste Informationen zur Verfügung. Die Abstraktionsebenen und Datenmengen sind zum Teil sehr unterschiedlich. Abbildung 1.3 zeigt mögliche Einsatzorte für ein Gedächtnis im visuellen Teil des SFBs.

1.3 Vorbild Biologie

Die Biologie bietet eine Fülle von Ideen und Realisierungen von Gedächtnisleistungen. Ich möchte nur einige grundsätzliche Phänomene herausgreifen, die weit über unterschiedliche Tierklassen innerhalb der Wirbeltiere verbreitet sind. Einen umfassenden Überblick über den aktuellen Stand der Forschung auf dem Gebiet der Neurobiologie bekommt man in [KSJ96]. Die Autoren Eric R. Kandel, James H. Schwartz und Thomas M. Jessell legen die neurobiologischen Grundlagen dar und entwickeln darauf aufbauend Modelle zur Wahrnehmung und Speicherung von Informationen. Eine populärwissenschaftliche, aber fundierte und gut verständliche Heranführung an das Thema leistet das Buch “Denken, Lernen, Vergessen” [Ves96]. Frederic Vester beschreibt eine Reihe von Phänomenen und deren in der Wissenschaft vorherrschenden Erklärungsversuche. Aus [Bad79] von Alan D. Baddeley entnahm ich viele Ideen zu psychologischen Aspekten des Gedächtnisses. Neben den Unterschieden zwischen Kurz- und Langzeitgedächtnis wird dort auch das Vergessen ausführlich diskutiert. Baddeley gibt eine Zusammenfassung mehrerer Artikel über Modalitätenmodelle und geht auf die visuelle Kurzzeitspeicherung ein.

Die genannten Bücher eröffnen viele Einblicke in die Möglichkeiten der Verarbeitung von Informationen in biologischen Systemen. Zur Unterscheidung der Verarbeitungsstadien möchte ich den Weg der Informationen von der Aufnahme (Wahrnehmung), über die Verarbeitung/Speicherung (Gedächtnis) bis zur Ausgabe (Reproduktion) verfolgen. Ich greife einige mir wichtige Aspekte heraus, an denen ich phänomenologisch die zu entwerfende Architektur anlehnen möchte. Die von der Natur gezeigten Leistungen sind das Ziel; die Verarbeitung von Informationen im Programm und im Organismus ist grundlegend verschieden. Die internen Verarbeitungsschritte biologischer Systeme regen manche Programmierprinzipien an, sind aber im Detail wesentlich komplexer und programmtechnisch nicht nachzuempfinden.

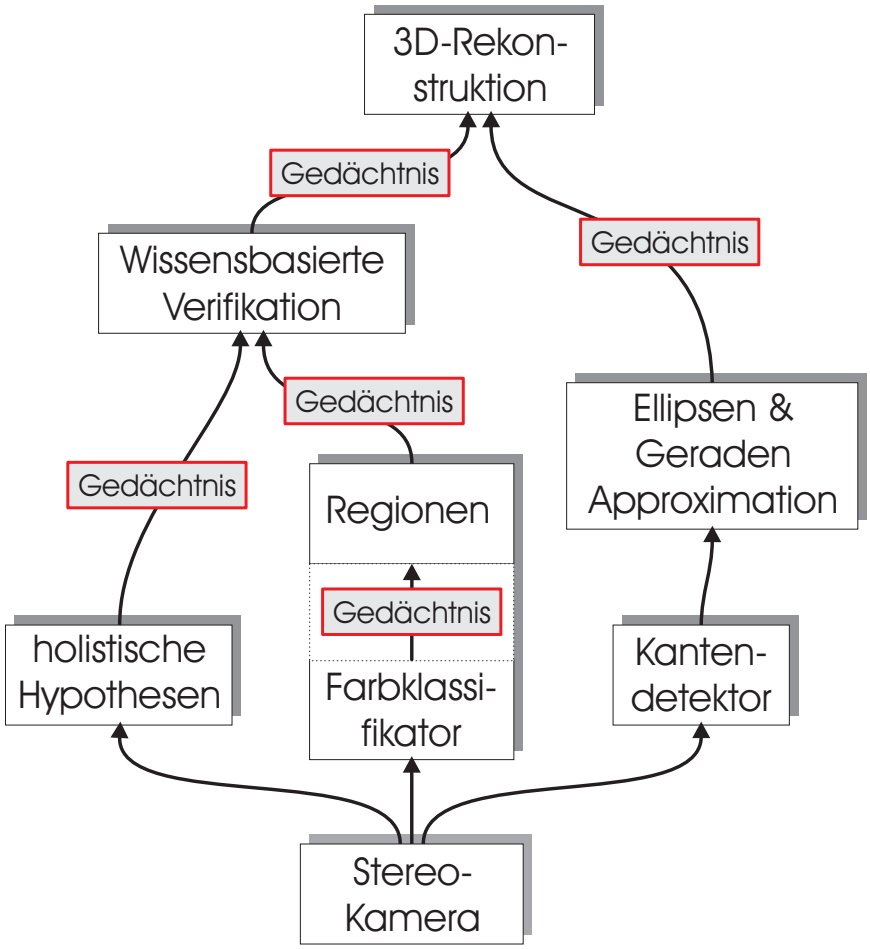


Abbildung 1.3: Mögliche Einsatzorte des Gedächtnisses im SFB

Wahrnehmung

Um als Tier (und als Mensch) Informationen zu verarbeiten, ist es nötig, Informationen aufzunehmen. Die Umwelt umgibt einen Organismus und dieser nimmt die Umwelt mit seinen Möglichkeiten wahr. “Neue”, kreativ erfundene Informationen, die aus dem Organismus heraus entstehen, lasse ich hier bewußt beiseite, da diese Art von Information noch viel schwieriger zu fassen und zu formalisieren ist als die von außen aufgenommenen Informationen.

Die Aufnahme von Informationen erfolgt allgemein durch Sensoren, die in der Biologie als *Rezeptoren* bezeichnet werden. Die Sinnesorgane (zum Beispiel Augen) setzen sich aus einer ganzen Reihe von Rezeptoren zusammen, die jeweils den Zustand oder die Veränderung einer physikalischen oder chemischen Größe messen. Die gemessene Größe wird als adäquater *Reiz* bezeichnet, wenn sie eine Erregung des Rezeptors verursacht. Der Rezeptor wandelt die Erregung (die Information über die gemessene Größe) in elektrische Impulse (Aktionspotentiale), die über sensorische Bahnen (Nerven) weiter bis zum Cortex geleitet werden. Die Art der gemessenen Größe (zum Beispiel Luftschwingung = Schall) wird als *Modalität* bezeichnet. Der Reiz einer spezifischen Modalität löst im Cortex eine bestimmte Empfindung aus (sehen, hören, fühlen, etc.). Ein Rezeptor

nimmt für ihn spezifische Reize auf, und verschiedene Rezeptoren können derselben Modalität zugeordnet sein. Die Reize werden vom Rezeptor bis zum Cortex in getrennten sensorischen Bahnen verarbeitet.

Die Reizqualität oder Reizstärke ist eine Spezifizierung innerhalb der Modalität (zum Beispiel rot, grün; süß, sauer). Die Stärke eines Reizes kann präzise trotz großer Wertebereiche wahrgenommen werden, indem sich der Rezeptor adaptiert. Diese neuronale Adaptation reduziert schon bei der Wahrnehmung das elektrische Signal der Rezeptorzelle (siehe Kapitel 7 über das visuelle System aus [ZR94]).

Die Aktivierung eines Rezeptors findet in einem begrenzten Bereich, seinem rezeptiven Feld, statt. "Die Größe dieses Feldes hat entscheidende Bedeutung für die räumliche Auflösung eines sensorischen Systems: Eine höhere Auflösung wird durch Nervenzellen mit kleineren rezeptiven Feldern erreicht" (aus Kapitel 20. [KSJ96]: DIE SENSORISCHEN SYSTEME). Die Unterscheidbarkeit zweier Reize gleicher Modalität wird durch laterale Hemmung benachbarter rezeptiver Felder verbessert. Außerdem ist die räumliche Verteilung der Rezeptoren den verschiedenen Anforderungen angepaßt, wie aus Untersuchungen der Mechanorezeptoren in der Haut bekannt ist.

Verarbeitung

Jede weitere Verarbeitung eines Reizes nach der Aufnahme durch die Rezeptoren ist mit der Funktion des Gedächtnisses verknüpft. Zur Gedächtnisleistung rechne ich nicht nur die bloße Speicherung von Wissen, sondern auch die Wirkung des Gespeicherten auf die weitere Aufnahme von Reizen und deren Interpretation.

Die Verarbeitung der aufgenommenen Reize wird im Cortex weitgehend getrennt und unabhängig für die verschiedenen Modalitäten durchgeführt. Unser Gehirn analysiert in getrennten Bahnen die Form, Farbe und Bewegung von Objekten, bevor das "innere Bild" der Szene aktiv (re-)konstruiert wird. An Patienten mit lokalen Verletzungen des Gehirns (Läsionen) können viele Phänomene beobachtet werden, die Rückschlüsse auf die Verarbeitungswege zulassen. Die "Auswertung" der Modalitäten geschieht parallel und spezifisch. Dennoch sind zum Teil Verknüpfungen zwischen den verschiedenen Modalitäten für deren Interpretation und weitere Verarbeitung wichtig. In der visuellen Verarbeitung kommt es zur Verknüpfung der Bahnen für Bewegung, Form und Farbwahrnehmung (nach Untersuchungen von Van Essen et al. an Makaken [VG94]).

Die Berührungsempfindungen der gesamten Körperoberfläche werden im primären somatosensorischen Cortex in topologieerhaltenden Karten repräsentiert (nach Kapitel 18 in [KSJ96]). Auch für die anderen Sinnesmodalitäten (zum Beispiel Muskel- und Gelenkinformationen) gibt es solche Karten, wobei jeweils besondere Anforderungen an die neuronale Architektur der Verarbeitung gestellt werden. Durch den intensiven Gebrauch oder auch den Nichtgebrauch der repräsentierten Rezeptor-Bereiche lassen sich die neuronalen Verknüpfungsmuster drastisch verändern, wie verschiedene Versuche mit Affen zeigen.

Der einzelne Reiz hat neben seiner Modalitätsspezifität eine Dauer und eine zeitliche Einordnung, die eine spätere zeitabhängige Verarbeitung erlauben. Die Dauer wird im biologischen System durch eine Kombination von schnell und langsam adaptierenden Rezeptoren "berechnet". Durch den direkten neuronalen Vergleich zweier schnell adaptierenden Rezeptoren kann eine Vorher/Nachher-Relation für zwei Reize aufgestellt werden.

Durch eine Überlagerung von ähnlichen Reizen bei der Speicherung kommt es zu einem Verschwimmen einzelner Reize. Das Vergessen von Information steht im direkten Zusammenhang hiermit. Dabei ist noch ungeklärt, ob Vergessen durch das Überlagern von alter Information mit neuer Wahrnehmung oder durch einen reinen Zerfallsprozeß zu erklären ist. Die zeitliche Einordnung alter Empfindungen ist schwieriger und unzuverlässiger. Ein früher empfundener Reiz kann dennoch in seiner Reizstärke mit einem neuen Reiz verglichen werden; die Speicherung der Zeitinformation scheint von der Reizstärke unabhängig zu sein.

Bei der Verarbeitung werden Reize zu abstrakten Größen zusammengefaßt. Die Erkennung von Objektgrenzen beruht nach Untersuchungen von Treismann und Julesz [Tre86] auf der Verarbeitung von elementaren visuellen Eigenschaften wie Farbe, Helligkeit und Orientierung. Eine bewußte Verarbeitung als Einheit, das heißt als Eindruck, wird damit erleichtert und findet in der Kombination verschiedener sensorischer Bahnen im Cortex statt.

Reproduktion

Die Reaktion auf die Aufnahme von Information äußert sich zum Beispiel durch die Reproduktion der von uns verstandenen Information. Wir antworten in einer Diskussion unserem Partner mit dem, was wir von seinem Beitrag verstanden haben und betonen dabei die von uns gleichermaßen dargestellten Punkte und die Unterschiede zu unserer Meinung. Die Ausgabe von Informationen ist eine Form von Reaktion. Ich möchte nun drei Reaktionsinhalte zusammenfassen, die eng mit der Assoziation verknüpft sind:

- Es werden Reize zu Eindrücken zusammengefaßt und als Gesamtheit gespeichert. Die einzelnen Reize treten innerhalb des Eindrucks zurück, welcher als Ganzes wahrgenommen wird. Dieser Kontext wird auch als Situation bezeichnet.
- Anhand weniger Reize kann eine ganze Situation assoziiert werden. Einige kleinere Details können uns noch Jahre später an einen Kontext erinnern, in dem wir ähnliche Reize empfunden haben.
- Es gibt eine Art Aufmerksamkeitssteuerung, die auf der Änderung von Reizen oder anderen abstrakten Größen beruht. Treismann schlägt eine Kombination verschiedener Merkmalskarten zu einer Positionskarte vor, durch die unsere Aufmerksamkeit gesteuert werden könnte.

1.4 Ziele

Aus dem Kontext des SFB und den dargelegten Vorstellungen aus der Biologie ergeben sich folgende Anforderungen an ein Gedächtnis, das in seiner Funktionsweise und in seiner Leistung biologischen Systemen nahekommen soll. Dabei kommt es nicht auf die präzise Imitation im Detail an, sondern auf die Verallgemeinerbarkeit und die Assoziationsfähigkeit des Gesamtsystems.

Technische Aspekte

Das Hauptaugenmerk auf der technischen Seite liegt auf der einfachen und vollständigen Integration des Gedächtnisses in das bestehende SFB-Demosystem. Das Gedächtnis nutze alle Möglichkeiten der DACS-Kommunikation, um eine flexible Anbindung an verschiedene Programme des SFBs zu bieten. Es soll sowohl auf unterschiedlichen Abstraktionsebenen als auch für unterschiedliche Datentypen als Lang- oder Kurzzeitgedächtnis einsetzbar sein. Diese universelle Einsetzbarkeit erfordert eine universelle Grundstruktur des internen Aufbaus. Es müssen allgemeine Prinzipien umgesetzt werden, da eine oder mehrere Speziallösungen nicht in Betracht kommen.

Dabei soll die Effizienz nicht vernachlässigt werden. Das Gedächtnis muß mit den im SFB üblichen Datenraten von etwa einem Bild pro Sekunde zurechtkommen. Darüber hinaus ist eine hohe Parallelität erwünscht; die Parallelität biologischer Prozesse ist unerreichbar, aber dennoch sollen möglichst viele der Einzelaufgaben entkoppelt gelöst werden.

Die Architekturvielfalt im SFB, die von Linux/PC-Systemen über OSF/Alpha bis zu Solaris/Sparc und Irix/Silicon Graphics reicht, muß unterstützt werden, sofern die Betriebssysteme eine Parallelisierung auf Prozeßebene (Threads) zulassen. Weiter Hinweise gebe ich speziell in Abschnitt 3.1 und Anhang A.

Biologische Aspekte

Die oben gezeigten Möglichkeiten, Ideen aus der Biologie zu übernehmen, sind sehr vielfältig; deshalb beschränke ich mich auf einige grundsätzliche Forderungen. Informationen werden vom Gedächtnis aufgenommen und als Eindruck in räumlichen, zeitlichen und modalen Kontexten gespeichert. Die Speicherung soll wertabhängig sein, um Assoziation zu ermöglichen. Die Repräsentation von Raum und Zeit, sowie der verschieden-dimensionalen Modalitäten, paßt sich dynamisch den beobachteten Werten in der Auflösungsfähigkeit und im Wertebereich an.

Die Eindrücke werden mit dem Wissen aus dem Gedächtnis assoziativ verknüpft und es erfolgt eine Identifikation wiederkehrender Eindrücken. Dabei ist es wünschenswert, daß kleinere Veränderungen akzeptiert und detektiert werden. Durch assoziative Ergänzung fehlender Information soll über die Zeit hinweg der Datenstrom stabilisiert werden.

Es soll die Möglichkeit geschaffen werden, abstraktere Informationen wie Position, Veränderung, Bewegung und Dauer aus den beobachteten Daten zu gewinnen. Durch selbständige Beobachtung solcher Größen kann das Gedächtnis eigene Aktivität entwickeln und seine Aufmerksamkeit auf Besonderheiten lenken.

Eine hohe Parallelität in der Datenverarbeitung und eine Orientierung an biologischen Prinzipien sind Leitfäden für implementatorische Entscheidungen. Ein zusätzlicher linearer Komplexitätsaufwand ist dabei in Kauf zu nehmen, da er durch erhöhten Hardwareaufwand aufzufangen sein wird. Die Trennung in die Schritte Aufnahme, Verarbeitung und Reaktion soll sich in der Architektur widerspiegeln und kann die Parallelität des Systems fördern.

1.5 Begriffsdefinitionen

Um im folgenden präzise mit Begriffen umgehen zu können, aber auch um die von mir gemeinte Bedeutung der benutzten Bezeichnungen ein wenig von der alltäglichen Verwendung abzuheben, möchte ich einige Begriffe, die auch schon in der Einleitung Verwendung fanden, umschreiben, wobei ich versuche, die Unterschiede zwischen biologischen und technischen Systemen zu berücksichtigen.

Ein *Objekt* ist ein Gegenstand, über den konkrete Informationen bekannt sind. In unserer Umwelt erfahren wir das Objekt durch unsere Wahrnehmung. Ein Objekt hat eine Identität, die erfahrungsgemäß in der Ausprägung variiert, aber dennoch erhalten bleibt. Wir nehmen das Objekt wahr und ergänzen nach und nach die Informationen, die wir über das Objekt zusammengetragen haben. Im SFB-Kontext könnte zum Beispiel ein beliebiger Würfel ein Objekt sein. Dieser Würfel hat *seine Identität* und soll nicht mit anderen Würfeln verwechselt werden. Demgegenüber sind Objekte im programmiertechnischen Sinne Einheiten, in denen Daten und Funktionen zusammengestellt sind; die Funktionen arbeiten speziell mit diesen Daten und erlauben einen kontrollierbaren Zugriff auf die Daten. Dieses Konzept ist aus objektorientierten Sprachen bekannt.

Bei der Wahrnehmung eines Objektes durch *Eindrücke* erhalten wir Informationen in Form von einzelnen Reizen. Ein Eindruck besteht aus einer Menge von Reizen, die ein Objekt in seinen Eigenschaften beschreiben. Jeder einzelne *Reiz* beschreibt die Ausprägung des Objektes bezüglich einer Eigenschaft. Diese beschriebene Eigenschaft nennt man *Modalität*. Der Reiz beschreibt als Information den Wert innerhalb seiner Modalität; er wird zusätzlich durch den Ort und den Zeitpunkt seines Auftretens sowie seine Genauigkeit bezüglich der Sicherheit der Wahrnehmung beschrieben. Jede Modalität ist die Repräsentation einer physikalischen oder chemischen Größe, die durch adäquate Reize wahrgenommen werden kann. Die Reize, die einer Modalität zugeordnet werden können, beschreiben demnach eine bestimmte gemeinsame Eigenschaft verschiedener wahrgenommener Eindrücke.

Das *Gedächtnis* beschränkt sich in seiner Funktion nicht nur auf die Speicherung der wahrgenommenen Reize; es wirkt vielmehr auf den gesamten Informationsfluß von der Wahrnehmung über die Verarbeitung und Speicherung bis zur Ausgabe oder Verwendung des gespeicherten Wissens. Ich fasse den Begriff so weit, da schon bei der Wahrnehmung die ersten Veränderung der aufgenommenen Information geschieht und damit die Speicherung der Information beeinflußt wird. Zudem wirkt die Reproduktion der gespeicherten Information auf die Speicherung selbst zurück. Es handelt sich also nicht um einen linearen Informationsfluß, sondern vielmehr um ein verstricktes Netzwerk von immer wieder sich wechselseitig ändernden Informationsflüssen.

Kapitel 2

Architektur

2.1 Überblick

Das Gedächtnis soll nach den Anforderungen aus Abschnitt 1.4 sehr universell einsetzbar sein. Dies erfordert eine ebenso universelle Architektur, die im folgenden motiviert und erläutert werden soll. Mein hier vorgestellter Entwurf ist eine von vielen Möglichkeiten, die gestellten Anforderungen zu erfüllen. Ich habe diese Architektur so entwickelt und an vielen Stellen Designentscheidungen so getroffen, daß die Architektur flexibel, erweiterbar und dennoch handlich erscheint. Die Modularisierung und Parallelität waren häufig die ausschlaggebenden Kriterien.

Ich werde zunächst auf die einzelnen Teile des Gedächtnisses eingehen und den groben Datenfluß des Programms beschreiben. Die Abbildung 2.1 faßt einige der wichtigsten Aspekte der Speicherung im Gedächtnis zusammen und vermittelt eine Übersicht über die verschiedenen Bestandteile. In den folgenden Abschnitten gehe ich näher auf die Datenspeicherung und Vernetzung ein, beschreibe die Assoziation zwischen den Daten und binde das Gedächtnis mit der DACS-Kommunikation in den SFB ein. Zum Schluß dieses Kapitels beschreibe ich die Möglichkeiten der Visualisierung des Gedächtnisinhalts. Ich möchte betonen, daß die hier beschriebene Architektur nur in ganz wenigen Punkten speziell auf die Erfordernisse des SFBs eingeht. Sie bietet vielmehr eine universelle Struktur, die in vielen anderen Zusammenhängen ebenso einsetzbar ist.

Teile des Gedächtnisses

Mein Entwurf der Architektur ist modular aufgebaut und ermöglicht damit eine einfache Erweiterung und eine flexible Anpassung an andere Erfordernisse. Das eigentliche Gedächtnis ist vollständig abgekoppelt von der externen Datenrepräsentation, dem SFB-Format. Daraus ergibt sich die grobe Gliederung des Gesamtsystems in drei Abschnitte: Die externe Kommunikation (DACS), die Schnittstelle zum Gedächtnis (Port) und das Gedächtnis selbst (Memory).

Das DACS (Distributed Application's Communication System) realisiert im SFB die Kommunikation und Systemintegration (siehe [FJRS95]). DACS bietet mehrere Kommunikationsprimitiva: Sogenannte *Demand Streams* und *Remote Procedure Calls (RPC)*. Streams sind Datenströme, die

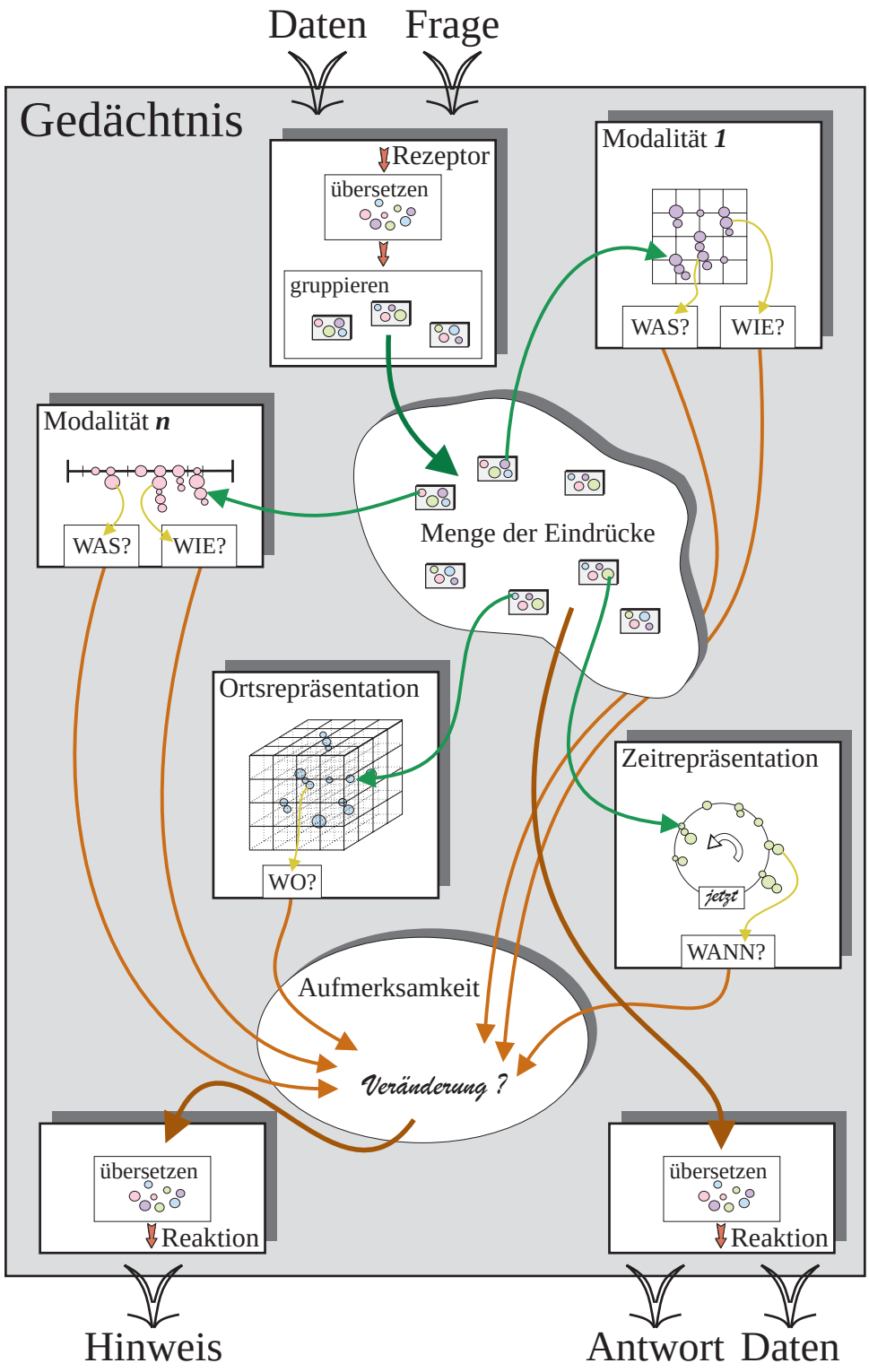


Abbildung 2.1: Informationsfluß im Gedächtnis

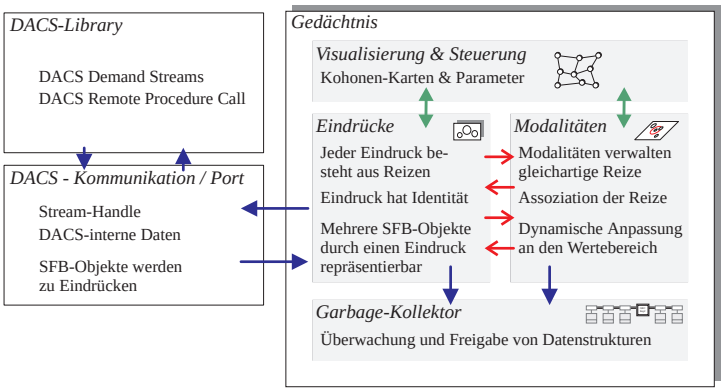


Abbildung 2.2: Modularer Aufbau des Gedächtnisses

auf Bestellung Daten von einem Programm zu einem anderen befördern. RPCs sind Funktionsaufrufe, die von einem Server angeboten werden und von einem Client genutzt werden können. Der Client kann eine Berechnung beim Server in Auftrag geben und bekommt das Ergebnis zurück.

Zwischen dieser Kommunikation mit DACS und dem eigentlichen Gedächtnis befindet sich eine Zwischenschicht (Port). Hier werden die Daten vom allgemeinen SFB-Format in das interne Format des Gedächtnisses übersetzt. Diese Schicht ist austauschbar und das Gedächtnis damit in anderen Anwendungen einsetzbar, sofern die Daten eine sinnvolle Speicherung in diesem Gesamtkonzept erlauben und von einer entsprechenden Schicht passend umgesetzt werden können.

Das Gedächtnis selbst gliedert sich in weitere Einzelteile, die in Abbildung 2.2 dargestellt sind. Die Trennung von funktionaler und Datenabstraktion ist in diesem Fall nicht durchgehend möglich, da einige der beschriebenen Module selbständig arbeiten und andere Module Dienste zur Bearbeitung von Daten anbieten. Ich trenne deshalb die einzelnen Aufgaben und Module in Verwaltungseinheiten.

Die Daten werden in *Reizen* gespeichert und verarbeitet. Reize, die das Gedächtnis zur selben Zeit aufnimmt, werden zu *Eindrücken* zusammengefaßt.[1] Ein Eindruck wird durch ein charakteristisches Reizmuster repräsentiert. Haben mehrere Eindrücke ein sehr ähnliches Reizmuster, sollen sie als ein Eindruck behandelt werden, der damit mehrere SFB-Objekte verknüpft. Die Menge aller Eindrücke bildet somit das Wissen und den Datenspeicher des Gedächtnisses.

Da im biologischen System die verschiedenen Reize anfangs getrennt ausgewertet werden, erfolgt in der technischen Abbildung eine Verarbeitung der Reize im Kontext ihrer Modalität. Innerhalb einer Modalität gelten spezifische Abstands- und Ähnlichkeitsmaße, so daß Vergleiche zwischen mehreren Reizen modalitätsspezifisch erfolgen können. Die Modalität paßt sich dem wahrgenommenen Wertebereich der angebotenen Reizen an und verfeinert ihr Auflösungsvermögen mit der dargebotenen Dichte der einzelnen Reize. Von der Modellierung der einzelnen Modalitäten hängt später die Assoziationsfähigkeit des Gedächtnisses ab.

Das technische System erfordert einige Einzelheiten, auf die das biologische System verzichten kann. Die Speicherverwaltung ist in dieser Architektur ein zentraler Punkt, da die Daten vernetzt gespeichert sind und zum Teil parallel von mehreren Stellen aus auf sie zugegriffen wird. Da der

[1] In anderen Zusammenhängen heißt eine solche Zusammenstellung Daten- oder Merkmalsvektor.

Hauptspeicher der verwendeten Rechner stark begrenzt ist, muß das Datenaufkommen in Grenzen gehalten werden. Ein aktives “Vergessen” sorgt dafür, daß alte und uninteressante Daten von neuen Daten überlagert werden. Eine Begrenzung der Dauer der Speicherung steuert somit die Aufnahmekapazität des Gedächtnisses und den Charakter als Lang- oder Kurzzeitgedächtnis.

Eine Visualisierung der Vorgänge im Gedächtnis ist notwendig, um verschiedene Steuerungsaufgaben wahrnehmen zu können und einen Eindruck von der Funktion des Systems zu bekommen. Zum einen kann die Assoziation über Demand Streams beobachtet werden, da auf verschiedenen Streams das “Wiedererkannte”, “Neue” und das “Unbekannte” ausgegeben wird. Ein weiterer Stream mit interessanten Daten, die einer offensichtlichen Änderung unterliegen, kann andere Programme von der Verarbeitung immer gleichbleibender Daten entlasten. Zum anderen soll der Benutzer Einsicht in die Dynamik der Speicherung in den verschiedenen Modalitäten erhalten und so die Fähigkeiten des Gedächtnis beurteilen und gezielt verändern können.

Datenfluß im Gedächtnis

Zwischen den oben skizzierten Verwaltungsmodulen ist ein Datenaustausch bzw. eine Vernetzung der Daten notwendig, um die geforderten Fähigkeiten zu realisieren. Eine Übersicht bietet die Abbildung 2.3. Die einzelnen Kästen stellen Funktionen oder eigenständige Prozesse innerhalb des Systems dar.

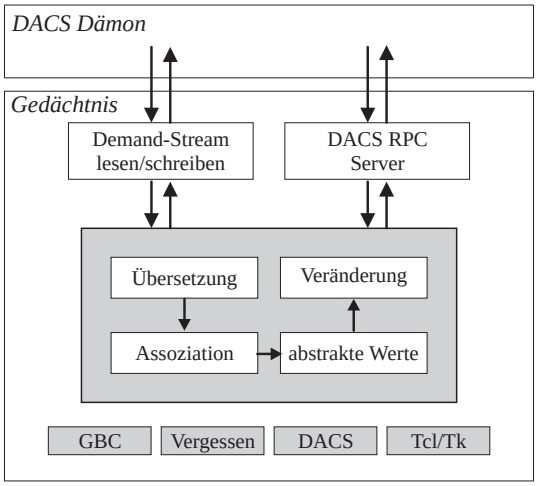


Abbildung 2.3: Datenfluß zwischen den funktionalen Einheiten

Die Aufnahme der Daten erfolgt über DACS Demand Streams oder die direkte Anfrage mit Hilfe der DACS RPCs. Die Daten werden in jedem Fall sofort in das Gedächtnisformat (also in Reize) umgewandelt. Jedes einzelne SFB-Objekt wird selbst weiter gespeichert, liegt aber zusätzlich als “Reizmuster” in einem Eindruck vor. Der Eindruck ist eine Art Datencontainer, in dem die Daten die Verarbeitungsschritte durchlaufen und ihren Zusammenhalt behalten. Die einzelnen Reize werden zusätzlich in den jeweiligen Modalitäten verwaltet, um eine wertabhängige Speicherung zu realisieren.

Der so gespeicherte Eindruck wird mit der bestehenden Datenbasis abgeglichen, indem nach schon bekannten ähnlichen Eindrücken gesucht wird. Diese Assoziation bildet den Kern des Wiedererkennens von Objekten und der Stabilisierung des Datenstroms. Wird der neue Eindruck als ähnlich

zu einem bekannten Eindruck bewertet, verschmelzen die beiden Eindrücke, indem die Daten (Reize und SFB-Objekte) gemeinsam als ein Eindruck weiterverarbeitet werden.

Diese aktuellen Reize eines neuen Eindrucks werden zur Berechnung von abstrakten Größen herangezogen. Abstrakte Größen liefern Maßzahlen für nicht sensorisch wahrnehmbare Eigenschaften. Zum Beispiel wird die Größe der Veränderung eines Eindrucks berechnet, indem die Summe der Abstände aller Reize innerhalb bestimmter Modalitäten gebildet wird. Diese berechnete Zusatzinformation wird von sog. *Fokusfunktionen* beobachtet und der Eindruck bei bestimmten Konstellationen der abstrakten Werte als interessant eingestuft und gesondert auf einem Stream ausgegeben. Diese Fokusfunktionen sollen die Aufmerksamkeitssteuerung des biologischen Vorbildes nachbilden.

Im normalen Datenfluß (Verarbeitung des Demand Streams) wird das neue oder das wiedererkannte Reizmuster als SFB-Objekt auf einem Stream anderen Programmen wieder zur Verfügung gestellt. Es erfolgt hier keine aktive Wandlung in ein neues SFB-Objekt, sondern ein Anpassung der Werte des Objektes an die gelernten ähnlichen Daten.

Wird dagegen der Datenfluß des Gedächtnisses funktionsorientiert mit RPCs benutzt, so erhält das “fragende” Programm als Antwort auf ein SFB-Objekt das im Gedächtnis gespeicherte SFB-Objekt des assoziierten Eindrucks. Man kann dieses Verhalten mit dem Beantworten von Fragen vergleichen, wobei auch an der Frage selbst gelernt werden soll und die Frage nicht nur einen Abruf von Wissen darstellt.

Die Verarbeitungsschritte sind also auf beiden “Wegen” durch das Gedächtnis gleich und unterscheiden sich nur in der direkten Zuordnung von Frage zu Antwort in dem Fall des RPCs. Werden dagegen die Daten über einen Demand Stream aufgenommen, so werden in ungeordneter Reihenfolge die mit diesen Objekten assoziierten Eindrücke wieder ausgegeben. Daneben entwickelt das Gedächtnis eigene Aktivität, indem es durch die Fokusfunktionen auf interessante Objekte, die einer Veränderung unterliegen, hinweist.

2.2 Speicherung von Daten

Wie beim biologischen Vorbild erfolgt die Aufnahme der Information durch Reize. Diese definieren sich durch die Reizstärke, eine Modalität und einen Reizort. In der Biologie wird meist eine Reizdauer verarbeitet, die im technischen System durch den Zeitpunkt des Auftretens des Reizes modelliert wird, da die zur Verfügung stehenden Daten jeweils nur die Momentaufnahme einer Situation darstellen und eine Dauer erst aus dem zeitlichen Abstand zweier einander zugeordneter Reize berechnet werden kann. Weiter verfügt ein biologisches System häufig über eine Größe, welche die Reizwichtigkeit bzw. Genauigkeit modelliert (manche Reize werden in ihrer Verarbeitung anderen “bevorzugt”, wenn diese zum Beispiel mit Angst, Freude oder Schmerz verbunden sind). Im technischen System betrachte ich eine solche Maßzahl als Wahrscheinlichkeit des Auftretens des Reizes bzw. als die Genauigkeit, mit der das System diesen Reiz “wahrgenommen” hat. Die Genauigkeit eines Reizes kann unabhängig von der Häufigkeit des Reizes modelliert werden.

Lernen

Gelernt wird in dieser Architektur in zweierlei Hinsicht: Zum einen werden die Reize als Reizmuster eines Eindrucks betrachtet und ein Eindruck damit nach und nach durch die Assoziationen vervollständigt und erweitert; zum anderen lernt die Modalität, zwischen ähnlichen Reizen zu unterscheiden, indem der Wertebereich sich dynamisch den angebotenen Reizen anpaßt.

Vergessen

Das biologische System hat im Bereich der Kurzzeitspeicherung nicht das “Problem des Vergessens” sondern vielmehr das “Problem des Merkens”. Das technische System muß dagegen aktiv vergessen, um der wachsenden Datenmenge zu begegnen. Um trotzdem dem biologischen System möglichst nahe zu kommen, sollen zwei Prinzipien umgesetzt werden: Gleiche Reize überlagern sich häufig und ältere Reize werden vor neueren Reizen vergessen.

Deshalb gibt es in dieser Architektur zwei Stellen, an denen entschieden wird, was zu vergessen ist (und was nicht). Einerseits wird in regelmäßigen Abständen die Datenbasis von den ältesten Reizen angefangen durchkämmt und es werden Reize, die ein Mindestalter erreicht haben, als “zu vergessen” markiert. So ist es möglich, daß neuere Reize diese markierten Reize innerhalb der Modalitäten überlagern. Daneben ist es möglich, eine Speicherung zu vieler gleicher Reize in den Modalitäten zu vermeiden. Dazu werden diese Reize *aktiv* vergessen.

2.3 Vernetzung der Daten

Da die Daten sowohl eindrucksbezogen als auch modalitätsbezogen gespeichert werden sollen, der Speicherplatz aber begrenzt ist, müssen die Reize vernetzt gespeichert werden. Ein Eindruck setzt sich aus den Reizen zusammen, die ihn als Gesamtheit charakterisieren. Jeder einzelne Reiz wird zudem in bis zu vier Modalitäten verwaltet, um eine modalitätsspezifische Verarbeitung zu ermöglichen. Dabei werden die räumlichen Bezüge in der Ortsmodalität, der zeitliche Kontext in der Zeitmodalität und das Genauigkeitsmaß in der Genauigkeitsmodalität gespeichert. Eine der weiteren Modalitäten (Farbe, Umfang, etc.) repräsentiert den modalitätsspezifischen Wert des einzelnen Reizes. Da jeder Reiz einem Eindruck angehört, kann aus der Sicht aller beteiligter Modalitäten eine Ähnlichkeit von Eindrücken definiert werden bzw. zu einem Reiz eine Menge von Eindrücken angegeben werden, die ähnliche Reize bezüglich dieser Modalität enthalten.

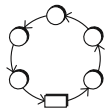
Es gibt verschiedene Möglichkeiten, die Speicherung der Reize, Eindrücke und Modalitäten zu realisieren. Die jeweilige Organisationsform hängt stark von der Zielsetzung der Vernetzung zwischen den Daten ab. Die notwendigen Zugriffe sollen effizient sein. Im folgenden werden Besondere Vor- und Nachteile von Listen, Feldern und wertabhängiger Speicherung mit Kohonenkarten sowie deren Anwendbarkeit erläutert.

Liste

In einer Liste werden Daten linear angeordnet werden, wobei eine Sortierung der Daten möglich ist, da neu dazukommende an beliebiger Stelle einsortiert werden können. Verzichtet man auf die Sortierung, so kann ein Datum sehr einfach in die Liste aufgenommen werden, indem es an den Anfang der Liste gestellt wird. Die Suche innerhalb der Daten ist hier aufwendiger, da alle Elemente berücksichtigt werden müssen, wenn das gesuchte nicht vorhanden ist. In sortierten Listen dagegen kann man Zeit sparen, da das Datum sich nur an einer – durch die Sortierung festgelegten – Stelle befinden kann. Ist die Verarbeitung unabhängig von einer Sortierung möglich und soll nur jedes Elementes garantiert bearbeitet werden, ist die Liste eine gute Lösung für dynamische Speicherung von Daten.

Die Listenstruktur ist Basis der meisten Verknüpfungen innerhalb des Gedächtnisses. Es werden doppeltverkettete Listen benutzt, um verteiltes Arbeiten mehrerer Prozesse an einer Liste zu vereinfachen und Rückwärtssuchen zu ermöglichen.

Ring

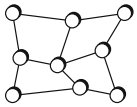


Der Ring ist eine Spezialform der Liste, bei der Ende und Anfang verbunden sind. Meist wird das erste Element bevorzugt behandelt, da dieses den “Aufhänger” des Rings bildet. Die Zeitmodalität nutzt diese Art der Speicherung, da alle Reize wie in einer Warteschlange sortiert vorliegen sollen. Neue Reize werden an der einen Seite kontinuierlich eingefügt; dadurch ergibt sich eine natürliche Sortierung, welche die Suche nach den ältesten Reizen erübrigt: diese sind einfach von der anderen Seite des Rings her erreichbar.

Feld

In einem Feld (*Array*) können Daten leicht abgelegt werden, wenn der Mehraufwand durch das Verwalten und Suchen in einer Liste gespart werden soll. Die Suche nach einem Element läßt sich durch Hashing vereinfachen und effizient gestalten (siehe [Sed92], Kapitel 16). Felder werden in meiner Architektur innerhalb der Modalitäten zur Speicherung der Verweise auf die beteiligten Reize verwendet. Wird ein Feld zu voll, muß es entweder reorganisiert oder ein weiteres Feld eröffnet werden. Die Felder werden dann in Listen gespeichert. Die Reorganisation kann entweder alte Reize freigeben oder das Feld in einer Karte umbauen.

Karte



Innerhalb der Modalitäten stellt sich das Problem, schnell möglichst viele Reize abhängig von ihrer Reizstärke zu speichern. Als Lösungsansatz bietet sich hier die Speicherung mit Kohonenkarten an. Kohonenkarten sind zum Beispiel in [Zel96] beschrieben und eignen sich in diesem Fall besonders, da sie sich selbständig an einen benutzten Wertebereich anpassen können und über die nötige Dynamik für die Speicherung innerhalb der Modalitäten verfügen. Weitere Bemerkungen und Erklärungen finden sich in Kapitel 8 von [Gör93] und in [RMS91].

Um die Vorteile der Kohonenkarte zu nutzen, aber auch die Reize später abrufen zu können, wird an die Knoten der Karte nicht nur der Gewichtsvektor, sondern zusätzlich ein Feld gebunden.

Dieses kann in der oben beschriebenen Weise die Daten aufnehmen und ist zur Abfrage und Suche gut geeignet. Alle Reize eines solchen angebundenen Feldes haben eine gewisse Ähnlichkeit, da die Suche in der Karte den gleichen Knoten ergab. Diese Zusammenstellung von Karten und Feldern wird Assoziativstruktur genannt, da ihre Funktionsweise die Assoziation beeinflußt. Die schnelle, wertabhängige Suche innerhalb der Karte durch Gradientenabstieg und die Suche in dem entsprechenden Feld mittels einer Hashfunktion ermöglichen gemeinsam einen effizienten, wertabhängigen Zugriff auf die gespeicherte Information.

Statt eines Feldes kann sich an jedem Knoten auch eine “Subkarte” befinden, um eine dort geforderte höhere Auflösung zu erreichen. Diese Subkarten müssen sich nicht notwendig überlappen, sondern sie passen sich dynamisch dem gegebenen Wertebereich an.

Durch Kombination dieser Organisationsformen erreicht die vorliegende Architektur insgesamt eine sehr dynamische Anpassung an den Wertebereich und an das geforderte Auflösungsvermögen unterschiedlicher Daten. Die Vernetzung zwischen Modalitäten und Reizen bzw. Eindrücken ist komplex und erlaubt vielfältige Manipulationen und Berechnungen, die in anderen Organisationsformen wesentlich aufwendiger wären.

2.4 Assoziation

Die Assoziation bildet das Herzstück der Gedächtnisleistung. Nach den biologischen Vorbildern wird die Assoziation zwischen zwei verschiedenen Reizen immer innerhalb einer Modalität erfolgen. Jede Modalität verfügt über eine charakteristische Vergleichs- und Abstandsfunktion, welche die Geometrie des Raumes dieser Modalität definiert. Durch die Abstandsfunktion können zwei Reizstärken innerhalb einer Modalität als ähnlich erkannt werden. Abschnitt 3.4.3 geht näher auf die verschiedenen Abstandsmaße ein.

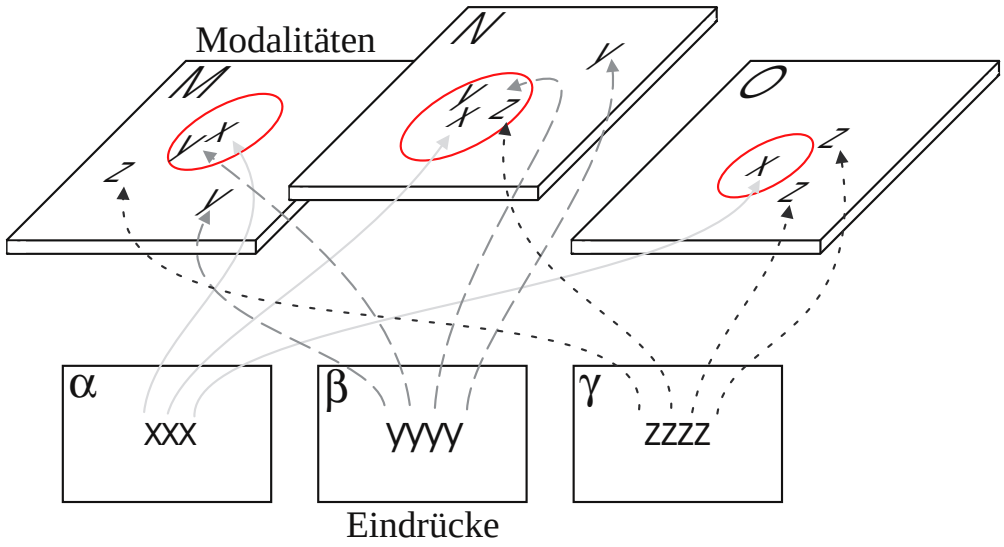


Abbildung 2.4: Assoziative Vernetzung von Modalitäten und Eindrücken

Abbildung 2.4 zeigt beispielhaft, wie die Reize **x** bis **z** der Eindrücke α bis γ in den Modalitäten M bis O eingeordnet sind. Die Kreise innerhalb der zweidimensionalen Modalitäten deuten den Abstandsradius um die Reize **x** an, in dem Reize als ähnlich empfunden werden.

Die Ähnlichkeit zwischen zwei Eindrücken setzt sich nun aus den Ähnlichkeiten der einzelnen Reize innerhalb der entsprechenden Modalitäten zusammen. Wir können einen Eindruck mit einem anderen assoziieren, wenn in genügend vielen Modalitäten die Reize des einen Eindrucks an die Reize des anderen Eindrucks *erinnern*, also ihnen ähnlich sind. In dem obigen Beispiel könnte man sagen, daß Eindruck α mit Eindruck β assoziiert wird, da zwei der drei Modalitäten eine Ähnlichkeit auf Reizebene aufweisen.

In der oben beschriebenen Organisation der Modalitäten bedeutet dies, daß in einer als Karte organisierten Modalität die Assoziation nur einen Vergleich zwischen Reizen eines Knotens, der als Feld organisiert ist, erfordert. Die Zugriffe und Vergleiche sind somit effizient implementierbar.

2.5 Kommunikation mit dem DACS

Im SFB wird die Kommunikation und Systemintegration durch das DACS (Distributed Application's Communication System) von Nils Jungclaus und Gernot A. Fink (siehe [FJRS95] und [FJK+96]) realisiert. Die Schnittstelle ist für die Programmiersprache C in Form einer Benutzerbibliothek gegeben. Der einzelne Programmierer einer Applikation muß sich dadurch nicht mit der Netzprogrammierung auseinandersetzen. Abbildung 2.5 zeigt, wie die verschiedenen Applikationen auf verschiedenen Rechnern verteilt sind. Die Kommunikation wird von einem DACS-Dämonen lokal auf jedem der beteiligten Rechner organisiert. Die Dämonen *verschicken* die einzelnen Datenpakete und erhalten ihrerseits Informationen über den Empfänger von dem zentralen "Nameserver", in dem alle Informationen über Applikationen abgelegt sind.

Jedes Programm meldet sich mit einem Funktionsaufruf beim laufenden DACS an und hat somit einen festen Namen und eine feste Adresse im Rechnernetz. Die Kommunikation im DACS ist mit Datenströmen (sogenannte *Demand Streams*) oder funktionsorientiert durch *Remote Procedure Calls* (RPCs) möglich. Das Programm kann nun Demand Streams bestellen oder bereitstellen. RPCs können nach der erfolgreichen Anmeldung beim DACS von einem Programm angeboten oder benutzt werden. Die Demand Streams sind in der Abbildung mit durchgezogenen Pfeilen dargestellt, um den Informationsfluß "durch" die Applikationen zu symbolisieren.

Die Demand Streams werden von einem Programm angeboten, das als Datenquelle dient. Eine beliebige andere Applikation kann die angebotenen Daten *bestellen* und mit DACS-Funktionen in normale C-Objekte wandeln. Das Datenformat der SFB-Objekte ist in Anhang B abgedruckt. Jeder Stream enthält nur SFB-Objekte eines Typs. Bestimmte Synchronisierungsmarken erlauben eine Gruppierung und Taktung des Datenstromes.

Die Kommunikation mit RPCs wird bisher wenig im SFB benutzt, erlaubt aber eine geschickte Verteilung von Aufgaben. Die Applikationen im SFB können zum Teil Dienste anbieten, die von anderen Applikationen benutzt werden. Das benutzende Programm kann jedem Aufruf das berechnete Ergebnis zuordnen. Bei der Verarbeitung der Daten durch Streams entfällt diese Zuordnung, da ein Programm, das die Daten bereitstellt, in der Regel nicht an dem Ergebnis interessiert ist. In der Abbildung ist durch die unterbrochenen Pfeile angedeutet, wie die Wissensbasierte Verifikation bei dem Gedächtnis für Regionen eine Anfrage über RPC stellt.

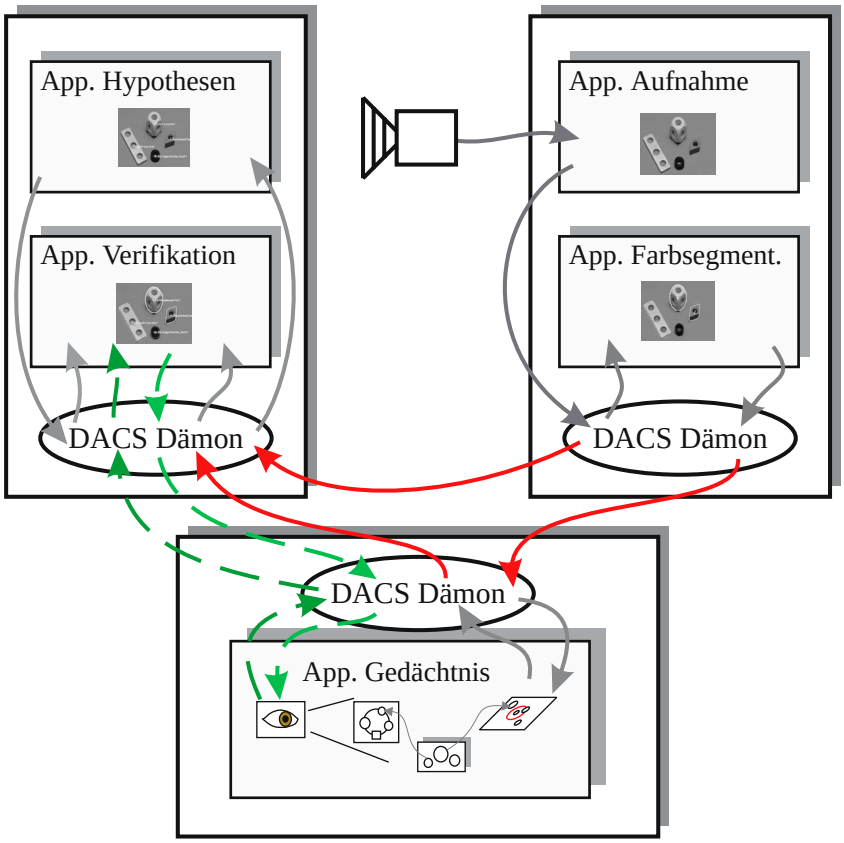


Abbildung 2.5: Beispiel für Kommunikation mit dem DACS

In meiner Architektur ist eine Anbindung mit Streams und RPCs vorgesehen. Die “normale” Benutzung erfolgt über einen Eingabe-Stream und einige Ausgabe-Streams. Diese sollen Applikationen die Auswertung der Daten erleichtern. Das Gedächtnis stellt jeweils einen Demand Stream für die von ihm erkannten und nicht erkannten Daten bereit. Die Daten, welche mit den neuen Daten assoziiert werden konnten und jene, bei denen im Zuge der Assoziation eine Veränderung festgestellt werden konnte, werden in zwei weiteren Demand Streams zur Verfügung gestellt. Daneben kann das Gedächtnis von außen mit RPCs “gefragt” werden, und das aufrufende Programm erhält die Antwort über das DACS zurück. Die Antwort beinhaltet das zur Anfrage assoziierte Objekt.

2.6 Visualisierung

Die Visualisierung des Gedächtnisses ist sinnvoll, um die Funktionsweise kontrollieren und steuern zu können. Es müssen eine Reihe von Parametern von Hand optimiert werden, die an den verschiedensten Punkten der Architektur eingreifen; dazu erweist sich die optische Kontrolle als unerläßlich. Eine Zusammenstellung der Parameter liegt in Abschnitt 4.2 vor. Durch die interaktive Bedienung ist das Programm leicht zu konfigurieren und an spezielle Anforderungen anzupassen.

Kartenvisualisierung

Eine spezielle Möglichkeit, in das Gedächtnis hineinzuschauen, bietet die Visualisierung der Kohonenkarten. Aus der Darstellung lassen sich Schlußfolgerungen über den wahrgenommenen Wertebereich und das erreichte Auflösungsvermögen des Gedächtnisses in jeder Modalität ziehen.

Neben den üblichen Darstellungen der Gewichtsvektoren, die den Wertebereich erschließen, bietet sich auch die Visualisierung der gespeicherten Informationsmenge an. An jedem Knoten können verschieden viele Reize gespeichert sein, so daß eine proportionale Darstellung der Inhaltsmenge jedes Knotens weitere Schlüsse auf die Dynamik der Karte erlaubt.

Streamvisualisierung

Eine andere Visualisierung kann mit dem Viewer des SFBs (siehe voraussichtlich [Jun97]) realisiert werden. Der Viewer stellt beliebige Demand Streams auf dem Bildschirm dar. Der Output kann während der Verarbeitung beobachtet werden und das Ergebnis über die Parametersteuerung weiter beeinflußt werden.

2.7 Parallelität von Aufgaben

In dieser Architektur ergeben sich einige Aufgaben, die parallel ausgeführt werden sollten und einige, die parallel ausgeführt werden können. Eine Parallelisierung ist wünschenswert, wenn Mehrprozessorsysteme zur Verfügung stehen. Darüberhinaus lassen sich einige Aufgaben eines solchen Projektes mit den Techniken der parallelen Programmierung einfacher umsetzen als dies mit “event-orientierten” Ansätzen möglich wäre.

In jedem Fall ist darauf zu achten, daß die Parallelität der Aufgaben nicht den Programmfluß beeinträchtigt und keine sogenannten *Deadlocks*, also Punkte, an denen kein Programmteil weiterkommen kann, auftreten können. Die parallele Programmierung mit Threads ist unter anderem in [LB96] beschrieben.

Pflicht-Parallelität

Parallel sollten folgende Verarbeitungsschritte laufen: Speicherverwaltung, Datenaufnahme und Verarbeitung, aktives Vergessen und Visualisierung. Diese Verarbeitungseinheiten sind funktional unabhängig und dementsprechend parallel entworfen und implementiert. Die bearbeiteten Daten überschneiden sich zum großen Teil; daher kann eine Verarbeitungseinheit in der Ausführung auf Mehrprozessorsystemen kurzzeitig die anderen hemmen. Dennoch ermöglicht das read ever/write once-Prinzip (Erläuterungen in Abschnitt 3.1 über Threads) den Verarbeitungsfluß in allen Programmteilen, da nach der Aufnahme der Daten nur noch selten Änderungen an den Daten vorkommen: Der Aufbau und das Vergessen von Daten sind meist die einzigen Schreibzugriffe auf den Datensätzen.

Wahl-Parallelität

Die Datenaufnahme kann intern in verschiedener Weise parallel ablaufen. Es ist möglich, jeweils ein neu aufgenommenes SFB-Objekt auf dem Weg durch das Gedächtnis von einem “Thread” (einem eigenständigen Programmteil) verarbeiten zu lassen. Eine andere Möglichkeit besteht darin, die einzelnen oben aufgeführten Verarbeitungsschritte in jeweils einem Thread laufen zu lassen und zwischen ihnen Warteschlangen einzurichten, um unterschiedliche Laufzeiten auszugleichen. Bei der zweiten Möglichkeit kann es trotzdem zu “Stauungen” kommen, die weitere Probleme im Programmlauf verursachen.

Ich entscheide mich für die erste Variante und biete dem Benutzer an, mit Hilfe eines Parameters die Parallelität durch die Anzahl der gleichzeitig möglichen objektgebundenen Threads einzustellen. Zusätzlich wird für jeden RPC ein Thread mit der Berechnung der Anfrage beauftragt.

Die Berechnung der abstrakten Werte und die Aktivität der Fokusfunktion kann zusätzlich parallel ausgeführt werden, da hier keine Stauungen entstehen können.

Kapitel 3

Implementierung

Es gibt immer mehrere Wege zum Ziel — dem Programm.

3.1 Einführung

Ich habe zur Implementierung die Sprache C gewählt, was mehrere Gründe hat: relativ leichte Portabilität, gute Unterstützung durch die Tools des SFBs, vorhandene Thread-Implementierung und die Tcl/Tk-Schnittstelle, mit der eine komfortable Benutzeroberfläche eingerichtet werden kann. Meiner Meinung nach ist auch in C eine "saubere" Implementierung einer solch komplexen Architektur möglich und übersichtlich zu realisieren. Hinweise für übersichtliches Projektmanagement findet man in [Str93]. Der modulare Entwurf ist mit den Ideen der objektorientierten Programmierung umgesetzt. Das Hauptaugenmerk liegt bei dieser Implementierung auf der Erweiterbarkeit, der Sicherheit und der Parallelität des Programms.

Ich möchte im folgenden nicht jede Zeile des Sourcecodes durchgehen und erläutern, sondern vielmehr auf einige Grundkonzepte hinweisen, die diese spezielle Implementierung grundlegend von anderen möglichen Realisierungen der gleichen Architektur unterscheiden.

Namenskonvention

Eine sinnvolle Namenskonvention hilft, sich im Source zurechtzufinden. Die Gliederung in einige verschiedene Module und Sourcefiles ist bei der Größe des Programms unerläßlich. Es gibt nur wenige Regeln, die die Namenskonvention betreffen:

- Alle eigenen Files, Typen, Makros und Funktionen beginnen mit einem Präfix, das hoffentlich noch nirgends verwendet wird. Ich nutze hier das Akronym "VDMF", das man als **V**isual **D**ynamic **M**emory **F**unctions interpretieren kann. In Makros ist es groß und sonst klein zu schreiben.
- Bei Funktionsnamen besteht der zweite Teil aus dem Modulnamen; dieser entspricht in aller Regel auch dem File, in dem sie definiert sind.
- Privat genutzte Funktionen, die zum Beispiel indirekt aufgerufen werden oder nur an speziellen Stellen aufgerufen werden dürfen, beginnen zusätzlich mit einem "_" vor dem Namen.

Source Dokumentation

Der Sourcecode ist in vielen Teilen ausführlich dokumentiert, und zu den meisten Funktionen wird eine Beschreibung geliefert. Die Gesamtkonzeption läßt sich aus dem Quelltext nur sehr mühsam erschließen. Dieser Teil der Diplomarbeit ermöglicht einen Überblick. Wer Interesse an der einen oder anderen Realisierung hat, kann gezielt danach suchen, indem er die HTML-Form des Quelltextes heranzieht (siehe Anhang C).

Threads

Die Threadprogrammierung ermöglicht die hohe Parallelität des vorliegenden Programms und erfordert einigen Mehraufwand, der sich bei Mehrprozessorsystemen aber schnell rentiert. Die Implementierung mit Threads läßt sich am besten mit dem Zusammenarbeiten von mehreren Prozessen auf einem UNIX-System vergleichen. Die Threads sind einzelne Programmteile, die unabhängig voneinander ausgeführt werden können. Sie verfügen jedoch über einen gemeinsamen Daten- und Adreßraum, so daß die Zusammenarbeit an einem Problem einfach zu realisieren ist.

Die Zugriffe auf den gemeinsamen Speicher können durch einen sog. Mutex gegenseitig geschützt werden. Ein Thread darf nur ein Datum benutzten, wenn er im “Besitz” des Mutex ist. Dies ist erforderlich, wenn der Zugriff nicht auf einen “atomaren” Prozessorzugriff begrenzt ist, und eine ungünstige Schedulingreihenfolge der Threads zu ungültigen Daten führen würde. Die Techniken der Threadprogrammierung definieren *read/write*-Locks (siehe [Jun96]) oder auch read multiple/write once-Locks, um effektiv auf den gemeinsamen Datenbestand zugreifen zu können. Diese Lock-Mechanismen erlauben keinen Lesezugriff, solange ein Thread auf den Daten schreibt, also im Besitz des write-Locks ist. Da in der beschriebenen Architektur die Daten vorwiegend an einer Stelle neu erstellt werden und dann fast nur noch zu Lesezwecken verwendet werden, bietet es sich an, das Lesen von Daten zu jeder Zeit zu ermöglichen. Deshalb habe ich eine eigene Zugriffsart definiert.

Diese Zugriffsart habe ich *read ever/write once* genannt. Gerade bei Listenoperationen ist es wichtig, daß viele verschiedene Threads auf einer Liste arbeiten können, ohne sich gegenseitig zu blockieren. Es soll nur verhindert werden, daß zwei Threads gleichzeitig auf einem Datensatz schreiben, also zum Beispiel jeweils ein Element aus einer Liste entfernen wollen. Die einzige Schwierigkeit bei dieser Überlegung ist, daß beim Schreiben von Daten immer gewährleistet sein muß, daß andere diesen Datensatz lesende Threads immer gültige, wenn auch dann etwas veraltete Daten lesen können. Die Schreiboperationen werden zusätzlich komplizierter, dafür sind die Leseoperationen immer möglich.

Ein weiteres Konstrukt der Threadprogrammierung sind sog. *Conditions*. Bei Eintreten eines bestimmten Zustandes kann ein Thread einen anderen darauf aufmerksam machen, wenn dieser darauf “gewartet” hat. Mit dieser Signalverarbeitung ist eine Synchronisation der Threads möglich. Es gibt noch eine Reihe weiterer Konstrukte der parallelen Programmierung, die in [Jun96] oder [LB96] behandelt werden. Meine Implementierung nutzt nur die Konstrukte Mutex und Condition.

Objekte in C

Auch in der Sprache C ist es möglich, Module zu entwickeln und in Anlehnung an C++ objektorientiert zu programmieren. Ich danke hier nochmals meinem Betreuer Nils Jungclaus für die Anregungen und Tips bei dem Entwurf der Module.

Der hauptsächliche Unterschied zu C++ liegt in der nicht automatisierten Vererbung und der fehlenden Kontrolle über private und geschützte Funktionen und Variablen eines Typs. Hier ist die Disziplin des Programmierers auf eine harte Probe gestellt, denn bei Unsauberkeiten können Fehler entstehen, die im Nachhinein schwer zu finden sind. Die Vorteile von C überwiegen klar, bedenkt man die Schwierigkeiten der Portierung von C++ oder anderen Sprachen auf mehrere Architekturen, so daß der erhöhte Aufwand ausgeglichen wird.

3.2 Konzepte

Ich werde im Kapitel 3.3 einige Datenstrukturen kurz vorstellen, die ich zur Realisierung der Architektur entworfen habe. Zwei grundlegende Konzepte gelten für fast alle diese Datenstrukturen; deshalb stelle ich diese Konzepte den Strukturen voran.

3.2.1 Common-Konzept

Das Common-Konzept soll die Erweiterbarkeit und die Sicherheit der Pointerarithmetik vereinen. Es definiert die Eigenschaften, die für alle Objekte des Programms gelten sollen:

- Das Objekt erhält eine Erkennungsmarke (*self*), die es eindeutig als Objekt des Programms ausweist. Diese Marke ist ein Pointer, der in der Regel auf das Objekt selbst (self) verweist, der aber auch andere Werte annehmen und damit anzeigen kann, in welchem Zustand dieses Objekt sich befindet (siehe Speicherverwaltung mit dem GBC in Abschnitt 3.2.2) .
- Jedes Objekt bekommt einen Typ (*TypId*) zugeordnet, an dem eine Funktion erkennt, ob sie diesen Objekttyp verarbeiten kann. Die TypId wird bei der Initialisierung gesetzt und darf nicht verändert werden.
- Die Listenfunktionalität ist so zentral, daß jedes Objekt eine entsprechende Struktur beinhaltet, mit der es möglich ist, das Objekt in einer Liste zu verwalten. Gespeichert werden in der Listenstruktur die beiden Pointer für eine doppeltverkettete Liste.
- Das *read ever/write once*-Konzept erfordert einen Referenz-Zähler (*refs*), mit welchem festgehalten werden kann, wieviele verschiedene Verweise auf ein Objekt durch andere Funktionen oder Objekte existieren. Darüber hinaus ist ein write-Lock erforderlich, um das *write once*-Prinzip zu implementieren.
- Um von jeder Stelle des Programms aus ein Objekt freigeben oder als Klartext ausgeben zu können, gibt es zwei Funktionen `freeobj()` und `sprint()` für jedes Objekt. Die Ausgabe als Text hilft beim Debugging und kann für eine interaktive Oberfläche in Tcl/Tk genutzt werden.

Die Zusammenstellung dieser Daten in einer Common-Struktur ermöglicht es, sehr komplexe Abhängigkeiten für verschiedene Objekte in einer Multithread-Umgebung aufbauen zu können. Es werden im Programm keine Objekte dieses Typ dynamisch alloziert, da ein solches Objekt noch keine speziellen Daten enthält. Dennoch werden an vielen Punkten Zeiger dieses Typs an Funktionen übergeben, wenn es für die Funktion unerheblich ist, welchen Objekttyp sie genau bearbeitet. Man kann dieses Konzept mit den abstrakten Klassen von C++ vergleichen. Es ermöglicht, bestimmte Verarbeitungsschritte allgemein für Objekte zu definieren, ohne speziell mit den eigentlichen Daten des Objektes umgehen zu müssen. Die Listenverwaltung ist mit diesem Grundsatz realisiert.

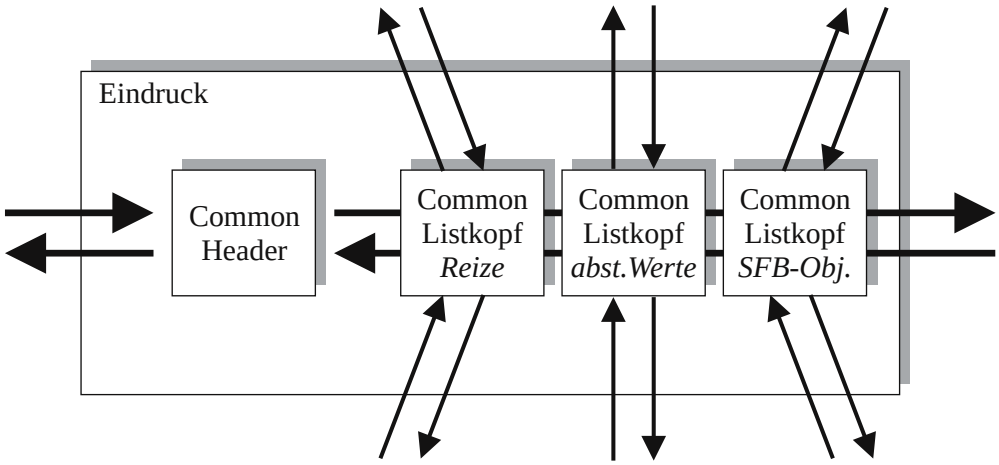


Abbildung 3.1: Eindruck mit Common-Header und eingelagerten Strukturen

Abbildung 3.1 zeigt eine typische Verwendung der Common-Struktur. In dem gezeigten “Eindruck” werden neben dem eigenen Kopf der Struktur auch die Listenköpfe für Reize, abstrakte Werte und SFB-Listenelemente gespeichert.

Die Common-Struktur bildet den Kopf, den Anfang jedes Objektes, welches von den allgemeinen Funktionen zu Listenverwaltung verwendet werden soll. An vielen Stellen werden Zeiger zunächst überprüft, ob sie den richtigen Objekttyp ansprechen, bevor sie benutzt werden. Dieser Overhead rentiert sich durch eine hohe Stabilität des Programms und hilft, verdeckte Pointerfehler schnell zu finden. In der fertigen Implementation können diese Sicherheitsabfragen entfernt werden, um eine höhere Performance zu erzielen.

Teilweise werden innerhalb von Objektstrukturen neue Listen “verankert”. In Abbildung 3.1 sind drei Listen im Eindrucks-Objekt zusammengestellt. Die Verankerung geschieht wieder mit Hilfe der Common-Struktur, da diese die Listenverwaltung beherbergt. Diese eingelagerten Strukturen bekommen gesonderte TypIds, um in der Verarbeitung Rücksicht auf den Header dieser Liste nehmen zu können. Bei der Freigabe von dynamisch angefordertem Speicher spielen diese Köpfe von Listen eine weitere wichtige Rolle (siehe dazu nächsten Abschnitt über das GBC-Konzept).

Da jede Struktur im Common-Kopf die Listenverwaltung implementiert hat, kann jedes Objekt in Listen geführt werden. Alle Listen sind doppeltverkettet, um einen leichten lokalen Zugriff auf die Struktur der Elemente zu ermöglichen. Damit wird es möglich, für alle Objekte folgende Listenfunktionen zu definieren, ohne die Forderung nach read ever/write once zu verletzen:

- Entfernen eines Elementes aus seiner Liste an beliebiger, aktueller Position
- Zusammenfügen von zwei Listen durch Aneinanderhängen
- Einfügen von neuen Elementen an beliebiger Position
- Löschen der ganzen Liste

3.2.2 GBC-Konzept

Eine sichere Speicherverwaltung wird in dem Moment notwendig, in dem verschiedene Threads berechtigt sind, Daten aus einer gemeinsam genutzten Datenbasis zu löschen. Durch die Common-Struktur verfügt jedes Objekt über einen Referenzzähler (*refs*), so daß an der Stelle, an der eigentlich ein Objekt freigegeben werden sollte, festgestellt werden kann, ob noch andere Funktionen auf dieses Objekt zugreifen. Die Lösung dieses Konflikts ist ein verzögertes Freigeben mit Hilfe eines Garbagecollectors (GBC). Die Objekte werden demnach nicht sofort freigegeben, sondern erst, wenn alle Funktionen, die noch damit gearbeitet haben, ihren Referenzzeiger "zurückgegeben" haben.

Diese Strategie erlaubt jeder Funktion ein Übertragen eines Objektes in einen freigabefähigen Zustand. Der eigenständige GBC untersucht nun von Zeit zu Zeit die Menge der "zur Freigabe" markierten Objekte und gibt diejenigen zum Löschen frei, die nicht mehr von anderen Funktionen benutzt werden. Der Prozeß des Löschens selbst ist zusätzlich mehrstufig organisiert, dadurch daß die Entscheidung vom GBC mehrmals getroffen werden muß, bevor das Objekt wirklich gelöscht wird. Dies ist notwendig, um das read ever-Prinzip zu realisieren und bei Löschvorgängen keinen Speicherfehler (Segmentation Fault) zu riskieren.

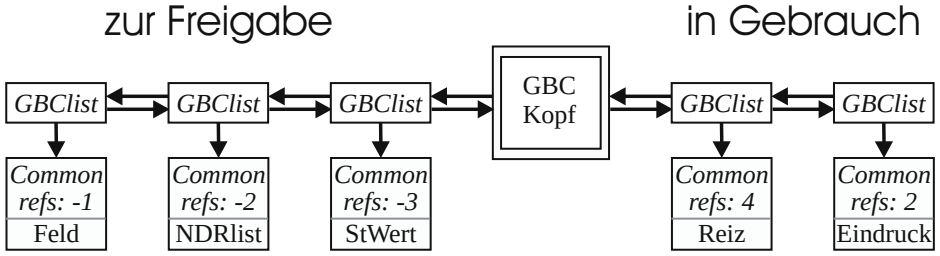


Abbildung 3.2: GBC-Liste mit "zu löschenden" Elementen

Ein weiteres Problem ergibt sich bei eingelagerten Strukturen. Diese sollen genau wie normale Strukturen löschbar sein, dürfen aber nicht mit dem Systemaufruf `free()` tatsächlich freigegeben werden. Der Speicherbereich, in dem die eingelagerten Strukturen stehen, darf erst dann freigegeben werden, wenn alle Substrukturen bereits "bereinigt" sind. Dieser Fall tritt besonders bei Listenköpfen, Feldern und Karten auf. Es wird deshalb für jede eingelagerte Common-Struktur eine weitere Referenz beim Übergeben in den GBC angefordert, um sicherzustellen, daß das Mutterobjekt erst nach allen Substrukturen freigegeben werden kann. Die bei den Common-Konzept erwähnte Erkennungsmarke (*self*) wird in diesem Fall als Zeiger auf das Mutterobjekt verwendet. Im Normalfall zeigt dieser Pointer auf das Objekt selbst. Bei der Verarbeitung durch den GBC werden für normale Objekte andere Pointerwerte gespeichert, um den Zustand des Objektes auch "von außen" erkennen zu können.

Ich zeichne kurz den “Lebensweg” eines Beispielobjektes nach, um den genauen Ablauf dieser Speicherverwaltung zu verdeutlichen:

- **Initialisierung:** Das Beispielobjekt `obj` vom Typ `vdmf_bsp_t` wird mit seiner Funktion `vdfm_bsp_init()` initialisiert und die evtl. eingelagerten Common-Strukturen werden rekursiv initialisiert. Die Daten werden meist schon hier in das Objekt eingetragen. Ebenso wird bereits die passende Funktion zur Freigabe dieses Objektes als `obj` $\rightarrow$`freeobj()` verankert. Dies kann entweder die allgemeine Funktion `vdmf_common_for_free()` oder eine objektspezifische Funktion `vdmf_bsp_for_free()` sein, welche die weiter unten beschriebenen Arbeitsschritte vollzieht.

- **Gebrauch:** Das Objekt ist nun für den Gebrauch geeignet. Sein Typ ist immer feststellbar; seine Daten können evtl. noch geändert werden. Es ist in Listen, Feldern oder Karten verwaltbar und kann von jeder Funktion an den GBC übergeben werden.

- **Übergabe an den GBC:** Jede Funktion, die eine Referenz auf ein Objekt hat, kann die Übergabe des Objektes an den GBC veranlassen. Die Objektfunktion `obj` $\rightarrow$`freeobj()` stellt durch einen Lock-Mechanismus sicher, daß ein Objekt nur einmal in den GBC übertragen wird. Diese im Objekt verankerte Funktion veranlaßt für eingelagerte Common-Strukturen ebenfalls eine Freigabe und Übergabe an den GBC. Anschließend wird das Objekt als gelöscht markiert; es kann aber noch immer von anderen Funktionen verwendet werden, sollte jedoch nicht mehr in Listen aufgenommen oder komplexeren Verarbeitungsschritten unterzogen werden. Funktionen, die ein “als gelöscht” markiertes Objekt finden, müssen ihren Referenzzeiger sobald wie möglich zurückgeben.

- **Verwaltung im GBC:** Das Objekt wird im GBC in einer speziellen Liste, auf die von außen nicht zugegriffen werden kann, verwaltet. Der GBC-Thread wartet darauf, daß alle Referenzzeiger auf dieses Objekt zurückgegeben werden. Durch ungünstige Verkettungen kann es vorkommen, daß nur Objekte, die sich selbst schon im GBC befinden, noch Referenzen auf unser Beispielobjekt besitzen. Es sollte immer einen Anfang dieser Kette von Abhängigkeiten zwischen von Objekten im GBC geben, so daß sichergestellt ist, daß sie nach und nach freigegeben werden können.

- **Mehrstufige Verwaltung im GBC:** Wird dem Objekt im GBC die letzte Referenz genommen, so wird der GBC-Thread durch eine Condition[1] darauf aufmerksam gemacht. Nach einer einstellbaren Anzahl von Conditions beginnt der GBC seine Liste mit freizugebenden Objekten zu durchsuchen. Findet er ein Objekt, daß keine Referenz mehr hat, wird dieses Objekt in einen anderen Teil (siehe linken Teil der Abbildung 3.3) der Liste verschoben und als “gesammelt”[2] markiert. Dieser Teil der Liste wird vor jeder neuen Sammlung durchsucht und die Objekte, die eine einstellbare Anzahl dieser Sammlungen überstanden haben, endgültig zur Freigabe bereitgestellt. Die Freigabe selbst erfolgt erst bei der Überlagerung des Objektes von einem neuen Objekt, das gerade in den GBC aufgenommen wird.

- **Freigabe der Objekte:** Die Freigabe geschieht im Rahmen der neuen Eintragung eines Objektes in den GBC mit `obj` $\rightarrow$`freeobj()`. Im GBC werden die internen Listenelemente,

[1] Dies ist ein Konstrukt der Threadprogrammierung, mit dem ein Thread einen anderen auf das Eintreten eines Zustandes aufmerksam machen kann.

[2] Der Referenzzähler *refs* ist ab jetzt negativ.

die auf zu löschende Objekte verwiesen werden, wiederverwendet; alte noch von diesen Elementen bezeichnete Objekte werden nun mit `vdmf_gbc_common_free()` freigegeben. Diese Funktion beachtet außerdem die Nichtfreigabe für eingelagerte Common-Strukturen, zum Beispiel Listenköpfe. Diese Kaskade von Verknüpfungen garantiert eine Arbeitsverteilung, bei der es nicht zu Stauungen kommen kann. Die Systemaufrufe zur dynamischen Speicherverwaltung sind aufwendig und werden gleichmäßig auf alle Threads, die Objekte freigeben wollen, verteilt.

- **Verweise auf Nicht-Common-Strukturen:** Es gibt Objekte, in denen Strings oder ähnliche Nicht-Common-Objekte verwaltet werden. Die Strukturen geben ihren zugeordneten Speicher erst kurz vor der eigenen Freigabe in der Funktion `vdmf_gbc_common_free()` an das System zurück.

  Alle dynamischen Speicheranforderungen und -freigaben werden durch eigene Funktionen gekapselt, um das Debugging von Speicherlecks zu vereinfachen.

3.3 Strukturen

Ich möchte in diesem Abschnitt auf diejenigen Strukturen eingehen, die im folgenden noch tragende Rollen haben und nicht nebenbei umfassend erklärt werden können. Ziel ist, die Übersicht über die komplette Pointerstruktur des Gedächtnisses zu bekommen, ohne den Faden zu verlieren oder die Sicht für das Ganze einzubüßen.

Datenorganisation

Das Hauptobjekt des Gedächtnisses ist `vdmf_memory_t`. Es beinhaltet die Listenköpfe für Eindrücke, Modalitäten und den GBC. Daneben gibt es einen Parametersatz vom Typ `param_t`, der die von Tcl/Tk steuerbaren Parameter beinhaltet und bei jedem Neustart aus einem Konfigurationsfile eingelesen werden kann oder mit Standardwerten belegt wird. Hier wird für Objekte, die noch keinem Memory-Objekt eindeutig zugeordnet wurden, ein Defaultwert für ein Memory festgelegt. Ebenso ist hier eine Struktur vom Typ `port_t` verankert, in der die Informationen über die Schnittstelle zum DACS gespeichert werden.

Objekte vom Typ `eindruck_t` verwalten zwei Listenköpfe für *Reize* und *abstrakte Werte*, sowie den Kopf der Liste der SFB-Objekte. Abstrakte Werte und Reize sind beide vom Typ `reiz_t` und unterscheiden sich nur in der Entstehung und Freigabe.

Ein Objekt vom Typ `reiz_t` speichert die eigentlichen Daten. Es enthält darüber hinaus Verweise auf alle Modalitäten, in denen dieser Reiz gespeichert wurde.

Modalitäten sind Objekte vom Typ `modal_t` und verwalten die Verweise auf die Reize einer bestimmten Modalität. Die Struktur, mit der die Verweise angeordnet werden, hängt von der Art der Reize ab. Es gibt für jede Modalität eine festgelegte Organisationsform, die nach der Initialisierung nicht mehr geändert werden sollte. Die Zeitmodalität wird in einer Ringstruktur (siehe Abbildung 3.3), die übrigen Modalitäten werden in einer Assoziativstruktur organisiert.

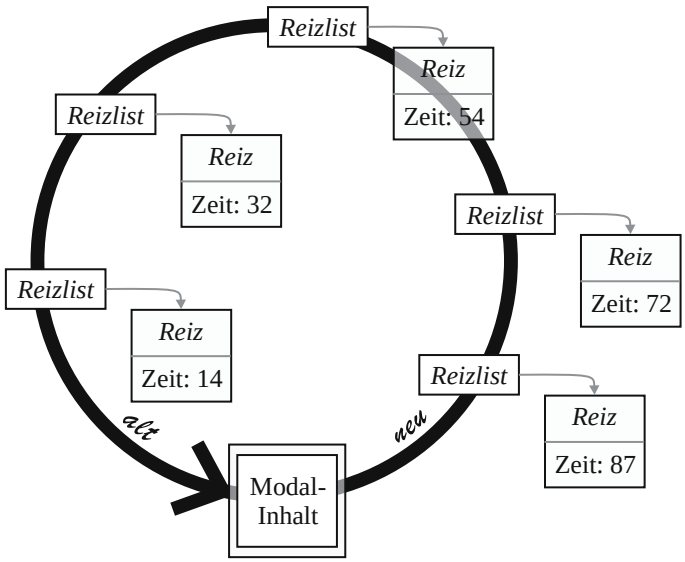


Abbildung 3.3: Ringförmige Anordnung der Zeitmodalität

Eine Assoziativstruktur ist vom Typ `assoz_t` und kann intern mehrere Erscheinungsformen haben. Sie kann leer sein, einen Reiz speichern, ein Feld speichern oder eine ganze Karte von weiteren Assoziativstrukturen verwalten.

Objekte vom Typ `feld_t` sind Hasharrays, in denen Verweise auf Reize gespeichert werden. Felder haben eine konstante Größe, lassen sich aber bei Bedarf zusätzlich in Listen anordnen, wobei die Suche nach einem Element dann für jedes Feld der ganzen Liste unternommen werden muß. Innerhalb eines Feldes wird der Speicherindex durch ein “hashen” von Feld- und Reizadresse berechnet. Die Kollisionsauflösung erfolgt durch das Speichern an der nächsten freien Speicherstelle.

Neben diesen komplexen Objekten gibt es eine Reihe von einfachen Listen, welche auf Objekte verweisen können. Die GBC-Liste ist ein Spezialfall, da sie nur innerhalb des GBCs verwendet werden darf. Außerdem gibt es die SFB-Objekt-Liste und Stringlisten. Auch das folgende Statistikobjekt benutzt eine spezielle Form der Werteverwaltung.

Für die Assoziation werden Statistiken zur Berechnung von Maxima und Anzahlen verschiedener Werte gebraucht. Der Typ `statistik_t` implementiert Objekte, die über Funktionen zur Aufnahme von neuen Werten und zur statistischen Auswertung verfügen. Die interne Liste der Werte (Typ `stwert_t`) liegt sortiert nach Werten vor, um mehrmals eingetragene Werte effizient finden zu können.

Pointerstruktur

Um die Funktionsweise des Gedächtnisses zu verstehen, ist es nicht notwendig, die Pointerstruktur genau nachzuvollziehen. Ein Grundverständnis ist jedoch unerläßlich. Die Speicherung der Eindrücke erfolgt in Listen des Memory-Objektes, welches auch die Modalitäten verwaltet, die zur Kategorisierung der Reize notwendig sind. Die einzelnen Reize sind in einer Liste des Eindrucks gespeichert, den sie als Reizmuster beschreiben. Die Reize haben, um eine Assoziation innerhalb

der Modalitäten zu ermöglichen, einen Zeiger auf den Eindruck, zu dem sie gehören. Die Repräsentation in Ort, Zeit und Genauigkeit ermöglicht neben der reizspezifischen Repräsentation eine Verwendung des Reizes in allgemeinen Zusammenhängen, die nicht von der Spezifität des Reizes abhängen. Die Reize werden demnach in dem Kontext von vier Modalitäten gespeichert, behalten aber den Zusammenhang in dem Eindruck, den sie repräsentieren. Die Verweise innerhalb der Modalitäten sind unterschiedlich realisiert und lassen verschiedene Zugriffe zu. Die Assoziation kann von der Ähnlichkeit auf Reizebene auf die Ähnlichkeit auf Eindrucksebene schließen. Dies gilt für die reizspezifische wie für die allgemeinen Modalitäten.

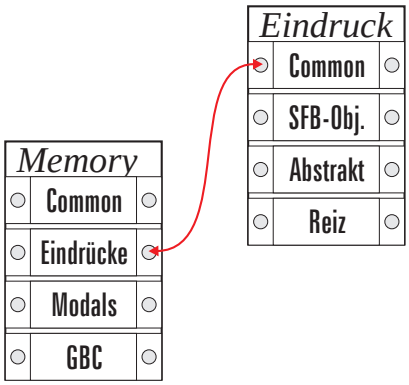


Abbildung 3.4: Pointerstruktur von Memory und Eindruck

Um tiefer in die Verwaltung und Verweise der Datenstrukturen einzusteigen, sei dem interessierten Leser im folgenden ein kleiner Leitfaden an die Hand gegeben. Ich versuche die Struktur zu vereinfachen, indem ich von nur einem Memory-Objekt mit einem Eindruck ausgehe. Im normalen Betrieb werden im Gedächtnis mindestens 100 Eindrücke und über 5000 Reize gleichzeitig benötigt, um überhaupt sinnvolle Assoziationen beobachten und einen Nutzen aus dieser Struktur ziehen zu können.

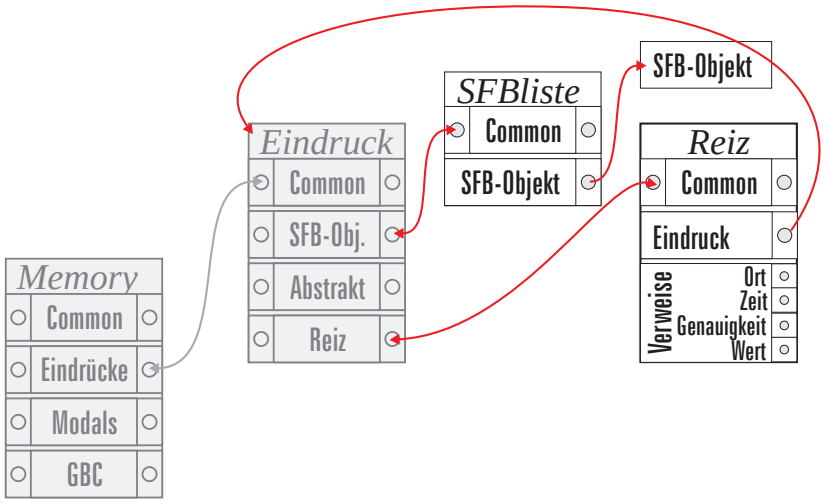


Abbildung 3.5: Pointerstruktur vom SFB-Objekt und Reiz

Das Hauptobjekt des Gedächtnisses ist das Memory-Objekt (siehe Abbildung 3.4). Es kann mehrere solcher Objekte geben, aber es wird zur Zeit nur ein Objekt dieses Typs initialisiert. Dieses Objekt ist gleichzeitig das Default-Memory-Objekt, falls ein Objekt nicht einem bestimmten Memory zugeordnet werden kann. Die Zuordnung ist zwar nicht zwingend notwendig, ermöglicht aber für spätere Erweiterungen eine saubere Trennung zwischen Objekten mehrerer Gedächtnisse. Das Memory-Objekt ist der Ausgangspunkt für alle weiteren Objekte, die nicht einen globalen Charakter wie zum Beispiel der Parametersatz oder der Port zu DACS haben. Im Memory-Objekt ist der GBC verankert. Daraus folgt, daß jedes Memory einen eigenen GBC haben kann. Die Menge der Eindrücke, die im Gedächtnis gespeichert sind, wird in einer Liste des Memory-Objektes verwaltet. Die dazu erforderlichen Modalitäten sind in einer anderen Liste des Memory-Objektes gespeichert. Das Memory-Objekt verfügt über eine Reihe von Funktionen, die den Zugriff auf die internen Listen kapseln.

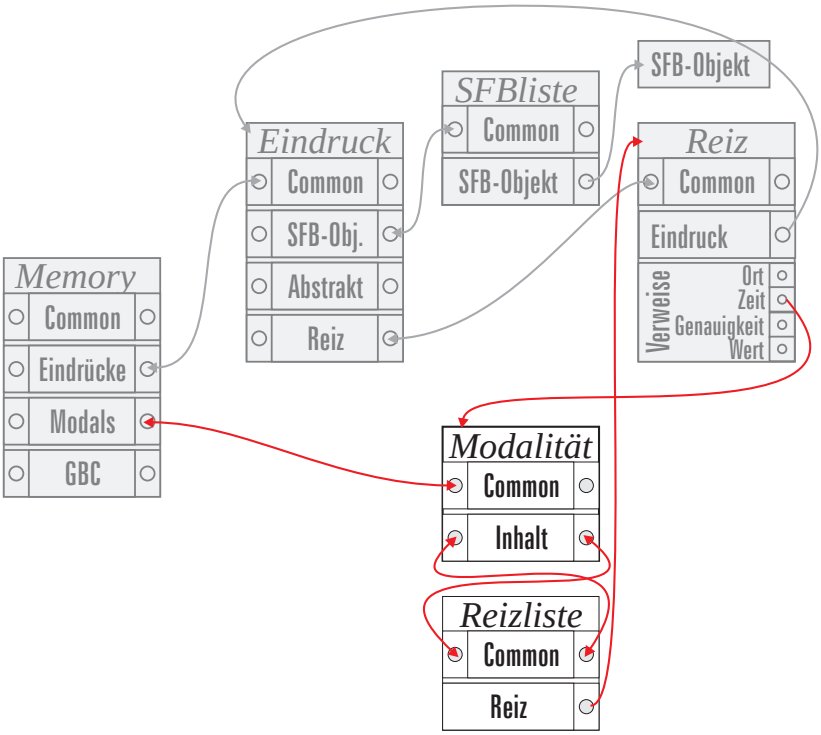


Abbildung 3.6: Pointerstruktur der Zeitmodalität

Der Eindruck setzt sich der Einfachheit halber nur aus einem Reiz und einem SFB-Objekt zusammen (siehe Abbildung 3.5). Für die abstrakten Werte läuft die Zeigerverwaltung analog zu den Reizen. Der typische Eindruck beim Betrieb des Programms hat einige abstrakte Werte und meist mehr als 20 Reize. Die Information zu dem Reiz kommt aus dem SFB-Objekt, das in der SFBListe des Eindrucks gespeichert ist. Die Daten selbst sind im Reiz gespeichert und der Reiz ist nach der Art der Daten in die Modalitäten eingetragen. Für Ort, Zeit und Genauigkeit werden immer die jeweiligen Modalitäten genutzt. Die vierte Modalität des Reizes wird passend zur Reizmodalität selbst, also zur Art des Reizes gewählt. Die Verweise auf die hier beteiligten Modalitäten sind im Reiz fest eingetragen. Der abstrakte Wert wird auf ähnliche Weise in den Modalitäten eingetragen. Der Modalitätstyp des abstrakten Wertes ist ein anderer als der Typ des Reizes und deshalb wird

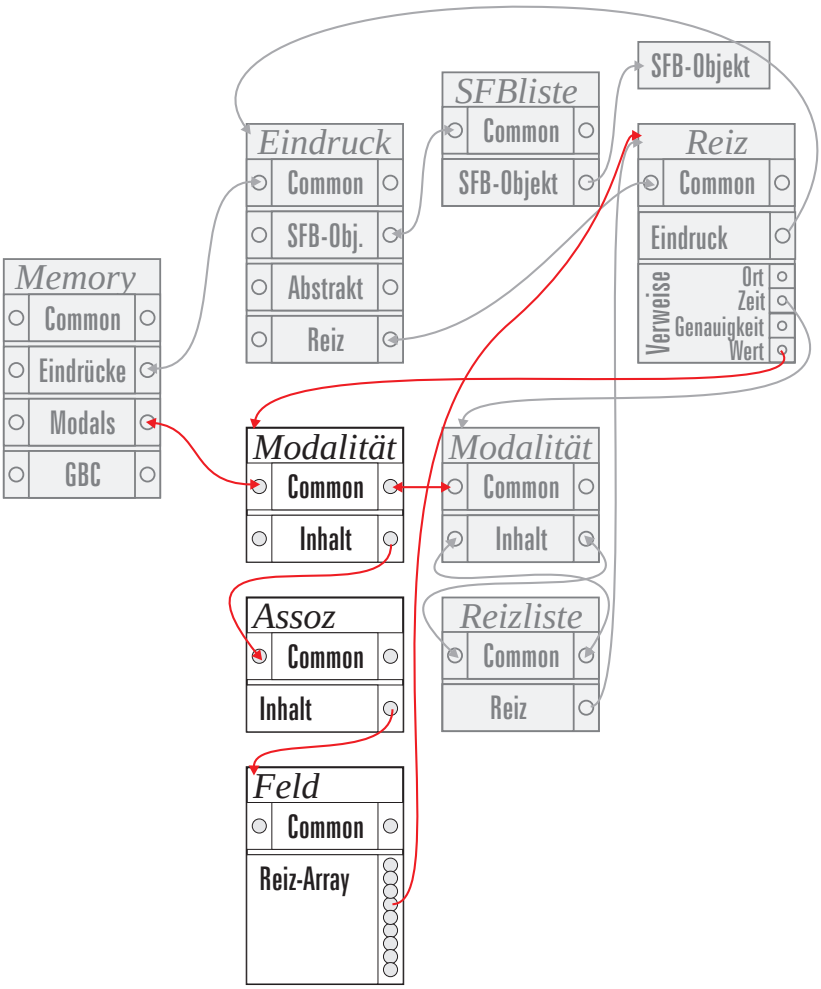


Abbildung 3.7: Pointerstruktur der Wertmodalität mit Feld

der abstrakte Wert in einer anderen Modalität eingetragen. Jeder einzelne Reiz hat zudem noch einen Verweis auf den Eindruck, zu dem er gehört. Dadurch ist es möglich, von einer Modalität aus die Ähnlichkeit auf Eindrucksebene einzuschätzen.

Die Organisation der Modalitäten bestimmt, wie die Modalität mit dem Eintrag des Reizes umgeht. Die Zeitmodalität ist in einer ringförmigen Struktur organisiert (siehe Abbildung 3.3), um immer die natürliche Sortierung in der Zeitreihenfolge beibehalten zu können. Der Reiz wird hier durch den Eintrag in eine Reizliste chronologisch abgelegt (siehe Abbildung 3.6). Die anderen Modalitäten sind meist assoziativ organisiert; das heißt, der Reiz wird in einem Assoziativobjekt gespeichert. Ist er der erste Reiz, so zeigt das Assoziativobjekt direkt auf ihn. Ansonsten wird der Reiz in einem Feld gespeichert (siehe Abbildung 3.7), denn Assoziativobjekt kann als Feld oder Kohonenkarte (siehe Abschnitt 2.3) organisiert sein. Im letzteren Fall handelt es sich um ein Array aus weiteren Assoziativobjekten, die dann wiederum den Reiz, das Feld oder weitere Subkarten speichern können (siehe Abbildung 3.8). Auf diese Weise wird im Assoziativobjekt eine wertabhängige Repräsentation erreicht.

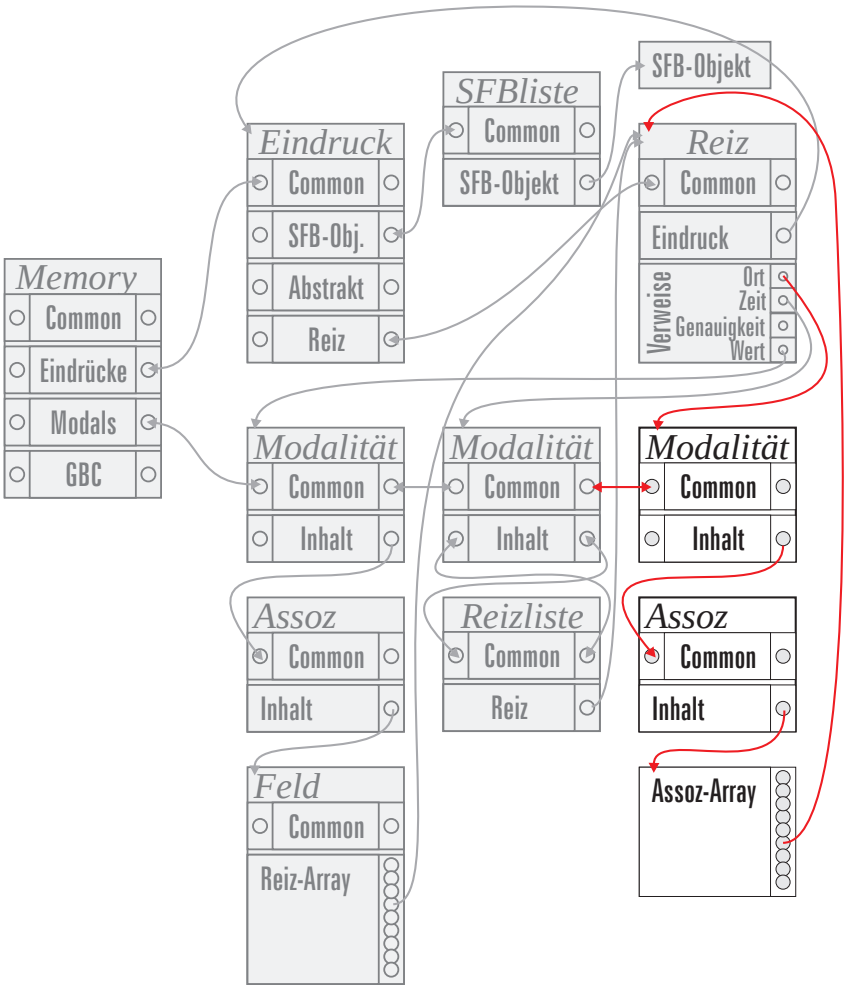


Abbildung 3.8: Pointerstruktur der Ortsmodalität mit Karte

3.4 Gedächtnis

Bisher haben wir nur die Speicherung der Daten in komplexen Strukturen behandelt und kommen nun zu dem Punkt, an dem die bedeutendere Gedächtnisleistung, die Assoziation, zur Sprache kommt. Die Speicherung von Daten alleine reicht noch nicht aus, um die Fähigkeiten biologischer Systeme zu erreichen. Sie ist dennoch eine Voraussetzung, um abstraktere Zugriffsoperationen effizient und elegant implementieren zu können. Ich werde nun den Weg der Information beschreiben, um an den verschiedenen Punkten auf die Gedächtnisleistungen einzugehen.

3.4.1 Schnittstelle zum SFB

Ein Teil der Assoziationsleistung wird bei der Aufnahme der SFB-Objekte erbracht, indem die Daten in das interne Gedächtnisformat übersetzt werden. Die Aufnahme der Daten geschieht im Modul “Port” und ist speziell auf das SFB-Demosystem und die darin vorkommenden Daten zugeschnitten. Die Demand Streams werden hier empfangen und Anfragen über RPCs entgegengenom-

men. Die von DACS-Library-Funktionen gelieferten Daten werden typisiert in einem Listenelement abgelegt und durch die Umwandlung in Reize in ihren Werten, aber nicht in der eigentlichen Struktur verändert (siehe Wert-Assoziation im nächsten Abschnitt).

Die adäquate Umsetzung der SFB-Objekte in die internen Daten ist eine wichtige Voraussetzung zum Funktionieren der folgenden Verarbeitungsschritte, da diese mit den hier festgesetzten Daten arbeiten müssen. Die Umsetzung von SFB-Objekten ist unvollständig, da nicht alle Daten sinnvoll in interne Strukturen übersetzt werden können oder assoziationsfähige Informationen beinhalten. Zum Beispiel werden Sequenznummern oder Aufnahmegeräte bei der Übersetzung ignoriert. Andere Daten des SFB-Objektes sind für die Assoziation von zentraler Bedeutung, wie zum Beispiel der Umfang oder der Schwerpunkt von Regionen. Da die SFB-Objekte rekursiv aufgebaut sind, wird die Übersetzung der Daten in einer rekursiven Funktionskette vorgenommen. Für jedes SFB-Objekt gibt es eine Funktion, welche die wichtigen Daten jeweils in entsprechende Reize übersetzt oder in weitere Substrukturen mit weiteren Funktionen auflöst. Zu den Interna des SFB-Formates sei hier auf Anhang B verwiesen.

3.4.2 Assoziation

Wie schon früher angedeutet, werden zwei Formen der Assoziation implementiert. Zum einen gibt es die Wert-Assoziation bei der Aufnahme eines SFB-Objektes; zum anderen findet die Eindrucks-Assoziation beim Abgleich der aufgenommenen Daten mit der bekannten Datenbasis statt.

Wert-Assoziation

Schon bei der Aufnahme eines SFB-Objektes wird die Wert-Assoziation vorgenommen. Zu einem neuen Reiz wird in der bestehenden Datenbasis — der entsprechenden Wert-Modalität — nach ähnlichen Reizen gesucht. Aus diesen ähnlichen Reizen wird der Durchschnitt berechnet, der als Reiz nie selbst wahrgenommen wurde. Der neue Reizwert wird bezüglich dieses Durchschnitts korrigiert und dann in die Datenbasis aufgenommen. Dieser Wert wird in das SFB-Objekt zum Zwecke späterer Ausgabe zurückgeschrieben. Bei Aufzählungstypen wie zum Beispiel der Farbnummer von Baufixteilen ist diese Wert-Assoziation nicht sinnvoll und es wird kein anderer Wert für das SFB-Objekt aus den bestehenden Daten berechnet.

Die Ähnlichkeiten auf Reizebene werden zur Aufnahmezeit genutzt, um die Daten der SFB-Objekte zu ändern. Später ausgegebene Objekte sind daher ein Abbild des Wissens des Gedächtnisses zur Aufnahmezeit und nicht nur “auswendig gelernte” Daten.

Eindrucks-Assoziation

Wenn ein SFB-Objekt komplett in Reize übersetzt ist, wird die neue Reizkette zusammen mit dem SFB-Objekt in einem neuen Eindruck gespeichert. Dieser Eindruck wird nun in der Eindrucks-Assoziation mit anderen, schon bestehenden Eindrücken verglichen. Die Eindrucks-Assoziation kann man in mehrere Schritte einteilen:

- Für jeden Reiz der Reizkette des neuen Eindrucks wird modalitätsspezifisch nach ähnlichen Reizen gesucht. Das heißt, für jede Modalität wird eine Statistik mit den Eindrücken angelegt, die einen zu dem Reiz aus der Reizkette ähnlichen Reiz besitzen. In der Statistik wird festgehalten, welche Eindrücke überhaupt Ähnlichkeiten aufweisen, und wie oft sie bei dieser Assoziation genannt wurden; außerdem wird der durchschnittliche Abstand der Reize des neuen Eindrucks zu den Reizen des als ähnlich befundenen Eindrucks als statistische Größe berechnet.

- In einem zweiten Schritt werden die ähnlichsten Eindrücke aus den Statistiken gewonnen. Dies geschieht, indem nach den Eindrücken gesucht wird, die die häufigste Nennung in den Statistiken erhalten haben. Diese Eindrücke haben also in möglichst vielen Modalitäten Reize, die den Reizen des neuen Eindrucks ähneln.

- Aus diesen Kandidaten wird derjenige Eindruck ausgewählt, der den kleinsten kumulativen Abstand in den Modalitäten zu dem neuen Eindruck aufweist. Ist kein Eindruck eindeutig "im Vorteil", so kann keine Entscheidung getroffen werden, und die assoziierten Eindrücke gelten als gleich gut passend. Kann eindeutig ein Eindruck assoziiert werden, so wird der neue Eindruck mit dem assoziierten Eindruck vereinigt. Dies passiert, indem die Listen der Reize, der SFB-Objekte und der abstrakten Werte vereinigt werden und eine der beiden Grundstrukturen wieder freigegeben wird.

Diese Assoziationsmethode wendet einige Prinzipien der Biologie direkt an: Es werden keine absoluten Schranken gesetzt, um Entscheidungen zu treffen, sondern eine relative Abschätzung liefert das Ergebnis; Eindrücke werden durch Ähnlichkeit in mehreren Modalitäten assoziiert; die Ähnlichkeitsmaße aus verschiedenen Modalitäten werden verknüpft ausgewertet.

An dieser Stelle wird deutlich, wie wichtig die richtige Datenrepräsentation für die Leistung des Gedächtnisses ist. Neben Datenrepräsentation spielt besonders die Gestaltung der Modalitäten eine Schlüsselrolle. Wird innerhalb der Modalitäten kein "vernünftiges" Abstandsmaß definiert und auf die Daten angewendet, so kann mit dieser Architektur keine besondere Leistung erreicht werden.

3.4.3 Abstandsmaße

Die Abstandsmaße spielen in dreierlei Hinsicht eine Rolle:

- Sie entscheiden bei der Assoziation mit, welcher Eindruck der ähnlichste ist, indem jeder Abstand in die Bewertung mit einfließt.

- Schon im Vorfeld dieser Assoziation werden die Daten durch das Abstandsmaß bei der Wert-Assoziation beeinflußt.

- Außerdem werden bei der Speicherung die Daten mit Hilfe des Abstandsmaßes in die Felder einsortiert.

Um eine solche Fülle von Funktionen mit wenig Aufwand zu verwirklichen, bedarf es grundlegender Überlegungen über die Art des Abstandsmaßes. Ich lege folgende Normierung fest, die eine flexible Einsetzbarkeit ermöglicht: Abstandsmaße sind für je zwei Datenvektoren definiert und liefern ein Ergebnis im Bereich der reellen Zahlen größer gleich `0`; die Abstandsmaße sind intern so zu skalieren, daß ein Abstand von kleiner `1` noch als sehr ähnlich gelten kann, ein Abstand von `10` noch im Rahmen der Ähnlichkeit liegt und darüber die Vektoren nicht mehr als ähnlich behandelt werden sollen. Mit dieser Festlegung läßt sich jedes Abstandsmaß definieren und relativ zu den anderen Abstandsmaßen justieren, so daß typische Reizbeispiele als ähnlich eingestuft werden.

Die einzelnen Abstandsfunktionen sind logarithmisch, euklidisch, linear, exponentiell oder mit Arkustangens implementiert. Für Aufzählungstypen sind Verwechslungsmatrizen (Abstandstabellen) vorgesehen, um empirischen Tests möglichst nahe zu kommen, ohne eine mathematische Funktion angeben zu müssen. Jede Abstandsfunktion kann während der Laufzeit durch Parameter oder durch das Parameterfile justiert und verändert werden. Hierzu verweise ich auf den Abschnitt 4.2.

3.5 Visualisierung und Steuerung

Die Visualisierung und interaktive Steuerung des Gedächtnisses wird mit Tcl/Tk realisiert. Eine Anbindung an die Variablen des Programms wird mit der Funktion `Tcl_LinkVar()` an das Objekt vom Typ `param_t` erreicht. Einige C-Funktionen erlauben es, direkt von Tcl aus Aktionen einzuleiten; die Ergebnisse der C-Funktionen werden in Tcl weiter verarbeitet und interpretiert. Auf diese Weise ist die Visualisierung von Karten und die Übersicht über die verwendeten Objekte des C-Programms realisiert. Beim Beenden des Programms durch die Tk-Oberfläche wird die Verbindung zu DACS korrekt durch eine C-Funktion geschlossen und das Programm danach von Tcl beendet.

Ich habe viele Tips zur Oberflächengestaltung aus dem Lehr- und Handbuch zu Tcl/Tk von John K. Ousterhout [Ous94] entnommen. Auch das Buch von Brent B. Welch [Wel95] hat mir gute Dienste beim Design der Oberfläche geleistet. Ein Nachteil der jetzigen Lösung der Zusammenarbeit von C und Tcl/Tk ist die Voraussetzung von Threadsafe-X11-Library-Funktionen. Verfügt das Rechnerbetriebssystem nicht über diese Unterstützung, so kann das Programm nur ohne Visualisierung betrieben werden.

Kapitel 4

Ergebnis

Dieses Kapitel befaßt sich nicht nur mit den Ergebnissen, die mit dem vorliegenden Programm “VDMF” zu erzielen sind, sondern gibt auch Erklärungen, wie solche Phänomene zustande kommen, und Hinweise für die Steuerung, um solche Ergebnisse überhaupt erzielen zu können. Zunächst beschreibe ich die Steuerung mit dem Parameterfile bzw. der Tcl/Tk-Oberfläche des Programms. Danach gehe ich näher auf die verschiedenen Parametergruppen und ihren Wirkungsort im Programm ein. Als Resultat der Einstellungen kann ich dann die Fähigkeiten und Leistungen beschreiben sowie auf einzelne zu beobachtende Phänomene hinweisen. Ich schließe dieses Kapitel mit einer Zusammenfassung des Erreichten.

4.1 Steuerung

Im normalen Betrieb des Programms muß der Benutzer nicht aktiv eingreifen. Die wichtigsten Parameter lassen sich über die Kommandozeile dem Programm übergeben und ermöglichen so eine flexible und unkomplizierte Einbindung in das bestehende SFB-Demosystem. Alle im Tcl/Tk einstellbaren angegebenen Parameter lassen sich auch direkt im Parameterfile vor dem Programmstart mit einem Texteditor festlegen und unterstützen die Automatisierung weiter.

Um das Programm interaktiv steuern zu können gibt es die Tcl/Tk-Oberfläche. Es sind eine Reihe von Möglichkeiten in verschiedenen Fenstern zusammengestellt. Der Benutzer bekommt nach dem Programmstart (wenn per Kommandozeilen-Parameter erwünscht) das Hauptfenster (Abbildung 4.1) angezeigt. Dieses Fenster bietet die Verzweigung in die verschiedenen anderen Steuerungsfenster an. Zu jedem Fenster und zu jedem Parameter gibt es einen Hilfetext, in welchem knapp die Möglichkeiten und Hintergründe umrissen werden.

Das wohl am meisten benutzte Fenster ist das “Pausen-Fenster” (Abbildung 4.2). Hier kann der Benutzer verschiedene Komponenten oder das ganze Programm pausieren lassen. Hinter dem Button “Statistik” verbirgt sich eine immer wieder aktualisierbare Zusammenstellung der vom Programm benutzten C-Objekte (Abbildung 4.3). Das Fenster “Abstände” faßt statistische Daten der Abstandsberechnung (Abbildung 4.4) zusammen und ermöglicht damit eine Justierung der entsprechenden Parameter. Das “Stream-Fenster” (Abbildung 4.7) ermöglicht es dem Benutzer, die Namen der verschiedenen Streams für den nächsten Programmstart von “VDMF”

Abbildung 4.1: Hauptfenster von “VDMF”

Abbildung 4.2: Pausenfenster

neu festzulegen. Mit dem Button “Kohonen” gelangt der Benutzer in den Bereich der internen Gedächtnisdarstellung (Abbildung 4.6). Es wird der Zustand der Kohonenkarten visualisiert und der Benutzer kann mit dem Optionsfenster (Abbildung 4.5) weitere Einstellungen vornehmen. Unter dem Punkt “Parameter” (Abbildung 4.8) werden alle weiteren Steuerungsparameter für das Programm zusammengefaßt. In den beiden zuletzt genannten Fenstern können die Parameter verändert und dann in einem Paramterfile festgehalten werden.

4.2 Parameter

Ich möchte auf diejenigen Parameter (aus Abbildung 4.8) näher eingehen, welche für die verschiedenen Fähigkeiten und Leistungen von Belang sind; andere Parameter sind nur zur Steuerung der Visualisierung notwendig oder erklären sich selbst.

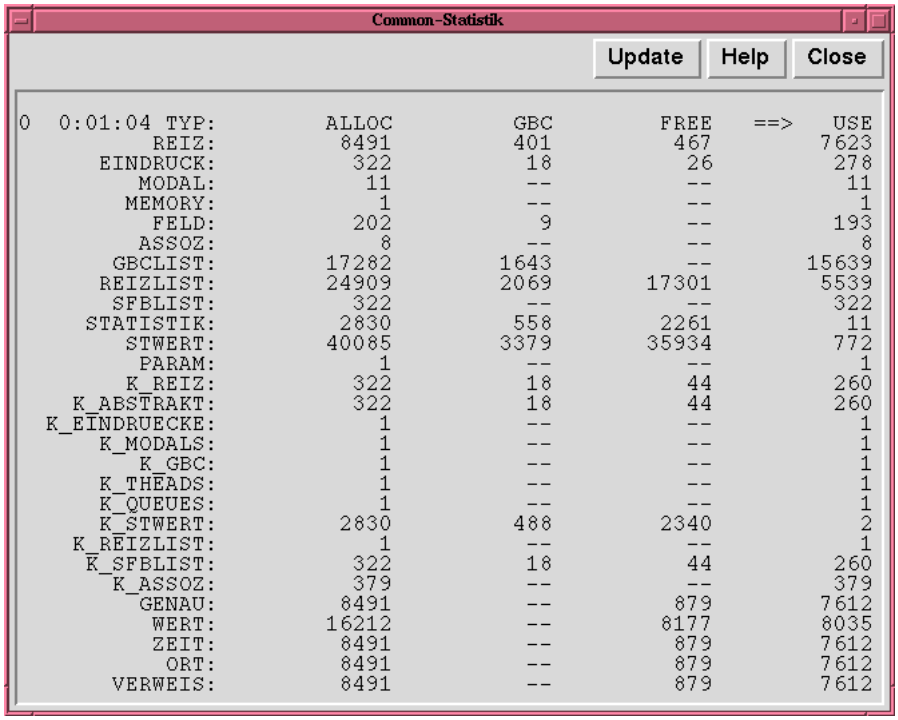


Abbildung 4.3: Statistik

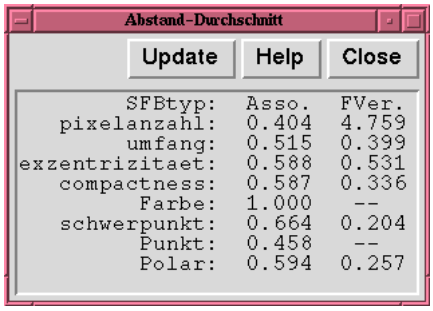


Abbildung 4.4: Abstände

Die drei Parameter `Zeitskalierung`, `Wandlungsthreads` und `Rauschen` steuern Teile der SFB-Objekt-Aufnahme. Die Zeitauflösung kann verändert werden, um bei sehr schnellen Rechnern Zeitunterschiede darstellen zu können. Momentan ist die minimale Zeitauflösung mit `1/1000` Sekunde eingestellt. Die maximale Threadzahl bei der Wandlung von SFB-Objekten zu Eindrücken bestimmt die mögliche Parallelität des Programms. Bei Mehrprozessorsystemen sollte hier eine den Prozessoren angemessene Zahl eingetragen werden. Das Rauschen im Port ist eine Möglichkeit, in Testumgebungen mit wenig variablen Eingabedaten eine Variation zu erzeugen und die Stabilität des Programms testen zu können. Es hat sich gezeigt, daß Rauschen eine Verbesserung der Gedächtnisleistung verursachen kann, da zu viele identische Werte die Assoziation durch fehlende Unterscheidbarkeit verschlechtern. Identische Werte verursachen in der Abstandsberechnung immer identische Abstände und führen so zu einem starren Verhalten bezüglich der

Abbildung 4.5: Kohonenkarten Optionen

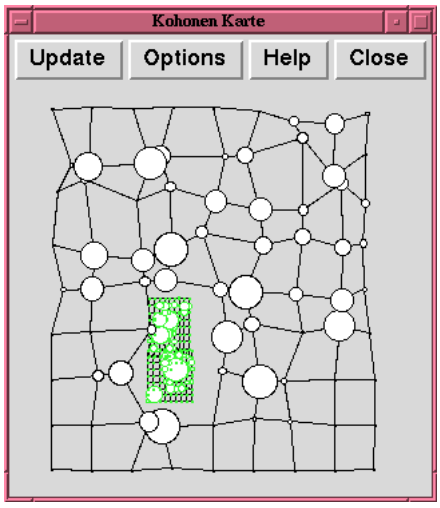


Abbildung 4.6: Kohonenkarte der Ortsmodalität mit Subkarten

Assoziation. Die Statistiken bei der Assoziation enthalten dann häufig sehr ähnliche Werte für verschiedene Eindrücke, so daß keine Unterscheidung zwischen den Eindrücken möglich ist.

Mit den Steuerungsparametern `Zeit: Reiz zu alt` und `Zähler: Reiz zu alt` sowie der `Zeitverzögerung`, kann das aktive Vergessen der zu alten Reize, mit den Parametern `Zeitverzögerung` und `Zähler: Eindruck leer` das Vergessen der Eindrücke beeinflußt werden. Die Zeitverzögerung ist die Mindestruhezeit zwischen zwei Durchgängen der jeweiligen Threads durch die Datenbasis. Ein Reiz wird nach seiner Altersüberschreitung als “zu vergessen” markiert und kann ab diesem Zeitpunkt von jedem beliebigen Thread dem GBC übergeben werden. Dies geschieht in der Regel durch das Überschreiben der Position in den Feldern, also durch ein Überlagern des Reizes durch neuere Daten. Dies entspricht einem Modell aus der Psy-

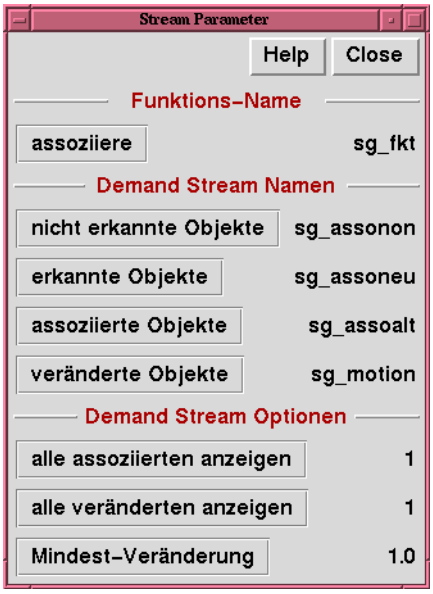


Abbildung 4.7: Stream-Fenster

chologie, in dem das Vergessen beim Menschen durch das Überlagern des Wissens durch neue, aktuelle Eindrücke beeinflußt wird (siehe [Bad79]). Trifft der Reiz-Vergessen-Thread nach mehr als der eingestellten Anzahl von Durchgängen immer noch auf den alten Reiz, so wird der Reiz an den GBC übergeben. Dies entspricht einem anderen Modell der Psychologie, welches das Vergessen durch einen reinen Zeitfaktor zu erklären versucht. Da es bisher keinen Versuch gibt, der eines der beiden Modelle widerlegt, habe ich beide Ansätze im Gedächtnis verwirklicht. Ein Eindruck wird vergessen, wenn er durch keine Reize mehr in der Datenbasis vertreten wird, also kein Reizmuster mehr hat. Nach der eingestellten Anzahl von Durchgängen wird der Eindruck aktiv vom Eindruck-Vergessen-Thread an den GBC übergeben. Mit diesen Parametern kann das Gedächtnis als Kurz- oder Langzeitspeicher konfiguriert werden.

Die GBC-Steuerung läßt sich mit `Zeitlimit`, `Objekt-Zähler` und `Mehrstufigkeit` beeinflussen. Es ist hier ein Mittelweg zwischen Geschwindigkeit eines Durchgangs und der Häufigkeit der Durchgänge zu finden. Wird der GBC nicht von einer Anzahl von Conditions der auf Null gesetzten Referenzen angestoßen, wird er spätestens nach einem einstellbaren Timeout aktiv. Die Sicherheit des GBC ist durch die Mehrstufigkeit gewährleistet.

Auf die Speicherung der Daten in Feldern kann mit den Parametern `Voreinstellung Größe`, `maximale Größe` und `Rekonfiguration` Einfluß genommen werden. Bei der Anlage einer neuen Modalität wird für diese die Einstellung aus dem Defaultwert für alle Felder der Modalität angenommen. Sind die Modalitäten bereits angelegt, so kann global für alle Felder eine maximale Speicherkapazität angegeben werden. Damit kann auf die Merk- und Erkennungsfähigkeit Einfluß genommen werden. Die Rekonfiguration von Feldern betrifft das Verhalten bei der Umwandlung von Feldern in Assoziativobjekte. Werden bei dieser Umwandlung keine verschiedenen Werte innerhalb eines Feldes gefunden, so werden alle alten Werte sofort als “überschreibbar” markiert, um neuen Werten Platz zu machen. Ist diese Rekonfiguration nicht erlaubt, so wird in die Liste der Felder ein neues Feld eingefügt, um dort weitere Werte zu speichern. Die Rekonfiguration ist sinnvoll, um das Gedächtnis nicht mit vielen gleichen Werten zu belasten.

Alle Parameter, deren Namen mit `Skal.` beginnen, sind für die Justierung der Abstandsmaße verantwortlich. Es gilt generell, daß ein größerer Parameter auch einen größeren Abstand bei der Berechnung zur Folge hat. Die Parameter sind untereinander kaum vergleichbar, da ihnen verschiedenste Formeln zugrunde liegen. Die Formeln sind jeweils mit dem individuellen Hilfetext einsehbar. Die relative Justierung der einzelnen Parameter beeinflußt das Gedächtnis an vielen Stellen (siehe Abschnitt 3.4.3).

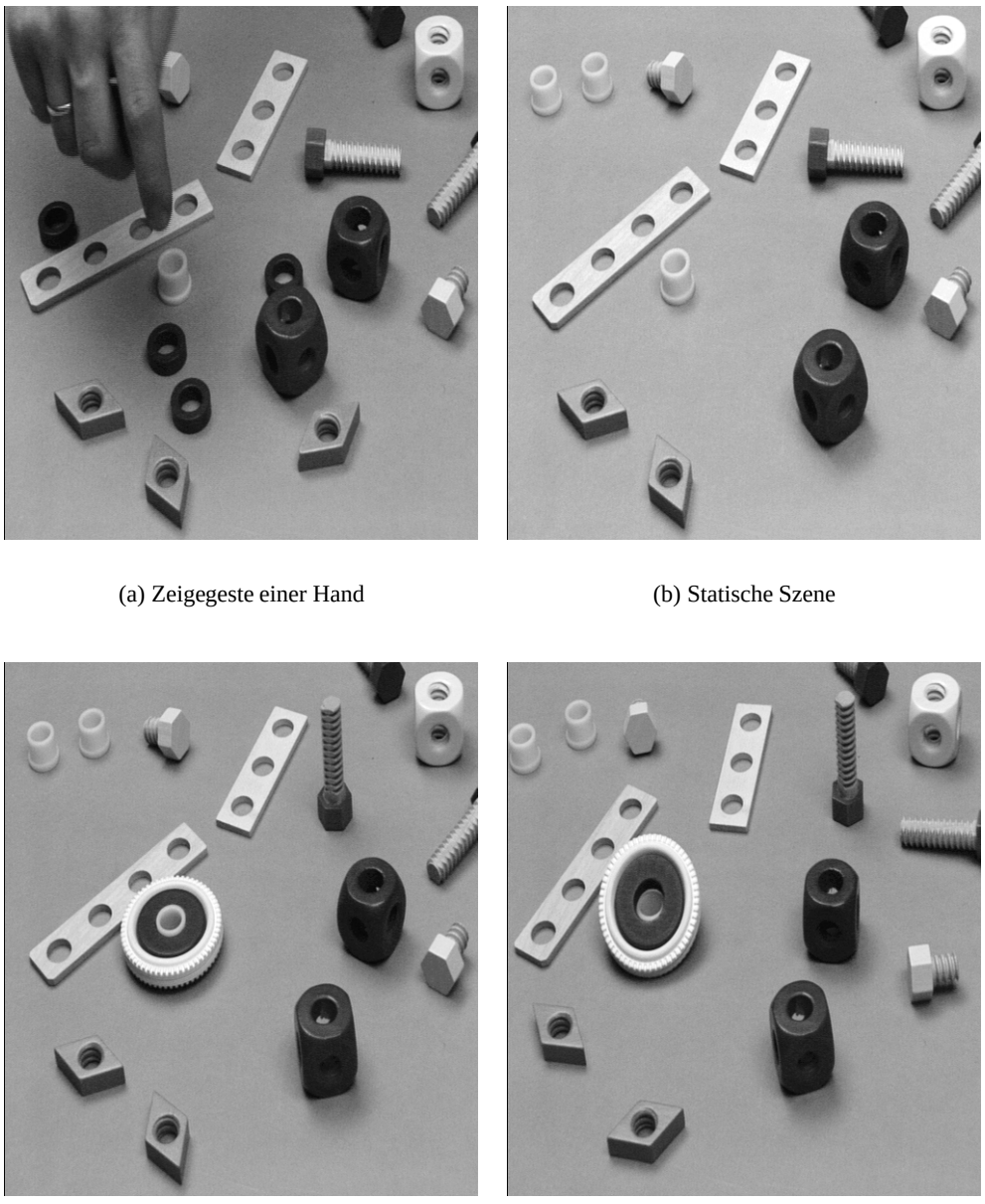
Die Parameter `maximaler Eindrucks-Abstand` und `Mindest-Genauigkeit`, sowie `Mindest-Modalanteil`, `Mindest-Eindrucksanteil` und `Vereinigungsschranke` wirken direkt auf die Assoziation und die weitere Verarbeitung ein. Der maximale Abstand bezieht sich auf die Eindrucksassoziation. Es werden nur Eindrücke in die Statistiken übernommen, deren Reizabstand unter dem angegebenen Maximalwert liegt. Zusätzlich müssen die Reize der Eindrücke eine Mindestgenauigkeit besitzen, also “glaubhaft” sein. Die Eindrücke werden in den Statistiken nur weiter betrachtet, wenn ihre Anteile an den assoziierten Modalitäten und aller assoziierter Eindrücke die angegebenen Grenzen überschreiten. Werden bei der Assoziation zwei zusammengehörige Eindrücke gefunden, so werden sie nur vereinigt, wenn sie die Bewertungsschranke zur Vereinigung überschreiten. Damit kann verhindert werden, daß zu viele Objekte sich in einem Eindruck ansammeln und das Gedächtnis keine Eindrücke mehr unterscheidet.

Abbildung 4.8: Parameterfenster

In der Wert-Assoziation werden die Schranken durch die Parameter (analog zu der Assoziation der Eindrücke) `maximaler Abstand`, `Mindest-Genauigkeit` und `Gewichtungsfaktor` festgelegt. Die Wertgewichtung legt fest, in welchem Maße der im Gedächtnis assoziierte Mittelwert bei der Wandlung von SFB-Objekten in Eindrücke den Wert beeinflussen soll.

Die beiden Parameter `Voreinstellung Kantenlänge` und `Rekonfiguration` definieren das Verhalten der Assoziativ-Objekte. Der Defaultwert legt für alle neuen Karten die Kantenlänge fest. Die Dimensionalität wird durch die Dimension des Reizes festgelegt. Verbietet man die Rekonfiguration, werden die Felder nicht mehr in Assoziativ-Objekte umgewandelt.

Die dynamische Anpassung der Kohonenkarten kann in dem Optionsfenster der Kartendarstellung eingestellt werden. Die Parameter `Lernradius`, `Lerngeschwindigkeit` und `Lernrate` ermöglichen eine Steuerung der neuronalen Netze in der üblichen Weise.



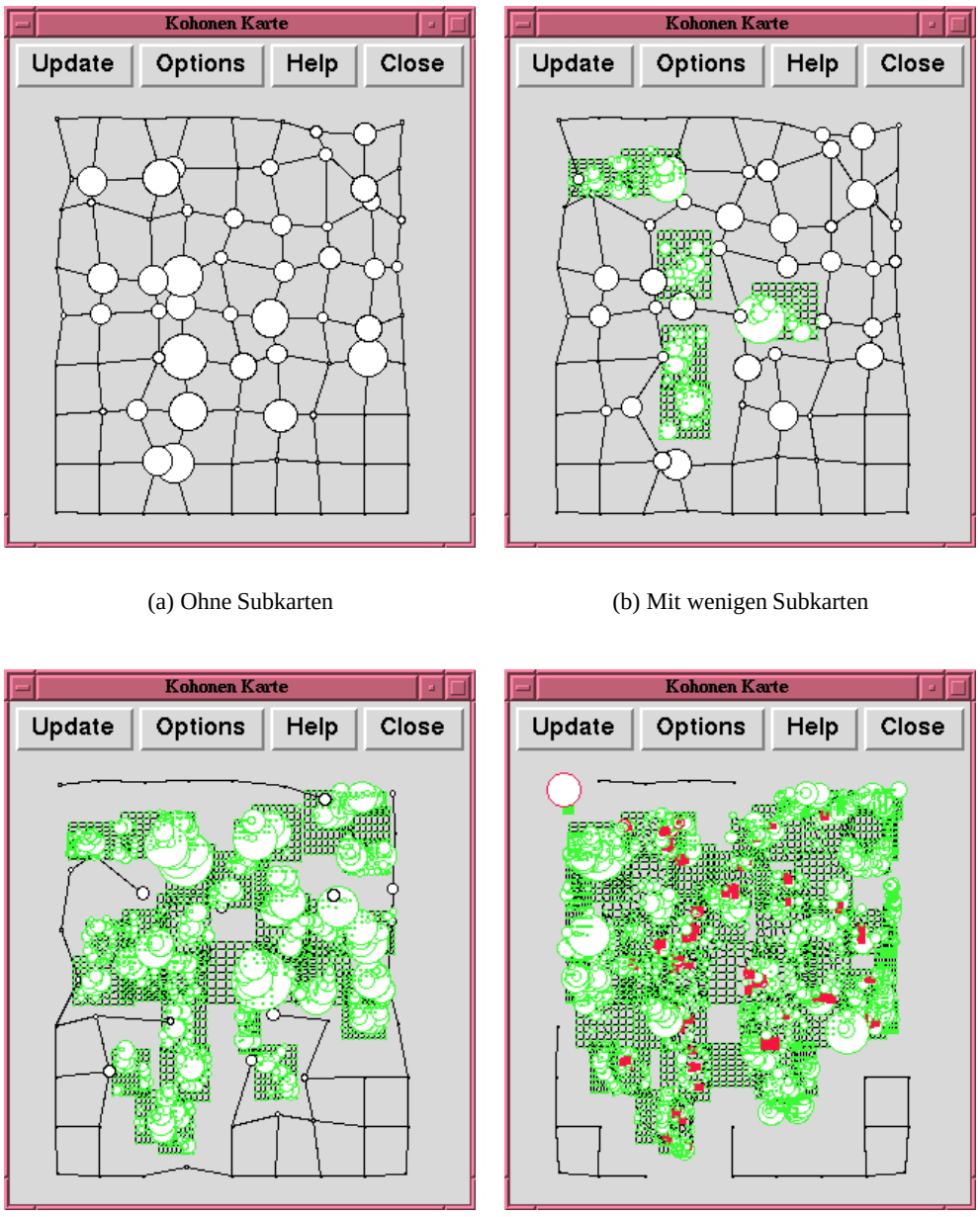
(a) Zeigegeste einer Hand

(b) Statische Szene

(c) Mit Veränderungen relativ zu (b)

(d) Mit Veränderungen relativ zu (c)

Abbildung 4.9: Kamerabilder der Testdaten



(a) Ohne Subkarten

(b) Mit wenigen Subkarten

(c) Mit vielen Subkarten

(d) Mit Subkarten zweiter Ordnung

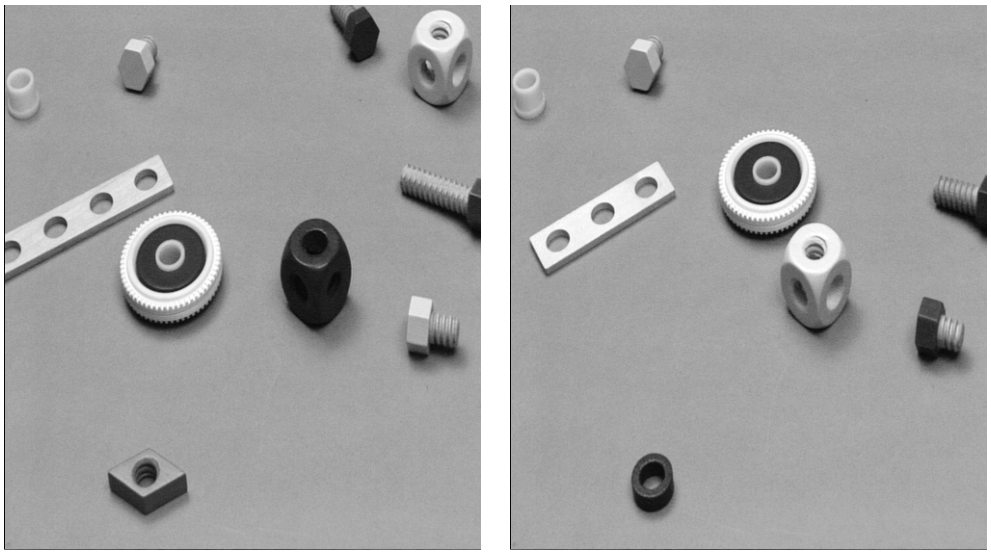
Abbildung 4.10: Kohonenkarten in verschiedenen Entwicklungsstufen

4.3 Fähigkeiten und Leistungen

Das vorliegende Programm erfüllt den Katalog von Anforderungen, der in Abschnitt 1.4 aufgestellt wurde. Die in der Biologie zu beobachtenden Leistungen werden natürlich nicht in jedem Detail erreicht, aber die Tendenz ist klar erkennbar. Die Architektur erlaubt ein Wiedererkennen von Eindrücken auf Grundlage der dargelegten Techniken. Es besteht eine hohe Abhängigkeit von der Güte der Umsetzung der Daten in die interne Repräsentation. Nur wenn die relevanten Da-

ten umgesetzt werden, an denen eine Gedächtnisleistung nachvollzogen werden kann, ist es der vorgestellten Architektur möglich, interessante Phänomene zu zeigen.

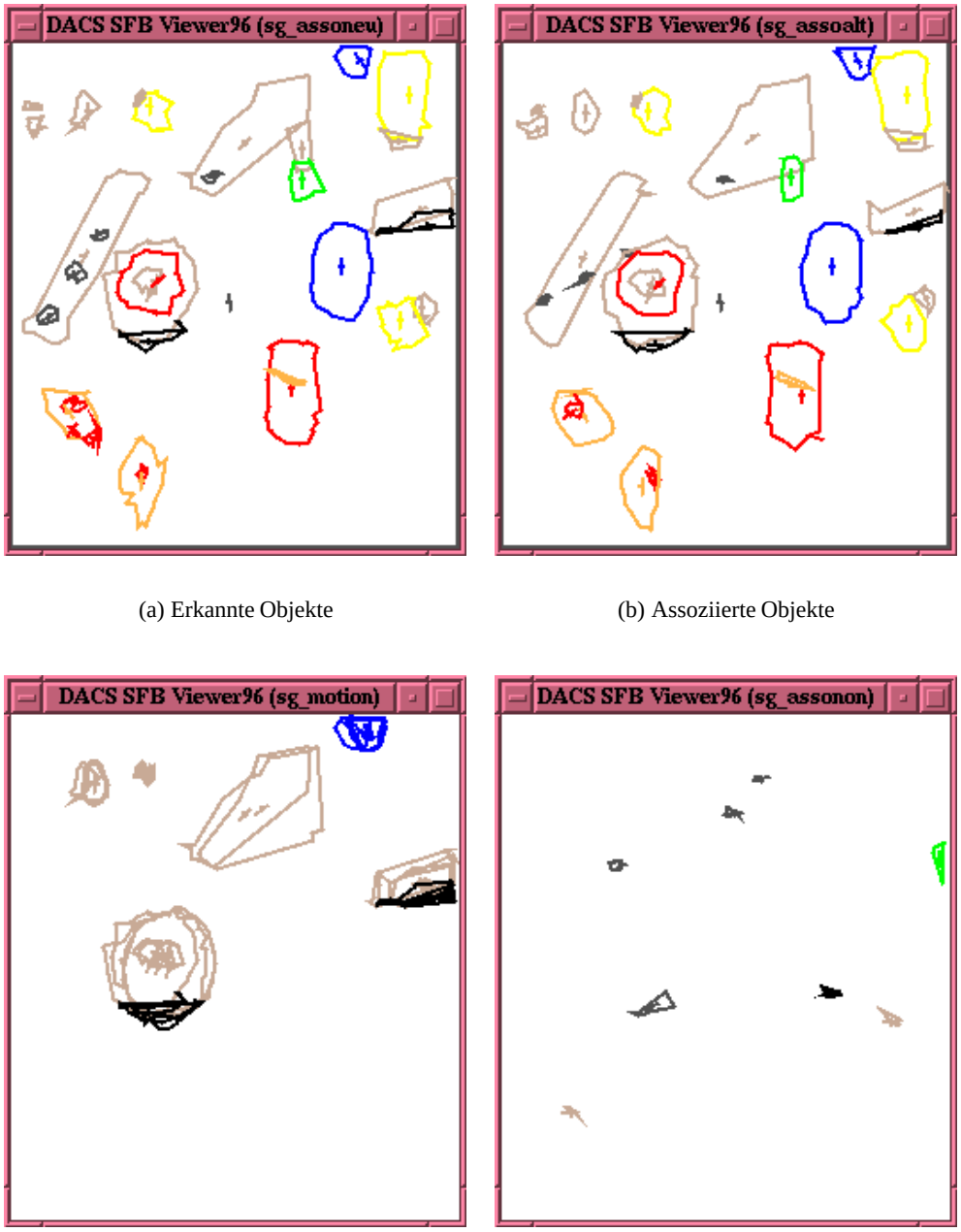
Viele der geforderten Architekturmerkmale sind in ihrer Funktionalität und ihrem Einfluß schwierig zu charakterisieren und zu beschreiben. Häufig kann das System nur als Ganzes beobachtet werden und der Einfluß des einzelnen Merkmals bleibt ungewiß. Andere Anforderungen können konkret als erfüllt bezeichnet werden und sind gut zu untersuchen oder schlagen sich offensichtlich in der Implementierung nieder.



(a) Ausgangsbild (b) Nach Änderung der Szene

Abbildung 4.11: Testbilder zur Provokation einer Farbverwechslung

Die technischen Anforderungen können bis auf leichte Abstriche in der Gesamtperformance als erfüllt gelten. Das Programm ist vollständig in den SFB integrierbar und nutzt sowohl Demand Streams als auch RPCs des DACS. Die verschiedenen SFB-Objekte werden vom Portmodul sinnvoll in die interne Repräsentation übertragen und damit ist eine Einsetzbarkeit auf vielen Ebenen des SFBs geschaffen. Die Einstellungen erlauben einen Einsatz als Lang- und Kurzzeitgedächtnis. Es sind durchgängig allgemeine Prinzipien und nur in der Anpassung an den SFB Speziallösungen umgesetzt. Die Geschwindigkeit des Programms liegt an der unteren Grenze des Geforderten. Sie ist jedoch leicht durch den Verzicht auf Wartbarkeit und Erweiterbarkeit zu steigern. Viele Sicherheitsabfragen sind im jetzigen Entwicklungsstadium überflüssig geworden, werden aber bei einer Weiterentwicklung wieder gebraucht. Die Parallelität des Programms erlaubt eine weitere Performancesteigerung durch den Einsatz weiterer Mehrprozessorsysteme. Tests auf einem Vierprozessorsystem belegen eine Nutzung von gut *90%* des Gesamtsystems. An vielen Stellen werden Speicherbereiche von mehreren Threads benutzt und können so auf Mehrprozessorsystemen zu Cache-Inkonsistenzen führen. Dies bedeutet in der Regel einen deutlichen Performanceverlust, der bei diesem Programm trotz der verteilten Daten noch nicht beobachtet werden konnte. Ob das Programm Systeme mit mehr als vier Prozessoren ebenso gut auslasten könnte, bleibt abzuwarten. Die meisten im SFB installierten Rechnerarchitekturen werden vom Programm unterstützt. Die



(a) Erkannte Objekte

(b) Assoziierte Objekte

(c) Veränderte Objekte

(d) Unbekannte Objekte

Abbildung 4.12: Phänomen Translation

Voraussetzungen in Bezug auf Threads sind im Abschnitt 3.1 und Anhang A aufgezeigt. Eine Portierung für das Irix/Silicon Graphics System wurde aus den dort dargelegten Gründen noch nicht vorgenommen.

Die Vorgaben der Biologie ließen sich gut umsetzen. Die einzelnen Stadien der Informationsverarbeitung von der Aufnahme über die Speicherung bis hin zur Reaktion können direkt im Programm nachvollzogen werden. Die Prinzipien der Biologie sind in vielen Teilen auf eine hohe Parallelität

angewiesen bzw. nutzen sie. Den größten Anteil an der Verwirklichung dieser Parallelität hat wahrscheinlich das read ever/write once Konzept, das sich durch die komplette Implementierung der Datentypen zieht. Die Speicherung der Daten erfolgt wie gefordert in räumlichen, zeitlichen und modalen Kontexten. Die Modalitäten implementieren eine reizspezifische Speicherung der Daten. Sie passen sich den gegebenen Daten schnell in ihrem repräsentierten Wertebereich an und haben ein flexibles Auflösungsvermögen. Die Abbildungen 4.10 stellen die Differenzierung der Ortsmodalität während des Betriebs dar. Die Modalitäten zeigen die Umsetzung von Eigenschaftsräumen verschiedener Dimensionalität, die nahe an den Beispielen der Biologie operiert.

Das System ist in der Lage, relativ konstante Eindrücke ohne Fehler wiederzuerkennen. Leichte Schwankungen, etwa durch die thermischen Veränderungen der Kamera oder andere Effekte, die bei der Aufnahme vorkommen, werden zum großen Teil toleriert. Auf weitere Phänomene höherer Ordnung gehe ich im nächsten Abschnitt näher ein. Die Eindrücke werden bei der Wiedererkennung verknüpft und damit eine Ergänzung von evtl. fehlender Information erreicht. Die somit mögliche Stabilisierung des Datenstroms wird durch den Verlust von Details erkauft. Das System adaptiert sich nur langsam an sehr komplexe Szenarien, da dort die Informationsdichte wesentlich höher und damit die Verallgemeinerbarkeit viel geringer ist.

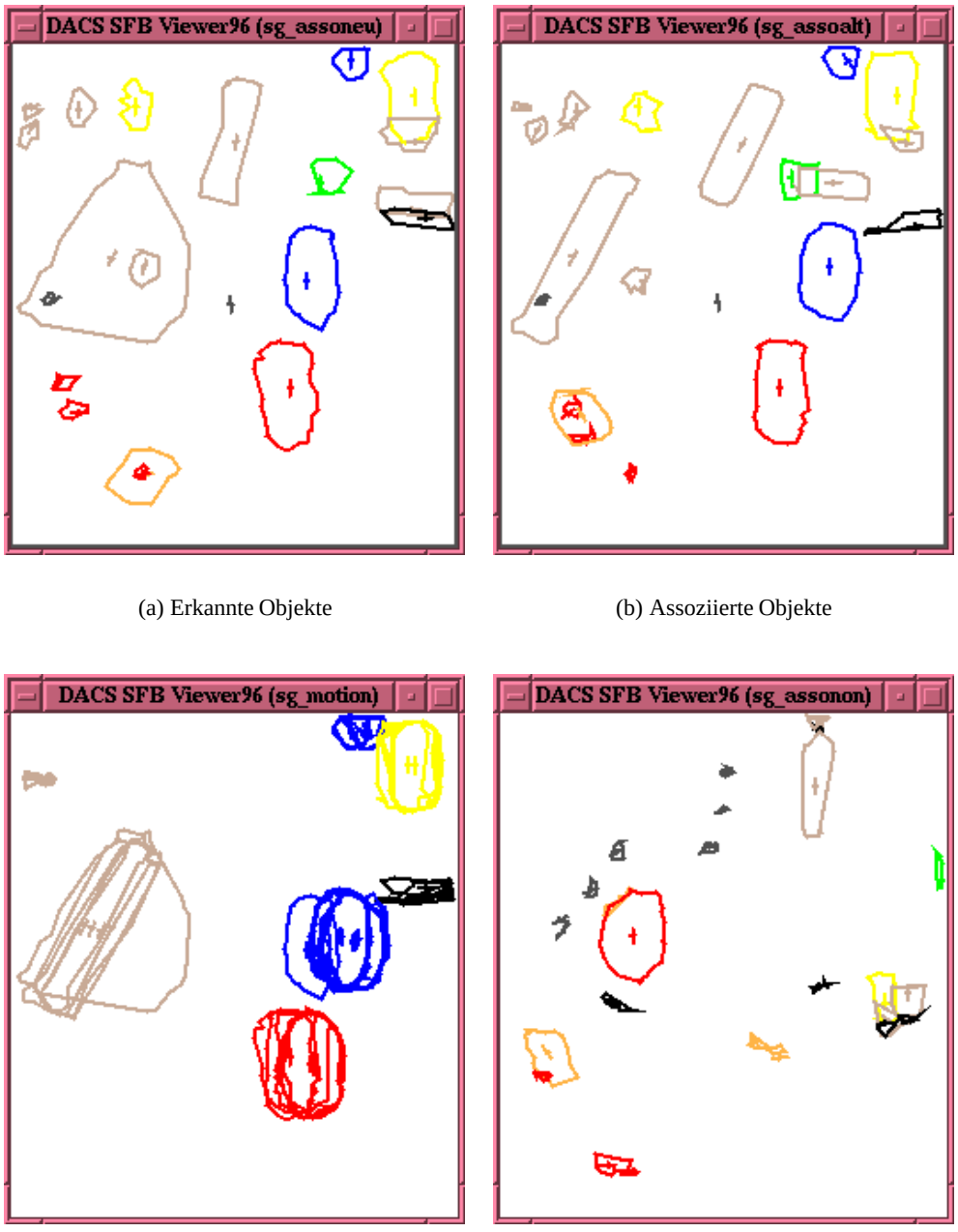
Im System ist es möglich, abstraktere Größen aus den Reizdaten neu zu berechnen. Ein wichtiges Beispiel stellt die Berechnung eines Veränderungsmaßes dar, welches auf die Veränderung in den meisten sonst stabilen Modalitäten reagieren kann. Damit ist eine Aufmerksamkeitssteuerung möglich, welche die Fokussierung von weiterverarbeitenden Programmen auf diese Bereiche der Daten lenkt.

Die Einsatzorte, die in Abschnitt 1.2 skizziert wurden, werden von dem Gedächtnismodul abgedeckt. Der Einbau in das SFB-Demo-System wurde noch nicht geleistet, da hierzu noch strukturelle Veränderungen des Gesamtsystems erfolgen müssen. Die Erläuterungen zu dieser weiteren Zielsetzung stelle ich in Abschnitt 5.2 dar.

4.4 Bewegungswahrnehmung

Aus Kamerabildern (Beispiele in Abbildungen 4.9 und 4.11) wurden, wie im SFB-Demo-System üblich, Regionendaten gewonnen, mit deren Hilfe die Leistung des Gedächtnisses dokumentiert wird. Bei Veränderungen der Szene kommt die eigentliche Fähigkeit des Gedächtnisses zur Assoziation zum tragen. Die im folgenden aufgezeigten Phänomene stellen keine Ausnahmereaktionen dar, können aber häufig nicht einfach am laufenden System provoziert werden. Die komplexe Repräsentation und nicht zu beeinflussende zeitliche Veränderungen des internen Zustandes verhindern eine deterministische Untersuchung, bei der nach dem Programmstart immer die gleichen Vorgänge ablaufen. Das Programm reagiert ohne Zufallsgenerator, durch die sich immer verändernde Schedulingreihenfolge des Systems, unterschiedlich auf den gleichen Input. Dennoch lassen sich mit etwas Geduld einige grundlegende Reaktionen beobachten, die auf den generellen Charakter der Programmarchitektur schließen lassen. Die einzelnen Beispiele zeigen einige sich ändernde Objekte. Das Programm selbst unterscheidet zwischen ursächlich verschiedenen Änderungen intern nicht, da ihm der Typ der Veränderung nicht zugänglich ist, sondern nur in bestimmten Schranken Veränderungen von Daten innerhalb der einzelnen Modalitäten wahrgenommen werden. Zum Beispiel erfordert die Unterscheidung zwischen Translation und Rotation

eine tiefergehende Analyse der sich ändernden Daten. Prinzipiell ist eine solche Analyse der Daten möglich, erweist sich aber als sehr zeitintensiv, wenig zuverlässig und für die Funktionalität des Gedächtnisses als unerheblich.



(a) Erkannte Objekte

(b) Assoziierte Objekte

(c) Veränderte Objekte

(d) Unbekannte Objekte

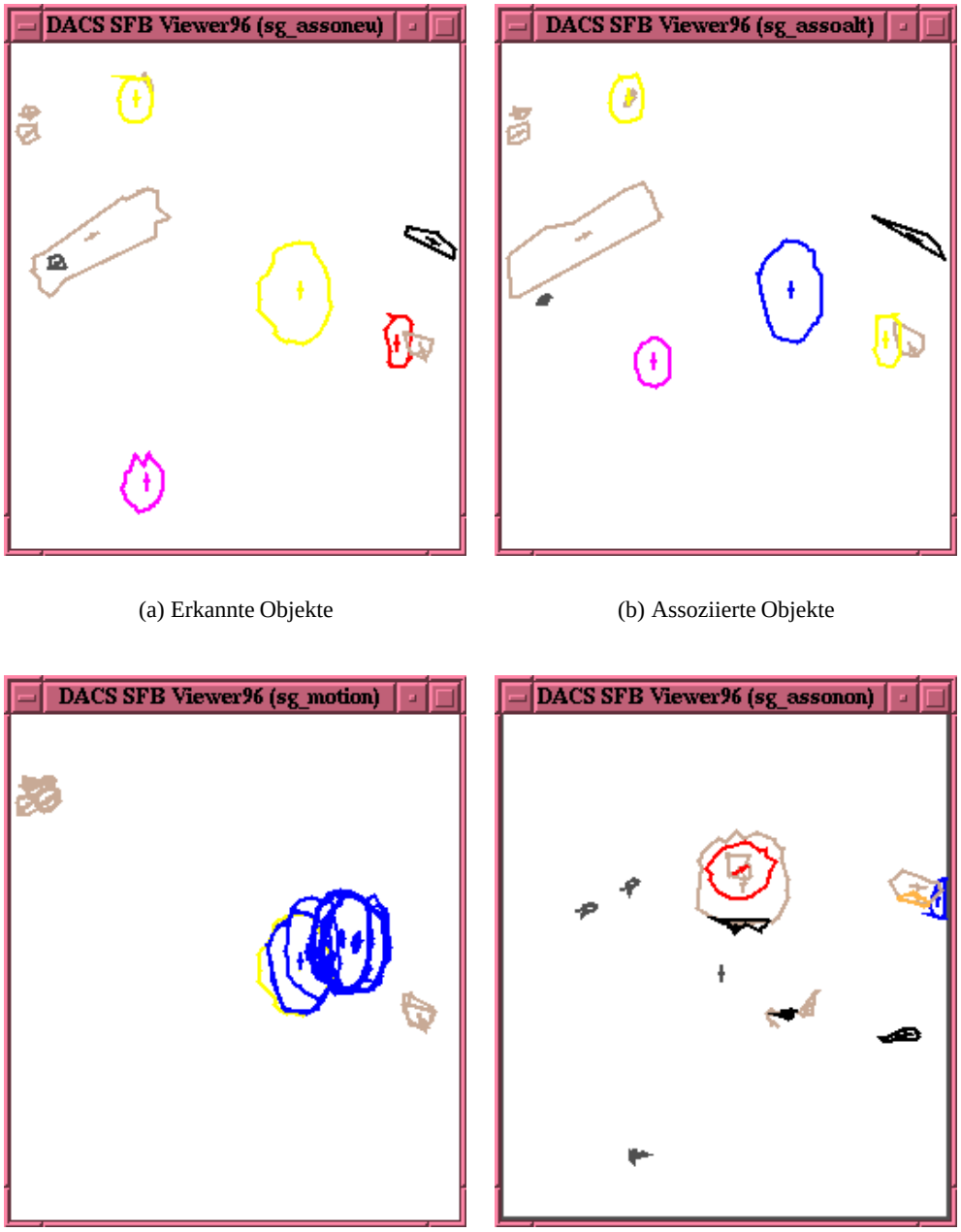
Abbildung 4.13: Phänomen Rotation

Translation

Bei der Translation ist die zurückgelegte Entfernung ausschlaggebend für die Erkennungsleistung bezüglich dieser Veränderung, weil mit zunehmender Entfernung das Abstandsmaß in der Ortsmodalität anwächst. Beispiele für erkannte Translation sind in Abbildung 4.12 dargestellt. Bei sich örtlich überlappenden Eindrücken werden in der Regel die Translationen erkannt, da in der Umriß-Modalität ein Polygonzug in Punkte zerlegt wurde und diese Punkte untereinander eine Assoziation zulassen. Daraus ergibt sich, daß überlappende Objekte — unabhängig von ihrer tatsächlichen Zusammengehörigkeit — in dieser Modalität häufig als ähnlich empfunden werden. Überlappen sich die Eindrücke nicht, so kann das System prinzipiell nicht den Unterschied zwischen dem Erscheinen mehrerer Objekte und der schnellen Bewegung eines Objektes feststellen. Dieser uns aus dem Kino und Trickfilm bekannte Effekt beruht in diesem Fall auf der Analyse von Momentaufnahmen in lokalen Bereichen. Der Assoziation steht die globale Sicht zwar zur Verfügung; sie kann aber nicht die tatsächliche Anzahl der gleichen Objekte “zählen”. Dennoch wird eine Translation auch über große Entfernungen akzeptiert, wenn das Objekt in den anderen Modalitäten eindeutig beschrieben wird. Wenn zum Beispiel das einzige rote Objekt eine Bewegung erfährt, so kann dies besser detektiert werden, als wenn eines von vielen holzfarbenen Objekten die Position gravierend ändert. Die Abbildung 4.11(b) zeigt ausgehend von Abbildung 4.9(a) die Translation einer Beilagscheibe über eine weite Strecke hinweg. Die in Abbildung 4.14 gezeigte Assoziation der entsprechenden Regionen ist möglich, führt jedoch wegen zu vieler Unterschiede zu keiner Vereinigung der Eindrücke.

Rotation

Rotationen schlagen sich in den SFB-Objekten verschiedenartig nieder. Auf den unteren Abstraktionsebenen findet zum Beispiel eine Veränderung der Form und Hauptachse statt, wohingegen auf höheren Verarbeitungsstufen die Ausrichtung eines Objektes in der 3D-Rekonstruktion berechnet wurde und in Form einer Rotationsmatrix feststeht. Interessant für die Leistungen unseres Gedächtnisses ist die Rotation auf dem Datentyp der Regionen, da sie sich hier in einer Verformung niederschlägt und gleichzeitig mehrere Modalitäten verändern kann. Je nach Typ des Objektes können einige verschiedene Formen beobachtet werden. Dreht sich zum Beispiel eine Holzleiste, so kann sich, wenn wir von Effekten am Bildrand absehen, die Hauptachse kontinuierlich drehen, da die Umrißlinie kaum einer Verformung unterworfen ist. Daher bleiben die Maßzahlen für Exzentrizität und Kompaktheit ähnlich. Betrachten wir die Drehung eines Würfels, so ändert sich dabei eher die Form; die Hauptachse des unförmigen Blobs bleibt in bestimmten Schranken. Die Erscheinung gleicht einem dicken wabernden Punkt, der nicht ohne weiteres als Würfel zu kennen ist. Das Programm verfügt auf dieser Ebene noch nicht über die Zuordnung der Würfellöcher zum Würfelkörper, mit der eine genauere Analyse des Vorgangs möglich wäre. Abbildungen 4.9(c) und (d) zeigen im oberen Bildteil eine gedrehte Dreilochleiste und eine Schraube. Die Assoziationsleistung von Abbildung 4.13(a) und (b) zeigt eine Identifizierung der entsprechenden Farbregionen. Die Detektion der Veränderung durch die zuvor durchgeführte Translation ist deutlich in Abbildung 4.13(c) zu sehen.



(a) Erkannte Objekte

(b) Assoziierte Objekte

(c) Veränderte Objekte

(d) Unbekannte Objekte

Abbildung 4.14: Phänomen Farbverwechslung

Farbverwechslung

Die Farbverwechslung ist ein Beispiel für die Detektion der Änderung eines Aufzählungstyps der SFB-Datenstrukturen. Die Aufzählungstypen werden nicht in einen virtuellen Raum projiziert, in dem einfache mathematische Funktionen das Abstandsmaß bilden, sondern werden, um ein flexibles Abstandsmaß zu erhalten, in Verwechslungsmatrizen als Zeilen bzw. Spaltenindex ver-

wendet. Die eigentliche Zahl, die eine Aufzählungsgröße repräsentiert, steht in keiner natürlichen Ordnung und definiert keine Nachbarschaftseigenschaften. Dennoch repräsentiert jede Zahl eine Eigenschaft des Eindrucks. Die Veränderung dieser Eigenschaft kann, wie in Abbildung 4.14 gezeigt, detektiert werden, wenn genügend andere Modalitäten die Ähnlichkeit von zwei Eindrücken bestätigen und eine Assoziation auch gegen einen "Fehler" in einer Modalität möglich ist. Der blaue Würfel wurde in Abbildung 4.11 gegen einen gelben Würfel ausgetauscht, um diese Farbverwechslung zu provozieren. Das Gedächtnis assoziierte die Farbregionen und vereinigte die Eindrücke. Im Demand Stream (Abbildung 4.14(c)) wird auf diese Regionen hingewiesen.

4.5 Zusammenfassung

Die Ergebnisse meiner Arbeit lassen sich wie folgt zusammenfassen: Die beschriebene Architektur und vorliegende Implementierung orientiert sich in vielerlei Hinsicht am Vorbild der Biologie und vermag Assoziationsleistungen zu erbringen. Die hohe Parallelität beruht auf dem Grundsatz des uneingeschränkten Lesezugriffs mehrerer selbständiger Prozesse auf gemeinsame Datengrundlagen. Die verschiedenen aufgenommenen Informationen werden spezifisch in dynamischen Modalitätskarten gespeichert, ohne den Aufnahmekontext zu verlieren. Die Ausgabe der Information erfolgt flexibel und unkompliziert in mehreren Demand Streams. Das Programm identifiziert in Grenzen Objekte nach einer Rotation, Translation oder Farbverwechslung und weist auf diese Veränderungen hin. Die Assoziationsfähigkeit wird durch die Verknüpfung der Ergebnisse einiger verschiedener, paralleler und voneinander getrennter Verarbeitungswege erreicht. Das Programm ist vollständig in das Prototypsystem des SFBs integrierbar und bietet neben der Stream-Verarbeitung auch Objektassoziation über RPCs an. Die Leistungen des Gedächtnisses kommen, sofern man nur die Verarbeitung eines kleinen Teils des visuellen Systems betrachtet, phänomenologisch denjenigen von einfachen Lebewesen nahe.

Kapitel 5

Ausblick

Das Programm “VDMF” kann in seiner momentanen Version im SFB-Demo-System eingesetzt werden. Es eignen sich hier besonders die Stellen in der bildverarbeitenden Hierarchie, die wenige Objekte mit komplexen Informationen enthalten. Die Repräsentation auf Gruppier-Hypothesen erfordert eine wesentlich höhere Verarbeitungsgeschwindigkeit, da sehr viele einzelne kleine Objekte für ein gesamtes Bild behandelt werden müssen. Neben den schon erbrachten Leistungen sind eine ganze Reihe von Verbesserungen möglich, die den Rahmen dieser Arbeit bei weitem gesprengt hätten. Ich möchte einige Hinweise geben, um den Weg zu solchen Erweiterungen zu ebnen.

5.1 Erweiterungen

Um die bereits entwickelte Architektur weiterhin deutlich von einer einfachen Datenbank oder ähnlichem abzugrenzen, empfehle ich die Beachtung der aufgezeigten und schon implementierten Konzepte. Bei Fragestellungen stehe ich gern zur Verfügung und freue mich, wenn dieses Programm weiterentwickelt wird. Die Erweiterungen sollten sich nicht nur auf programmtechnische Aspekte beschränken, sondern möglichst auch schriftliche Dokumentationen des Veränderten beinhalten.

Module

Die Ergänzung um Module zur Implementierung weiterer Funktionalität ist problemlos möglich. Das neue Modul sollte einen “sprechenden” Namen haben und mit in das Entwicklungsverzeichnis aufgenommen werden. Ein Eintrag im `Makerules` bindet es in den Compileraufruf mit ein. Ein entsprechendes Headerfile ist in `vdmf_include.h` einzutragen, das von dem C-File eingebunden werden sollte. Die bisherigen Programmierschnittstellen der einzelnen Module und Datentypen gehen aus den vorhandenen Files deutlich hervor und sind jeweils in einem Abschnitt “Description” dokumentiert.

Natürlich können auch Erweiterungen der bestehenden Module vorgenommen werden. Viele der im folgenden angesprochenen Ideen beziehen sich auf diesen Fall. Eine Kennzeichnung von Änderungen ist wünschenswert, damit spätere Dokumentationen speziell auf die Änderungen eingehen können.

Threads

Die Berechnung und Beobachtung von abstrakten Werten ist bisher nur beispielhaft implementiert. Sie wirkt sich noch nicht direkt auf die Assoziationsfähigkeit aus. Eine Verknüpfung auf dieser Ebene würde weitere interessante Leistungen des Programms ermöglichen. Die hierzu benötigte Rechenzeit stünde auf Mehrprozessorsystemen leicht zur Verfügung. Diese Erweiterung läßt sich prinzipiell auf zwei Wegen realisieren. Zum einen könnte die Bearbeitung, sofern sie wenig aufwendig ist, aber grundlegende Größen implementiert, in die normale Eindrucksverarbeitung nach dem Beispiel der Veränderungsdetektion aufgenommen werden. Der andere Weg ist die Implementierung eines Threads in der Art der Vergessen-Threads, der direkt und unabhängig auf den Daten des Gedächtnisses arbeitet.

Port

Das Gedächtnis kann an andere Datendefinitionen und Aufgaben angepaßt werden, indem ein neuer "Port" für die neuen Datenstrukturen definiert wird. Es ist denkbar, die Kommunikation auf ein anderes System umzustellen, sofern adäquate Kommunikationsprimitiva existieren. Weitere Anpassungen des Port-Moduls sind möglich, wenn zum Beispiel neue SFB-Objekte mit dem Gedächtnis verarbeitet werden sollen. Neue Funktionen des Typs `vdmf_port_Neu2reiz()` müssen die Übersetzung in die interne Repräsentation übernehmen. Dabei kann sich die Verarbeitung an den bestehenden Funktionen orientieren und weitgehend vorhandene Wandlungen der Basistypen benutzen.

Modalität

Die Erweiterungen im Bereich der Modalitäten versprechen eine Verbesserung der Assoziationsfähigkeiten des Gedächtnisses. Für eine neue Art von Modalität müssen Anpassungen stattfinden: Der Modalität ist eine "Nummer" im File `vdmf_enum.h` zuzuteilen. Eine der Wandlungsfunktionen im Port-Modul sollte die Wandlung in Daten dieser Modalität vorsehen. Es kann dort eine neue Abstandsfunktion eingebracht oder eine der vorhandenen Funktionen benutzt werden, wobei zu beachten ist, daß die Skalierungsfaktoren dann identisch sind und diese Abstandsmaße nicht relativ zueinander justiert werden können. Sollen in einer Modalität Reize mit mehr als drei Dimensionen verarbeitet werden, so muß der Wert `VDMF_ASSOZ_MAXDIMENSION` entsprechend erhöht und das System auf seine Funktion hin überprüft werden. Viele der Verarbeitungsschritte sind bereits auf diesen Fall vorbereitet.

Die dreidimensionale Ortsbeschreibung kann in das System mit einbezogen werden, wenn andere Applikationen im SFB diese Informationen liefern. Bisher werden Koordinaten in der Ortsmodalität nur zweidimensional gespeichert und ausgewertet. Ebenfalls wünschenswert ist die Einbeziehung der Genauigkeitsmaße in die Assoziation. Im SFB-System werden bisher nur wenige Objekte mit einer Maßzahl für Genauigkeit versehen und deshalb habe ich auf die Bearbeitung dieser Größen verzichtet.

Eine weitere Möglichkeit wäre, die verwendeten Abstandsmaße dynamisch den Wahrnehmungen anzupassen und damit die Darstellungseigenschaften der Modalitäten und die Leistungen der Assoziation zu verändern. Das System könnte durch Rückkopplung auf unbekannte Eingaberäume reagieren und Abstandsmaße so wählen, daß die Daten sinnvoll unterschieden, aber dennoch Ähnlichkeiten gefunden werden. Zur Realisierung böten sich neuronale Techniken an.

Datenorganisation

Die Datenorganisation der Modalitäten ist weiter entwickelbar, da bisher nur Kohonenkarten zur wertabhängigen Speicherung verwendet werden. Eine Verbesserung des Feld-Moduls könnte die Geschwindigkeit weiter erhöhen, wenn die Reorganisation verbessert würde und die Notwendigkeit zur Reorganisation einfacher als bisher festzustellen wäre. Für weitere Datenorganisationsformen sind im Eindrucks-Modul entsprechende Aufrufe zur Assoziation der Daten notwendig. Die vorhandenen Funktionen können als Beispiele genutzt werden, um die Aufgabe der einzelnen Abfragefunktionen zu verdeutlichen.

Tcl/Tk

Eine Erweiterung der Tcl/Tk-Oberfläche ist zum Teil ohne Änderungen am C-Code möglich. Ein Austausch des benutzten Tcl-Files könnte die Oberfläche komplett anders erscheinen lassen. Die Präsentation der Parameter ist bisher nur wenig vor der Eingabe ungültiger Werte geschützt. An einigen Stellen würden Schieberegler oder ähnliche Konstrukte die Eingabe vereinfachen.

Um den Zugriff weiterer Variablen und Parameter von Tcl/Tk aus zu ermöglichen, sind diese in das Parameter-Objekt aufzunehmen und in der Funktion `vdmf_Tcl_AppInit()` mit einem Link zu verknüpfen. Weitere Funktionen in der Art der Statistik können Informationen aus dem laufenden Programm zur Visualisierung zur Verfügung stellen. Die Erweiterung der Kohonenkarten-Visualisierung könnte zum Beispiel eine Auswahl der darzustellenden Dimensionen vorsehen.

Eine Oberfläche zum interaktiven Debugging ist mit den Ausgabefunktionen der Common-Objekte bereits vorbereitet. Das Datenformat geht deutlich aus den Funktionen hervor und eignet sich gut zum Parsen der an Tcl/Tk übergebenen Terme.

5.2 Einsatz des Gedächtnisses

Das Gedächtnismodul wurde in der vorliegenden Form noch nicht in das SFB-Demo-System integriert. Um Assoziationsleistungen beurteilen zu können und die Funktionsweise des Gedächtnisses zu demonstrieren, wurden speziell Sequenzen aufgezeichnet und mit den bestehenden Applikationen des SFBs Testdaten berechnet. Diese Testdaten wurden als Input für das Gedächtnismodul genutzt, aber der Output nur visuell beurteilt, ohne eine Wirkung auf weitere Verarbeitungsschritte zu untersuchen. Eine Verschaltung, wie sie in Abbildung 1.3 angedeutet wurde, ist in Zukunft noch vorzunehmen und in ihrer Leistungsfähigkeit zu untersuchen. Dazu muß ein Effizienzmaß so definiert werden, daß die Wirkung des Gedächtnisses beurteilt werden kann. Sicher müssen an den verschiedenen Einsatzorten spezielle Parametersätze im Zuge der Leistungsbewertung erarbeitet werden, um den Abstraktionsebenen gerecht zu werden. Ich habe auf die Untersuchungen am kompletten SFB-System verzichtet, da diese den Rahmen der Arbeit bei weitem gesprengt hätten. Nachfolgenden Arbeiten oder Projekten wünsche ich viel Erfolg beim Einsatz dieses Gedächtnismoduls.

Literaturverzeichnis

[Bad79] Alan D. Baddeley. *Die Psychologie des Gedächtnisses*. Klett – Cotta, Stuttgart, 1979.

[FJK+96] Gernot A. Fink, Nils Jungclaus, Franz Kummert, Helge Ritter und Gerhard Sagerer. A Distributed System for Integrated Speech and Image Understanding. In R. Soto, Hrsg., *International Symposium on Artifical Intelligence*, Seiten 117–126, Cancun, Mexico, 1996.

[FJRS95] Gernot A. Fink, Nils Jungclaus, Helge Ritter und Gerhard Sagerer. A Communication Framework for Heterogeneous Distributed Pattern Analysis. In V. L. Narasimhan, Hrsg., *International Conference on Algorithms and Applications for Parallel Processing*, Seiten 881–890, Brisbane, Australia, 1995. IEEE.

[GKP89] R. L. Graham, D. E. Knuth und O. Patashnik. *Concrete Mathematics*. Addison–Wesley, 1989.

[GMS94] Michel Goossens, Frank Mittelbach und Alexander Samarin. *Der LaTeXBegleiter*. Addison–Wesley (Deutschland) GmbH, Bonn – Paris, 1994.

[Gör93] Günther Görz. *Einführung in die künstliche Intelligenz*. Addison–Wesley (Deutschland) GmbH, Bonn – Paris, 1993.

[Jun96] Beiträge zum Seminar ”Paralleles und verteiltes Rechnen”, Sommersemester 96, 1996. Nils Jungclaus (Ed.), Universität Bielefeld – Technische Fakultät.

[Jun97] Nils Jungclaus. *Verteilte Systemintegration mit DACS*. Dissertation, Universität Bielefeld – Technische Fakultät, voraussichtlich 1997.

[Kla91] Herbert A. Klaeren. *Vom Problem zum Programm: eine Einführung in die Informatik*. Teubner, Stuttgart, 2. Auflage, 1991.

[Kop91] Helmut Kopka. *LaTeX: eine Einführung*. Addison–Wesley (Deutschland) GmbH, Bonn – München, 1991.

[KSJ96] Eric R. Kandel, James H. Schwartz und Thomas M. Jessell (Hrsg.). *Neurowissenschaften : Eine Einführung*. Spektrum Akademischer Verlag GmbH Heidelberg, Berlin – Oxford, 1996.

[LB96] Bil Lewis und Daniel J. Berg. *Threads Primer, A Guide to Multithreaded Programming*. Prentice Hall, California, 1996.

[Ous94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison–Wesley Publishing Company, Massachusetts, 1994.

[RMS91] Helge Ritter, Thomas Martinetz und Klaus Schulten. *Neuronale Netze: Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke*. Addison–Wesley (Deutschland) GmbH, Bonn – München, 2. Auflage, 1991.

[Sed92] Robert Sedgewick. *Algorithmen in C++*. Addison–Wesley (Deutschland) GmbH, Princeton, New Jersey, 1992.

[Str93] Bjarne Stroustrup. *C++ Programmiersprache*. Addison–Wesley (Deutschland) GmbH, Bonn, 2. Auflage, 1993.

[Tre86] A. Treismann. Features and objects in visual processing. *Sci. Am. 254(5)*, Seiten 114–126, 1986.

[Ves96] Frederic Vester. *Denken, Lernen, Vergessen*. Deutscher Taschenbuch Verlag GmbH & Co. KG, München, 23. Auflage, 1996.

[VG94] D. C. Van Essen und J. L. Gallant. Neural mechanisms of form and motion processing in primate visual systems. *Neuron 13(1)*, Seiten 1–10, 1994.

[Wel95] Brent B. Welch. *Practical programming in Tcl and Tk*. Prentice Hall PTR, New Jersey, 1995.

[Zel96] Andreas Zell. *Simulation neuronaler Netze*. Addison–Wesley (Deutschland) GmbH, Bonn – Stuttgart, 1996.

[ZR94] Karl Zilles und Gerd Rehkämper. *Funktionale Neuroanatomie: Lehrbuch und Atlas*. Springer-Verlag, Berlin, Heidelberg, New York, 2. Auflage, 1994.

Glossar

Algorithmus Ein Algorithmus ist eine Menge von Regeln für ein Verfahren, um aus gewissen *Eingabegrößen* bestimmte *Ausgabegrößen* herzuleiten, wobei die Bedingungen der Finitheit der Beschreibung, der Effektivität, der Terminiertheit und der Determiniertheit erfüllt sein müssen (nach [Kla91]).

Array In einem Array werden Elemente eines gleichen Typs zusammengestellt, auf die über einen Index zugegriffen werden. Arrays sind in den meisten Programmiersprachen grundlegende Datenstrukturen, da sie im direkten Zusammenhang mit der Speicherorganisation und der Maschinensprache der Rechnersysteme stehen. In C kann in Abhängigkeit der Größe der gespeicherten Elemente auch mit dem Index *gerechnet* werden.

Assoziation Das Verknüpfen von Erinnerungen durch zeitliche oder räumliche Nachbarschaft oder Ähnlichkeit von Einzelvorstellungen.

Client Empfänger von Information in der Kommunikation zwischen zwei Rechnersystemen (engl. Kunde). Der *Server* bietet seine Dienste dem Client an.

Condition Das Eintreten einer Bedingung oder eines Zustandes kann zur Synchronisierung von Prozessen benutzt werden. Dies ist insbesondere in der parallelen Programmierung wichtig (siehe Threads).

Cortex Bezeichnet die Großhirnrinde (Cortex cerebri). Der Cortex ist in verschiedene Bereiche zu gliedern, die vorwiegend sensorische, motorische oder *assoziative* Funktionen erfüllen.

DACS Das **D**istributed **A**pplication's **C**ommunication **S**ystem realisiert die Systemintegration und -kommunikation im gesamten SFB.

Deadlock Der Zustand, in dem *Threads* gegenseitig das weitere Ausführen des Programms behindern. Ein Deadlock kann zum Beispiel entstehen, wenn zwei Threads mit unterschiedlichen Reihenfolgen zwei *Mutex* anfordern und jeder der beiden Threads nun versucht, den anderen Mutex zu bekommen, um weiter zu arbeiten.

Demand Stream Ein Demand Stream ist ein Kommunikationsmittel des DACS. Es werden SFB-Objekte eines Typs durch einen Demand Stream von einem Programm zur Verfügung gestellt. Danach können beliebig viele andere Applikationen diesen Stream anfordern und erhalten vom *DACS* die entsprechenden Daten in Form von C-Strukturen.

Eindruck Gesamtheit der *Reize*, die als Reizmuster ein Objekt beschreiben und charakterisieren. Die einzelnen Reize treten innerhalb des Eindrucks zurück, welcher als Ganzes wahrgenommen wird. Dieser Kontext wird auch als Situation bezeichnet.

Feld Allgemein in der Informatik als *Array* bezeichnet. Speziell in diesem Programm als Objekt mit Hashzugriff verwendet (siehe *hash*).

Fokus Bezeichnet in der Physik den Brennpunkt einer Linse. Metaphorisch steht etwas im Brennpunkt, wenn wir unserer Aufmerksamkeit darauf richten. Der Fokus ist das Zentrum unseres Interesses.

GBC Die Speicherverwaltung wird zum großem Teil im **G**ar**b**age**c**ollector (engl. "Müllsammler") erledigt. Dieser eigenständige *Thread* verwaltet die dynamisch angelegten Objekte und gibt Systemresourcen frei, sobald sie nicht mehr von anderen Threads benutzt werden.

Gedächtnis bezeichnet die Fähigkeit, sich Inhalte unwillkürlich oder bewußt durch Lernen zu merken und zeitweilig zu behalten. Damit bildet sie die Grundlage für Erinnerung und Assoziation. In dieser Arbeit wird der Begriff eher als Bezeichnung des Ortes benutzt, an dem die bezeichnete Fähigkeit lokalisiert ist.

hash (engl. zerhacken) Bei dieser effizienten Methode, Daten in einem Feld zu speichern, wird das Datum zur Adreßberechnung innerhalb des Feldes benutzt (zerhackt) und die eigentliche Suche entfällt. Da für mehrere Daten eine gleiche Adresse berechnet werden könnte, müssen Strategien zur Kollisionsauflösung herangezogen werden.

holistisch eine ganzheitliche Betrachtung und Ableitung von Fakten aus dem betrachteten Bild mittels einfacher Filter und trainierter neuronaler Netze.

HTML Beschreibungssprache (**H**yper**t**ext **M**arkup **L**anguage) für vernetzte Information. Es werden im WWW mehrere Versionen vermischt und nebeneinander eingesetzt. Die beschriebenen Texte können mit einem "Browser" angeschaut werden. Der Vorteil dieser Informationspräsentation ist die Möglichkeit, Verweisen unmittelbar, ohne weitere Suche folgen zu können.

Hypothese (griech. Unterstellung) Eine für die Erklärung verschiedener Tatsachen eingeführte Annahme, die zur Ableitung neuer Tatsachen herangezogen wird. Die Hypothese eines SFB-Objektes ist eine über die Beschreibung des Objektes hinausgehende Zusammenstellung von Daten, die eine weitere Einordnung und Beschreibung des einzelnen Objektes erlaubt.

Integration Die Integration bezeichnet den Zusammenschluß von einzelnen Komponenten, um eine übergeordnete Ganzheit zu erreichen. Die Komponenten werden integriert, also dem Ganzen als Bestandteil untergeordnet.

Irix/Silicon Graphics Rechnerbetriebssystem und Herstellerfirma von professionellen Grafikworkstations. Die Systeme werden auf Grund ihrer Fähigkeiten häufig für die photorealistische Darstellung virtuelle Szenen genutzt.

Kohonenkarte Kohonenkarten stammen aus dem Bereich der künstlichen neuronalen Netze und repräsentieren durch Gewichtsvektoren topographische Merkmalskarten, welche die Eigenschaft zur Selbstorganisation besitzen.

Komplexitätsaufwand Ziel der Untersuchung der Berechnungskomplexität eines *Algorithmus* ist, zu zeigen, daß der Aufwand (die Laufzeit) für den ungünstigsten Fall eine obere und eine untere Schranke besitzt. Diese Schranken werden in der O-Notation angegeben (siehe Details in [GKP89]).

LaTeX LaTeX ist eine Zusammenstellung von Tools, die auf Laslie Lamport zurückgeht und den Umgang mit dem professionellen Satzprogramm TeX von Donald E. Knuth erleichtert. LaTeX übersetzt die angegebene logische Struktur von Kapiteln und ähnlichem im Rahmen vorgegebener Layout-Stile in entsprechende TeX-Anweisungen, die den Textsatz beschreiben.

Linux/PC Betriebssystem, das von Linus B. Torvalds in Anlehnung an UNIX für den IBM-PC entwickelt wurde. Heute wird das System von einer weltweit verteilten "Gemeinde" von Linux-Programmieren täglich weiter entwickelt und die Unterstützung neuer Hardware realisiert.

Liste Grundkonstrukt der Informatik zur dynamischen Verwaltung von einzelnen Elementen. Die einfachste Form besitzt nur die "Vorwärtsverkettung" der Elemente und erlaubt keinen Rückgriff auf vorangegangene Elemente. Die Erweiterung zu doppeltverketteten Listen erlaubt eine Verarbeitung in beiden Verkettungsrichtungen.

Loadaverage Maßzahl für die durchschnittliche Warteschlangenlänge im Systemscheduling der einzelnen Prozesse eines UNIX-Systems. Die Zahl vermittelt einen Eindruck von der Auslastung des Prozessors, da die Wartezeiten des Systems nicht in die Berechnung eingehen.

Makro Definition für einen einfachen Textersatz, der vor dem eigentlichen kompilieren durchgeführt wird. Ein Makro kann variable Teile enthalten, die erst bei der Ersetzung konkretisiert werden.

Mechanorezeptor Es gibt verschiedene Rezeptoren in der Haut, die unterschiedlich auf Druck, Berührung oder Vibration reagieren. Auf Grund ihrer Antworteigenschaften kann man die Mechanorezeptoren in langsam, schnell und extrem schnell reagierende Rezeptoren gliedern (nach [ZR94]).

Modalität Die Eigenschaft eines Objektes, deren Ausprägung durch einen Reiz beschrieben wird, nennt man Modalität. Diese Qualität wird durch spezifische Rezeptoren wahrgenommen. Die Verarbeitung von Reizen geschieht im Cortex in modalitätsspezifischen Bahnen, welche zur bewußten Empfindung verknüpft werden. Die Vorgänge dieser Verknüpfung sind weitgehend unbekannt und Gegenstand aktueller Forschung. Im Kontext dieser Arbeit bezeichnet der Begriff die Repräsentation von Objekteigenschaften im Gedächtnis und läßt sich mit den philosophischen Erkenntniskategorien vergleichen.

Mutex Ein Mutex ist eine Synchronisationsvariable, die für Daten, welche sich im gemeinsamen Zugriff mehrerer Threads befinden, "gleichzeitige" Zugriffe verhindert. Ein Thread darf eine durch Mutex geschützte Variable nur ändern, wenn er im Besitz des Mutex ist. Threads, die nicht den Mutex bekommen konnten, gehen in den "sleep" Zustand und werden wieder geweckt, wenn der Mutex freigegeben wird.

OSF/Alpha Betriebssystem und Hardwarefirma für Workstations. Der *SFB* verfügt über ein Vierprozessorsystem mit Alphaprozessoren zum Test paralleler Programme.

Pointer (engl. für Zeiger) Die Verweise zwischen C-Strukturen werden durch sog. Pointer oder auch Zeiger realisiert. Im klassischen Computer enthält die zeigende Variable eine physikalische Speicheradresse des Datums, auf das verwiesen wird.

Port (engl. auch Umschlaghafen) Soll die Übersetzung zwischen zwei Formaten symbolisieren. Das Port-Modul von *VDMF* realisiert die Schnittstelle zwischen dem SFB-Objekten und dem internen Format der *Eindrücke* und *Reize*.

Region Bezeichnet ein SFB-Objekt, das beschreibende Daten einer Farbregion eines Bildes enthält. Die Region wird durch einen umgebenden Polygonzug, eine Farbe, einen Schwerpunkt, eine Hauptachse und die Exzentrizität und Kompaktheit beschrieben.

Reiz Physikalische oder chemische Größe, deren Zustand oder Zustandsänderung auf ein Sinnesorgan (Rezeptor) einwirkt und den Grad der Erregung bestimmt. Ein Eindruck wird in

dieser Arbeit wird durch die Gesamtheit der Reize definiert, die ein Objekt beschreiben.

rezeptives Feld Das rezeptive Feld ist der Bereich, im dem ein Rezeptor durch Reize erregt werden kann. Die Größe des rezeptiven Feldes hat entscheidenden Einfluß auf die räumliche Auflösung des sensorischen Systems.

Rezeptor Sinneszelle, mit der die Außenwelt wahrgenommen wird. Der einzelne Rezeptor reagiert auf eine spezifische Art eines physikalischen oder chemischen Zustandes oder seiner Veränderung.

Ring Spezialform einer *Liste*, bei der Anfang und Ende verbunden sind. Der Ring ermöglicht eine kontinuierliche und wiederholte Verarbeitung der Listenelemente, wobei die Anzahl der Elemente jederzeit dynamisch verändert werden kann.

RPC Ein **R**emote **P**rocedure **C**all ist der Aufruf einer Funktion auf einem entfernten, über ein Datennetz erreichbarem Rechner. In der Kommunikation des *DACS* bietet eine Applikation als (*Server*) einen Dienst an. Die angebotenen Dienste verschiedener Programme können durch einen *Client* genutzt werden.

Scheduling Mit Scheduling bezeichnet man die Verteilung einer Resource auf mehrere Nutzer. Das Systemscheduling verteilt zum Beispiel die Rechenzeit des Hauptprozessors (oder auch mehrerer Prozessoren) auf die Prozesse (Programme), die auf dem System laufen.

Server Anbieter von Diensten in einem Rechnerverbund. In der Kommunikation des DACS bezeichnet der Begriff das Programm, das eine Funktion anbietet und für andere Applikationen ausführt.

SFB Kurzform für **S**onder**f**orschungs**b**ereichs 360 “Situierte Künstliche Kommunikatoren”.

Solaris/Sparc Betriebssystem und Hardwarefirma der meisten Computer der Technischen Fakultät. Das Betriebssystem unterstützt auf Kernelebene die Programmierung mit *Threads*.

Tcl/Tk Die Sprache Tcl (**t**ool **c**ommand **l**anguage) und die Sammlung von Werkzeugen Tk (**t**ool**k**it) ermöglicht eine flexible und schnelle Gestaltung eines GUI (Graphical User Interface) für das X11-Window System.

Thread (engl. Handlungsfaden) Ermöglicht eine parallele Programmierung in einer UNIX-Umgebung. Verschiedene Konstrukte (*Mutex, Condition*) ermöglichen eine Synchronisierung der einzelnen selbständigen Programmteile (siehe Programmiertechnische Hinweise in [LB96]).

VDMF Akronym für das vorliegende Programm. Es kann als **V**isual **D**ynamic **M**emory **F**unctions interpretiert werden, was soviel wie Optisch Dynamische Gedächtnisfunktionen bedeutet.

Viewer Applikation zur Darstellung der verschiedenen Demand Streams des SFB-Systems. Das Programm interpretiert alle gängigen Typen der SFB-Objekte und kann teilweise die numerischen Daten einblenden.

wissensbasiert Auf Wissen beruhende Interpretation von SFB-Objekten. Das Wissen ist etwa in der Art: “Schraube besteht aus Schraubenkopf und Gewinde” repräsentiert.

X11 Fensteroberfläche für UNIX-Systeme.

Anhang A

Programmhinweise

In diesem Anhang sind einige Details zusammengetragen, die mir die Arbeit erleichtert, andere, die diese erschwert haben. Ich hoffe mit den Informationen denjenigen, die evtl. die Arbeit fortführen wollen, eine kleine Hilfestellung geben zu können und nicht nur knapp 400 Seiten Sourcecode zu hinterlassen.

Betriebssystem-Voraussetzungen

Um das vorliegende Programm auf andere Architekturen zu portieren, muß nur eine Voraussetzung erfüllt sein: Das DACS muß auf dem System installiert sein. Das DACS benötigt genau wie mein Programm einen C-Compiler und den Threadsupport. Wenn DACS an das System angepaßt ist, existieren auch für das “VDMF” die nötigen Anpassungen.

Einzig die Unterstützung von Tcl/Tk sowie X11 mit Threadsupport für Visualisierung gehen über die Anforderungen des DACS hinaus. Die Visualisierung vereinfacht das Testen und Überwachen des Programms. Es kann aber auch ohne dieses Unterstützung betrieben werden. Bisher werden die Threadsafe-X11-Libraries auf Linux/PC-System und OSF/Alpha angeboten. Andere Versionen von Solaris und Irix bieten dieses Unterstützung schon, sind aber noch nicht bei uns installiert. In Kürze werden auch diese Betriebssysteme mit Visualisierung nutzbar sein.

Entwickelt wurde das Programm in weiten Teilen auf einem Linux 2.0.29 der Debian 1.2 Distribution. Lauffähige Versionen existieren bereits für DEC OSF/1 V3.2 und für Solaris 2.4.

Fehlerquellen und Fehlersuche

Ich bin während der Entwicklung natürlich über viele eigene Fehler gestolpert, die ich nicht im einzelnen aufführen möchte. Es gibt aber auch Fehler, die sehr schwierig zu finden sind, da ungünstige Konstellationen von installierter Software oder andere Zufälle die Finger im Spiel hatten.

Die Funktion `random()` auf meinem Linux-System ist zum Beispiel nicht zu verwenden, wenn man mit Threads arbeiten möchte. Dieser Hinweis stand aber nicht in den Man-Pages von Linux, sondern in den Man-Pages von Solaris. Die aufgetretenen Effekte waren sehr merkwürdig, da sie

sich durch Bildschirmausgaben immer von selbst auflösten und nur bei Durchläufen ohne Textausgaben ein `Segmentation Fault` auftrat. Das Debugging war daher erschwert und zum Teil unmöglich. Offensichtlich hatte die Synchronisation der threadsafe `printf`-Funktion den eigentlichen Fehler unterbunden. Ich benutze nun die Funktion `drand48()`.

In vielen anderen Fällen hatte ich mich mit den Pointern selbst ausgetrickst und fand später den unauffälligen `*`oder das Fehlen eines solchen. So lernte ich, mit Klammern bei Pointerarithmetik vorsichtiger zu sein, da eine Klammerhierachie zuviel den Typ des Pointers ändert und damit auch die Adreßberechnung in Arrays beeinflußt, denn Typ((&obj)) $\neq$ Typ(&obj).

Andere unschöne Fehler ergaben sich aus dem hohen Ziel des read ever/write once in der Threadbenutzung. Manche Operationen sind eben doch nicht atomar, so daß Zustände entstehen können, mit denen ich beim Entwurf nicht gerechnet hatte. Die Funktionen für die nächsten bzw. vorhergehenden Elemente einer Liste erhielten einen “Retry”, da hier manchmal die eine tatsächlich vorkommende Referenz von einem lesenden und einem schreibenden Thread benutzt wurde. Der schreibende Thread änderte den Pointer und der lesende Thread hatte dann Probleme, mit diesem Pointer zu operieren, da er dann keine Referenz mehr für ihn hatte. Das Objekt unterlag nicht der Kontrolle des lesenden Threads.

In Multitaskumgebungen, in der Threads auf Prozesse abgebildet werden, steigt der *Loadaverage* entsprechend der Parallelität. Dies kann zu Verwirrungen führen, wenn das Programm `xload` zur Beobachtung des Loadaverages genutzt wird. Bei ungünstigen Schedulingverfahren kann das Programm deutlich im Vorteil gegenüber Singlethread-Programmen sein. Dies liegt dann daran, daß jeder Thread als ein Prozeß die gleiche Zeit zugeteilt bekommt und andere Programme im Nachteil sein können.

Programmierhilfen

Eine große Hilfe nicht nur bei Threadproblemen war mein Betreuer Nils Jungclaus. Häufig hatte er noch einige Ideen und Tips, an die ich noch nicht gedacht hatte. Er hat die Grundlage für die Portierung auf viele Systeme mit einer Schnittstelle zu den Threads ($dthread$) gelegt und half gern bei Problemen mit seinem DACS aus. Seine Beispiele für das DACS wiesen mir den Weg und erleichterten die Anbindung an und das Verständnis für das System.

Die Idee von Nils Jungclaus, daß ein Makro “talk” neben einer Meldung auch noch die Funktion, die Sourcezeile und das Sourcefile verrät, habe ich gerne aufgegriffen und erweitert. Neben dem farbigen Output, der eine bessere Orientierung zuläßt, kann man nun den dort laufenden Thread anhand seiner internen Nummer identifizieren. Eine unterschiedliche Parameterzahl hatte ich schon im Rahmen meiner HiWi-Arbeit im SFB hinzugefügt.

Die Tcl/Tk-Anbindung habe ich am Leitfaden und an Beispielen des veröffentlichten Sourcepaketes vorgenommen. Die dort angegebene Init-Funktion wurde erweitert und meinem Programm in der dort vorgeschlagenen Weise angepaßt. Die Bücher [Wel95] von Brent B. Welch und [Ous94] von John K. Ousterhout halfen mir sehr beim Design der interaktiven Oberfläche. Ich lernte aus ihnen den Umgang mit Tcl/Tk und fand dort viele praktische Anregungen und Beispiele.

Anhang B

SFB-Format

Das im Port übersetzte SFB-Format definiert einige Datentypen, mit deren Hilfe die Kommunikation in der Bildverarbeitung des SFB realisiert wird. Normalerweise enthalten Demand-Streams eine der “Hypothesen” der Typen `Region`, `Objekt2D`, `Objekt3D` oder `Gruppier`. Die folgenden Datenstrukturen führe ich auf, um das Verständnis der Übersetzung der SFB-Objekte in Reize des Gedächtnisses zu erleichtern und einen Überblick über die gespeicherte Information in diesen Objekten zu ermöglichen.

Das Format wurde dem momentanen verwendeten SFB-Header entnommen. Die Aufzählungstypen `Teil_t`, `Farb_t`, `Konturtyp_t`, `Gruppiertyp_t`, `Kamera_t` und `Geraet_t` werden intern mit Integern gespeichert.

```
typedef struct HypothesenKopf_t
{
   char         *typ;
   char         *quelle;
   double       bewertung;
   unsigned int sec;
   unsigned int usec;
   int          sequenz_nr;
} HypothesenKopf_t;

typedef struct Punkt_t
{
   double       x;
   double       y;
} Punkt_t;

typedef struct Polar_t
{
   double       winkel;         /* ... in Radiant [-pi .. pi] */
   double       radius;
} Polar_t;

typedef struct Koordinate_t
{
   int  x;
   int  y;
} Koordinate_t;

typedef double Rotation_t[3][3];

typedef double Translation_t[3];
```

```
typedef struct KameraParam_t {
   double       f;              /* Brennweite */
   double       Cx;             /* Hauptpunkt = Schnittpunkt der optischen */
   double       Cy;             /*      Achse mit der Bildebene */
   double       sx;             /* Skalierungsfaktoren = Sensorbreite nach */
   double       sy;             /*      Grabbing in X- und Y-Richtung */
} KameraParam_t;

typedef struct Polygon_t
{
   int          n_punkte;
   Punkt_t      **punkt;
} Polygon_t;

typedef struct Kontur_t
{
   Konturtyp_t          typ;
   Punkt_t              start;
   Punkt_t              end;
   double               orientierung;
   double               laenge;
   Punkt_t              mitte;
   double               radius_a;
   double               radius_b;
   double               bogen;
} Kontur_t;

/* Bild_t hat einen Sonderstatus; keine echte Hypothese mit Basistyp */

typedef struct Bild_t
{
   HypothesenKopf_t     kopf;
   Kamera_t             kamera;
   Geraet_t             geraet;
   int                  inhalt;         /* 'veroderte' defines s.o. */
   Koordinate_t         gesamt;
   Koordinate_t         offset;

   int                  anzahl;         /* muss mit #bits in Inhalt uebereinstimmen */
   Koordinate_t         delta;
   unsigned char        ***bild;
} Bild_t;

typedef struct Region_t
{
   Punkt_t              schwerpunkt;
   Polygon_t            polygon;        /* Umrisslinie */
   Farb_t               farbe;          /* 1. und ... */
   Farb_t               farbe2;         /* .. 2. Alternative der Farbklassif. */
   int                  pixelanzahl;
   int                  umfang;         /*  # Randpixel in 4-fach Nachbarsch. */
   Polar_t              hauptachse;
   double               exzentrizitaet; /* 0: Kreis, 1: Linie */
   double               compactness;    /* 1: Quadrat <1: ausgefranster Rand */
} Region_t;

typedef struct RegionHyp_t
{
   HypothesenKopf_t     kopf;
   Kamera_t             kamera;
   Geraet_t             geraet;
   Region_t             *region;
} RegionHyp_t;
```

```
typedef struct Gruppier_t
{
   Gruppiertyp_t        typ;
   int                  n_konturen;
   Kontur_t             **kontur;
} Gruppier_t;

typedef struct GruppierHyp_t
{
   HypothesenKopf_t     kopf;
   Kamera_t             kamera;
   Geraet_t             geraet;
   Gruppier_t           *gruppier;
} GruppierHyp_t;

typedef struct Objekt2D_t
{
   Teil_t               typ;
   double               typ_bewertung;
   Farb_t               farbe;
   double               farb_bewertung;
   Punkt_t              schwerpunkt;
   Region_t             *region;        /* umschreibende Region */
   int                  n_gruppier;     /* Liste von Gruppierungshypothesen */
   Gruppier_t           **gruppier;
   int                  n_teil;         /* Liste von Teilobjekten */
   struct Objekt2D_t    **teil;
} Objekt2D_t;

typedef struct Objekt2DHyp_t
{
   HypothesenKopf_t     kopf;
   Kamera_t             kamera;
   Geraet_t             __geraet__;     /* this should NOT be used
                                         * (for compatibility only)
                                         */
   Objekt2D_t           *objekt;
} Objekt2DHyp_t;

typedef struct Objekt3D_t
{
   Teil_t               typ;
   double               typ_bewertung;
   Farb_t               farbe;
   double               farb_bewertung;
   Rotation_t           rotation;       /* Rotationsmatrix */
   Translation_t        translation;    /* Translationsvektor */
   KameraParam_t        param_links;    /* Parameter der LINKEN Kamera */
   int                  n_teil;         /* Liste von Teilobjekten */
   struct Objekt3D_t    **teil;
} Objekt3D_t;

typedef struct Objekt3DHyp_t
{
   HypothesenKopf_t     kopf;
   Kamera_t             kamera;
   Geraet_t             __geraet__;     /* this should NOT be used
                                         * (for compatibility only)
                                         */
   Objekt3D_t           *objekt;
} Objekt3DHyp_t;

typedef struct Aeusserung_t
{
   char                 *text;          /* Text(-schablone) "%O" == Objekt */
   int                  n_objekt;       /* Liste von referierten Objekten */
   Objekt2D_t           **objekt;
} Aeusserung_t;
```

Anhang C

Programmcode

Der Umfang des Sourcecodes verbietet von sich aus einen Abdruck im Anhang. Eine Suche nach bestimmten Funktionen oder Realisierungen ist in der vorliegenden elektronischen Form wesentlich einfacher. Die zu dieser Arbeit angefertigte Diskette enthält neben dem eigentlichen C-Code auch eine Aufbereitung im HTML-Format. Sie wurde mit dem hilfreichen Tool `GTAGS` von Shigio Yamaguchi `(shigio@wafu.netgate.net)` erstellt, welchem an dieser Stelle unbekannterweise gedankt sei. Neben dem Überblick über die einzelnen Funktionen und ihre Aufrufstellen werden die einzelnen Sourcefiles[1] durch Links gut überschaubar aufbereitet. Die Daten sind entsprechend auch in den SFB-Entwicklungsverzeichnissen der Technischen Fakultät im Verzeichnis `\vol\d3\src\mvonderh\vdmf` abgelegt.

[1] Leider entfallen im HTML die “folding”-Funktionen