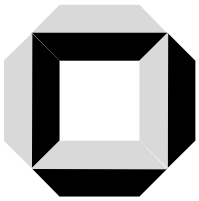


Universität Karlruhe (TH)
Fakultät für Informatik
Intititut für Prozeßrechentechnik und Robotik
Prof. Dr.-Ing. H. Wörn, Prof. Dr.-Ing. U. Rembold
und Prof. Dr.-Ing. R. Dillmann

Makrogenerierung aus Benutzervorführungen

Diplomarbeit

von Elmar Schwarz

Referent: Prof. Dr.-Ing. R. Dillmann
Korreferent: Prof. Dr.-Ing. H. Wörn
Betreuer: Dipl.-Inform. H. Friedrich

1. Februar 1998 - 31. Juli 1998

Erklärung

Hiermit erkläre ich, diese Diplomarbeit selbständig, ohne Hilfe Dritter und ohne Benutzung von Quellen oder Hilfsmitteln, welche meinen Betreuern nicht als solche bekannt sind, verfaßt zu haben.

Karlsruhe, den 31. Juli 1998

(Elmar Schwarz)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Die Programmierung von Robotern stellt in industriellen Anwendungen in mehrfacher Hinsicht ein großes Problem dar. Denn selten verfügen Personen, die ausreichendes Wissen und Fähigkeiten über die korrekte Durchführung einer Aufgabe durch einen Roboter besitzen, auch über ausreichende Programmierkenntnisse. Umfangreiches Spezialwissen zur konventionellen Programmierung eines Roboters in einer speziellen Programmiersprache für Robotikanwendungen ist oft nicht vorhanden. Selbst wenn dieser Fall gegeben ist oder zwei Experten entsprechende Kenntnisse und das benötigte Wissen austauschen können, bleiben die entstandenen Programme doch nicht frei von üblichen Programmierfehler. Diese Fehler können eine Reduktion der Qualität, Produktionsausfall oder gar eine Gefährdung der Gesundheit des Bedienpersonals nach sich ziehen. Je nachdem, ob ein Robotiker sich der Programmierung einer ihm an sich fremden Aufgabe annimmt oder sich ein Verfahrenstechniker die Programmierkenntnisse in der entsprechenden Programmiersprache aneignet, treten die möglichen Mängel unterschiedlich wahrscheinlich auf. Entweder vernachlässigt der erste wichtige Details des Herstellungsprozesses oder der zweite macht aus Mangel an Erfahrung gravierende Fehler in der technischen Programmierung.

Der Bedarf nach einem fehlertoleranten System ist offensichtlich, das von jedem Experten für eine zu lösende Aufgabe ohne Rücksicht auf dessen möglicherweise mangelnden Kenntnisse in der eigentlichen Programmierung bedient werden kann. Programming by Demonstration (PbD) bietet einen verbreiteten Ansatz, der es Programmierlaien ermöglichen soll, die benötigten Programme eigenständig und ohne Hintergrundwissen in einer Programmiersprache zu erstellen. Der Ansatz geht von einer Vorführung des Benutzers aus, in der er die gestellte manuell Aufgabe löst. Basierend auf dieser Vorführung leitet das System ein Programm ab, das dieselbe Aufgabe möglichst generell bewältigen soll.

Im besonderen liegt das Problem vor, das gewonnene Programm in einer Form zu sichern, das es sich später evtl. auch in anderen Kontexten wiederverwenden läßt. So ist es nicht nur in der Robotik üblich, daß sich ein Problem aus einer Sequenz von Teilproblemen zusammensetzt, für die möglicherweise schon Lösungen - also Programme - existieren. Ist das System in der Lage diese Teilprobleme als Makroprogramme zu repräsentieren und zu sichern, kann der Benutzer später komplexere Problemlösungen aus bereits bekannten

Komponenten zusammensetzen. Auf diese Weise wird es dem ungeschulten Benutzer auch möglich komplexere Aufgaben zu programmieren.

1.2 Robot Programming by Demonstration

Im Robot Programming by Demonstration (RPD) spielt Modularisierung eine besondere Rolle. Physische Vorführungen durch den Experten vermitteln dem System in der Regel eine Fülle von Information wie Positionen, Kraft- und Geschwindigkeitsmomente. Von diesen ist häufig nur ein relativ kleiner Ausschnitt relevant, während viele andere Komponenten der Vorführung redundant sind bzw. generalisiert und an veränderte Umwelten angepaßt werden müssen. Dies erfordert einerseits eine genaue Selektion der relevanten Informationen, aber gleichzeitig auch eine angemessene Generalisierung der Vorführung sowie evtl. eine Korrektur fehlerhafter oder unzulänglicher Sensordaten. Dies läßt es unzumutbar erscheinen, komplexere Vorgänge, die längere und möglicherweise sich analog wiederholende Sequenzen enthalten vollständig physikalisch vorzuführen. Dieses Vorgehen würde sowohl bei der Vorführung als auch in der notwendigen voll- oder halbautomatische Nachbereitung der Daten einen zu hohen Aufwand darstellten.

Demnach gliedert sich die Programmierung einer Bearbeitungssequenz für einen Roboter logisch in zwei Stufen. In der ersten Stufe werden die physischen Informationen einer Vorführung ausgewertet und gewichtet. Diese für Mensch und Maschine relativ komplexe Aufgabe, aus der großen Informationsfülle relevante Daten herauszufiltern, läßt sich effizienter gestalten, indem es ermöglicht wird, auf bereits vorhandene Teillösungen zurückzugreifen. Eine Vorführung findet dann nicht mehr physisch statt, sondern besteht lediglich aus einer Anordnung bereits vorhandener Teillösungen, die in Form von generischen Makroprogrammen zur Verfügung stehen.

1.3 Problemstellung

Damit stellt sich das Problem auf Basis der Analyse einer physischen Demonstration ein generalisiertes Makroprogramm zu erstellen, das möglichst auch in anderen Kontexten verwendbar sein sollte. Dieses Programm sollte weitgehend selbstständig von dem System erzeugt werden, so daß die notwendige Interaktion mit dem Benutzer auf ein Minimum beschränkt wird. Um den Aufwand und das benötigte Hintergrundwissen des Benutzer zu minimieren, sollte das System weitgehend vollautomatisch arbeiten und dem Benutzer lediglich im Falle falscher Ableitungen eine Möglichkeit zum Eingriff lassen. Wesentliche Komponenten eines Programmes, wie die Verzweigungslogik, Übergabe und Verteilung von Parametern und deren Absicherung gegenüber menschlichen Fehleingaben sollten weitestgehend autonom von dem System durchgeführt werden.

In der vorliegenden Aufgabe wurde das Problem der Generierung von Makroprogrammen dadurch komplexer, daß nicht nur physische Vorführungen als Grundlagen von Makroprogrammen dienen sollten, sondern auch solche, die nur noch aus abstrakten Vorführungen in Form von bereits existierenden Makrooperatoren bestehen. Dabei sollte möglichst in beiden Fällen auf identische Mechanismen zurückgegriffen werden, um homogene Ergebnisse der Makrogenerierung und eine einheitliche Benutzerschnittstellt bieten zu können.

Inhalt der vorliegenden Arbeit war es also, Mechanismen zu entwickeln und zu implementieren, die es möglich machen sollten mit möglichst geringem Aufwand an Benutzerinteraktion Makroprogramme aus physischen bzw. abstrakten Vorführungen zu erzeugen und zur weiteren Nutzung bereitzustellen.

1.4 Aufbau der Arbeit

Nach dieser Einleitung wird zunächst ein kurzer Überblick über wichtige Methoden maschinellen Lernens (siehe Kapitel 2) gegeben, soweit sie für die vorliegende Arbeit relevant sind. Gleichzeitig wird auf den Stand der Forschung auf dem Gebiet des Programming by Demonstration in der Robotik eingegangen (siehe Abschnitt 2.2). Darauf folgt eine ausführliche Motivation des Themas der Arbeit und deren Zuordnung zu dem Gesamtprojekt (siehe Kapitel 3). Der theoretische Hintergrund der Arbeit im sich anschließenden Abschnitt wird von einer detaillierten Abstraktion der Problematik (siehe Abschnitt 4.1) eingeleitet und beschreibt schließlich die theoretischen und inhaltlichen Zusammenhänge der einzelnen Komponenten und Arbeitsschritte des Systems (siehe Kapitel 4). In Kapitel 5 werden dann die zur Implementierung gewählten Paradigmen und Werkzeuge vorgestellt. In einer experimentellen Evaluation wird die Performanz des Systems anhand eines realen Beispiels demonstriert (siehe Kapitel 6). Im letzten Abschnitt der Arbeit wird schließlich ein Bewertung der Arbeit in Hinblick auf das Gesamtsystem gegeben und es werden mögliche Ansätze dargestellt, die Arbeit konstruktiv fortzusetzen (siehe Kapitel 7).

Kapitel 2

Stand der Forschung

Die vorliegende Arbeit beschäftigte sich vornehmlich mit verbesserten Realisierungen des Robot Programming by Demonstration (RPD) Paradigmas, einer Unterdisziplin des Programming by Demonstration (PbD). Hierzu wurden einige bewährte Verfahren maschinellen Lernens verwendet sowie einige neue Ansätze zur Steigerung der Effizienz entworfen.

Dieses Kapitel enthält deswegen wesentliche Grundlagen des PbD, einen kurzen Einblick in den wissenschaftlichen Stand verwendeter Verfahren sowie die theoretischen Grundlagen der Neuentwicklungen.

2.1 Maschinelles Lernen

Die vorliegende Arbeit baut in starkem Maße auf unterschiedliche Ansätze aus dem Bereich des maschinellen Lernens auf. Diese sollen hier erläutert und die Auswahl der verwendeten Verfahren motiviert werden.

2.1.1 Grundlagen des maschinellen Lernens (ML)

Maschinelles Lernen zeichnet sich dadurch aus, daß ein System maschinellen Lernens dazu in der Lagen ist, autonom oder halbautonom seine Fähigkeiten zur Lösung von Problemen zu verbessern bzw. zu erweitern. Dieses geschieht durch den *Erwerb von Fähigkeiten* (engl. *Skill Acquisition*), *Erwerb von Wissen* (engl. *Knowledge Acquisition*) und *Verbesserung bzw. Adaptierung von Wissen* (engl. *Knowledge Refinement* oder *Knowledge Adaptation*). Klassisch lassen sich verschiedene Dimensionen angeben nach denen sich maschinelles Lernen kategorisieren läßt. Als wichtigste Kategorien seien die folgenden genannt:

- **Symbolisches** vs. **Subsymbolisches Lernen**:
  - **Symbolisches Lernen.** Symbolisches Lernen zeichnet sich dadurch aus, daß Wissen in Form von *Relationen* bzw. *Regeln* abgebildet wird. Wissen wird also auf gewisse semantische Konstrukte abgebildet, die konkretes Wissen enthalten bzw. sich aus ihnen konkretes Wissen ableiten läßt. Lernen besteht hier darin, daß neue semantische Einheiten, also Relationen oder Regeln, dem System

hinzugefügt werden. Der Lernprozeß erfolgt vorwiegend *inzidentell*. Die Fähigkeiten des Systems verbessern sich also sprunghaft zu einem bestimmten Augenblick, zu dem die Erweiterung des Wissens erworben wurde. Dieses Alles-oder-Nichts-Prinzip beim Wissenserwerb, kann durchaus ein Problem darstellen, da falsches neuerworbenes Wissen vom System anstandslos und gleichberechtigt unmittelbar Anwendung findet und zu Fehlverhalten des Systems führen kann. Klassische Gebiete symbolischen Lernens sind intelligente regelbasierte Systeme oder semantische Netze.

  - **Subsymbolisches Lernen.** Subsymbolische Lernverfahren repräsentieren ihr Wissen oft verteilt über eine große Anzahl sogenannter Parameter oder über deren Anordnung (*Architektur*). Aus einem konkreten Parameter läßt sich oft kein konkreter semantischer Gehalt ableiten. Die Fähigkeit bzw. das Wissen des Systems manifestiert sich erst durch das Zusammenwirken der einzelnen Parameter. Lernen besteht in der Adaptierung von Parametern, etwa in Form besonderer Statistiken bzw. in der Reorganisation ihrer Struktur. Der Lernprozeß ist hier über einen längeren Zeitraum verteilt. Die Fähigkeiten des Systems verbessern sich *inkrementell* mit jedem Lernschritt. Oft läßt sich gar nicht genau bestimmen, ob das System seine Aufgabe schon beherrscht, manchmal fehlen sogar Kriterien bzw. es ist nicht objektivierbar, wie eine korrekte Beherrschung der Aufgabe gestaltet sein müßte. Dadurch daß sich den Parametern bzw. ihrer Struktur keine konkrete Bedeutung zuschreiben läßt, kann man letztlich auch nicht bzw. nur sehr schwer kontrollieren, ob das erworbene Wissen korrekt ist, was ein großes Problem darstellt. Oft hilft nur, die Performanz des Systems zu beobachten und daraus abzuleiten, ob das System seinen Aufgaben gewachsen ist, also brauchbares Wissen erworben hat. Typische Vertreter bilden klassische statistische bzw. stochastische Verfahren aber auch neuronale Netze.
  - **Mischverfahren.** In modernen Systemen werden beide Verfahren oft kombiniert. So werden etwa subsymbolische Verfahren eingesetzt, um symbolisches Wissen und Fähigkeiten zu bewerten, und eine in diesem Sinne optimale Hypothese über neue Relationen oder Regeln ableiten zu können.

- **Überwachtes** vs. **Unüberwachtes Lernen**
  - **Überwachtes Lernen.** Beim Überwachten Lernen liegt eine gewisse Zielvorgabe durch einen Lehrer vor; d.h. das System erhält nach jedem Lernschritt oder jeder Aktion Rückmeldung darüber, ob das erworbene Wissen korrekt oder inkorrekt ist, bzw. sich Leistung des Systems verbessert oder verschlechtert hat. Oft wird das erworbene Wissen durch *Kommentierungen* des Lehrer noch verbessert oder verfeinert. Lernen entsteht hier aus dem Zusammenspiel zwischen Lehrer und System. Jeder Lernschritt ist essentiell mit einer wertenden Interaktion zwischen Lehrer und System verbunden. Auch wenn der *Lehrer* nicht immer ein menschliches Wesen sein muß, ist diese Art der Wissensvermittlung sehr aufwendig, da im Prinzip eine Instanz existieren muß, welche die zu erlernende Aufgabe quasi allwissend und optimal beherrscht. Diese Situation ist in den seltensten Fällen gegeben.

  - **Unüberwachtes Lernen.** Unüberwachtes Lernen hingegen zeichnet sich dadurch aus, daß das System ohne jedes Feedback durch einen Lehrer neues Wissen erwirbt und anwendet. Das Lernen findet hier sogar oft vollkommen im Hintergrund statt, ohne daß der Benutzer des lernenden Systems Kenntnis davon nimmt. Zwar muß der Benutzer durchaus noch dem System Lernmaterial, etwa in Form von Trainingsbeispielen, geben, es ist aber nicht nötig diese weiter zu kommentieren. Der Vorteil der Methode liegt klar auf der Hand: Es ist sozusagen hinreichend das System mit den zu lösenden Problemen zu konfrontieren, die Lösung dazu wird vollkommen selbstständig generiert. Der Nachteil solcher Methoden ist ebenfalls offensichtlich: Es ist oft nur schwer kontrollierbar, ob das System in der Tat *sinnvolles* Wissen erwirbt, und zusätzlich, ob dieses Wissen korrekt ist. Typische Anwendungsgebiete liegen im Beobachtungslernen im symbolischen Bereich oder beim Erlernen automatischer Klassifikatoren etwa durch neuronale Netze auf der subsymbolischen Ebene.
  - **Mischformen.** Es ist offensichtlich, daß sich überwachtes und unüberwachtes Lernen gut miteinander kombinieren lassen, in der Form, daß ein Teil des Systems unüberwacht arbeitet, aber an den Schlüsselstellen des Wissenserwerb dem Benutzer die Möglichkeit zur Kommentierung oder zum Verwerfen bestimmter Hypothesen gegeben wird. Häufig kommen aber auch Verfahren zum Einsatz, die sich ihr Feedback selbst generieren und die sich iterativ optimieren (etwa Erwartungsmaximierung).[1]

- **Induktives** vs. **Deduktives Lernen**:[2]

  - **Induktives Lernen.** Induktives Lernen zielt auf die Ableitung allgemeingültigen Wissens aus einem oder mehreren Beispielen. Dies geschieht in der Regel durch *Generalisierung* der Beispiele. Da das Hauptziel dieser Lernverfahren das Erzeugen neuer Konstrukte ist, spricht man auch von *Synthetischen Lernverfahren*. Klassische induktive Verfahren werden etwa im Bereich des *Data Mining* angewandt.
  - **Deduktives Lernen.** Deduktives Lernen verfolgt genau das umgekehrte Ziel. Mithilfe genereller Regeln wird ein Beispiel deduktiv analysiert, so daß es effizient verarbeitet werden kann. Die eigentliche Wissensbasis, die der Analyse dient, wird dabei klassisch nicht modifiziert, weswegen genaugenommen also eigentlich gar kein neues Wissen erworben wird. Ein typisches Beispiel für rein deduktive Verfahren ist *Erklärungsbasiertes Lernen* auf das in Abschnitt 2.1.3 noch genauer eingegangen werden wird.

[1] Die genannten Beispiele werden oft auch zu den rein unüberwachten Verfahren gezählt, weil zu Beginn des Lernens keinerlei Vorgaben existieren, welche Lösungen korrekt sind. Betrachtet man sich die Verfahren aber im Detail, wird man feststellen, daß von einem Lernschritt zum nächsten durchaus eine ,,korrekte" Lösung impliziert wird, die den Lehrer an der jeweiligen Stelle ersetzt.

[2] Die Kategorien des Induktiven bzw. Deduktiven Lernens machen nur im Bereich der symbolischen Lernverfahren einen Sinn. Eine Anwendung der Kategorien auf nicht-symbolische Verfahren erscheint dagegen wenig sinnvoll. Manche Theoretiker ordnen subsymbolische Verfahren pauschal der induktiven Methodik zu, was in gewisser Weise auch Sinn macht, da jede Werteausprägung der Parameter auch der Synthese einer Hypothese über die optimale Werteausprägung entspricht; diese Diskussion soll hier aber nicht vertieft werden.

– **Hybride Methoden.** Auch hier ist einsichtig, daß sich beide Verfahren kombinieren lassen. Dabei werden die in der deduktiven Analyse erzeugten Erklärungsstrukturen von Beispielen dazu genutzt, die Hypothesenbildung im Induktionsschritt zu unterstützen. Typischer Vertreter einer solchen Vorgehensweise ist das analoge Lernen und Schließen (engl. *Case-Based Reasoning*, siehe 2.1.3).

2.1.2 Induktives Lernen

Induktive Lernverfahren zeichnen sich vor allem dadurch aus, daß in einem *Synthese-Schritt* neues Wissen aus einem oder mehreren Beispielen erworben wird. In diesem Schritt wird versucht aus einem oder mehreren konkreten Beispielen allgemeingültiges Wissen zu induzieren. Diese Induktion erfolgt in der Regel auf Basis einer *Generalisierung* des gegebenen Beispiels. Für die Induktion neuen Wissen auf Basis eines oder mehrerer Beispiele existieren oft mehrere *Hypothesen*, die unterschiedliche Möglichkeiten zur Generalisierung des Beispiels darstellen. Um zwischen unterschiedlichen Hypothesen diskriminieren zu können, sind sog. *Vorzugskriterien* zu finden, die es ermöglichen die ,,beste" Hypothese zu selektieren. Mögliche Vorzugkriterien sind:

- **Wähle verständlichste Hypothese.** Vom System synthetisierte Hypothesen werden danach ausgewählt, ob sie dem Benutzer verständlich sind. Dies ist sinnvoll, um dem Benutzer die Interaktion mit dem System zu erleichtern.
- **Wähle Hypothese mit geringstem Aufwand.** Es wird diejenige Hypothese gewählt, die während der Nutzung des Wissens, dem Wissensabruf bzw. -nutzung den kleinsten Berechnungaufwand verursacht.
- **Wähle einfachste Hypothese.** Man wählt die am wenigsten komplexeste Hypothese, wobei Komplexität nach verschiedenen Kriterien definiert sein kann.
- **Wähle informationstheoretisch kompakteste Lösung.** Es wird diejenige Lösung selektiert, die eine minimale Komplexität aufweist, dabei aber immer noch ein Maximum an Information konserviert (Occam's Razor). Diese Methode wird in Abschnitt 2.1.4 näher behandelt.

2.1.3 Erklärungsbasiertes Lernen und Schließen (EBL)

Die Terminologie des *Erklärungsbasierten Lernens* (engl. *Explanation-based Learning*) wurde erstmals in [DeJong 86] eingeführt. Diese Arbeit baut maßgeblich auf den Arbeiten von [Mitchell 86] auf, der das Paradigma des *Erklärungsbasierten Generalisierens* (engl. *Explanation-based Generalization*) eingeführt hat.

Das Grundproblem der Generalisierung liegt darin aus einem oder mehreren Beispielen eine semantische Beschreibung zu finden, die allen Beispielen gerecht wird, aber dennoch spezifisch genug ist, daß nur positive Beispiele der Beschreibung gerecht werden. Dementsprechend läßt sich die Aufgabe der Generalisierung auch so beschreiben, daß zwischen *beispielspezifischen* und *konzeptspezifischen* Eigenschaften der Beispiele zu unterscheiden ist. Beispielspezifische Eigenschaften werden dabei in der Generalisierung igno-

riert, während konzeptspezifische Eigenschaften Bestandteil der Generalisierung werden (siehe Abbildung 2.1).

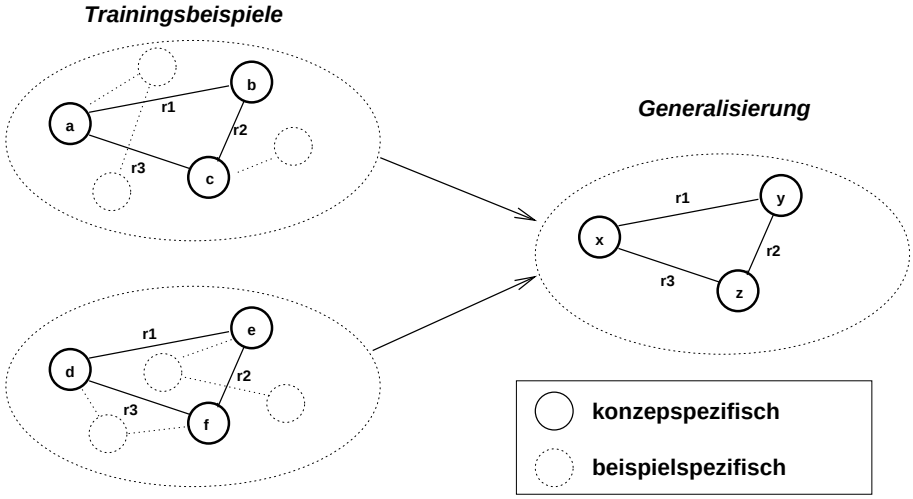


Abbildung 2.1: Generalisierung zweier Trainingsbeispiele.

Zur Generalisierung aus Beispielen werden häufig empirische Methoden verwendet, die basierend auf einer großen Menge von Beispielen versuchen gemeinsame und distinkte Merkmale der Beispiele zu extrahieren. Oft sind jedoch große Mengen von Beispielen nötig, um sinnvolle Generalisierungshypothesen zu generieren. Die ist oft nicht wünschenswert, so daß schon früh versucht wurde Hintergrundwissen zu involvieren, das eine sinnvolle Generalisierung unterstützt. Dazu werden häufig Methoden benutzt, die Hintergrundwisse über die Ähnlichkeit von Eigenschaften oder Objekten besitzen, das die Generalisierung unterstützt (engl. Similarity-based Learning, siehe [Lebowitz 86]). Über dieses Wissen läßt sich der Raum sinnvoller Generalisierungshypothesen einschränken.

Ein Schritt darüber hinaus gehen erklärungsbasierte Methoden. Diese versuchen zunächst eine *Erklärung* eines Beispiels finden, über das Schlüsse auf relevante Eigenschaften und deren Generalisierungsmöglichkeiten gezogen werden können. [Mitchell 86] formularisierte das Verfahren mit folgender theoretischer Grundlage der Anwendung des Verfahrens:

- **Zielkonzept.** Das Zielkonzept beschreibt das zu erlernende Konstrukt, für das Beispiele gegeben werdem.
- **Beispiel.** Ein Beispiel für das Zielkonzept.
- **Bereichtstheorie.** Hintergrundwissen in Form von Regeln und Fakten, die das Beispiel auf das Zielkonzept abbilden und somit eine *Erklärungsstruktur* für das Beispiel in Bezug auf das Zielkonzept bilden.
- **Operationalitätskriterium.** Das Operationalitätskriterium stellt die unmittelbare Anwendbarkeit gefundener Erklärungen sicher. Wichtig dabei ist, das diese Anwendung direkt, ohne Anwendung weiteren Hintergrundwissens erfolgen kann, erst dann liegt eine *vollständige Erklärung* im Sinne der Bereichetheorie vor.

Basierend auf diesem theoretischen Gerüst läßt sich erklärungsbasiertes Lernen in zwei Schritte einteilen:

1. **Generierung der Erklärungsstruktur.** Durch Anwendung der Regeln der Bereichstheorie wird rekursiv ein Ableitungsbaum erzeugt, bis es gelungen ist, das Zielkonzept abzuleiten. Die durch das rekursive Anwenden der Bereichstheorie erzeugte Struktur bildet eine Erklärung des Beispiels in Bezug auf das Zielkonzept.

2. **Generalisierung des Erklärungsstruktur.** Ist die Erklärungsstruktur einmal gefunden, läßt sich anhand der Struktur ablesbar, welche Eigenschaften des Trainingsbeispiels zur Ableitung der Erklärung notwendig waren. Diese müssen dann allgemein immer für das Zielkonzept gelten, während nicht involvierte Eigenschaften lediglich beispielspezifisch waren.

- **Zielkonzept**: Objektpaare (x,y), für die gilt SAFE-TO-STACK(x,y), wobei gilt: SAFE-TO-STACK(x,y) $\Leftrightarrow$ NOT(FRAGILE(y)) $\vee$ LIGHTER(x,y)

- **Trainingsbeispiel**:
  ON(OBJ1,OBJ2)
  ISA(OBJ1,BOX)
  ISA(OBJ2,ENDTABLE)
  COLOR(OBJ1,RED)
  COLOR(OBJ2,BLUE)
  VOLUME(OBJ1,1)
  DENSITY(OBJ1,0.1)
  $\vdots$

- **Bereichstheorie**:
  VOLUME(p,v) & DENSITY(p,d) $\Rightarrow$ WEIGHT(p,v*d)
  WEIGHT(p1,w1) & WEIGHT(p2,w2) & LESS(w1,w2) $\Rightarrow$ LIGHTER(p1,p2)
  ISA(p,ENDTABLE) $\Rightarrow$ WEIGHT(p,5)
  LESS(.1,5)
  $\vdots$

- **Operationalitätskriterium**: Das Zielkonzept muß in Termen der Beschreibung der Beispiele oder einfach auswertbarer Prädikate aus der Bereichstheorie (z.B. LESS(.1,5)) beschrieben sein.

Abbildung 2.2: Beispiel für EBG

Die Abbildung 2.2 zeigt ein einfaches Beispiel einer Bereichstheorie, eines Trainingbeispiels und des zugehörigen Operationalitätskriteriums.[3] Das Zielkonzept, auf welches

[3]Die Abbildungen 2.2, 2.3 und 2.4 wurden aus [Riepp 97] entnommen.

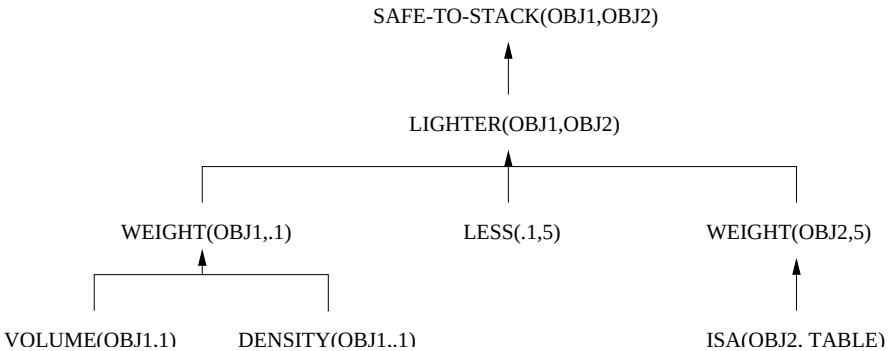


Abbildung 2.3: Erklärungsstruktur im EBL

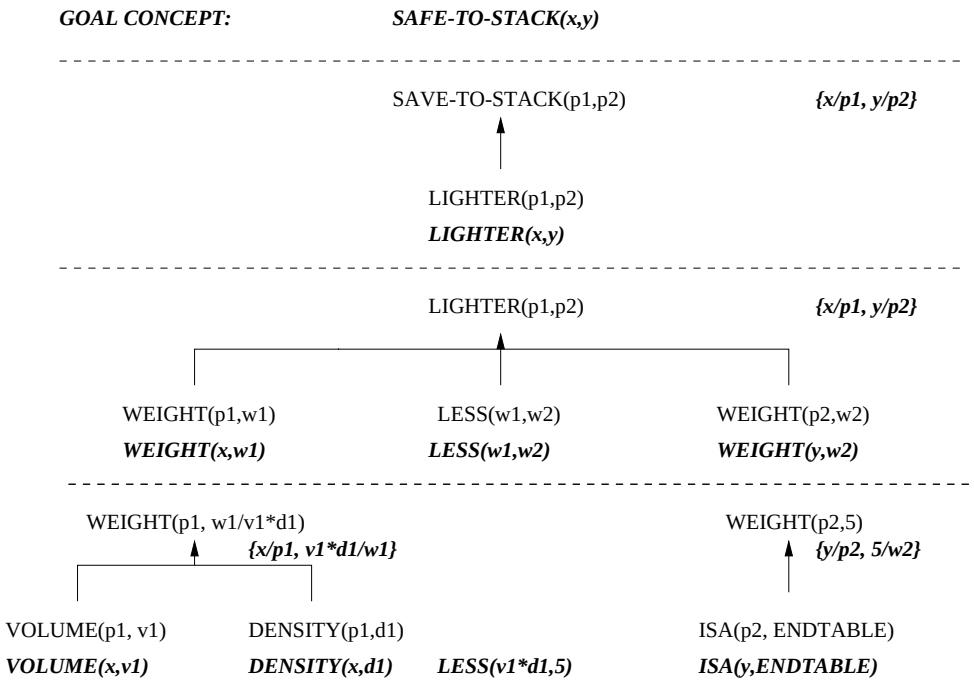


Abbildung 2.4: Generalisierte Erklärungsstruktur im EBL

das Trainingsbeispiel abgebildet werden soll, ist die *SAVE-TO-STACK*-Relation. Abbildung 2.3 zeigt die auf der Bereichstheorie aufbauende Abbildung des Trainingsbeispiels in Form einer Erklärungsstruktur. Diese Struktur wird dann systematisch generalisiert (siehe Abbildung 2.4): Alle Fakten des Trainingsbeispiels, die nicht zur Erklärung benötigt werden, fallen aus der Generalisierung als beispielspezifisch heraus. Die verbleibenden Relationen legen fest, welche Parameter *fest* und welche *frei* sind. *Feste Parameter* werden bereits durch die generalisierte Erklärungsstruktur festgelegt, *freie Parameter* werden in der Erklärungsstruktur von unten nach oben propagiert und schließlich zu den freien Parametern des Zielkonzeptes *SAVE-TO-STACK*.

Es ist einsichtig, daß auch diese Methode keine eindeutige Generalisierung zuläßt, da je nach Komplexität der Bereichstheorie oft mehrere Erklärungen möglich sind, wodurch auch unterschiedliche Generalisierungen abgeleitet werden können. Dadurch kann es mitunter sinnvoll sein, auch hier die sinnvollste Erklärungshypothese mit anderen Methoden zu diskrimieren (siehe Abschnitt 2.1.2).

Erklärungsbasiertes Lernen stellt damit eine Mischform zwischen induktivem und deduktivem Lernen dar, da zunächst im Analyse-getriebenen Erklärungsschritt Deduktion betrieben wird, die dann dazu dient, neues Wissen zu induzieren.

2.1.4 Informationstheoretische Ansätze

Grundlagen der Informationstheorie

Shannon legte in seinen Arbeiten die Grundlagen der Informationstheorie. Dabei sind Daten, die von einem *Sender* zu einem *Empfänger* durch einen *Übertragunskanal* übertragen werden, als stochastischer Prozeß aufzufassen. In der Übertragung dieses abstrakten Modelles auf das Problem des PbD implementiert der Benutzer den Sender und die kognitive Komponente des PbD-Systems den Empfänger. Der Übertragungskanal, über den die Vorführung, also die Daten vom Benutzer zum System übertragen werden, bilden die Sensoren des Systems.

Unserem mathematischen Modell der Benutzervorführung legen wir folgende Definitionen zugrunde, die an übliche Begrifflichkeiten der Sochastik angelehnt ist:

Definition 2.1 Prozeß: Ein *Prozeß* $\mathcal{P}$ ist definiert als eine Abbildung von $\mathbb{N}$ in einen endlichen Zustandsraum $\mathcal{Z}$:

$$\mathcal{P} : \mathbb{N} \rightarrow \mathcal{Z}$$

Für die Beziehung $\mathcal{P}(t) = z$, $t \in N$, $z \in \mathcal{Z}$ sagen wir auch, der Prozeß $\mathcal{P}$ ist zum Zeitpunkt t im Zustand z.

Definition 2.2 Stochastischer Prozeß: Ist über dem Zustandsraum $\mathcal{Z}$ eines Prozesses $\mathcal{P}$ zusätzlich eine zu jedem Zeitpunkt t des Prozesses eindeutig definierten Dichte $P_t : \mathcal{Z} \rightarrow [0,1] \subset \mathbb{R}$, also eine Abbildung P_t von $\mathcal{Z}$ nach $[0,1]$ aus $\mathbb{R}$ mit $\sum_{z \in \mathcal{Z}} P_t(z) = 1$ definiert, spricht man von einem *Stochastischen Prozeß* mit der *Verteilung* P. Für die *Wahrscheinlichkeit* p, daß ein Prozeß zum Zeitpunkt t sich im Zustand z befindet schreibt man $P(\mathcal{P}(t) = z)$ und meint damit $P_t(z)$. Gilt zusätzlich

$$\forall m, n \in \mathbb{N}, z \in \mathcal{Z} : P_m(z) = P_n(z)$$

so heißt der Prozeß *homogen*. $P(z)$ heißt dann die *Zustandwahrscheinlichkeit* des homogenen Prozesses.

Aufbauend auf dieser Definition können wir festlegen:

Definition 2.3 Informationsgehalt: Gegeben sei ein Stochastischer Prozeß $\mathcal{P}$ mit dem Zustandraum $\mathcal{Z}$ und einer darüber definierten *Verteilung* p. Der *Informationgehalt* I_z eines Ereignisses $z \in \mathcal{Z}$ zum Zeitpunkt t mit der Wahrscheinlichkeit $p = P(\mathcal{P}(t) = z)$ ist definiert als

$$I_z = log_2 \left(\frac{1}{p}\right). \tag{2.1}$$

Zur Bestimmung der in 2.2 definierten Zustandswahrscheinlichkeit läßt sich ein einfacher Maximum-Likelyhood-Schätzer bestimmen, der für unsere Anwendungen ausreicht und zudem erwartungstreu ist. Aus eine ausführliche Ableitung des Schätzers sowie einen Beweis seiner Erwartungstreue soll hier aber verzichtet werden:

Axiom 2.1 Schätzung der Zustandswahrscheinlichkeit. Die Zustandwahrscheinlichkeit $P(z)$ eines homogenen stochastischen Prozesses $\mathcal{P} : \mathbb{N} \to \mathcal{Z}$ läßt sich wie folgt schätzen:

$$\hat{P}(z) = \frac{\#(\mathcal{P}(t) = z)}{\sum_{z \in \mathcal{Z}} \#(\mathcal{P}(t) = z)} \tag{2.2}$$

Dabei bezeichnet $\#(\mathcal{P}(t) = z)$ die Anzahl der *beobachteten* Zustände eines Prozesses, in denen $\mathcal{P}(t) = z$ gilt, wobei t in einem beliebigen, aber endlichen und festem Intervall $[m, n]$ mit $m, n \in \mathbb{N}$ gewählt ist. Für den Grenzwert $n \to \infty$ gilt:

$$\lim_{n \to \infty} \hat{P}(z) \to P(z)$$

Der Schätzer $\hat{P}(z)$ heißt deswegen *erwartungstreu*.

Markov-Ketten

Die Theorie von Markov-Ketten verfeinert das Modell eines stochastischen Prozesses dahingehend, daß die Wahrscheinlichkeit eines Zustandes im Zustandsraum zum Zeitpunkt t lediglich von dem Zustand zum Zeitpunkt $t-1$ abhängig ist. Ist zudem die *Übergangswahrscheinlichkeit* vom Zustand zum Zeitpunkt $t-1$ zu dem zum Zeitpunkt t wohldefiniert für alle Elemente des Zustandsraumes so kann man definieren:

Definition 2.4 Übergangswahrscheinlichkeit. Gegeben sei ein homogener stochastischer Prozeß $\mathcal{P}$ mit dem Zustandsraum $\mathcal{Z}$ und einer darüber definierten Verteilung $P(z)$. Ist die bedingte Wahrscheinlichkeit

$$P\Big(\mathcal{P}(t+1) = z | \mathcal{P}(t) = y\Big) = \frac{P\Big(\mathcal{P}(t+1) = z \wedge \mathcal{P}(t) = y\Big)}{P\Big(\mathcal{P}(t) = y\Big)} \tag{2.3}$$

wohldefiniert über alle $t \in \mathbb{N}$ und alle $z, y \in \mathcal{Z}$, spricht man von der *Übergangswahrscheinlichkeit* des Prozesses zum Zeitpunkt t vom Zustand y in den Zustand z. Man schreibt auch kurz $P(z|y)$.

Mit dieser Definition können wir weiter festlegen:

Definition 2.5 Markov-Kette. Ein homogener stochastischer Prozeß $\mathcal{P}$ mit dem Zustandsraum $\mathcal{Z}$ und einer darüber definierten Verteilung $P(z)$ heißt dann eine *Markov-Kette*, wenn die Übergangswahrscheinlichkeit $P\Big(\mathcal{P}(t+1) = z|\mathcal{P}(t) = y\Big)$ für alle $y, z \in \mathcal{Z}$ wohldefiniert ist.

Auch für die Schätzung der Übergangswahrscheinlichkeit aus Definition 2.4 genügt ein Maximum-Likelyhood-Schätzer:

Axiom 2.2 Schätzung der Übergangswahrscheinlichkeit. Die Übergangswahrscheinlichkeit $P\Big(\mathcal{P}(t+1) = z|\mathcal{P}(t) = y\Big)$ einer Markov-Kette $\mathcal{P} : I\!N \to \mathcal{Z}$ läßt sich wir folgt schätzen:

$$\hat{P}\Big(\mathcal{P}(t+1) = z|\mathcal{P}(t) = y\Big) = \frac{\#\Big(\mathcal{P}(t+1) = z \wedge \mathcal{P}(t) = y\Big)}{\sum_{z\in\mathcal{Z}} \#\Big(\mathcal{P}(t+1) = z \wedge \mathcal{P}(t) = y\Big)} \tag{2.4}$$

Dabei bezeichnet analog zu Gleichung 2.2 $\#\Big(\mathcal{P}(t+1) = z \wedge \mathcal{P}(t) = y\Big)$ die Anzahl der Zustände z auf einem Beobachtungsintervall $[m, n]$ mit $m, n \in I\!N$, welche die Bedingung in den Klammern erfüllen. Für den Grenzwert $n \to \infty$ gilt:

$$\lim_{n\to\infty} \hat{P}(z|y) \to P(z|y)$$

Der Schätzer $\hat{P}(z|y)$ heißt deswegen *erwartungstreu*.

Schränkt man den Zustandsraum auf zwei Zustände, etwa $\{true, false\}$, ein, so läßt sich eine drastische Reduktion der freien Parameter erreichen; denn unter diesen Umständen ist die Übergangswahrscheinlichkeit zwischen beiden Zuständen zwangsläufig gleich. Da dieser Umstand unmittelbar einleuchtend ist formulieren wir:

Axiom 2.3 Axiom über zweielementige Zustandsräume. Gegeben sei ein Markov-Prozeß mit einem zweielementigen Zustandsraum $\{z_1, z_2\}$. Dann gilt für die Wahrscheinlichkeiten der Beobachtung eines Zustandsübergangs von z_1 nach z_2, $P(\mathcal{P}(t+1) = z_2|\mathcal{P}(t) = z_1) \cdot P(\mathcal{P}(t) = z_1)$, bzw. von z_2 nach z_1, $P(\mathcal{P}(t+1) = z_1|\mathcal{P}(t) = z_2) \cdot P(\mathcal{P}(t) = z_2)$:

$$P(\mathcal{P}(t+1) = z_2|\mathcal{P}(t) = z_1) \cdot P(\mathcal{P}(t) = z_1) = P(\mathcal{P}(t+1) = z_1|\mathcal{P}(t) = z_2) \cdot P(\mathcal{P}(t) = z_2) \tag{2.5}$$

2.2 Programming by Demonstration (PbD)

Programming by Demonstration (PbD) ist eine relativ junge Disziplin maschinellen Lernens. Die vorliegende Arbeit beschäftige sich intensiv mit der Frage, wie sich PbD in der Robotik mithilfe von maschinellen Lernverfahren optimieren und möglichst vollautomatisch gestalten läßt. Eine Einführung in Programming by Demonstration gibt [Cypher 93] bzw. [Bocionek 94].

2.2.1 Paradigma des PbD

Die Grundidee des Paradigmas liegt darin, daß der Benutzer ,,Programm" nicht wie herkömmlich in einer Programmiersprache verfaßt, sondern vielmehr durch Vorführung des auszuführenden Programms, dem PbD-System die Möglichkeit gibt, das Programm selbstständig aus der Vorführung zu abstrahieren.

Das Anforderungsniveau und die die Abstraktion des Programmes kann dabei unterschiedlich sein. Man kann prinzipiell drei Stufen unterscheiden:

1. **PbD mit direktem Nachmachen.** Diese Methode erlaubt es lediglich exakt dieselbe Sequenz wiederzugeben, die der Benutzer vorführte, ohne daß dabei über die Situation der Anwendung oder deren Inhalte generalisiert werden würde. Diese Technik erfordert kaum maschinelle Intelligenz.

2. **PbD mit generalisierendem Nachmachen.** In diesem Fall versucht das System Generalisierungen in Bezug auf die Instantiierung der manipulierten Objekte sowie verschiedener Ausführungsparameter zu finden, um den Anwendungsbereich des erzeugten Programms zu erweitern. Zu diesem Zweck bedarf es einer gewissen semantischen Interpretation der Umwelt als kodiertes Wissen darüber, welche Generalisierungen zulässig oder plausibel sind.

3. **PbD mit abstrahierendem Nachmachen.** Hier werden nicht nur Generalisierungen über die Ausführungsparameter und die Instantiierungen der manipulierten Objekte getroffen, sondern auch die Umwelt, in der das erzeugte Programm ausgeführt wird. Zu diesem Zweck ist das Programm derart abstrakt zu formulieren, daß es auf unterschiedlichen Umgebungen ausführbar wird bzw. über dessen Ausführbarkeit entschieden werden kann.

Die Quelle der Vorführung kann dabei unterschiedlicher Natur sein. Man unterscheidet zwei Fälle:

1. **Elementare Vorführungen.** Diese operieren direkt auf der Umwelt und manipulieren Objekte in dieser. Aus diesen Vorführungen können eine semantische Struktur sowie mögliche Parameter des letztlichen Programmes direkt extrahiert werden.

2. **Ikonische Programmierung.** In diesem Fall wird nicht wirklich die Umwelt des Systems manipuliert, vielmehr werden bereits vorhandene Programme (*Operatoren*) vom Benutzer in eine Sequenz gestellt, die ein Programm repräsentiert. Hier müssen zunächst die verwendeten Operatoren instantiiert und parametriert werden, um den semantischen Inhalt ihrer Manipulation erschließen zu können.

Bei den höheren Ebenen des PbD erscheint es einsichtig, daß eine derartig komplexe Aufgabe nicht immer automatisch korrekt bzw. vollständig bewältigt werden kann, d.h. es kommt zu fehlerhafter oder unzureichender Generalisierung von manipulierten Objekten, der Umwelt oder deren Parameter. Dies läßt sich niemals vollständig ausschließen, da oftmals durch nur eine Vorführung durch den Benutzer, nicht alle Aspekte und Varianten einer Vorführung erschlossen werden können. Zu diesem Zweck bedarf es zusätzlicher Interaktion mit dem Benutzer, die in drei Ebenen unterteilt werden können:

1. **Bestätigung.** Diese Interaktion überläßt dem Benutzer lediglich die Möglichkeit eine vom System vorgeschlagene Variante eines Programmen anzunehmen oder zu verwerfen. Obwohl diese Vorgehensweise sehr unbefriedigend für den Benutzer ist, wird diese Methode durchaus häufig eingesetzt.

2. **Hypothesenauswahl.** In diesem Falle werden dem Benutzer unterschiedliche Varianten (*Hypothesen*) eines möglichen Programmes vorgeschlagen, die oft unterschiedliche Interpretationen, der Situation wiederspiegeln.

3. **Parametrierung.** Der Benutzer erhält die Möglichkeit, die Parameter eines erzeugten Programmes unterstützt durch das System zu manipulieren. Dabei sollte der Benutzer aber nicht dazu gezwungen werden, eine Art Programmiersprache zu verwenden, vielmehr sollten alle Eingaben durch eine grafische Benutzeroberfläche oder andere Komponenten einer multimodalen Schnittstelle möglich sein.

4. **Nachbearbeitung.** Der Benutzer hat die Möglichkeit das erzeugte Programm in einer Programmiersprache abgebildet nachzubearbeiten. Die läßt zwar die größte Flexibilität offen, macht aber den größten Vorteil eines PbD-Systems zunichte, auch dem Laien zugänglich zu sein.

2.2.2 Beispiele für PbD

An dieser Stelle sollen einige Anwendungsbeispiele von PbD außerhalb der Robotik gegeben werden. In diesen Systemen wird zumeist auf *Elementare Vorführungen* der Benutzer zurückgegriffen, aus denen das System im Stile eines *Assistenten* versucht Programme abzuleiten.

Graphische Benutzeroberflächen

Das System PERIDOT (siehe [Myers 90]) ermöglichte es Benutzern durch Vorführen das anbinden von Benutzeraktionen auf einer graphischen Oberfläche an bestimmte Funktionen. Darauf aufbauend ermöglicht das Nachfolgeprojekt GARNET in [Myers 93] Generalisierungen entlang einer Vererbungshierarchie von Schnittstellenelementen. In beiden Systemen sind die jeweiligen Hypothesen über Programme vom Benutzer jeweils zu bestätigen oder zu verwerfen.

Grafikeditoren

METAMOUSE (siehe [Maulsby 93]) kann seinen Funktionsumfang durch Vorführungen des Benutzers erweitern. Aus bereits vorhanden Funktionen führt der Benutzer die neue Funktionalität vor. METAMOUSE verwendet hierbei ein *Antizipationsprinzip*, bei dem der Benutzer Aktionshypothesen des Systems auswählen oder verwerfen muß.

Programm- und Makroerstellung

Eines der bekanntesten und engagiertesten Systeme in diesem Bereich ist TINKER (siehe [Cypher 91]). Es ermöglicht dem Benutzer durch Vorführung von Beispielen LISP-Funktionen zu erzeugen. Eine Funktion, die der Berechnung der Reversen einer Liste dient,

kann durch Vorführung von ein- und zweielementigen Listen mit der jeweiligen Reversen induziert werden, wobei der Benutzer Hypothesen des Systems annehmen oder verwerfen muß.

2.2.3 PbD in der Robotik

Im Fall der Robotik wurde bereits in verschiedenen Systemen versucht sich die Vorteile des Programming by Demonstration Paradigmas zu Nutze zu machen. Für die Aufgabe des PbD in der Robotik gilt im besonderen (vgl. [Kuniyoshi 94]):

- Der Benutzer ist in der Regel Experte in der vom Roboter zu durchzuführenden Aufgabe, aber verfügt nicht über entsprechende Kenntnisse in der Roboterprogrammierung.
- Kostenerwägungen spielen im Robotikbereich oft eine besondere Rolle, so daß sich maximale Flexibilität nur durch Programming by Demonstration kosteneffektiv implementieren läßt.
- Erzeugte Roboterprogramme sollten möglichst generisch und wiederverwendbar sein.

Noch mehr als in anderen Bereichen des PbD unterscheidet man im RPD zwischen den beiden unterschiedlichen Abstraktionsebenen der Vorführung: der elementaren und der ikonischen Vorführung. *Elementare Vorführungen* werden dabei in einer realen physikalischen oder einer möglichst exakt simulierten Umwelt vorgeführt. In der *ikonischen Programmierung* werden dann durch elementare Vorführung erstellte Roboterprogramme aufgegriffen und abstrakt zusammengefügt, was den Prozeß der Vorführung effizienter gestaltet.

Bei elementaren Vorführungen können Vorführungen entweder direkt durch die Extremitäten des Benutzers aber auch schon mittels technischer Effektoren und Roboter vorgenommen werden.

2.2.4 Beispiele für PbD in der Robotik

In diesem Abschnitt werden verschiede Systeme vorgestellt, die Programming by Demonstration im Bereich der Robotik anwenden. Zur systematischen Einordnung der Systeme betrachte man Tabelle 2.1. Per Definition ist eine ikonische Vorführung in einer realen

	Physikalische Vorführung	**Virtuelle Vorführung**	**Ikonische Vorführung**
Hand	LFO, APO [Tung 95], [Tso 95]	TLT	—
Roboter	ETAR, RPD (APO)	—	SKORP [Matsui 89]

Tabelle 2.1: Systematische Einordnung von PbD-Systemen in der Robotik

Umwelt nur schwer zu realisieren bzw. würde die Vorteile dieser Methode weitestgehend zunichte machen.

Es existieren in der Tat noch keine Systeme die in einer simulierten Umgebung mit ebenfalls simulierten Robotern ausgeführt werden, obwohl diese Methode deutliche Vorzüge hat: Einerseits entfällt der relativ große Aufwand zur korrekten Modellierung bzw. Abstimmung zwischen simulierter und realer Umwelt. Auf der anderen Seite lassen sich viele Fragen wie Arbeitsraumprobleme von Robotern oder Abbildungen von Griffen auf den Roboter durch diese Methode sehr elegant lösen.

Beispiele für direkte Vorführungen durch den Benutzer sind:

APO (Assembly Plan from Observation)

APO (Assembly Plan from Observation, siehe [Kang 97]) extrahiert Montagepläne aus physikalischen Benutzervorführungen. Die Lösungen werden hierbei durch Kontaktzustände zwischen Objekten repräsentiert, sog. Kontaktrelationen. Eine Manipulation wird dann als Folge von Übergängen zwischen Kontaktzuständen betrachtet. Als Sensoren kommen in diesem System Stereokameras zum Einsatz.

Aufbauend auf den Sensordaten werden in dem System drei Abstraktionsebenen extrahiert:

- **Low-level-description.** Auf dieser Abstraktionsebene werden die absoluten Positionen des Effektors, also der Hand, repräsentiert.
- **Medium-level-description.** In der zweiten Abstraktionsebene werde aus Stereobildern die Fingerstellungen extrahiert und repräsentiert.
- **High-level-description.** In der obersten Abstraktionsebene werden Griffe erkannt und Grifftypen klassifiziert.

Die Klassifikation der Grifftypen erfolgt auf Basis von aus den Fingerstellungen abgeleiteten Kontaktrelationen. Diese beschreiben die Lage und Anzahl der Kontaktstellen der Hand mit dem manipulierten Objekt. Von diesen Kontaktrelation werden die Grifftypen aus einer Hierarchie von Griffklassen abgeleitet.

In dem System kommt den Greif- und Loslaßpunkten eine entscheidende Bedeutung zu. Prinzipiell wird deshalb in dem System jede Benutzervorführung in vier zyklische Phasen eingeteilt (siehe [Kang 94b]):

- **Phase vor dem Griff.** Diese Phase geht einem Griff unmittelbar voraus. Sie wird durch eine Annäherung der Hand an das zu greifende Objekt, eine Verringerung der Geschwindigkeit der Hand durch den Raum und eine Veränderung der Fingerstellung in Form eines leichten Öffnens der Hand zum aufnehmen der Objekte erkannt.[4]

[4]Wie noch später diskutiert wird, ist diese Annahme extrem fragwürdig. Vielmehr kommt es tatsächlich zu einem Übergang der Handstellung von einer entspannten *Relaxationsstellung* zu einer an den Griff eines bestimmten Objektes angepaßten *adaptierten Fingerstellung*. Je nach Größe des Objektes und der personenspezifischen kann *Relaxationsstellung* der Finger kann diese Anpassung aber auch ein Schließen der Finger bedeuten oder im Extremfall auch keine Änderung (siehe 4.2.2).

- **Greifphase.** Mit dem Berühren eines Objektes beginnt diese Phase. Sie endet, wenn das Objekt fest gegriffen ist, die Kontaktrelationen zwischen Objekt und Hand also konstant bleiben. Die eigentliche Erkennung des Griffes basiert auf der Bestimmung eines Minimums der Geschwindigkeit der Hand in Kombination mit den Fingerstellung. Dazu läßt man die Fingerspitzen der Hand ein Polygon aufspannen und betrachtet das Volumen zwischen zwei solchen in der Benutzervorführung aufeinanderfolgenden Polygonen. Das Minimum dieses Volumens bestimmt die Position des Griffs (siehe [Kang 92]).

- **Manipulationsphase.** In der Manipulationsphase wird ein Objekt bewegt. Bewegungen eines Objektes werden in dem System ausschließlich über Änderungen von Kontaktrelationen von Objekten zur Hand bzw. auch zu anderen Objekten beschrieben. Wird etwa ein Objekt auf einen Tisch abgelegt, so bildet die Unterseite des Objekts und die Tischfläche eine Kontaktrelation. Wird das Objekt auf dem Tisch verschoben, ändert sich mit dem Kontaktpunkt des Objektes auf dem Tisch auch die Kontaktrelation zwischen Tisch und Objekt. Aber auch Änderungen des Griffes in der Hand, etwa durch Drehen eines Objekts in einer Hand, stellen Änderungen von Kontaktrelationen dar, in der der Form, daß sich die Kontaktpunkte der Hand mit dem Objekt ändern.

- **Loslaßphase.** Alle Kontakte zwischen Hand und Objekt lösen sich und die Hand entfernt sich.

Das aus dieser Analyse abgeleitete Programm besteht, wie bereits angedeutet, lediglich aus einer Sequenz von Zustandsübergängen zwischen Kontaktrelationen. Die Fingerstellungen der einzelnen Griffe werden dabei automatisch auf den Effektor des Roboters abgebildet (siehe [Kang 97]), während die eigentlichen Zustandübergänge zwischen einzelnen Kontaktrelationen als vorhanden angenommen werden. Die interessante Repräsentationsform der Vorführung einer Manipulationsaufgabe als Übergänge zwischen einzelnen Kontaktzuständen, stellt aber gleichzeitig den größten Schwachpunkt des Systems dar. Denn die Implementierung dieser Kontaktübergänge auf anderen Effektoren wird nicht durch das System erlernt, so daß letztlich $O(n^2)$ Übergänge von Kontaktrelationen für *jeden* Roboter neu zu programmieren sind, so daß Lernen in dem System nur auf einem sehr eingeschränkten Bereich stattfindet, um dann umso größeren Aufwand für konventionelle Programmierung zu hinterlassen. Des weiteren sind verschiedene Annahmen und Verfahren auf sehr idealtypische Umwelten und Aufgaben zugeschnitten (siehe Fußnote 4), so daß eine Generalisierung der Methoden, etwa auf komplexere Umwelten oder Objekte, im Hinblick auf die Berechenbarkeit und Verdeckungen, noch große Probleme aufwirft.

In einer Erweiterung des APO-Systems wird in [Onda 97] ein System vorgestellt, in dem die Vorführung mit Hilfe eine Roboterarms vorgenommen wird.

LFO (Learning from Observations)

LFO (Learning from Observations, siehe [Kuniyoshi 94]) ist in der Lage mit Hilfe einer Stereokamera Manipulationen durch den Benutzer zu erlernen und auf ähnlichen Umgebungen auszuführen. Das System ist allerdings auf eine Erkennung einfacher Zweifingergriffe und

Manipulationsaufgaben beschränkt, die zudem für das Stereokamerasystem gut sichbar ausgeführt werden müssen.

Das System ist in der Lage die Umwelt einer Manipulationsaufgabe selbstständig zu interpretieren und zu analysieren. Darauf aufbauend werden in Echtzeit alle Bewegungen der Hand und etwaige Manipulationen registriert, abstrahiert und aufgezeichnet. Ein interessanter Aspekt ist dadurch gegeben, daß auch der Plan der Benutzervorführung in Echtzeit analysiert und interpretiert wird. Zu diesem Zweck werden sogenannte *Ereignisse* zu Hilfe gezogen, auf deren Basis eine Segmentierung der Vorführung erzeugt wird. Diese *Segmentierung* stellt in gewisser Weise eine semantische Erklärung der Vorführung im mitchellschen Sinne (siehe [Mitchell 86], Abschnitt 2.1.3). Hierbei wird logisch auf unterschiedlichen Ebenen segmentiert:

1. Zunächst wird eine grobe Segmentierung anhand der Greif- und Loslaßpunkt in der Benutzervorführung definiert. Diese Einteilung in Greif- und Loslaßphasen stellen die wichtigste Segmentierungsebene des Systems dar (engl. *Pick&Place*) .

2. Auf dieser Grundeinteilung aufbauend werden verschiedene Attribute aus der Vorführung abgeleitet, wie etwa Geschwindigkeit und Richtung der Bewegung der Hand durch den Raum, aber auch Abstände und Lage von Objekten bzgl. der Hand. Diese Attribute dienen dazu eine feinere Segmentierung zu gewinnen, etwa wird Ablegen eines Objektes in *Annäherung* (engl. *Approach*), *Exaktes Bewegen* (engl. *Fine Motion*) und Verlassen (engl. *Depart*) untergliedert. Aus der gefundenen Segmentierung werden einzelne Operatoren generiert.

3. Zwischen den gebildeten Operatoren werden dann semantische Abhängigkeiten gesucht, wie etwa das Zusammenfassen von Annäherung, Exaktem Bewegen und Verlassen auf einer höheren Abstraktionsebene in ein *Ablegen* (engl. *Place*).

Bei der Ausführung des generierten Roboterprogramms ist das System in der Lage die Ausführumgebung des Programmes mit derjeniger der Vorführung zu vergleichen, wobei Anzahl und Art der Objekte identisch zur Vorführung sein müssen. Lediglich die Position der Objekte ist generalisiert.

Ähnliche Systeme wurden auch von [Tung 95] und [Tso 95] vorgestellt, die allerdings vor allem dem direkten Nachmachen der Vorführung ohne jede Form der Generalisierung dienen. Diese sollen hier aber nicht näher behandelt.

Für Systeme mit robotergestützter Vorführung lassen sich folgende Beispiele geben:

ETAR (Example-based Task Acquisition in Robots)

ETAR (Example-based Task Acquisition in Robots, siehe [Heise 92]) erzeugt aus mehreren Benutzervorführungen mit einem Roboterarm als Effektor Sequenzen von Elementaroperatoren, in die Verzweigungs- und Schleifenbedingungen eingebaut werden.

RPD (Robot Programming by Demonstration)

RPD (Robot Programming by Demonstration, siehe [Friedrich 96b]) bildet die Vorführungen eines Benutzers mit Hilfe eines Robotors auf Sequenzen von symbolischen Operaratoren ab. Die Segmentierung der einzelnen Phasen erfolgt mittels neuronaler Netze, die Greif- und Loslaßsequenzen voneinander trennen. Es werden die Operatoren *Abfahren*, *Greifen*, *Loslassen* und *Transferbewegungen* unterschieden. Die erzeugten Operatoren werden mit Vor- und Nachbedingungen versehen, über die eine Verzweigungslogik der Operatorsequenz definiert wird. Diese Vor- und Nachbedingungen werden aus Kommentierungen des Benutzers, der seine *Intention* angeben muß, generiert und die Operatorsequenz generalisiert.

Ikonische Programmierung in der Robotik: SKORP

Der Begriff *Ikonische Vorführung* wurde bereits bei der Paradigmenbeschreibung des PbD erläutert (siehe Abschnitt 2.2.1). Hierbei besteht die Vorführung aus einer Sequenz von bereits vorhandenen Roboterprogrammen bzw. Fragmenten von Programmen, aus denen ein neues, komplexeres Programm generiert wird.

Als Beispiel für diesen Ansatz soll hier das von [Archibald 93] vorgestellte System SKORP (Skill Oriented Robot Programming. In diesem System wird ein Pool elementarer kognitiver Operatoren dem Benutzer zur Verfügung gestellt, aus dem er in einer graphischen Benutzeroberfläche Programme generieren kann. Deren Operatoren und Kontrollstrukturen muß er mit der Hand parametrieren. Der Pool dieser elementaren kognitiven Operatoren (engl. *Skills*) läßt sich erweitern. Allerdings läßt es das System nicht zu durch ikonische Vorführung erstellte Programme in neuen Vorführungen zu Verwenden, um ein noch höheres Abstraktionsniveau zu erreichen, weswegen der Grad der Abstraktion einer ikonischen Vorführung starken Beschränkungen unterliegt.

Kapitel 3

Problembeschreibung

Das essentielle Problem dieser Arbeit bestand darin, aus Benutzervorführungen generalisierte Makrooperatoren zu erzeugen, die zur Ausführung auf Robotersystemen geeignet sind. Dabei sollte nicht nur über Ausführungsumwelt und -kontexte generalisiert werden, sondern auch eine interne Verzweigungslogik bestimmt werden, die redundante bzw. optionale Aktionen automatisch detektiert und Verzweigungsbedingungen erzeugt, um eine optimale Ausführung des resultierenden Roboterprogramms zu gewährleisten.

Dieser Abschnitt soll darstellen, auf welcher Grundlage die Arbeit aufsetzte, und wie sich darauf basierend die Aufgabe definierte. Dabei wird ein besonderer Wert auf die Einbettung der Arbeit in den Kontext des bestehenden Projektes gelegt. Abschließend wird eine prägnante Definition der Problemlösungsschritte formuliert sowie eine Lösungsskizze gegeben.

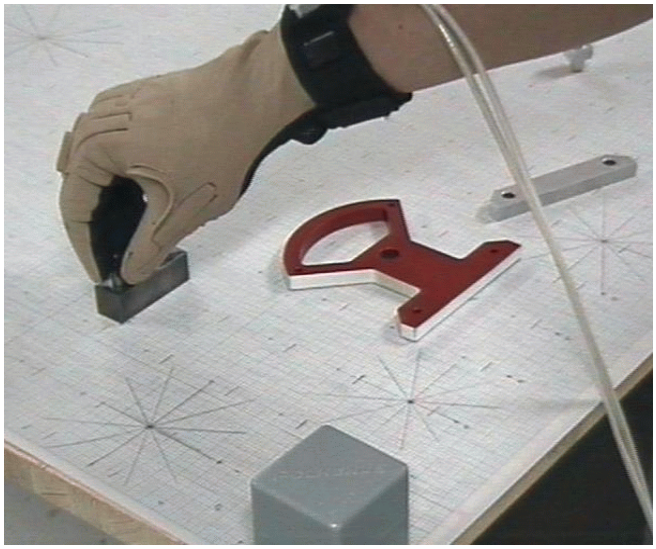
3.1 Ausgangspunkt

3.1.1 Vorarbeiten

Mit den Arbeiten von [Holle 97], [Holle 98] und [Riepp 97] wurde die Grundlage des bestehenden Systems geschaffen. Die Arbeiten ermöglichten eine interaktive Erstellung einerseits von *Elementaroperatoren* andererseits die Erstellung und Parametrierung von *Makrooperatorsequenzen.*

KaVis

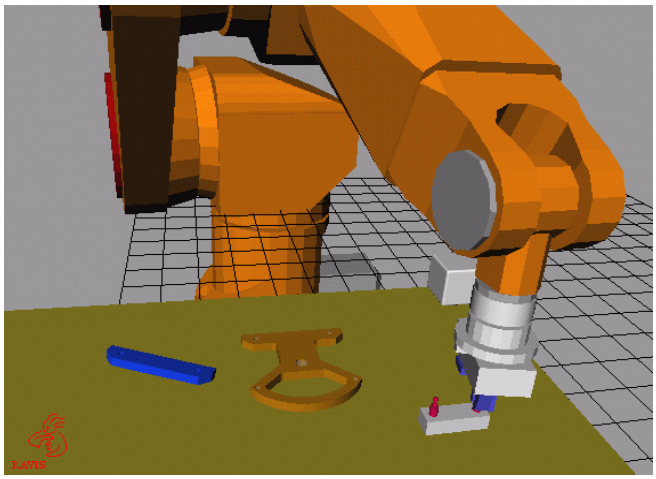
In [Holle 97] wurde das bereits bestehende Visualisierungssystem *Karlsruher Visualisierer (KaVis)* (siehe [Schaude et al.]) um die Möglichkeit erweitert, räumliche relationale Zusammenhänge zwischen den simulierten Objekten zu bestimmen. Mit KaVis stand schließlich ein mächtiges Simulationswerkzeug zur Verfügung, das es ermöglichte, Objekte im vierdimensionalen Raum zu simulieren und zu animieren, sowie deren räumliche Lage relational zu analisieren. Auf diesem Werkzeug bauten die Arbeiten von [Holle 98] und [Riepp 97] auf.



Benutzervorführung

Makrooperator
INSERT-SPACER

Makrogenese



Ausführung

Abbildung 3.1: Skizze der Aufgabenstellung der Arbeit

Interaktive Erzeugung von Operatorsequenzen

[Holle 98] implementiert eine Umgebung zur Programmierung von interaktiv erstellten Operatorsequenzen auf Basis von Benutzervorführungen. In der dort gewählten Lösung hat der Benutzer die Möglichkeit mit der Hand vorgeführte Trajektorien mit Hilfe von Kommentierung zu segmentieren und semantisch zu kommentieren. Dabei müssen die Greif- und Loslaßpunkte vom Benutzer manuell vorgegeben werden; nach den Greif- und Loslaßpunkten wird die Benutzervorführung in *Phasen* eingeteilt, die jeweils um die Greif- und Loslaßpunkte liegen und durch *Kontextwechsel* getrennt werden (siehe Abbildung 3.2). Innerhalb einer Phase bzw. eines Kontextes wird angenommen, daß sich der Effektor relativ zu einem bestimmten Objekt bewegt. Diese Grobsegmentierung muß in dem System vollständig manuell durchgeführt werden und sowohl die manipulierten Objekte als auch diejenigen Objekte, relativ zu denen sich der Effektor bewegt, sind vom Benutzer durch Kommentierung vorgegeben.

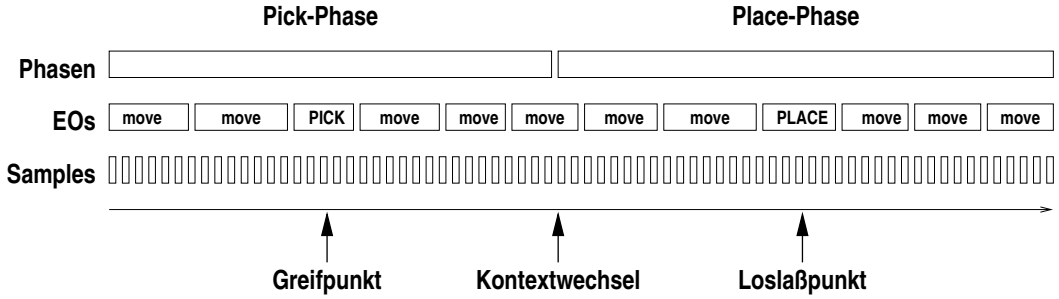


Abbildung 3.2: Segmentierungsmodell nach [Holle98]

Lediglich eine Einteilung der Trajektorie in Liniensegmente erfolgt automatisch mit Hilfe des *Iterative-Endpoint-Fit*-Verfahrens (siehe [Duda 73] und Abbildung3.2). Die gefundenen Liniensegmente werden dann auf *Elementaroperatoren (EOs)* abgebildet (siehe Abbildung 3.3)[1]. Bewegungen werden anhand von *Pfadlisten* repräsentiert, die eine Liste von Objekttransformationen (siehe 4.3) und der Bezugsobjekte, auf die sich die Objekttransformationen jeweils beziehen enthält. Die Trajektorie wird also bereits über die Objektpositionen generalisiert und relativ zu Objekten angegeben. Die manipulierten Objekte bzw. die Objekte, relativ zu denen Bewegungen stattfinden, sind freie Parameter der einzelnen Elementaroperatoren, so daß die Sequenz prinzipiell auch auf anderen Umwelten ausführbar wäre.

Die zwischen den Phasen liegenden Greifoperationen lassen sich durch den Benutzer mit *Greifkontexten* und *Loslaßkontexten* kommentieren. Diese Kontexte spezifizieren generalisierte Bedingungen der Ausführumgebung, unter denen die vorgeführte Sequenz ausführbar ist. Auch diese Kommentierung erfolgt in dem von [Holle 98] vorgestellten System vollständig manuell, allerdings mit Unterstützung der Simulation.

Als Ausgabe erzeugt das System eine generalisierte Sequenz von Elementaroperatoren, die mit den Objekten der Vorführumwelt instantiiert ist.

[1] Abbildung 3.3 wurde entnommen aus [Holle 98]

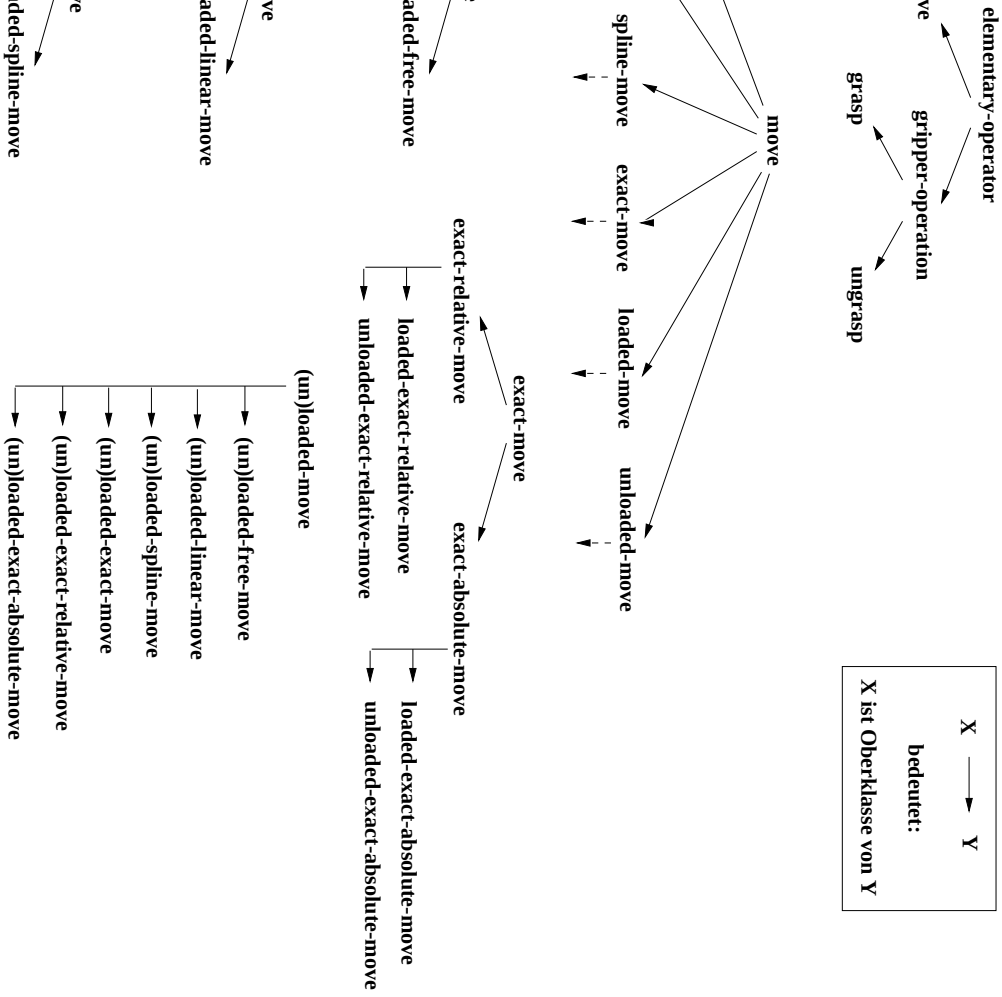


Abbildung 3.3: Hierarchische Aufstellung der Elementaroperatorklassen

Ikonische Programmsequenzen

[Riepp 97] implementiert ein System zur *ikonischen Programmierung*, das ebenfalls unter der KaVis-Simulation operiert. Der Benutzer kann einzelne *Makrooperatoren* zu einer Sequenz zusammenfügen, die eine Manipulationsaufgabe löst. Aufgrund der Kontexte der einzelnen Makrooperatoren sowie einer simulierten Ausführung lassen sich alle möglichen Instantiierungshypothesen berechnen. Diese werden über eine empirische Methode bewertet und danach sortiert. Der Benutzer kann letztlich die tatsächliche optimale Instantiierung aus der sortierten Liste auswählen.

Ererbte Operatorstruktur

In den Vorarbeiten wurde eine generische Operatorstruktur für Makro- und Elementaroperatoren entwickelt, die sich von konventionellen Definitionen unterscheidet. Jedem Operator ist ein sog. *Ausführungskontext* und eine *Objektliste* zugeordnet. Der Ausführungskontext legt fest, unter welchen Umweltzuständen ein Operator angewendet werden darf. Umgekehrt kann er aber auch dazu dienen, *Instantiierungen* eines Operators zu finden. Mit Instantiierung ist die Belegung der Objektliste mit bestimmten Objekten einer gegebenen Umwelt gemeint. Daneben enthalten Elemtaroperatoren generisch instantiierbare operatorspezifische Parameter, wie etwa die *Pfadliste*, welche die Trajektorie eines Bewegungsoperators beschreibt. Makrooperatoren enthalten eine Liste von *Nachfolgern*, welche ihrerseits Operatoren darstellen (siehe Abbildung 3.4). Aus einem korrekt instatiierten Makrooperator sind über Zuordnungslisten für die einzelnen Parameter und Nachfolger auch immer alle Nachfolger eindeutig instantiierbar. Damit reicht es aus einen Makrooperator so zu instanziieren, daß sein Kontext erfüllt ist um sicherzustellen, daß der gesamte Operatorbaum korrekt instanziiert werden kann.

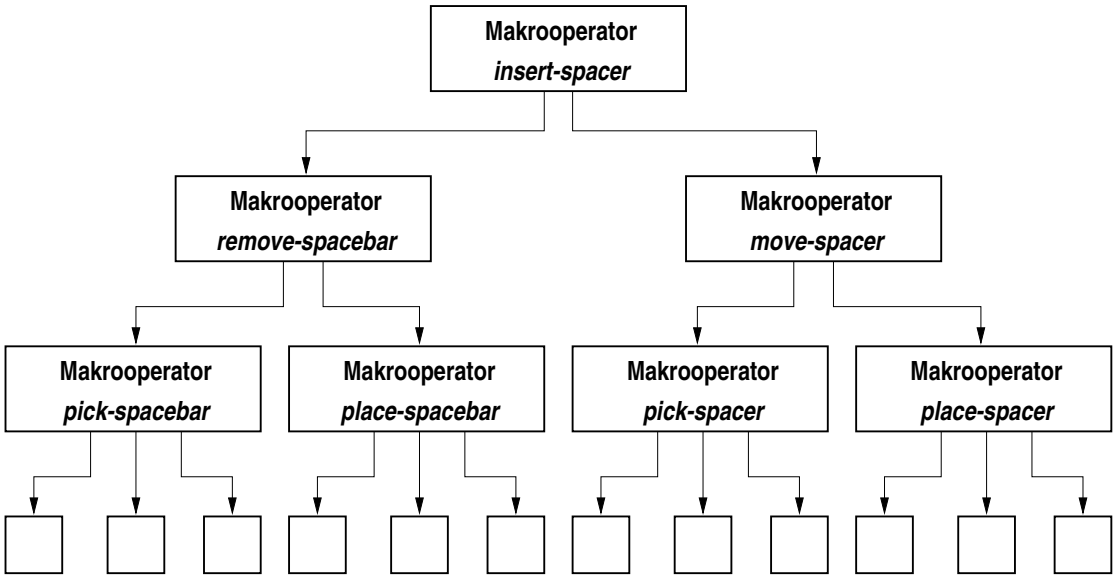


Abbildung 3.4: Exemplarischer Operatorbaum

3.1.2 Gründe für Modifikationen

Der Ausgangpunkt der Arbeit bestand ursprünglich darin, vollständig auf den Vorarbeiten aufzusetzen, welche die unter Abschnitt 2.2.1 aufgelisteten Eingabemöglichkeiten eines PbD-Systems bieten sollten. Bei diesem Ansatz sollten die Sensordaten bzw. ikonischen Vorführungen von seperaten Vorverarbeitungsmodulen in Sequenzen von initialisierten Operatoren bereitgestellt und von dem zu entwerfenden System weiterverarbeitet werden, das daraus Makrooperatoren generieren sollte. Während der Implementierung erwies es sich aus mehreren Gründen als sinnvoll diese strikte Trennung aufzugeben:

- Die Eingabe von Kommentierungen durch den Benutzer erwies sich ohne jede Kontrollmöglichkeit des Systems als zu unzuverlässig. Der Grund für diese Unzuverlässigkeit des Benutzer ist in der Trennung von Vorverarbeitungs- und Analysekomponente zu sehen, also der logischen Analyse und der Kommentierung der Benutzervorführung. Vollständig erzeugte Makroprogramme geben dem Benutzer durchaus die Möglichkeit seine Eingaben zu verifizieren. Etwa können die Instantiierungshypothesen des Systems Aufschluß auf korrekte Generalisierung geben. Die Ausführung der Programme in Simulationen auf anderen Umwelten kann Aufschluß auf die Korrektheit der Verzweigungslogik des Programmes und damit die korrekte Spezifikation der Ausführungskontexte enthalten. Durch die Trennung der Vorverarbeitungs- und der Analysekomponente wird der Benutzer dazu gezwungen in jeder Kommentierung der Vorführung den Analyseschritt des Systems zu antizipieren; denn nur so ist es möglich korrekte Kontexte und Benutzerintentionen zu spezifizieren. Es ist einsichtig, das dieses dem ungeübten Benutzer nicht zugemutet werden kann.

- Mit der Erzeugung des Makroprogrammes stehen dem Benutzer zwar zusätzliche Möglichkeiten zur Verifikation seiner Kommentierungen zur Verfügung, eine Verwertung ist aber immer damit verbunden, sowohl die Vorverarbeitung als auch die inhaltliche Erschließung komplett neu durchzuführen. Diese Vorgehensweise erwies sich als ineffizient.

- Die vorverarbeiteten Sequenzen waren bereits in ihrer Information reduziert, d.h. die Information war bereits auf die Kommentierungen des Benutzers beschnitten. Diese Information erwies sich in Kombination mit fehlerhaften Kommentierungen als nicht immer ausreichend.

- Generell zeigte sich, daß selbst geübte Benutzer mit einer korrekten bzw. sinnvollen Kommentierung der Vorführung stark gefordert waren und auch bei Experten noch Fehler bei der Kommentierung auftraten. Dies machte die Notwendigkeit evident, den Benutzer so weit als möglich als Fehlerquelle zu eliminieren und wichtige Verarbeitungsschritte halbautomatisch durchzuführen; d.h. das System führt den Schritt selbstständig aus, und der Benutzer erhält lediglich die Möglichkeit wichtige Parameter zu verändern. Diese Vorgehensweise erwies ich als komfortabler und konnte generell die Verarbeitungszeit von Vorführung zum fertigem Oeprator beschleunigen.

Aufgrund dieser Erkenntnisse wurde das Thema der Arbeit dahingehend erweitert, daß der bestehende Ansatz enger in das zu implementierende System integriert werden sollte.

Verschiedene bislang rein manuelle Arbeitsschritte wurden dabei durch halbautomatische Komponenten ersetzt. Der Analyseschritt wurde enger mit der Vorverarbeitung verzahnt. Letztlich steht damit der Analysekomponente noch die vollständige Information aus der Verführung, alle Kommentierungen des Benutzer und die Hypothesen des Systems bei der Analyse zur Verfügung. Dies machte das System insgesamt flexibler. Außerdem sollte in der endgültige Umgebung eine interaktive Nachbesserung aller Kommentierungen möglich sein, so daß eine Verfeinerung der erzeugten Makrooperatoren inkrementell und interaktiv auf der Basis von Versuchen mit den neuen Operatoren möglich werden sollte.

3.2 Motivation

Wie bereits in der Einleitung zu diesem Kapitel erwähnt, bestand die Hauptaufgabe dieser Arbeit darin, aus den Benutzervorführungen ein generisches Makroprogramm abzuleiten ist, das in der Lage sein sollte,sich an andere Objektkonstellationen anzupassen und gegenüber Veränderungen der Problemstellung flexibel zu reagieren. Um ein solches Programm erstellen zu können sind drei Schritte unumgänglich:

1. **Semantische Erschließung.** Um eine Benutzervorführung zu generalisieren ist zunächst eine abstrakte semantische Beschreibung der Umwelt zu generieren, welche die Umwelt an festzulegenden *Synchronisationspunkten* in ihrem Zustand während der Vorführung zu beschreiben. Daraus läßt sich eine semantische Beschreibung der *Dynamik* einer Vorführung, also der Veränderungen und Manipulationen in der Welt, ableiten.

2. **Generalisierung der Benutzervorführung.** Es gilt aus der Benutzervorführungen diejenigen Parameter zu extrahieren, die für die Vorführung essentiell sind, also auch bei der Ausführung des Makroprogramms auf anderen Umwelten auftreten müssen, damit das Makroprogramm fehlerfrei arbeitet. Dies bedeutet, daß einerseits generalisierte *Ausführungskontexte* andererseits aber auch eine generalisierte *Dynamik* der Benutzervorführung erschlossen werden muß. Die Ausführungskontexte beschreiben hierbei Bedingungen, unter denen einzelne Operatoren korrekt ausgeführt werden. Die Dynamik legt fest, welche Veränderungen bzw. Manipulationen in der Umwelt durch die Vorführung und deren Segmenten hervorgerufen wird. Die Generalisierung der Vorführung legt aber auch gleichzeitig die *Intention* des Benutzers fest, die wir auch die *Pragmatik* der Vorführung nennen.

   Zur Generalisierung der Benutzervorführung wurde in erster Linie ein deduktiver, erklärungsbasierter Ansatz gewählt. Wie aber bereits der der Vorstellung dieser Theorie angedeutet (siehe Abschnitt 2.1.3), ist bei komplexen Bereichstheorien bzw. einem komplexen Beispiel nicht von einer eindeutigen Erklärung des Trainingsbeispiels auszugehen, so daß die Erklärungsstruktur nur zur sinnvollen Einschränkung des Hypothesenraums für die Generalisierung genutzt werden kann. Daneben kommen unüberwachte, empirische und informationsbasierte Lernverfahren zum Einsatz, über die eine optimale Hypothese ausgewählt wird (siehe Abschnitt 2.1.4).

3. **Verzweigungsanalyse bzw. Inhaltliche Analyse.** Nachdem eine Erklärungsstruktur erzeugt und über informationsbasierte Ansätze mit generalisierten Ausfüh-

rungskontexten und der Benutzerintention versehen wurden, lassen sich sinnvolle Schlüsse über innere Abhängigkeiten der Benutzervorführung und daraus eine Verzweigungslogik des Makroprogramms generieren. Zusätzlich ist es möglich einen *Gesamtkontext* der Benutzervorführung zu ermitteln, über den sich mögliche Ausführumgebungen und Instantiierungen gegen den Makrooperator testen lassen.

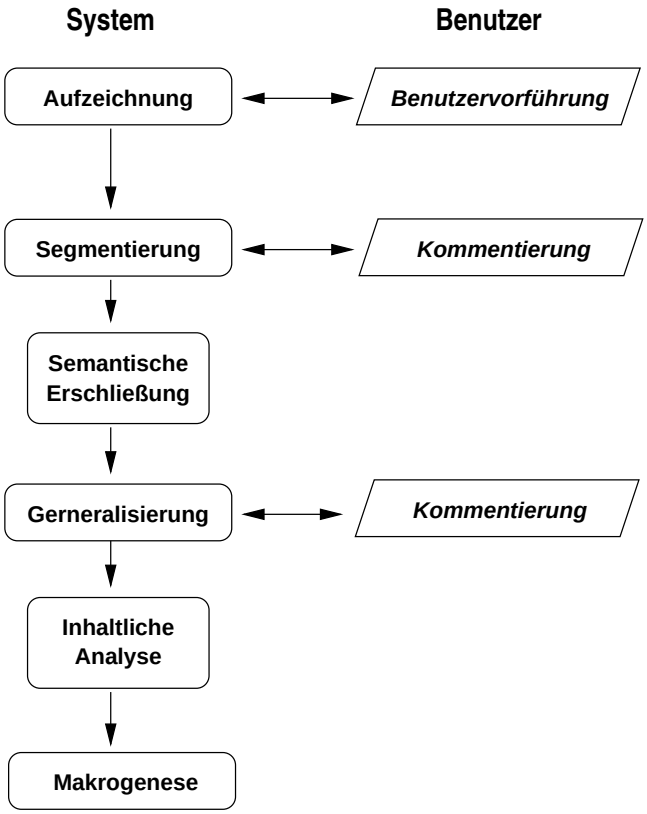


Abbildung 3.5: Verarbeitungsschritte des Lösungskonzeptes

Schließt man die Revision der Vorverarbeitungsschritte in die Arbeit mit ein so ergibt sich eine Skizze der Verarbeitungsschritte des angestrebten Systems in Abbildung 3.5.

3.3 Problemdefinition

Unter der Berücksichtigung der Vorarbeiten und der Überlegungen zur Erzeugung von generischen Makroprogrammen gliedert sich die Aufgabe der Arbeit in folgende Schritte.

1. **Vorverarbeitung des Datenmaterials aus der Vorführung.** Das durch den Benutzer in einer Vorführung eingegebene Datenmaterial muß in eine semantische Repräsentation überführt werden, die für das System verwertbar ist. Einhergehend mit der semantischen Repräsentation wird in diesem Schritt eine sinnvolle Vorseg-

mentierung erzeugt, die sich anhand von sog. *Elementarereignissen* festmachen läßt. Damit kommen der *Vorverarbeitung* zwei fundamentale Aufgaben zu:

(a) Datengetriebene *Segmentierung* der Verführung sowie

(b) *Semantische Erschließung* der Sensordaten und

(c) einer semantischen Beschreibung deren *Dynamik*.

2. **Generalisierung der Vorführung.** Die funktionale Abbildung der Sensordaten auf eine semantische Struktur der kognitiven Komponente des Systems allein reicht nicht für die Erzeugung einer hinreichend generischen Wiedergabe der Benutzervorführung aus. Um dies zu ermöglichen, gilt es den Raum für Hypothesen über die Intention des Benutzers sowie die Relevanz verschiedener Umweltzustände sinnvoll einzuschränken. Dazu wird die vorgeführte Sequenz analysiert und sowohl in ihrem Ausführungskontext als auch den Manipulationen darauf abstrakt beschrieben und generalisiert. Die *Inhaltliche Erschließung* der Benutzervorführung dient deswegen zur

   (a) *Generalisierung* der Benutzervorführung auf Datenebene,

   (b) *Generalisierung der Ausführungskontexte* und

   (c) Erschließen deren *Pragmatik* bzw. der *Benutzerintention* der Gesamtvorführung.

3. **Inhaltliche Analyse.** Aufbauend auf dem generalisierten *Kontext* und der Dynamik der zu analysierenden Vorführung können Hypothesen über die inneren Abhängigkeiten und die *Pragmatik* der gesamten Vorführung bzw. einzelner Segmente erschlossen werden. Damit wird in diesem Schritt abgeleitet:

   (a) Die *pragmatische Relevanz* einzelner Segmente und damit deren

   (b) *Ausführungsbedingung*

   (c) sowie der *Gesamtkontext* der Benutzervorführung.

4. **Implementierung einer Schnittstelle.** Um die Integration von Altdaten sowie die Möglichkeit der weiteren Verwendung von Ansätzen aus [Holle 98] und [Riepp 97] zu gewährleisten, gilt es das System für die Ausgaben der jeweiligen Systeme offen zu halten. Dazu gilt es

   (a) eine *Schnittstelle* zum Einlesen und der Weiterverarbeitung dieser Daten,

   (b) ein *modulares Segmentierungverfahren*, die auch auf den Altdaten fehlerfrei arbeitet,

   (c) sowie ein weitgehend *domänenunspezifische* Verfahren zur inhaltlichen Erschließung und zur Analyse der Benutzervorführung

   zu implementieren.

Kapitel 4

Theoretische Grundlagen

In diesem Kapitel wird beschrieben, wie sich die Verarbeitungschritte des Programming by Demonstration auf das spezielle Problem des PbD in der Robotik abbilden lassen. Es werden die in der Arbeit verwendeten Ansätze theoretisch beschrieben und klargestellt, worin der Beitrag der Arbeit besteht, und auf welchen Komponenten und Ansätzen sie aufbaut. Dabei wird auf die spezielle Problematik genauso eingegangen, wie auf mögliche generelle Anwendbarkeit der implementierten Ansätze.

4.1 Begriffsbestimmung

Bevor wir uns der genauen Spezifikation der Aufgabenstellung dieser Arbeit zuwenden, soll in diesem Abschnitt zunächst eine grundlegende Definition der notwendigen Begrifflichkeiten gegeben werden.

4.1.1 Welten

Bei den hier gegebenen Definitionen ist zu beachten, daß sie rein semantisch motiviert sind; d.h. es wird davon ausgegangen, daß sich eine gleichermaßen korrekte sowie vollständige Beschreibung der Umwelt allein durch Mittel der Prädikatenlogik geben läßt. Wichtige Parameter, die üblicherweise in ähnlichen Definitionen zu Manipulationsaufgaben ebenfalls berücksichtigt werden, wie Transformationen zwischen Objekten und physischen Parameter derselben, werden entweder auf eine homogene semantische Struktur abgebildet oder in diesen theoretischen Betrachtungen vernachlässigt. Zur semantischen Repräsentation einer Benutzervorführung benötigen wir zunächst ein gewisses prädikatenlogisches Grundrepertoir. Deshalb definieren wir:

Definition 4.1 Unäre Relation: Gegeben sei eine endliche Menge von *Objekten* $\mathcal{O}$ und eine abzählbare Menge von *Werten* $\mathcal{V}$. Dann heißt das Tupel $u = (\mathcal{O} \times \mathcal{V})$ eine *unäre Relation* über dem *Objektraum* $\mathcal{O}$ und dem *Wertebereich* $\mathcal{V}$. Man schreibt für den Objektraum $\mathcal{O}$ von u: $\mathcal{O} = \mathcal{O}(u)$ und für den Wertebereich $\mathcal{V}(u)$.

Definition 4.2 Binäre Relation: Gegeben sei eine endliche Menge von *Objekten* $\mathcal{O}$ und eine abzählbare Menge von *Werten* $\mathcal{V}$. Dann heißt das Tripel $b = (\mathcal{O} \times \mathcal{O} \times \mathcal{V})$ eine *binäre Relation*

über dem *Objektraum* $\mathcal{O}$ und dem *Wertebereich* $\mathcal{V}$. Man schreibt für den Objektraum $\mathcal{O}$ von u: $\mathcal{O} = \mathcal{O}(u)$ und für den Wertebereich $\mathcal{V}$: $\mathcal{V} = \mathcal{V}(u)$.

Mit diesem Grundgerüst können wir den Begriff der *Welt* definieren, der dazu dient, semantisch den Zustand und die Veränderungen darauf zu beschreiben, die während einer Benutzervorführung auftreten:

Definition 4.3 Welt: Gegeben sei eine Menge von Relationen $\mathcal{R}$, die über dem Objektraum $\mathcal{O}(r)$ und dem Wertebereich $\mathcal{V}(r)$ definiert sind. Dabei soll für die Objekträume gelten: $\forall r_1, r_2 \in \mathcal{R} : \mathcal{O}(r_1) = \mathcal{O}(r_2)$. Dann heißt das Tupel $W = (\mathcal{O}, \mathcal{R})$ *Welt* über dem Objektraum $\mathcal{O} = \mathcal{O}(W)$ und der *Beschreibungssprache* $\mathcal{R} = \mathcal{R}(W)$.

Nachdem wir Konstrukte definiert haben, die eine Welt inhaltlich beschreiben zu können, fehlen noch entsprechende Definitionen, die deren Zustände differenzieren:

Definition 4.4 Ausprägung einer Relation: Gegeben sei eine unäre Relation $u = (\mathcal{O} \times \mathcal{V})$ bzw. eine binäre Relation $b = (\mathcal{O} \times \mathcal{O} \times \mathcal{V})$ mit dem Objektraum $\mathcal{O}$ und dem Wertebereich $\mathcal{V}$. Dann heißt jedes Tupel (o, v) bzw. Tripel (o_1, o_2, v) mit $o, o_1, o_2 \in \mathcal{O}$ und $v \in \mathcal{V}$ eine *Ausprägung* der Relation u bzw. b über dem Objektraum $\mathcal{O}$ und dem Wertebereich $\mathcal{V}$. Man schreibt auch $u(o) = v$ bzw. $b(o_1, o_2) = v$ und meint damit, daß die Ausprägung der Relationen in den oben genannten Tupeln vorliegt.

Eine Ausprägung einer Relation beschreibt also mögliche Assozationen zwischen dem Objektraum und dem Wertebereich einer Relation. Diese muß allerdings weder vollständig noch eindeutig sein.

Um den Sprachgebrauch zu vereinfachen, wird von nun an der Begriff „Menge von Relationen" verwendet, der eine Menge aus unären und binären Relationen meint. Wo nicht weiter zwischen Ausprägungen differenziert wird, ist die Schreibweise $u = (o_1, o_2, v)$ bzw. $u(o_1, o_2) = v$ für unäre Relationen so zu verstehen, daß sie analog auf die eigentlich korrekte zu transformieren ist: $u = (o_1 \times v)$ bzw. $u(o_1) = v$; wie man sieht, ist o_2 hierbei als auf ein beliebiges Objekt aus dem Objektraum festgelegt zu betrachten.

Definition 4.5 Konsistenz: Gegeben seien zwei Ausprägungen $r_1 = (o_{11}, o_{12}, v_1)$ und $r_2 = (o_{21}, o_{22}, v_2)$ einer Relation $r = (\mathcal{O} \times \mathcal{O} \times \mathcal{V})$ über dem Objektraum $\mathcal{O}$ und dem Wertebereich $\mathcal{V}$. Die Ausprägungen heißen *konsistent*, wenn entweder $v_1 = v_2$ oder $(o_{11} \neq o_{21}) \vee (o_{12} \neq o_{22})$. Man schreibt $r_1 \bowtie r_2$.
Ist entsprechend eine Menge von Ausprägungen $\mathcal{I}$ einer Relation r gegeben, so gilt diese als konsistent, wenn die Ausprägungen paarweise konsistent sind. Man Schreibt $\bowtie \mathcal{I}$

Eine Menge von Ausprägungen ist also als konsistent anzusehen, wenn sich eine eindeutige Zuordnung von jedem Objekttupel zu seiner Wertausprägung ableiten läßt. Letztlich beschreibt dieser Begriff nichts anderes als die einleuchtende Tatsache, daß eine Relation nicht zwei Werteausprägungen gleichzeitig annehmen kann.

Definition 4.6 Vollständigkeit: Gegeben seien eine Menge von Ausprägungen $\mathcal{I}$ einer Relation $r = (\mathcal{O} \times \mathcal{O} \times \mathcal{V})$ über dem Objektraum $\mathcal{O}$ und dem Wertebereich $\mathcal{V}$. $\mathcal{I}$ heißt vollständig wenn gilt:

$$\forall o_1, o_2 \in \mathcal{O}, o_1 \neq o_2 : \exists v \in \mathcal{V} : (o_1, o_2, v) \in \mathcal{I}.$$

Eine vollständige Menge von Ausprägungen weist also jedem Objekttupel immer einen Wert zu, der aber noch nicht notwendigerweise eindeutig ist. Es ist also nur folgerichtig zu definieren:

Definition 4.7 Implementierung: Eine Menge von Ausprägungen $\mathcal{I}$ einer Relation $r = (\mathcal{O} \times \mathcal{O} \times \mathcal{V})$, die konsistent und vollständig ist, heißt *Implementierung* einer Relation.

Fassen wir nun die Implementierung mehrerer Relationen zusammen erhalten wir einen generelleren Begriff und erhalten damit den zentralen Begriff zur Beschreibung unserer Umwelten:

Definition 4.8 Zustand: Gegeben sei eine Welt W. Zusätzlich sei für jede Relation $r \in \mathcal{R}(W)$ eine Implementierung $\mathcal{I}_r$ gegeben, so daß sich

$$\mathcal{I}(S) := \bigcup_{r \in \mathcal{R}} \{(o_1, o_2, v) \in \mathcal{I}_r : (r, o_1, o_2, v)\} \tag{4.1}$$

schreiben läßt.[1] Dann heißt das Tupel $S = (W, \mathcal{I})$ *Zustand* in der Welt W. Für die dem Zustand S zugeordnete Implementierung $\mathcal{I}$ schreiben wir $\mathcal{I}(S)$.

Wir hatten bereits in der Einführung zu diesem Abschnitt darauf abgehoben, daß nicht alle Komponenten eines physischen Umweltzustandes relational vollkommen beschrieben werden können. Um diesem Umstand gerecht zu werden, muß man sich vor Augen führen, daß jedem Zustand nach Definition 4.8 immer mehrere physische Umwelten entsprechen. Damit definieren wir:

Definition 4.9 Realisierung eines Implementierung: Eine physische Umwelt heißt *Realisierung* einer Implementierung $\mathcal{I}$ eines Zustandes S, wenn die Implementierung aller Relationen der zugeordneten Welt $S.W$ der Auswertungslogik dieser physischen Umwelt entspricht, d.h. alle in W beschriebenen in der physischen Umwelt diesselbe Ausprägung haben. Für die Menge aller Realisierungen einer Implementierung schreiben wir $R(\mathcal{I})$. Umgekehrt muß einer Realisierung r immer eindeutig *genau eine* Implementierung zugeordnet sein, wofür wir $\mathcal{I}(r)$ schreiben.

4.1.2 Episoden

Im letzten Abschnitt habe wir alle Begrifflichkeiten definiert die zur semantischen Beschreibung einer statischen Welt notwendig sind. Zur korrekten Beschreibung der Dynamik einer Manipulation definieren wir weiter:

Definition 4.10 Episode: Gegeben sei eine Welt $W = (\mathcal{O}, \mathcal{R})$. Zusätzlich sei eine *Sequenz* von Zuständen $(S)_0^n = (S_0, \ldots, S_n)$ in der Welt W gegeben. Dann heißt das Tupel $E = (W, (S)_0^n)$ *Episode* in der Welt W. S_0 heißt *initialer Zustand* oder *Ausführungskontext* der Episode, ihr S_n *Endzustand*.

Eine Episode beschreibt also eine Folge von Zuständen in *einer* Welt. In der Folge dieser Zustände können einzelne Relationen ihre Ausprägungen verändern, so daß wir definieren:

[1] Man führe sich vor Augen, daß durch diese Definition auch stets wieder eine Rücktransformierung von $\mathcal{I}$ in die einzelnen Implementierung $\mathcal{I}_r$ möglich ist.

Definition 4.11 Manipulation einer Episode: Gegeben sei eine Episode $E = (W, (S)_0^n)$ mit dem initialen Zustand S_0, dem Endzustand S_n und den zugeordneten Implementierungen $\mathcal{I}(S_0), \ldots, \mathcal{I}(S_n)$. Dann heißt die Menge $\Delta(E)$:

$$\Delta(E) := \mathcal{I}(S_n) \setminus \mathcal{I}(S_0) \tag{4.2}$$

Manipulation der Episode E. Gilt für die Manipulation einer Episode $\Delta(E) = \emptyset$, dann heißt sowohl die Manipulation als auch die dazugehörige Episode *trivial*. Der Zustand S_0 heißt auch *Kontext* der Manipulation $\Delta(E)$ bzw. der Episode.

Die Manipulation einer Episode gibt also die Unterschiede der Ausprägungen der einzelner Relationen zwischen Anfangs- und Endzustand.

Definition 4.12 Verkettung von Episoden: Gegeben seien zwei Episoden $E_1 = (W, (S)_{10}^{1n})$, $E_2 = (W, (S)_{20}^{2m})$ mit ihren Manipulationen $\Delta(E_1)$, $\Delta(E_2)$. Dann heißt die Episode:

$$V = E_1 \circ E_2 = \left(W, \quad \begin{pmatrix} (S)_{20}^{2m} \\ (S)_{10}^{1n} \end{pmatrix} \right)$$

Verkettung der Episoden E_1 und E_2 mit der Manipulation $\Delta(E_1 \circ E_2)$.

Wir definieren für Manipulationen weiter:

Definition 4.13 Verkettung von Manipulationen: Gegeben sei eine Verkettung zweier Episoden $\Delta(E_1 \circ E_2)$ mit den dazugehörigen Manipulationen $\Delta(E_1)$ und $\Delta(E_2)$. Dann definieren wir die Verknüpfung $\circ$:

$$\Delta(E_1) \circ \Delta(E_2) := (\Delta(E_1) \cup \Delta(E_2)) \setminus \left\{ \begin{array}{l} (r, o_1, o_2, v_1) \in \Delta(E_1) : \\ \exists (r, o_1, o_1, v_2) \in \Delta(E_2) : v_1 \neq v_2 \end{array} \right\},$$

als die Verkettung von Manipulationen. Gilt zusätzlich $\Delta(E_1 \circ E_2) = \Delta(E_1) \circ \Delta(E_2)$ heißt die Verkettung der Episoden bzw. Manipulationen auch *Fortschreibung* der einzelnen Teile der Verkettung.

Die Begriffe gelten analog für Mengen von Episoden, wenn sich durch Rekursion jeweils Paare finden lassen, die die Voraussetzungen erfüllen. Drei Episoden E_1, E_2, E_3 gelten etwa dann als Verkettung $E_1 \circ E_2 \circ E_3 = E_1 \circ (E_2 \circ E_2)$. Ihre Manipulationen gelten als Fortschreibung $\Delta(E_1) \circ \Delta(E_2) \circ \Delta(E_3)$, wenn $\Delta(E_2) \circ \Delta(E_3)$ und $\Delta(E_1) \circ \Delta(E_2 \circ E_3)$ Fortschreibungen darstellen. Analog läßt sich der Begriff aus $n \in I\!N$ Episoden ausdehnen.[2]

Der Begriff der Fortschreibung läßt sich so verstehen, daß zwei Episoden sich logisch aneinanderfügen lassen. Das bedeutet einerseits, daß die Episoden in einer zeitlich sinnvollen logischen Reihenfolge stehen, zwischen Ihnen aber auch keine Lücke besteht in der relevante Operationen fehlen. Andererseits bedeutet das, daß die beiden Episoden konsekutiv korrekt und außerdem inhaltlich lückenlos sind. Andererseits schließt dies auch ein,

[2]Man beachte, daß die Assoziativität für die Manipulationen von Verkettungen von Episoden nicht generell gilt, sondern daß gerade dieses mit der Definition der Fortschreibung gemeint ist, also durchaus nicht immer Manipulationen von Episoden eine Fortschreibung in ihrer Verkettung eine Fortschreibung darstellen.

daß die beiden Episoden in sich geschlossene Aktionen bilden, daß heißt mögliche Manipulationen auf den Welten ausschließlich innerhalb der einzelnen Episoden stattfinden, so daß keine zusätzlichen Zustandsänderungen mehr auftreten, die erst in der Verkettung beider Episoden relevant werden. Man lese die Definition der *Geschlossenheit* (Definition 4.19) nach, um eine noch bessere Motivation zu erhalten.

Definition 4.14 Operator: Gegeben sei ein auf der Welt W definierter Zustand S und dessen Implementierung $\mathcal{I}(S)$. Eine Abbildung O, welche jede Realisierung $r \in R(\mathcal{I})$ eindeutig auf eine Realsierung r' abbildet, heißt *Operator* auf der Welt W. Analog zur Manipulation einer Episode definiert

$$\Delta(O(r)) := \mathcal{I}(r) \setminus \mathcal{I}(r')$$

die *Manipulation* dieses Operators auf dem Zustand S bzgl. der Realisierung r. Die Menge aller Operatoren auf einer Welt W beschreiben wir als $O(W)$.[3]

Mit dem definierten Operatorbegriff können wir analog zu [Mitchell 86] den Begriff der Operationalisierung und damit der Operationalisierbarkeit einer Episode festlegen (siehe Abschnitt 2.1.3):

Definition 4.15 Verkettung von Operatoren: Gegeben seien zwei Operatoren O_1, O_2, sowie ein Zustand S mit der Implemtierung (I). Gilt $O_1(r) = r'$ und $O_2(r') = r''$ für zwei Implementierung $(I)'$ und $(I)''$, dann heißt $O_1(r) \circ O_2$ *Verkettung* der beiden Operatoren. Der Begriff gilt analog für $n \in I\!N$ Operatoren.

Definition 4.16 Erklärung einer Episode: Gegeben sei eine Episode E mit ihrer Manipulation $\Delta(E)$, dem initialen Zustand S_0, dessen Implementierung $\mathcal{I}$. Eine Verkettung von Operatoren $O_1 \circ \ldots \circ O_n$ heißt *Erklärung* eines Segmentes bzgl. einer Realisierung $r \in R(\mathcal{I})$, wenn gilt:

$$\Delta(E) = \Delta(O_1) \circ \ldots \circ \Delta(O_n).$$

Die Definition bedeutet, daß eine Realisierung existieren muß, für welche die Verkettung aller Manipulationen der einzelnen Operatoren exakt der Manipulation der Episode der Episode entsprechen.

Um eine Episode aber sinnvoll operationalisieren zu können, ist es sehr hilfreich, diese vor ihrer Erklärung zu untergliedern. Dazu wird die Episode rekursiv immer weiter zergliedert, bis sich schließlich Teilepisoden ergeben, die direkt auf Operatoren abbildbar sind. Die Generierung dieser hierarchischen Erklärungsstruktur nennen wir die Segmentierung einer Episode:

Definition 4.17 Segment einer Episode: Eine Episode $P = (W, (P)_0^m)$ heißt dann *Segment* $[S]_i^{i+m}$ einer anderen Episode $E = (W, (S)_0^n)$, wenn gilt:

$$\exists i \in [0, n] : \forall k \in [0, m] : P_k = S_{k+i}.$$

[3] Von nun an betrachten wir die Menge aller Operatoren, doe auf einer Welt definiert sind stets als endlich und fest vorgegeben.

Zwei Segmente $[S]_k^1$ und $[S]_m^n$) einer Episode heißen *disjunkt*, wenn

$$\forall i \in [k,l], j \in [m,n] : i \neq j.$$

Gilt zusätzlich $l < m$, heißen die Segmente *konsekutiv* und man schreibt $[S]_k^1 \rhd [S]_m^n$.

Definition 4.18 Segmentierung einer Episode: Eine Folge von paarweise disjunkten und konsekutiven Segmenten $([S]_{i_0}^{j_n})_0^n$ einer Episode $E = (W, (S)_0^n)$ heißt *Segmentierung* einer Episode. Ist eines der Segmente der Segmentierung trivial, so heißt auch die Segmentierung *trivial*.

Definition 4.19 Geschlossene Epsiode bzw. Segment: Gegeben sei eine Episode E mit dem initialien Zustand S und dessen Implemenierung $\mathcal{I}$. Die Episode E heißt *geschlossen*, wenn eine Verkettung von Operatoren $O_{1^n} = O_1 \circ \ldots \circ O_n$ exisitert, so daß O_1^n bzgl. aller Realisierung $r \in R(\mathcal{I})$ eine Erklärung darstellt. Alle nicht-geschlossenen Segmente heißen *offen*.

Nun liegt der Verdacht nahe, daß jedes Segment geschlossen ist. Dies trifft aber nicht zu. Der Grund dafür liegt wieder im endlichen Auflösungsvermögen einer relationalen Beschreibung einer physischen Umwelt. Man betrachte etwa Manipulationsaufgaben und eine mögliche relationale Beschreibung derselben. Nehmen wir nun an, daß in der relationalen Beschreibung keine Relation exisitiert, die beschreibt, ob ein Objekt gegriffen ist oder nicht. Je nachdem, ob aber ein Objekt gegriffen ist oder nicht, haben Bewegungsoperatoren Einfluß auf die Position eines Objektes und damit der relationalen Beschreibung seiner Lage oder nicht. Eine Extrapolation einer solchen Überlegung zeigt recht schnell, daß bei dieser Konstellation erst Segmente einer Episode geschlossen sein können, wenn innerhalb dieser ein Objekt angefahren, das Objekt gegriffen, ein Ablagepunkt angefahren und das Objekt dort losgelassen wird. Diesen Ablauf nennt man generell eine *Pick&Place*-Operation, welche in Abschnitt 4.2 genauer motiviert wird. Tatsächlich war es in dem vorgegebenen System so, daß keine solche Relation zur Beschreibung gegriffener Objekte exisiterte, weswegen die *Pick&Place*-Operationen in dem implementierten System auch die kleinsten möglichen geschlossenen Segmente darstellten.

4.1.3 Pragmatik und Generalisierungen

Die weitere Vorgehensweise im EBL (siehe Abschnitt 2.1.3) legt nahe, daß nun eine Generalisierung der gefundenen Erklärungen zu generieren ist. Hierzu definieren wir:

Definition 4.20 Pragmatik einer Episode: Gegeben sei eine Episode E und deren Manipulation $\Delta(E)$. Jede Teilmenge $\mathcal{P}$ von $\Delta(E)$ heißt *Pragmatik* der Episode E. Eine Pragmatik $\mathcal{P}$ heißt *trivial*, wenn $\mathcal{P} = \emptyset$.

Definition 4.21 Generalisierung des initialen Zustandes: Gegeben sei eine Episode E, deren initialer Zustand S mit der Implementierung $\mathcal{I}$ und eine nicht-triviale Pragmatik $\mathcal{P}$ der Episode. Zusätzlich existiere eine Erklärung O_1^n. Eine Teilmenge $\mathcal{G}$ der Implementierung $\mathcal{I}$ heißt *Generalisierung* des initialen Zustandes von E bzgl. der Pragmatik $\mathcal{P}$, wenn gilt:

$$\forall r \in R(\mathcal{I}) : \mathcal{P} \subset \Delta(O_1(r)) \circ \ldots \circ \Delta(O_n).$$

4.2 Vorverarbeitung der physischen Vorführung

In Schritt der Vorverarbeitung werden die ersten beiden Schritte des Lösungskonzeptes implementiert (siehe Abschnitt 3.2, Abbildung 3.5). Nach der *Benutzervorführung* mit Hilfe des Datenhandschuhs (siehe Abschnitt 4.2.1), wird zunächst eine *Simulation* der Szene generiert, die eine identische Abbildung der Umwelt in Echtzeit ermöglicht. Ziel dieser Simulation ist einerseits die *Schlüsselereignisse* der Benutzervorführung zu detektieren, um darauf aufbauend eine *Segmentierung* der Vorführung vorzunehmen. Andererseits erfolgt in der Simulation die *Semantische Erschließung* der Episode. Die beiden Schritte dienen der Gewinnung der episodischen Daten, die den Definitionen im Abschnitt 4.1. Insgesamt ist die Vorgehensweise mit der des LFO-Systems (siehe [Kuniyoshi 94]) vergleichbar, das bereits in Abschnitt 2.2.4 vorgestellt wurde. Die resultierende semantische Repräsentation in beiden System ist allerdings vollständig verschieden.

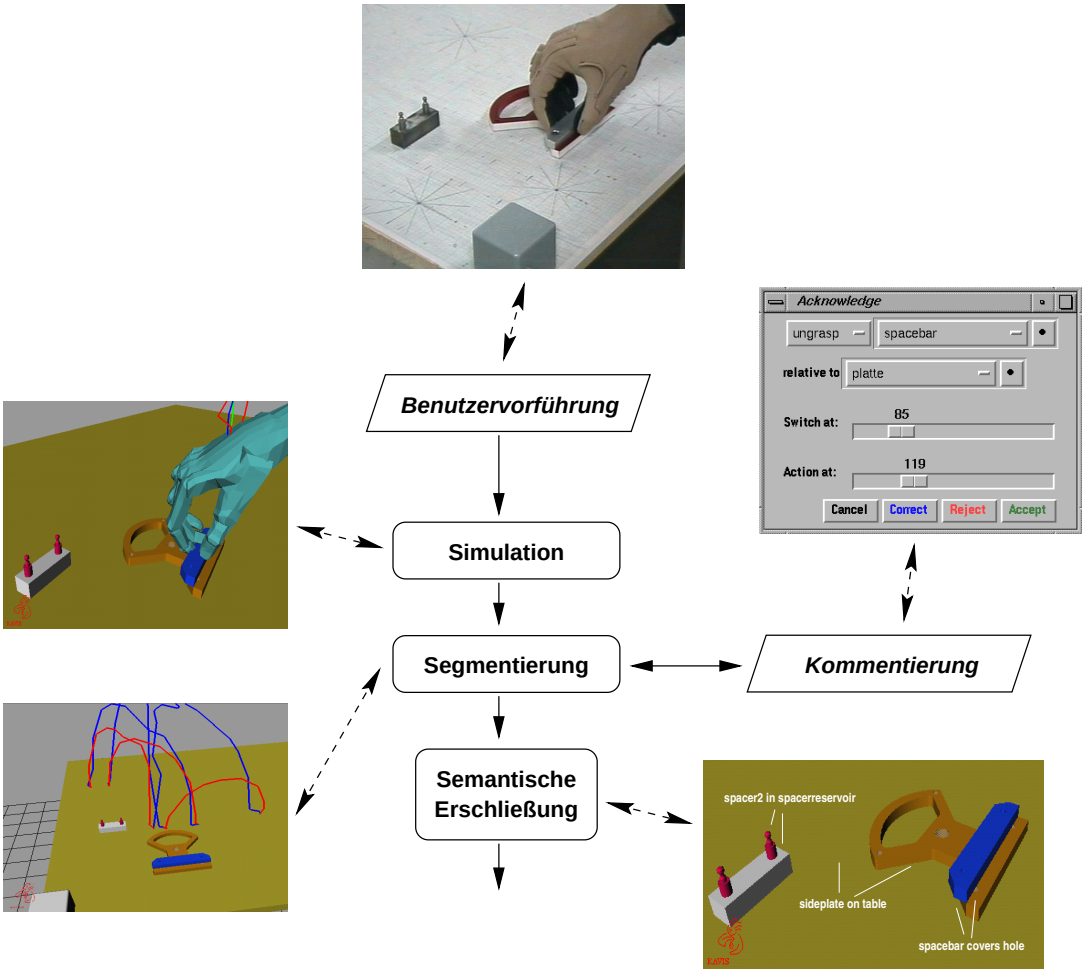


Abbildung 4.1: Schritte der Vorverarbeitung

4.2.1 Sensordaten und Simulation

Zur Aufzeichnung der Benutzervorführungen wurden in dieser Arbeit ein sog. Datenhandschuh (siehe Abbildung 4.2) verwendet. Dieser Datenhandschuh liefert die sechs Freiheitsgrade der Position des Handrückens in der Szene sowie je vier Gelenkwinkel pro Finger und zwei Gelenkwinkel des Handwurzelgelenkes. Aus diesen Daten ist eine relativ präzise Rekonstruktion der Handposition sowie der Hand- und Fingerstellung möglich. Die Aufzeichnung der Daten erfolgt mit einer festen Samplerate; die Folge dieser Datenvektoren wird auch als *Trace* einer Benutzervorführung bezeichnet. Die Samplepunkte dieses Traces entsprechen im Prinzip den Indizes der Zustände eines Prozesses (siehe Definition 2.1) bzw. einer Episode (siehe Definition 4.10), sind aber nicht nicht direkt mit der Episode zu verwechseln, da diese erst nach der inhaltlichen Erschließung entsteht.

Abbildung 4.2: Datenhandschuh mit Tracker

Die Daten werden in unmittelbar vom Datenhandschuh zur Simulationskomponente übertragen. Dort werden die Daten dazu genutzt über die Simulation direktes visuelle Rückmeldung über seine Vorführung zu geben: Simultan zur Aufzeichnung wird die Position des Datenhandschuhs in der Simulation nachgefahren und die jeweils aktuellen Hand- und Fingerstellungen werden visualisiert. Die Position der Hand wird in der Simulation bzgl. des *Tool-Center Points (TCP)* der Hand angegeben, der sich im Zentrum der Handfläche befindet (siehe Abbildung 4.3).

Darüber hinaus besteht aber auch die Möglichkeit durch den Benutzer vorgenommene Manipulationen direkt zu visualisieren. Zu diesem Zweck muß die simulierte und die reale Umwelt der Benutzervorführung in Einklang gebracht werden. Danach ist das System in der Lage, alle mit dem Datenhandschuh vorgenommenen Manipulationen simultan zur Vorführung nachzuvollziehen, so daß der Vorführer direkte Rückmeldung über die Greif- und Loslaßpunkte sowie die Möglichkeit, die neuen Positionen der Objekte während ihrer Manipulation und nach dem Loslassen zu verfolgen. Die gewählte Technik ließe auch prinzipiell die Möglichkeit offen, die Benutzervorführung alleine in der Simulation ohne reale physische Umwelt vorzunehmen. Diese Funktionalität stand zwar zum Abschluß der Arbeit zur Verfügung, wurde aber noch nicht weiter genutzt.

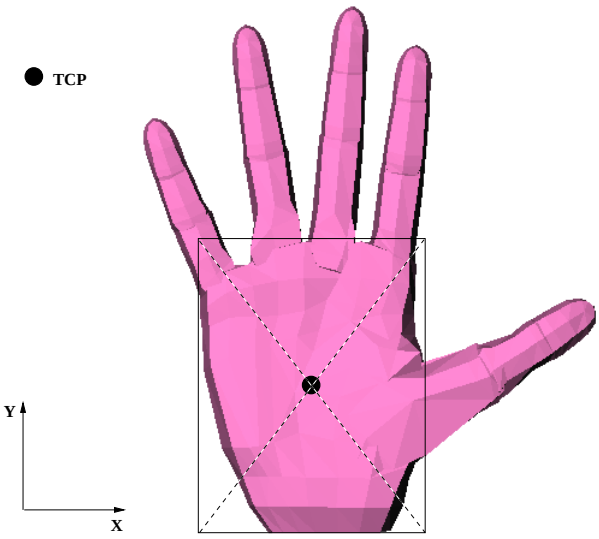


Abbildung 4.3: Tool-Center Point des Datenhandschuhs

4.2.2 Detektion von Schlüsselereignissen

Analog zu [Kuniyoshi 94] oder [Kang 97] besteht die erste Aufgabe zur Verarbeitung der Benutzervorführung darin sog. *Schüsselereignisse* zu detektieren. Diese bestehen bei einfachen Manipulationsaufgaben in den Greif- und Loslaßoperationen. Es gilt also diejenigen Zeitpunkte in der Trajektorie zu bestimmen, in denen Objekte gegriffen bzw. losgelassen werden. Das implementierte Verfahren motiviert sich durch einen Blick auf Abbildung 4.4. Diese zeigt die Geschwindigkeit des TCP (*grün*), Fingerkrümmung (*blau*) und erkannte Griffe (*rot*) darstellt. In der Grafik sind insgesamt drei Greif/Loslaßsequenzen (*Pick&Place*) dargestellt. Zur Motivation der dargestellten Vorgehensweise muß man sich vor Augen führen, daß in dem implementierten System nur statische Griffe zulässig waren; d.h. die Fingerstellung bleibt vom Greifen des Objektes bis zum Loslassen desselben fest.

Der typische Verlauf einer solchen Sequenz beginnt mit einer raschen *Annäherung* an das zu greifende Objekt, die sich durch eine große Geschwindigkeit des TCP und eine Abnahme des Abstands zum Objekt auszeichnet. Während der Annäherung wird eine objektspezifische Griffstellung eingenommen, die je nach Größe des Objektes stark variieren kann. Schließlich kommt es zum Greifen des Objektes mit einer Zunahme der Fingerkrümmung. Das Greifen wird durch das *Abrücken* des TCP vom Ort des Greifens abgeschlossen, das durch eine erneute Zunahme der Geschwindigkeit gekennzeichnet ist. Wie man Abbildung 4.4 entnehmen kann, bleibt die Fingerkrümmung bis zum Ablegen des Objektes weitgehend konstant. Aufgrund dieses Profils bietet es sich an den *effektiven Greifzeitpunkt* anhand der minimalen Geschwindigkeit des TCP in Kombination mit der maximalen Fingerkrümmung zu bestimmen (siehe Algorithmus 4.1).

Beim *Loslassen* des Objektes ergibt sich ein ähnlicher Verlauf. Vor dem *effektiven Loslaßzeitpunkt* kommt es zur *Annäherung* an den *Ablagepunkt*, die mit relativ hoher

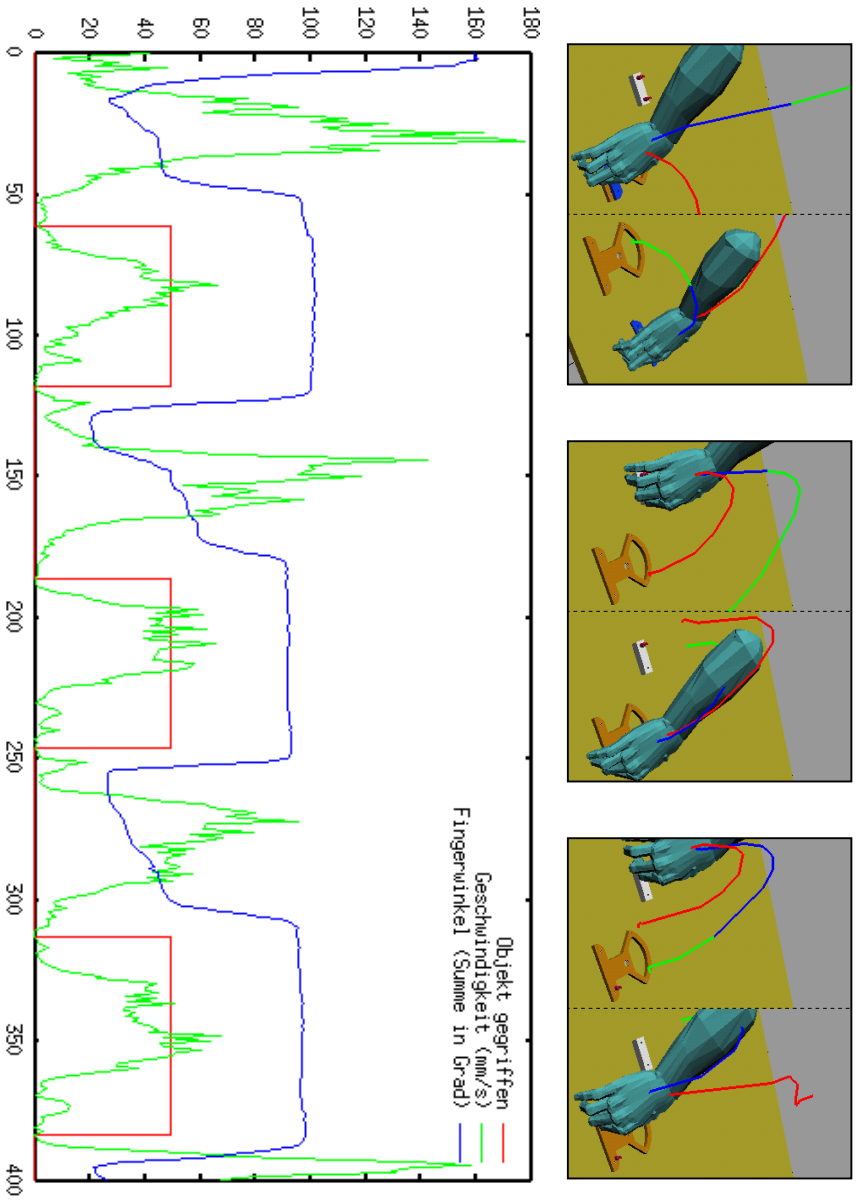


Abbildung 4.4: Greif- und Loslaßdetektion.

Eingabe: $start, speed[\,], jonts[\,]$
$start$**:** Anfangszeitpunkt des Erkennung (Samplepunkt)
$speed[\,]$**:** Array mit Handgeschwindigkeit
$joints[\,]$**:** Array mit Fingerkrümmung
Ausgabe: $begin, pos, end, object$
$begin$**:** Anfangszeitpunkt der Greifumgebung
pos**:** Griffzeitpunkt
end**:** Endzeitpunkt der Greifumgebung
$object$**:** Gegriffenes Objekt

1: $i \leftarrow start$
2: $mindist[i] \leftarrow computemindist(i)$
3: $minobj[i] \leftarrow computeminobj(i)$
4: **repeat**
5: {**1. Anfangszeitpunkt der Greifumgebung**}
6: **repeat**
7: $i \leftarrow i + 1$
8: $mindist[i] \leftarrow computemindist(i)$
9: $minobj[i] \leftarrow computeminobj(i)$
10: **until** $(mindist[i] < minth) \wedge (mindist[i-1] > mindist[i]) \wedge (speed[i]) < speedth)$
11: $begin \leftarrow i$
12: {**2. Endzeitpunkt der Greifumgebung**}
13: **repeat**
14: $i \leftarrow i + 1$
15: $mindist[i] \leftarrow computemindist(i)$
16: $minobj[i] \leftarrow computeminobj(i)$
17: **until** $(mindist[i] > minth) \vee (minobj[i-1] \neq minobj[i])$
18: $end \leftarrow i$
19: $pos = start$
20: $minscore = speed[start] - joint[start]$
21: {**3. Griffzeitpunkt**}
22: **for** $i = begin$ **to** end **do**
23: **if** $(speed[start] - joint[start]) < minscore$ **then**
24: $pos = i$
25: $minscore = speed[start] - joint[start]$
26: **end if**
27: **end for**
28: {**4. Absicherung des Griffes**}
29: **until** $(joints[begin] + jointth) < joints[pos]$
30: **return** $begin, pos, end, minobj[pos]$

Algorithmus 4.1: Detektion des Greifzeitpunktes ($lookforpick$).

Geschwindigkeit erfolgt und durch ein starkes Abbremsen kurz vor dem Erreichen des eigentlichen Ablagepunktes gekennzeichnet ist. Wurde das Objekt erst einmal abgelegt kommt es zum *Abrücken* des TCP der Hand vom Ablagepunkt, wozu die Geschwindigkeit der Hand wieder zunimmt. Hier machen wir uns den Umstand zunutze, daß die Fingerkrümmung, während ein Objekt gehalten wird, fast konstant bleibt. Zu dem Zeitpunkt, zu dem die Fingerkrümmung relativ zu der Krümmung zum Greifzeitpunkt signifikant abfällt, wurde das Objekt abgelegt. Verfolgt man den Trace dann noch bis zu dem Punkt zurück, zu dem die Geschwindigkeit des TCP ein relatives Minimum erreicht, erhält man zuverlässig den *Loslaßzeitpunkt*. Während des Abrückens kommt es meist zu einer weiteren Entspannung der Hand bis zu einer Relaxationshaltung.

Greiferkennung

Damit ergibt sich eine Detektion von Greifoperationen in vier Phasen (siehe Algorithmus 4.1):[4]

1. **Erkennung einer Greifstelle.** Die *Greifstelle* beschreibt den Zeitpunkt in der Trajektorie, in der sich der Benutzer darauf vorbereitet, ein Objekte zu greifen. [Kang 94c] schlägt als Kriterien zur Detektion dieses Zeitpunktes die Annäherung an ein Objekt, eine Verringerung der Geschwindigkeit der Hand sowie ein leichtes Öffnen der Hand zum Zweck des Aufnehmens eines Objektes vor. Bereits in Abschnitt 2.2.4 wurde die Unangemessenheit des letzten Kriteriums diskutiert, das nur bei großen Objekten zuverlässig auftritt; deswegen wurde auf ein Heranziehen dieses Kriteriums verzichtet. Während der Implementierung traten bei der Bestimmung der Geschwindigkeit der Hand signifikante Probleme auf, da diese oft großen Schwankungen unterworfen war, ohne daß dies durch die physikalische Vorführung gerechtfertigt gewesen wäre.[5] Aus diesem Grund wurde an dieser Stelle auf dieses Kriterium verzichtet.

   In der vorliegenden Implementierung wurde ausschließlich auf die Annäherung der Hand an ein Objekt zurückgegriffen. Eine mögliche Greifstelle wird also dadurch erkannt, daß der TCP der Hand einen gewissen Abstand zum nächsten Objekt unterschreitet und sich über einen bestimmten Zeitraum weiter an das Objekt annähert. Zusätzlich wird nach endgültiger Detektion des Griffpunktes geprüft, ob die Hand sich tatsächlich geschlossen hat (siehe unten). Damit ist die im ersten Schritt des Algorithmus 4.1 bestimmte Griffstelle eher als Hypothese einer möglichen Greifstelle zu interpretieren, die erst noch näher zu prüfen ist.

2. **Bestimmung einer Greifumgebung.** Nachdem mit der Greifstelle, der Anfang der Greifumgebung bestimmt wurde, wird im zweiten Schritt das Ende der Greifumgebung und damit eine grobe Lokation des Griffes vorgenommen. Zu diesem Zweck

[4] Man beachte, daß das Verfahren stark vereinfacht symbolisiert wurde, um es auf eine überschaubare Komplexität zu reduzieren. In der eigentlichen Implementierung wird zusätzlich sichergestellt, daß eine Annäherung an das Objekt tatsächlich über mehrere Samplepunkte anhält. Außerdem fehlen Abbruchkriterien für den Fall, daß kein Greifpunkt mehr detektiert wird. Es wird also quasi angenommen, daß der Datenstrom nicht endlich ist.

[5] Man betrachte hierzu Abbildung 4.4.

wird nach dem Zeitpunkt in der Benutzervorführung gesucht, an dem der TCP der Hand entweder wieder zu allen Objekten ein bestimmte minimale Distanz angenommen Hand, oder sich das Objekt, das zum TCP die minimale Distanz aufweist, ändert; dieses Kriterium ist so zu interpretieren, daß sich die Hand dann wieder von einem potentiellen Greifpunkt entfernt, also daß Objekt in dem bestimmten Zeitraum gegriffen wurde.

3. **Bestimmung des Greifpunktes.** Nachdem eine Greifumgebung in den ersten beiden Schritten festgelegt wurde, in der potentiell ein Griff stattgefunden haben könnte, wird im dritten Schritt der endgültige Greifzeitpunkt bestimmt. Zu diesem Zweck wird über die Gerifumgebung eine Optimierung über ein Minimum der Geschwindigkeit der Hand und ein Maximum der Fingerkrümmung durchgeführt. Der Punkt, in dem der Term *TCP-Geschwindigkeit - Fingerkrümmung* minimal ist, legt den Greifzeitpunkt fest. Die Kombination aus Festlegung eines Hypothesenintervalles und einer Optimierung auf dem *gesamten* Intervall macht das Verfahren extrem robust, vor allem gegenüber lokalen Minima der Optimierungskriterien. Dieses Verfahren wird durch einen Blick auf Abbildung 4.4 motiviert: Kurz vor dem effektivem Griff kommt es typischerweise zu einem starken Abfall der Geschwindigkeit, bis sich schließlich die Hand schließt, und es zu einem starken Anstieg der Fingerkrümmung kommt. Anschließend beginnt das Abrücken der Hand vom Greifpunkt mit einem starken Anstieg der Geschwindigkeit.

4. **Absicherung des Griffes.** Im Unterschied zu den meisten anderen existierenden Verfahren, etwa [Kang 94c], arbeitet das implementierte Verfahren mit Hypothesen über eine potentielle Greifumgebung, die zusätzlich ausschließlich über den Abstand des TCP zu anderen Objekten festgelegt wird. Das eigentlich intuitive Kriterium, die Fingerstellung der Hand, kommt in diesen ersten beiden Verarbeitungsschritten noch gar nicht zur Anwendung. Diese Methode hat den Vorteil, daß Optimierungskriterium und Optimierungsintervall vollständig unabhängig voneinander sind, was wechselweise Fehler ausschließt und damit zu der bereits angesprochenen Robustheit führt. Andererseits ist das Abstandkriterium aber noch nicht hinreichend einen möglichen Griff sicher vorherzusagen. Aus diesem Grund wird im letzten Schritt abgesichert, daß tatsächlich ein Griff stattgefunden hat, indem geprüft wird, ob sich die Hand vom Anfang der Greifumgebung zum erkannten Greifpunkt tatsächlich signifikant geschlossen hat.

Loslaßerkennung

Die Loslaßerkennung gestaltet sich einfacher als die Greiferkennung, da sich sehr einfach eindeutig festlegen läßt, wann ein Loslassen erfolgt, wenn man erst einmal ein gegriffenes Objekt unterstellt. Damit gliedert sich die Loslaßerkennung in drei Phasen (siehe Algorithmus 4.2):

1. **Grobbestimmung des Loslaßzeitpunktes.** Da die Fingerkrümmung sich über den Zeitraum, in dem ein Objekt fest gegriffen bleibt, fast konstant bleibt, läßt bereits die Detektion eines signifikanten Öffnens der Hand in Form eines leichten

Abfalls der Fingerkrümmung einen zuverlässigen Schluß aus den Loslaßzeitpunkt zu. Dies ist Grundlage des ersten Verarbeitungsschrittes.

2. **Bestimmung der Loslaßumgebung.** Analog zur Grifferkennung ergibt sich das Ende der Loslaßumgebung daraus, daß alle Objekte der Szene einen bestimmten Abstand zum TCP aufweisen, oder ein neues Objekt den minimalen Abstand zum TCP einnimmt. Anders als bei der Greifumgebung stellt die Loslaßumgebung nicht gleichzeitig auch das Optimierungsintervall für den Optimierungsschritt dar, sondern dient lediglich der Sicherstellung eines hinreichenden Abstands zwischen einzelnen Greifhypothesen, was das Verfahren insgesamt robuster gegen fälschliches Erkennen von Greifoperationen im Bereich des Loslassens von Objekten macht.

3. **Optimierung des Loslaßzeitpunktes.** Im letzten Schritt wird der *effektive Loslaßzeitpunkt* bestimmt, zu dem das Objekt abgelegt wird. Nach dem effektiven Loslassen des Objektes kommt es relativ schnell zum Abrücken vom Loslaßpunkt, also einem Anstieg der Geschwindigkeit. Währenddessen entspannt sich die Hand, was ja nach Objektgröße zu einem weiteren Öffnen bzw. zu einem Schließen der Hand führen kann. Deswegen kann der effektive Loslaßpunkt gefunden werden, indem man den Trace bis zu dem Punkt zurückverfolgt, in dem die Geschwindigkeit ein relatives Minimum einnimmt, das Abrücken also noch nicht begonnen hat.

Gegriffene Objekte und Ablagepunkt

In den Algorithmen 4.1 und 4.2 werden neben den eigentlichen Zeitpunkten, zu denen die Greif- und Loslaßoperationen stattfinden, auch das *gegriffene Objekt* sowie das *Objekt des Ablagepunktes* bestimmt. Das Objekt des Ablagepunktes beschreibt jenes, relativ zu welchem das Objekt abgelegt wird. Dabei wird relativ zu diesem Objekt ein fester Ablagepunkt bestimmt, an dem das Objekt dann exakt abgelegt wird.

Zum besseren Verständnis des implementierten Verfahrens zur Bestimmung des gegriffenen Objektes und des Objektes des Ablagepunktes, soll hier zunächst ein kurzer Einblick in das verwendete Umweltmodell gegeben werden, da dieses über die allgemeinere Begriffsbestimmung aus Abschnitt 4.1 hinausgeht. Alle Objekte sind in ihrer Position durch eine Transformationsmatrix T bestimmt:

$$
\begin{aligned}
T = & \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(r_x) & -sin(r_x) & 0 \\ 0 & sin(r_x) & cos(r_x) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \\
& \cdot \begin{pmatrix} cos(r_y) & 0 & sin(r_y) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(r_y) & 0 & cos(r_y) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} cos(r_z) & -sin(r_z) & 0 & 0 \\ sin(r_z) & cos(r_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\end{aligned} \tag{4.3}
$$

Dabei sind x, y, z die kartesischen Koordinaten und r_x, r_y, r_z die zu den Achsen gehörigen eulerschen Rotationswinkel. Wichtig ist, daß diese Transformationen die Positionen der

Eingabe: $start, speed[\,], jonts[\,]$
$start$**:** Anfangszeitpunkt der Erkennung (Samplepunkt)
$speed[\,]$**:** Array mit Handgeschwindigkeit
$joints[\,]$**:** Array mit Fingerkrümmung
Ausgabe: $begin, pos, end, object$
$begin$**:** Anfangszeitpunkt der Loslaßumgebung
pos**:** Loslaßzeitpunkt
end**:** Endzeitpunkt der Loslaßumgebung
$object$**:** Objekt des Ablagepunktes

1: $i \leftarrow start$
2: $joint \leftarrow joints[start]$
3: {**1. Zeitpunkt des Loslaßstelle**}
4: **repeat**
5: $\quad i \leftarrow i + 1$
6: **until** $(joints[start] - jointth) > joint$
7: $pos \leftarrow i$
8: {**2. Endzeitpunkt der Loslaßumgebung**}
9: $mindist[i] \leftarrow computemindist(i)$
10: $minobj[i] \leftarrow computeminobj(i)$
11: **repeat**
12: $\quad i \leftarrow i + 1$
13: $\quad mindist[i] \leftarrow computemindist(i)$
14: $\quad minobj[i] \leftarrow computeminobj(i)$
15: **until** $(mindist[i] > minth) \vee (minobj[i-1] \neq minobj[i])$
16: $end \leftarrow i$
17: {**3. Loslaßzeitpunkt**}
18: **while** $speed[pos] > speed[pos-1]$ **do**
19: $\quad pos \leftarrow pos - 1$
20: **end while**
21: **return** $start, pos, end, computeminobj(pos)$

Algorithmus 4.2: Detektion des Loslaßzeitpunktes ($lookforplace$)

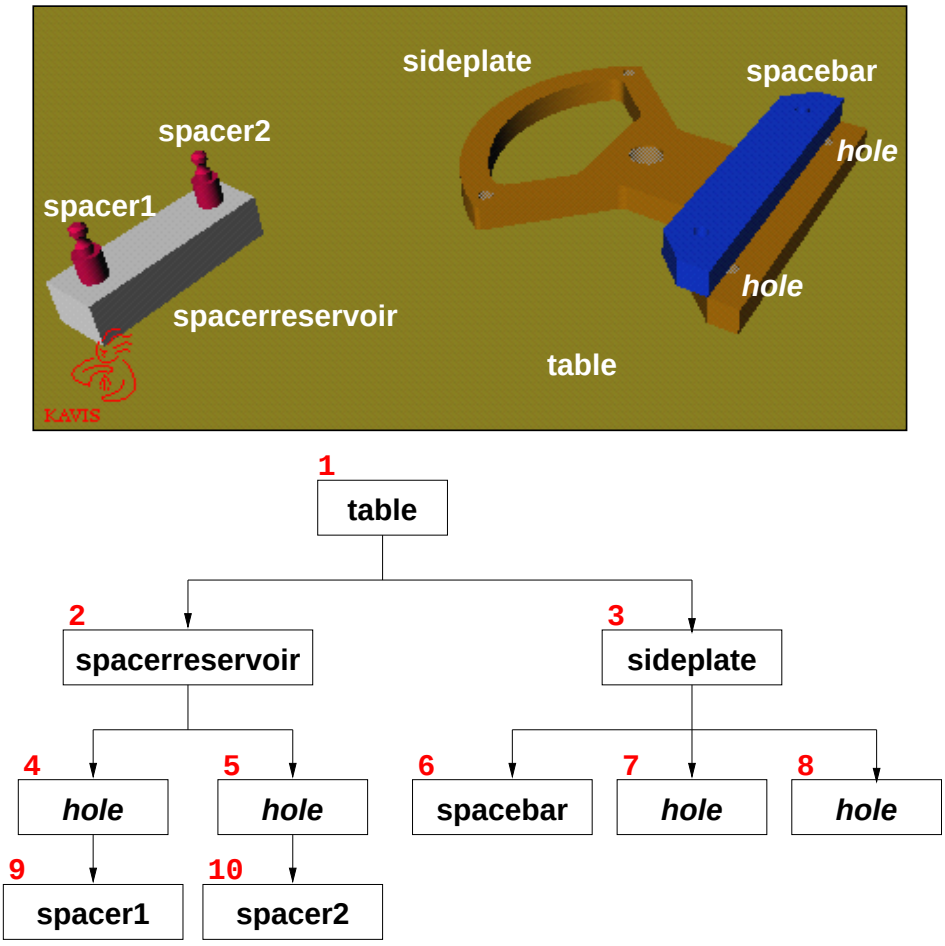


Abbildung 4.5: Szenegraph einer Welt im Simulator

Objekte jeweils relativ zu einem anderen Objekt beschreiben. Abbildung 4.5 zeigt eine einfache simulierte Szene und den dazugehörigen *Szenegraphen*. Man sieht wie in einer *echten* Baumstruktur die Objekte relativ zueinander positioniert sind. Aufbauend auf diesem Szenegraphen wird eine *topologische Sortierung* berechnet, die sicherstellt, daß jeder Knoten in einem Teilbaum hinter dem Wurzelknoten des Teilbaumes einsortiert wird. So ist sichergestellt, daß die topologisch herausragenderen Objekten auch weiter hinten einsortiert werden. Ein weiterer wichtiger Gesichtspunkt besteht in der Unterscheidung von Objektklassen: Wir unterscheiden *statische Objekte*, die in der Szene als unbeweglich angenommen werden (etwa *table*), *dynamische Objekte*, die beweglich sind (etwa *spacer1*), und *technischen Objekten* (etwa *hole*). Technische Objekte sind nicht als eigentliche physische Objekte anzusehen, sondern lediglich abstrakte Konstrukte, die bestimmte Konstruktionseigenschaften anderer Objekte wiedergeben. Diese Objekte können also nur zusammen mit ihrem Vaterknoten bewegt werden.

Diese Kategorisierungen sind wichtig, wenn das gegriffene Objektes bestimmt werden soll. Es ist offensichtlich, daß statische und technische Objekte nicht als gegriffene Objekte in Frage kommen können. Unter den dynamischen Objekten wird nun nach zwei Kriterien ausgewählt: Das primäre Kriterium wählt dasjenige Objekt, das den geringsten Abstand zum TCP aufweist. Insbesondere bei technischen Objekten kann es aber vorkommen, daß mehrere Objekte exakt denselben Abstand zum TCP einnehmen. In diesem Fall wird die topologische Sortierung zu Hilfe gezogen, die dasjenige Objekt bevorzugt, das topologisch herausragender ist, also in der Topologie weiter hinten liegt. Zur Bestimmung des Ablagepunktes werden alle Objektklassen gleichberechtigt herangezogen. Anders als bei der Bestimmung gegriffener Objekte kommt hier aber der minimale Abstand von dem gegriffenen Objekt zum Zuge. Analog ist allerdings die Verwendung der topologischen Sortierung, wenn mehrere Objekte denselben Abstand haben sollten.

Dem Algorithmus 4.3 liegt die Annahme einer Welt nach Definition 4.3 und den damit verbundenen Objekten zugrunde. Zusätzlich wird eine Funktion $distance(objekt1, objekt2)$ unterstellt, die den aktuellen Abstand zwischen den Transformation zweier Objekte berechnet. Diese Funktion wurde im System durch KaVis implementiert. Zusätzlich enthält die Variable *objectheld* ein ein möglicherweise gegriffenes Objekt. Ist kein Objekt gegriffen, so ist der entsprechende Wahrheitswert *falsch*. Je nachdem, ob der Algorithmus mit *computeminobject* oder *computemindist* aufgerufen wird, gibt das Verfahren *minobject* oder *mindist* zurück.

Festlegung der Kontextwechsel

Für das Verständnis der weiteren Vorgehensweise ist der Begriff des *Kontextwechsels* essentiell. Ähnlich dem LFO-System aus [Kuniyoshi 94]) erfolgt die Segmentierung der Benutzervorführung nicht nur an den Greif- und Loslaßpunkten. Zusätzlich wird unterschieden relativ zu welchen Objekten sich der TCP der Hand gegenwärtig bewegt und danach segmentiert. Den Zeitraum, in dem sich die Hand relativ zu einem bestimmten Objekt bewegt, nennt man einen *Bewegungskontext*. Die Stellen an denen sich das relative Objekt ändert bestimmen die *Kontextwechsel*.

Betrachten wir etwa eine typische *Pick&Place*-Aktion, so nähert sich die Hand zunächst dem zu greifenden Objekt, die Hand bewegt sich also relativ zum gegriffenen Objekt. Wir

Eingabe: $pos, world$
pos**:** Samplepunkt
$world$**:** Welt nach Definition 4.3
Ausgabe: $mindist, minobj$
$mindist$**:** Minimaler Abstand des TCP von beliebigem Objekt
$minobj$**:** Objekt mir minimalem Abstand

```
1:  minobj ← last(O(world))
2:  if heldobject then
3:    mindist ← distance(heldobject, last(O(world)))
4:  else
5:    mindist ← distance(tcppos, last(O(world)))
6:  end if
7:  for each object in reverseO(world) do
8:    if heldobject then
9:      {Ablagepunkt bestimmen:}
10:     if distance(heldobject, objekt) < mindist then
11:       mindist ← distance(heldobject, objekt)
12:       minobject ← object
13:     end if
14:   else
15:     {Gegriffenes Objekt bestimmen:}
16:     if isdynamic(object) ∨ istechnical(objekt) then
17:       if distance(tcppos, objekt) < mindist then
18:         mindist ← distance(heldobject, objekt)
19:         minobject ← object
20:       end if
21:     end if
22:   end if
23: end for
24: return minobj, mindist
```

Algorithmus 4.3: Minimaler Abstand (*computeminobjekt*, *computemindist*)

sprechen hier von der *Annäherung* der Hand an das Objekt. Ist das Objekt erst gegriffen, entfernt sich die Hand beim *Abrücken* vom Ort des Greifens. Auch in dieser Phase bewegt sich die Hand noch relativ zur ursprünglichen Position des gegriffenen Objektes. Doch jetzt wechselt der Bewegungskontext, da sich die Hand nun dem Ablagepunkt annähern muß. Ist das Objekt abgelegt, rückt die Hand von der Ablageposition ab und entfernt sich von dem Objekt. Eine *Pick&Place*-Aktion enthält also in der Regel zwei Bewegungskontexte.

Die Festlegung der Kontextwechsel ist relativ unkritisch, da diese keine unmittelbare Auswirkung auf die semantische Analyse des Systems haben. Allerdings können ungünstig gewählte Kontextwechsel bei der Ausführung der generierten Makrooperatoren kritisch sein. Der ursprüngliche Ansatz, die Kontextwechsel an den Punkt der maximalen Geschwindigkeit der Hand zwischen Greif- und Loslaßoperationen zu legen, wurde aufgrund der in Abschnitt 4.2.1 beschriebenen Ungenauigkeiten des Sensors in diesem Wert aufgegeben. Letztendlich erwies es sich als zuverlässiger den Punkt zu wählen, an dem der euklidsche Abstand zu Ablage- und Greifpunkt maximal ist.

Linearisierung der Trajektorie

In [Holle 98] wurde ein Segmentierungsverfahren implementiert, das es erlaubt dreidimensionale Trajektorien in lineare Segmente zu zergliedern. Das Verfahren beruht darauf hierarchisch Punkte zusammenzufassen, die von einer gedachten Linie nur einen bestimmten maximalen Abstand aufweisen. Aufbauend auf dieser Linearisierung werden von [Holle 98] elementare Bewegungsoperatoren nach Abbildung 3.3 abgeleitet. Diese Linearisierung und Klassifikation der Elementaroperatoren wurde auch in dieser Arbeit verwendet, um eine Feinsegmentierung der Trajektorie vorzunehmen.

4.2.3 Segmentierung einer Sequenz

Der Segmentierung der Benutzervorführung kommt eine zentrale Bedeutung in der Vorverarbeitung der Trajektorie zu. Wie in Abschnitt 4.1 dargelegt, gilt es im Segmentierungsschritt eine Segmentierungshierarchie zu erzeugen, die den Voraussetzungen für den Analyseschritt in Abschnitt 4.4 hinreichend ist. Das bedeutet, daß die zunächst Benutzervorführung operationalisiert und schließlich durch eine hierarchische Struktur erklärt werden muß.

Ereignisgestützte Segmentierung

Die erfolgreiche Detektion der wichtigsten *Schlüsselereignisse* legt bereits die Grundlage zu der Segmentierung der Benutzervorführung. Ähnlich zu [Kuniyoshi 94] dient die Erkennung der Schlüsselereignisse dem Aufbau einer hierarchischen Segmentierung. Abbildung 4.6 zeigt wie auf unterster Ebene Ereignisse datengetrieben aufgedeckt werden. Wie in Abschnitt 4.2.2 beschrieben werden verschiedene Ereignisse, wie Kontextwechsel, Greif- und Loslaßpunkte sowie Trajektorienpunkte extrahiert und auf die Benutzervorführung abgebildet.

Eingabe: $superclass, start, end$
$superclass$**:** Klasse des zu segmentierenden Segmentes
$start$**:** Anfang des zu segmentierenden Segmentes
end**:** Ende des zu segmentierenden Segmentes
Ausgabe: $segments$
1: $segments \leftarrow \emptyset$
2: $lastsample \leftarrow start$
3: **for** $sample$ **from** $start + 1$ **to** end **do**
4: **if** $fittingevent(sample, superclass)$ **then**
5: $class \leftarrow detsegclass(lastsample, sample, superclass)$
6: **collect** $[class, lastsample, sample]$ **into** $segments$
7: $lastsample \leftarrow sample$
8: **end if**
9: **end for**
10: **return** $segments$

Algorithmus 4.4: Generische Beschreibung einer Segmentierfunktion ($xsegment$).

Wie in Algorithmus 4.4 beschrieben, sucht eine ereignisgestützte *Segmentierfunktion* in einem vorgegebenen Segment einer Benutzervorführung nach Elementarereignissen. Dabei sind für jede Segmentierfunktion nur spezifische Ereignisse relevant. So ist etwa eine Segmentierfunktion für Kontextwechsel nur für Kontextwechsel empfänglich, andere reagieren auf Trajektorienpunke oder Greifoperationen. Dementsprechend ist die Funktion $fittingevent$ für jede Segmentierfunktion getrennt zu bestimmen. Sie legt fest, an welchen Punkten ein neues Untersegment beginnt. Diese Funktion kann von der Klasse des ursprünglichen Segmentes abhängig sein, da etwa ein Greifsegment nur eine Greifoperation enthalten sollte und deswegen Loslaßoperationen ignoriert werden können. Ist erst einmal die Lage eines Untersegmentes bestimmt, gilt es nun festzulegen, um welche Segmentklasse

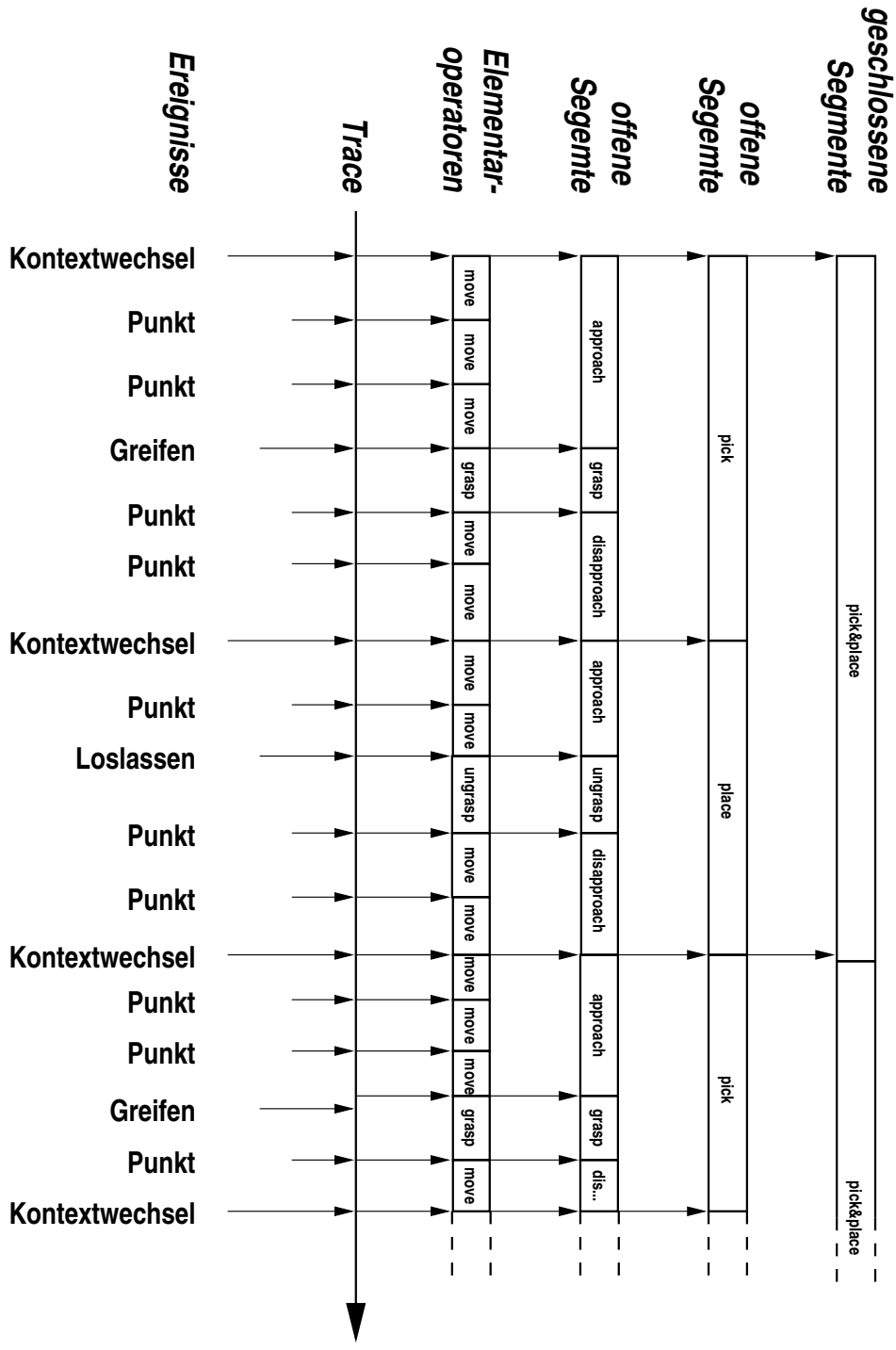


Abbildung 4.6: Ereignisgetriebenes hierarchisches Segmentierungsmodell

es sich handelt. So kann etwa ein Segment zwischen zwei Kontextwechseln eine Greif- (engl. *pick*) aber ebenso eine Ablegephasen (engl. *place*) darstellen. Diese könnten etwa dadurch differenziert werden, daß eine Greifphasen eine Greifoperation (engl. *grasp*), während Loslaßphasen Loslaßoperationen (engl. *ungrasp*) enthalten. Die Segmentierfunktion für die Trajektorie greift auf die vorher bestimmten elementaren Bewegungsoperatoren zurück. Man sieht also, daß auch die Funktion $detsegclass$, welche die Klasse eine Untersegmentes bestimmt, für jeden Segmentierer getrennt programmiert werden muß. Eine Segmentierfunktion, wie sie in Algorithmus 4.5 generisch beschrieben wird, erzeugt also auf der Basis zuvor berechneter Ereignisse eine Untersegmentierung eines Segmentes einer Benutzervorführung und gibt eine Liste mit den Untersegmenten mit der jeweiligen Segmentklasse zurück.

Genese der Erklärungshierachie

Um daraus eine Erklärungshierarchie nach Abschnitt 4.1 zu generieren, ist es notwendig solche Segmentierfunktionen in sich verschachtelt bzw. rekursiv auszuführen sind. Abbildung 4.6 zeigt, auf der obersten Ebene *Pick&Place*-Operationen. Diese setzen sich aus mehreren Kontexttexten zusammen, die von einem Greifsegment (engl. *pick*) eingeleitet und von einer Loslaßoperation (engl. *pick*) abgeschlossen werden. Diese Segmente untergliedern sich weiter in die Annäherung (engl. *approach*) an den Greif- bzw. Loslaßpunkt, die eigentliche Greifoperation und das Abrücken (engl. *disapproach*). Auf unterster Ebene liegt schließlich die Segmentierung der Trajektorie. Diese Einteilung ist vergleichbar mit der von [Kuniyoshi 94]) und stellt im Vergleich zum Segmentierungsmodell von [Holle 98] (siehe Abbildung 3.2) eine Weiterentwicklung dar. Damit ergeben dich für das in Abbildung 4.6 gegebene Beispiel folgende verschachtelte Ebenen von Segmentierungen:

1. **Pick&Place-Segmente.** Segmente vom Greifen bis zum Ablegen eines Objektes.
2. **Kontextwechsel-Segmente.** Segmente zwischen Kontextwechseln.
3. **Griff-Segmente.** Segmente eines Kontextes, die durch Greif- oder Loslaßoperationen unterteilt werden.
4. **Trajektorien-Segmente.** Liniensegmente einer Trajektorie.

Bei der Betrachtung dieses Beispiel wird aber schnell klar, daß diese Segmentierung zwar sinnvoll, aber durchaus nicht notwendigerweise genau so vorgenommen werden muß. Etwa ist der Kontextwechsel-Segmentierung unter bestimmten Umständen verzichtbar. Andererseits kann es aber durchaus sinnvoll werden, die Elementaroperatoren noch feiner zu untergliedern, etwa auf der Basis von bestimmten Regelzuständen während der Benutzervorführung. Es ist offensichtlich, daß ein modularer Ansatz, mit dem sich unterschiedliche Segmentierungsebenen kombinieren und ergänzen lassen, gewisse Vorteile ausweist. Wie in Abschnitt 3.1 beschrieben wurden, gilt es aber auch unterschiedliche Eingabedaten zu verarbeiten, die möglicherweise unterschiedliche Ereignisse zur Segmentierung und andere Segmentierungshierarchien erfordern. Auch hierzu erscheint ein flexibles modulares Segmentierungsverfahren unverzichtbar.

Eingabe: $segmenter, superclass, start, end$
segmenter: Noch ausstehende Segmentierer
class: Klasse des zu segmentierenden Segmentes
start: Anfang des zu segmentierenden Segmentes
end: Ende des zu segmentierenden Segmentes
Ausgabe: *segmentation*

```
1:  constant algorithm ← xalgorithm
2:  result ← ∅
3:  segments ← xsegment(superclass, start, end)
4:  if (length(segments) > 0) ∧ (length(segmenters) > 0) then
5:    for each s in segments do
6:      collect
7:        call (first segmenter) with (rest segmenter, s.class, s.start, s.end)
8:      into result
9:    end for
10: end if
11: if length(result) = 1 then
12:   return first result
13: else
14:   return [algorithm, superclass, start, end, result]
15: end if
```

Algorithmus 4.5: Generische Beschreibung eines Segmentierers ($xsegmenter$).

Deswegen wurde ein modulares Verfahren implementiert, mit dem sich sog. *Segmentierer* miteinander derart kombinieren lassen, daß sie sich rekursiv aufrufen und damit die rekursive Struktur erzeugen, wie sie Abbildung 4.6 zeigt. Zu diesem Zweck wird dem Algorithmus 4.5 eine Liste mit nachfolgenden Segmentierern erzeugt, die jeweils die Untersegmentierung eines Teilsegmentes erzeugt. Jedem Segmentierer ist eine Segmentierfunktion nach dem Muster von Algorithmus 4.4 zugeordnet. Diese berechnet zu einem gegebenen Segment die der Segmentierungsebene angemessenen Untersegmente. Nach der Berechnung der Untersegmente wird der erste übergebene Segmentierer mit den verbleibenden, nachfolgenden Segmentierern und den Parametern der einzelnen Untersegmente aufgerufen. So werden sukzessive alle übergebenen Segmentierer aufgerufen, bis entweder der letzte Segmentierer aufgerufen wurde, oder das Resultat einer Segmentierfunktion ein leeres Ergebnis ergibt. Das Verfahren gibt eine verschachtelte Liste mit der Segmentierungshierarchie des Segmentes zurück. Ein Spezialfall liegt vor, wenn die Untersegmentierung nur ein Segment ergibt; dann wird nämlich die aktuelle Segmentierungsebene übersprungen und das eine Untersegment direkt als Resultat zurückgeben ohne eine Struktur für die aktuelle Ebene zu erzeugen. Dies ist etwa bei Greifoperationen sinnvoll, die auf derselben Ebene wie Annäherungs- und Abrücksegmente liegen, die weiter durch Trajektorienpunke untersegmentiert werden, während es keinen Sinn macht die nach unserem Modell bereits elementare Greifoperation selbst zu untergliedern. Diese elementare Greifoperation wird also durch den Trajektoriensegmentierer nicht weiter untergliedert und somit eine Ebene nach oben durchgereicht.

Jeder Segmentierer benötigt aber nicht nur eine eigene Segmentierfunktion, sondern den einzelnen Segmentierungsebenen, denen die Segmentierer zugeordnet sind, angemessene inhaltliche Analysealgorithmen. In Abschnitt 4.4 wird gezeigt werden, daß offene und geschlossene Segmente (siehe Definition 4.19) und andere Segmentierungsebene verschiedene Algorithmen zur inhaltlichen Analyse benötigen. Diesem Umstand wird durch die je nach Segmentierer verschiedenen Konstante *algorithm* Rechnung getragen, die den entsprechenden Funktionsaufruf enthält. Dieser wird bei der Definition eines Segmentierers für diesen einmal global als Konstante festgelegt.[6] Tabelle 4.1 zeigt Beispielhaft, wie die Segmente der einzelnen Ebenen Algorithmen aufgrund ihrer Offenheit bzw. Geschlosseheit zugeordnet sind. Die aufgelisteten Algorithmen werden in Abschnitt 4.4 motiviert und vorgestellt.

Ebene	**Segmentart**	**Algorithmus**
Pick&Place-Segmente	geschlossen	*analyzeclosed* (4.12)
Kontextwechsel-Segmente	offen	*analyzeopen* (4.13)
Griffsegmente	offen	*analyzeopen* (4.13)
Trajektoriensegmente	offen	*analyzeelemtary* (4.14)

Tabelle 4.1: Zuordnung von Segmentierungsebenen zu Analysealgorithmen

4.2.4 Semantische Erschließung

Nachdem die vorangehenden Abschnitte sich noch weitgehend auf die Ebene der Sensordaten bzw. auf direkt daraus abzuleitende Ereignisse bezogen, beginnt mit der semantischen Erschließung der Benutzervorführung die abstrakte Bearbeitung der Benutzervorführung. Die folgenden Betrachtungen stützen sich ausschließlich auf die Definitionen in Abschnitt 4.1. Dementsprechend wird in der Folge beschrieben, wie die bislang in Form von Sensordaten verarbeiteten Benutzervorführungen in eine semantische Repräsentation in Form einer *Episode* überführt wird, also eine Folge von *Zuständen* und *Manipulationen.*

Damit erfolgt die Repräsentation des Umweltzustandes in Form von Relationen nach Definition 4.2. Mit Hilfe dieser Relationen wird der Zustand der Welt beschrieben. Der initiale Zustand der Welt wird berechnet und im Verlauf der Vorführung inkrementell weitergeführt, bis schließlich der Endzustand angenommen wird Abbildung 4.8 zeigt eine schematische Darstellung dieses Vorgangs, während Abbildung 4.11 eine exemplarische Episode mit ausgewählten Relationen und Manipulationen darstellt. Da die Sensordaten mit einer sehr hohen Rate aufgezeichnet werden, würde es einen unverhältnismäßig hohen Aufwand bedeuten, den Umweltzustand in jedem Samplepunkt neu zu bestimmen. Aus diesem Grund führt man *Synchronisationspunkte* ein, die festlegen, an welchen Punkten in der Benutzervorführung eine Aktualisierung des Umweltzustandes sinnvoll und notwendig ist. Abschnitt 4.2.3 beschreibt, wo die Synchronisationspunkte sinnvoll sind und wie sie im konkreten System gesetzt werden.

[6] In diesem Punkt hat sich die in Abschnitt 5.2.2 motivierte dynamische Programmiersprache CLOS besonders bewährt, da ihr Makrokonzept einfache Mechanismen bereitstellt, funktional ähnliche Prozeduren ohne Codeduplizierung zu definieren

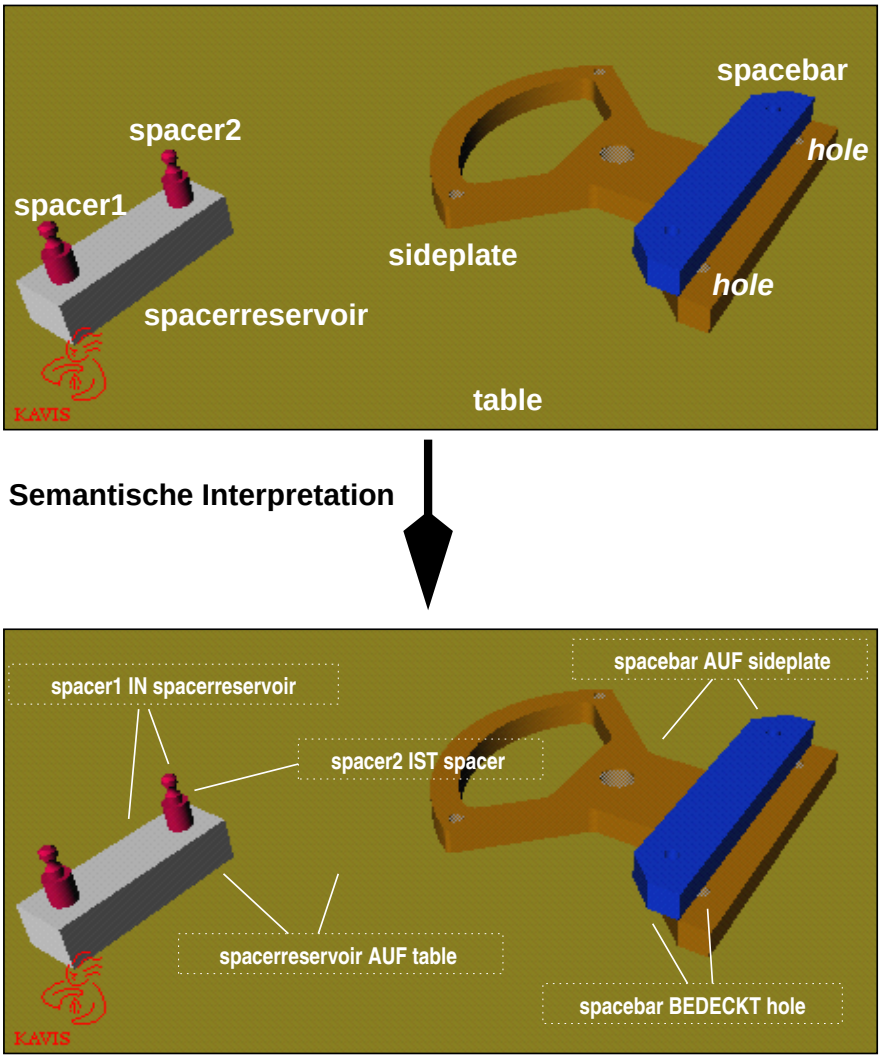


Abbildung 4.7: Semantische Erschließung einer statischen Szene

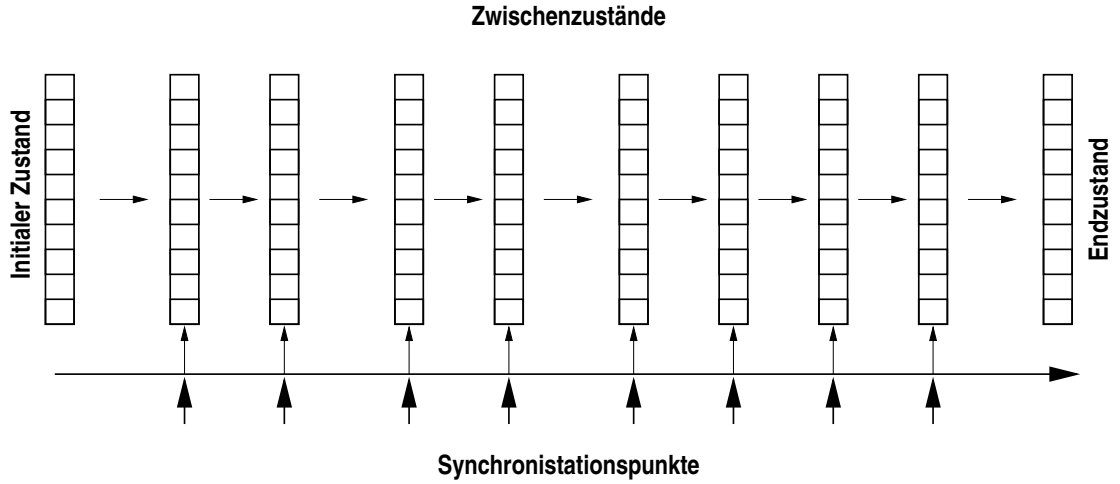


Abbildung 4.8: Skizze der inkrementellen semantischen Erschließung

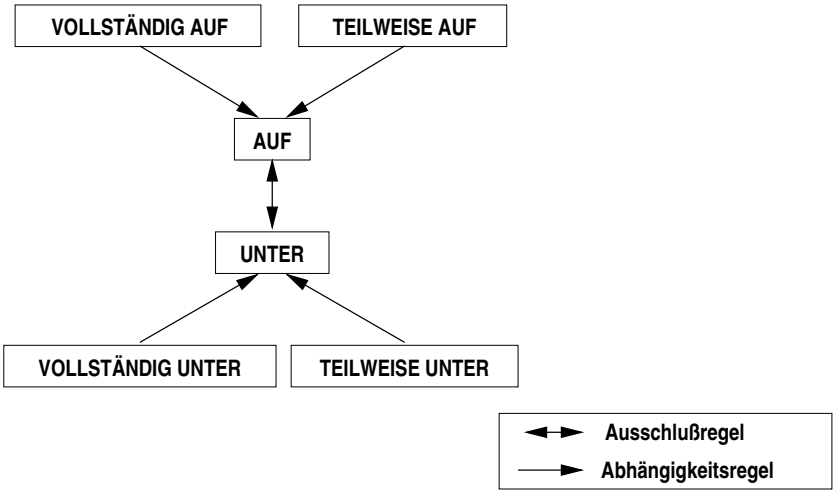


Abbildung 4.9: Ausschluß- und Abhängigkeitsregeln

Eingabe: $state, object1, object2$
$state$**:** Aktueller Auswertungszustand
$object1$**:** Erstes Objekt
$object2$**:** Zweites Objekt
Ausgabe: $state$

```
 1: for each rule in regelbasis do
 2:   {if- oder else-Zweig auswählen.}
 3:   if condition(rule, object1, object2) then
 4:     actions ← ifbranch(rule)
 5:   else
 6:     actions ← elsebranch(rule)
 7:   end if
 8:   {Aktionen des Zweiges ausführen.}
 9:   for each action in actions do
10:     if issetaction(action) then
11:       {set-Aktion ausführen.}
12:       relation ← relationofaction(action)
13:       value ← valueofaction(action)
14:     else
15:       {evaluate-Aktion ausführen.}
16:       relation ← relationofaction(action)
17:       value ← evaluate [relation, object2, object1, value]
18:     end if
19:     {Resultat in Zustand sichern.}
20:     store [relation, object1, object2, value] in state
21:     {Bei Bedarf Wert für symmetrische Relation sichern.}
22:     if symmetric(relation) then
23:       store [symmetric(relation), object2, object1, value] in state
24:     end if
25:   end for
26: end for
27: return state
```

Algorithmus 4.6: Berechnung der Relationen zwischen zwei Objekten (*snapstate*).

In Abschnitt 4.2.1 wurde beschrieben wie aus der physischen Vorführung des Benutzers eine Simulation generiert wurde. Innerhalb dieser Simulation standen verschiedene Relationen zur Verfügung (siehe [Holle 97]), die im jeweiligen Zustand der Simulation abgefragt werden konnte. Um nun den Zustand der Umwelt und die Manipulationen darauf im Sinne der Definition 4.8 bzw. 4.11 verfolgen zu können, sind drei Schritte notwendig:

1. **Initialer Zustand.** Zunächst muß der initiale Zustand der Welt berechnet werden.
2. **Inkrementelle Änderungen.** Aufbauend auf dem initialen Zustand werden die neuen Zustände inkrementell berechnet.
3. **Bestimmung der Manipulation.** Abschließen ist die Manipulation zwischen der Folge der einzelnen Zustände zu bestimmen sowie die Gesamtmanipulation der Episode.

Regelbasierte Auswertungsoptimierung

Da die Abfrage verschiedener Relationen in der Simulation mitunter recht kostenintensiv ist, wurde versucht, die Anzahl der tatsächlichen Anfragen an die Simulation so weit als möglich zu reduzieren. Um dies zu tun, macht sich das implementierte Verfahren die Eigenschaft der Relationen zunutze, daß diese sich einerseits teilsweise gegenseitig ausschließen, etwa die Relation *auf* und *unter*, andererseits sich gegenseitig bedingen, etwa *in* und *vollständig in*. Auf dieser Grundidee aufbauend kann man *Ausschlußregeln*, *Abhängigkeitregeln* und *Symmetrieregeln* aufstellen (siehe Abbildung 4.9):

- **Ausschlußregel:**

```
if (evaluate 'auf') then
   set 'unter' to false;
else
   evaluate 'unter';
endif
```

- **Abhängigkeitsregel:**

```
if (evaluate 'auf') then
   evaluate 'vollständig auf';
   evaluate 'teilweise auf';
else
   set 'vollständig auf' to false;
   set 'teilweise auf' to false;
endif
```

- **Symmetrieregel:**

```
if (evaluate 'auf(o1,o2)') then
   set 'unter(o2,o1)' to true;
endif
```

Nach diesem Konstruktionsprinzip läßt sich eine Regelbasis erstellen, welche die absolute Anzahl der benötigten Evaluationsaufrufe an die Simulation auf weniger als fünf Prozent der auszuwertenden Relationen reduzieren. Die Regelbasis bezieht sich immer auf die Relationen zwischen zwei Objekten. Die Auswertung der Regelbasis erfolgt sequentiell. Allerdings ist zu beachten, daß die resultierenden Wahrheitswerte aus den Evaluationsaufrufen bis zur nächsten Zustandsauswertung gespeichert werden, so daß ein erneuter Aufruf einer Evaluierung innerhalb *einer* Zustandsauswertung mit denselben Parametern, nicht tatsächlich ausgeführt wird, sondern lediglich der zwischengespeicherte Wert ausgelesen wird. In der Ausführungsbedingung sind zusätzlich Negationen sowie verschachtelte Konjunktionen und Disjunktionen zulässig, so daß sich auch komplexere Abhängigkeiten ausdrücken lassen. Bei der Auswertung der Ausführungsbedingungen werden die Relationen wie in einer *evaluate*-Aktion ausgewertet, also gegebenenfalls aus dem Zwischenspeicher ausgelesen oder tatsächlich abgefragt, aber dann auch im Zustand gespeichert. Symmetrieregeln werden in einer seperaten Regelbasis gesichert. Immer wenn eine neue Ausprägung einer Relation im Zustand gesichert wird, wird die Basis mit den Symmetrieregeln abgefragt und für den Fall, daß eine symmetrische Relation existiert, auch der Wert für die symmetrische Relation und mit vertauschten Objekten als Ausprägung im Zustand gesichert.

Eingabe: *world*
*world***:** Welt im initialen Zustand
Ausgabe: *state*
1: $state \leftarrow \emptyset$
2: **for each** $object1$ **in** $\mathcal{O}(world)$ **do**
3: **for each** $object2$ **in** $\mathcal{O}(world)$ **do**
4: **if** $object1 \neq object2$ **then**
5: $state \leftarrow snapstate(state, object1, object2)$
6: **end if**
7: **end for**
8: **end for**
9: $S(world) \leftarrow state$
10: **return** *state*

Algorithmus 4.7: Berechnung des initialen Weltzustandes (*computeinistate*).

Berechnung des initialen Weltzustandes

Zur Anwendung des in der Arbeit implementierten Verfahrens ist es erforderlich den Weltzustand zu jedem Zeitpunkt der Vorführung zu bestimmen. Um möglichst effizient zu arbeiten, geschieht dies weitgehend inkrementell. Lediglich der initiale Weltzustand, also der erste Zustand einer Episode muß aufwendig bestimmt werden, indem der Algorithmus 4.6 zwischen allen Objekt ausgeführt wird, wie es in Algorithmus 4.7 gezeigt wird. Damit ergibt sich ein quadratischer Aufwand für die Berechnung des initialen Weltzustandes. Allerdings muß man sich vor Augen führen, daß die symmetrischen Relationen und die Pufferung der Ergebnisse aus Symmetrieregeln dazu führt, daß später ausgewertete

Objekte nur einen geringeren Aufwand verursachen, da alle symmetrischen Relationen aus den Berechnungen für die früheren Objekte bereits bekannt sind. Da die Mehrzahl der berechneten Relationen symmetrisch sind, ergibt sich hier eine erhebliche Einsparung des Berechnungsaufwandes. Zudem wurde im System auch ein Mechanismus implementiert, der einmal berechnete initiale Umweltzustände auf der Festplatte sichert, so daß die Berechnung des initialen Weltzustandes mit dem hohen Aufwand nur selten wirklich notwendig wird.

Eingabe: $world, focus$
$world$**:** Welt der Vorführung
$focus$**:** Liste mit Objekten im Update-Fokus
Ausgabe: $state$

```
 1: state ← ∅
 2: oldstate ← S(world)
 3: {1. Primären Fokus auswerten.}
 4: for each object1 in focus do
 5:    for each object2 in O(world) do
 6:       if object1 ≠ object2 then
 7:          state ← snapstate(state, object1, object2)
 8:       end if
 9:    end for
10: end for
11: {2. Sekundären Fokus auswerten.}
12: focus2 ← ∅
13: for each relation in delta(oldstate, state) do
14:    add relation.object2 to focus2
15: end for
16: for each object1 in focus2 do
17:    for each object2 in focus do
18:       state ← snapstate(state, object1, object2)
19:    end for
20: end for
21: {3. Unveränderte Relationen übernehmen.}
22: for each r in oldstate do
23:    if not ([r.relation, r.object1, r.object2, ∗] in state) then
24:       store r in state
25:    end if
26: end for
27: focus ← ∅
28: S(world) ← state
29: return state
```

Algorithmus 4.8: Inkrementelle Fortschreibung des Weltzustandes (*snapshot*).

Inkrementelle Fortschreibung des Umweltzustandes

Nachdem der initiale Zustand einer Welt bekannt ist, gilt es nun die semantische Beschreibung des Umweltzustandes an den Synchronisationspunkten zu aktualisieren (siehe Abbildung 4.8). Es ist hier aber nicht notwendig den Zustand an jedem Synchronisationspunkt komplett neu zu berechnen; vielmehr läßt sich ein Verfahren angeben, das es erlaubt den Umweltzustand mit linearem Aufwand zu aktualisieren.

Algorithmus 4.8 basiert darauf, daß zwischen den einzelnen Synchronisationspunkten der sogenannte *Update-Fokus* verwaltet wird. Diese globale Variable beschreibt alle Objekte, für die sich zwischen den einzelnen Synchronisationspunkten Veränderungen ergeben haben könnten, etwa indem sie manipuliert wurden. Basierend auf dieser Fokus-Variable läßt sich ein Verfahren in drei Schritten angeben:

1. **Auswertung des Primärer Update-Fokus.** Zunächst werden die Relationen aller Objekte des Update-Fokus zu allen Objekten der Welt neu berechnet.

2. **Auswertung des Sekundärer Fokus.** Im zweiten Schritt wird der sekundäre Fokus bestimmt, der alle Objekte enthält, für die sich in dem bisherigen Aktualisierungsvorgang Veränderungen bzgl. des primären Fokus ergeben haben. Für die Objekte des sekundären Fokus werden nun alle Relationen zu Objekten des primären Fokus neu bestimmt.

3. **Propagierung des Restzustandes.** Alle Relationen, die durch diese Prozedur nicht bestimmt wurden, werden im letzten Schritt aus dem alten Umweltzustand extrahiert und in den neuen propagiert, so daß sich stets eine vollständige Repräsentation dieses Zustandes ergibt.

Eingabe: $state1, state2$
$state1$**:** Zustand vorher
$state2$**:** Zustand hinterher
Ausgabe: *delta*
1: $delta \leftarrow \emptyset$
2: **for each** *relation* **in** $state2$ **do**
3: **if not** (*relation* **in** $state1$) **then**
4: **add** *relation* **to** *delta*
5: **end if**
6: **end for**
7: **return** *state*

Algorithmus 4.9: Manipulation zwischen zwei Zuständen (*delta*).

Berechnung der Manipulationen

Zur Berechnung der Manipulationen zwischen einzelnen Zuständen wird ein einfaches Verfahren verwendet, die es in Algorithmus 4.9 dargestellt wird. Diese folgt im wesentlichen der Definition 4.11 der Manipulation, die alle Ausprägungen von Relationen, die im Folgezustand aber nicht im Ausgangszustand vorkommen als Veränderung erkennt. Damit läßt

sich die Manipulation der Gesamte Episode als $\Delta(S_0, S_n)$, wobei S_0 der initiale und S_n der Endzustand der Episode bezeichnet.

4.3 Generalisierung

Um aus der Benutzervorführung ein generalisiertes Makroprogramm erzeugen zu können gilt es im ersten Schritt, die Parameter der Benutzervorführung auf ihre essentiellen Werte zu reduzieren. Um das Makroprogramm auf veränderten Umwelten ausführen zu können, ist es zunächst nötig die Trajektorie der Vorführung zu verallgemeinern, was in Abschnitt 4.3.1 beschrieben wird. Wichtiger aber ist die Extraktion der essentiellen Bestandteile der Ausführungskontexte, also der Generalisierung der semantischen Beschreibung der Kontexte, um Ausführ- und Verzweigungsbedingungen bestimmen zu können. Die Generalisierung der Ausführungskontexte und der Manipulation der Benutzervorführung wird in Abschnitt 4.3.2 geschildert.

Zu diesem Zweck kommen primär erklärungsbasierte Ansätze nach Abschnitt 2.1.3 zum Einsatz. Die Komplexität der Bereichtheorie und der Umwelt läßt aber keine eindeutige Erklärung zu, weswegen die erklärungsbasierten Verfahren durch informationstheoretische Ansätze nach Abschnitt 2.1.4 unterstützt werden müssen, um die sinnvollste Erklärungshypothese wählen zu können.

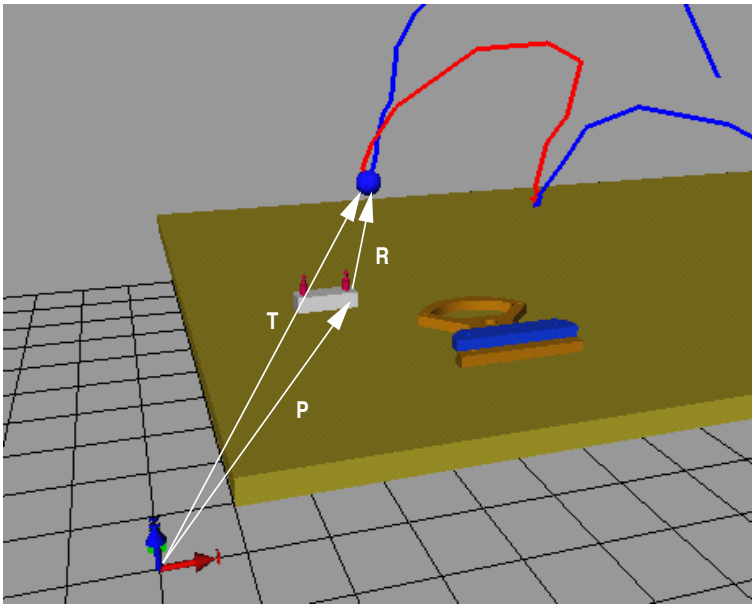


Abbildung 4.10: Generalisierung der Trajektorie über die Objektpositionen

4.3.1 Trajektorie

Nach der Generierung der Erklärungsstruktur, wie in Abbildung 4.6 dargestellt, läßt sich die Trajektorie über die Objektpositionen generalisieren und wird damit auf anderen Umwelten ausführbar. Die zentrale Erklärungsebene bei diesem Generalisierungsschritt ist die Segmentierung nach Kontextwechseln (siehe Tabelle 4.1). Auf dieser Ebene werden Segmente eingeteilt, die sich zwischen den Kontextwechseln erstrecken. Wir hatten die

zwischen den Kontextwechseln liegenden Bewegungskontexte so motiviert, daß sich die Hand innerhalb dieser Bewegungskontexte immer relativ zu einem bestimmten Objekt bewegt, etwa dem zu greifenden Objekt oder dem Ablagepunkt. Auf diesen Gedanken aufbauend kann die Trajektorie über die Objektpositionen generalisiert werden, indem die Positionen der einzelnen Trajektorienpunkte nicht mehr absolut sondern relativ zu dem einen Bewegungskontext bestimmenden Objekt gesichert werden. Diese Umrechnung wird von Abbildung 4.10 verdeutlicht und erfolgt nach folgender Formel:

$$R = P^{-1} \cdot T \tag{4.4}$$

Dabei beschreibt T die Transformationsmatrix des Trajektorienpunktes und P die des Objektes. Beide Transformationsmatrizen sind analog zu Gleichung 4.3 definiert.

4.3.2 Ausführungskontext und Manipulationen

Auch bei der Generalisierung der Ausführungskontexte ist eine Erklärungsstruktur wie in Abbildung 4.6 gezeigt hilfreich. Es ist aber einsichtig, daß niemals eindeutig geschlossen werden kann, welche Elemente der Ausführungskontexte tatsächlich für die Vorführung relevant sind, da Benutzervorführungen prinzipiell nicht eindeutig erklärbar sind. Um aber wenigstens eine möglichst plausible Erklärungshypothese zu wählen, greift das implementierte System auf informationstheoretische Ansätze zurück, die es erlauben die plausibelste Hypothese zu wählen.

Informationstheoretische Einbettung des Problems

Abbildung 4.11 zeigt eine Folge von Greif- und Loslaßpunkte mit drei *Pick&Place*-Operationen, die sich auf dieselben Objekte bezieht, wie Abbildung 4.7. In dem initialen Weltzustand liegt die *Spacebar* auf der *Sideplate* und die beiden *Spacer* stecken im *Spacerreservoir*. Zunächst wird die *Spacebar* entfernt, um die beiden Löcher (engl, *hole1, hole2*) freizulegen und anschließend die beiden *Spacer* in diese Löcher einfügen zu können. Die Abbildung zeigt wie sich ausgewählte Relationen über die Episode verändern und damit Teil der Manipulation der Episode werden.

Die Grundidee der Hypothesenauswahl besteht nun darin, die Veränderungen als stochastischen Prozeß im Sinne von Definition 2.2 aufgefaßt. Die Benutzervorführung wird also als Kommunkationsprozeß zwischen Benutzer und PbD-System aufgefaßt. Die Zustände des Prozesses werden durch die Relationen der Zustände der Episoden gegeben. Durch die Schätzer aus den Gleichungen 2.2 und 2.4 lassen sich empirisch die Wahrscheinlichkeit für bestimmte Zustände und Zustandsübergänge und damit auch deren Informationgehalt nach Gleichung 2.1 bestimmen.

Allerdings muß hier hervorgehoben werden, daß die direkte Abbildung eines stochastischen Prozesses auf das vorliegende Problem eine starke Vereinfachung der Situation darstellt. Schließlich wird ein Episodenzustand nicht nur durch eine Relation beschrieben, sondern durch eine große Menge von Relationen beschrieben. Auch Zustandübergänge ergeben sich nicht aus der Änderung einer Relation. Vielmehr können sich mehrere Relationen gleichzeitig ändern. In der tatsächlichen Implementierung wird also nicht die

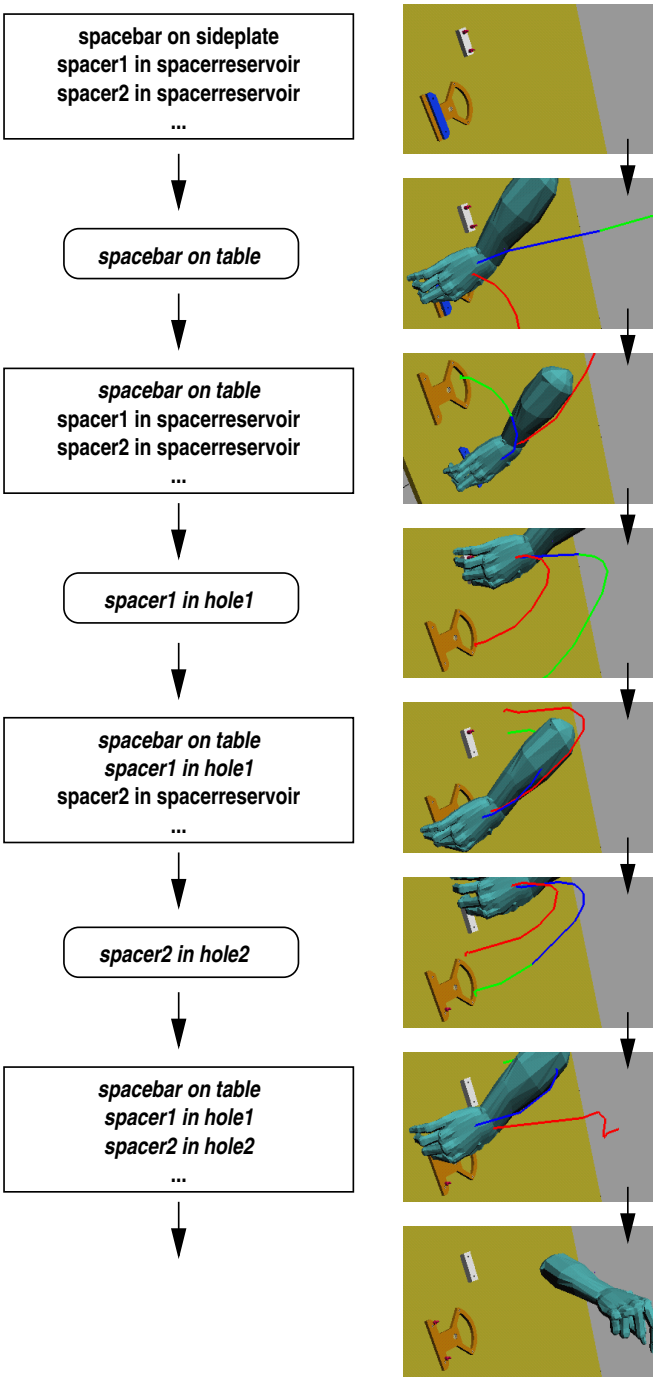


Abbildung 4.11: Episode und deren Manipulationen

Wahrscheinlichkeit eines Gesamtzustandes geschätzt sondern vielmehr die Wahrscheinlichkeit, daß eine zufällig betrachtete Relation, genau die bezeichnete Relationen mit dem gegebenen Wahrheitswert ist. Analog wird die Wahrscheinlichkeit bestimmt, daß eine beobachtete Änderung einer Relation die bezeichnete Relationen zwischen zwei gegebenen Objekten betrifft.

Zur Auswahl der Hypothese wird nun das in Abschnitt 2.1.2 dargestellt Verfahren *Occam's Razor*. Dieser Ansatz versucht mit minimaler Komplexität der Hypothese ein Maximum an Information zu konservieren. Der Sinn des Verfahren liegt darin möglichst kompakte Hypothesen zu erzeugen, ohne dabei die relevante Information zu verlieren.

Eingabe: $state, filter$
$state$**:** Zustand
$filter$**:** Filterobjekte
Ausgabe: *selectedrelations*
1: $relations \leftarrow \emptyset$
2: {**Beteiligte Relationen filtern**}
3: **for each** *relation* **in** *state* **do**
4: **if** ($relation.object1$ **in** $filter$) $\vee$ ($relation.object2$ **in** $filter$) **then**
5: **collect** *relation* **into** *relation*
6: **end if**
7: **end for**
8: {**Relationen nach Informationsgehalt filtern**}
9: **sort** *relations* **by** $information(x)$
10: $totalinformation \leftarrow 0.0$
11: {X **Prozent der Information bewahren**}
12: **for each** *relation* **in** *relations* **do**
13: $totalinformation \leftarrow totalinformation + information(relation)$
14: **end for**
15: $selectedrelations \leftarrow \emptyset$
16: $partialinformation \leftarrow 0.0$
17: **for each** *relation* **in** *relations* **do**
18: $partialinformation \leftarrow partialinformation + information(relation)$
19: **if** $partialinformation < (totalinformation * percentage)$ **then**
20: **collect** *relation* into *selctedrelations*
21: **end if**
22: **end for**
23: **return** *selectedrelations*

Algorithmus 4.10: Relevante Relationen aus Zustand (*getstatesig*).

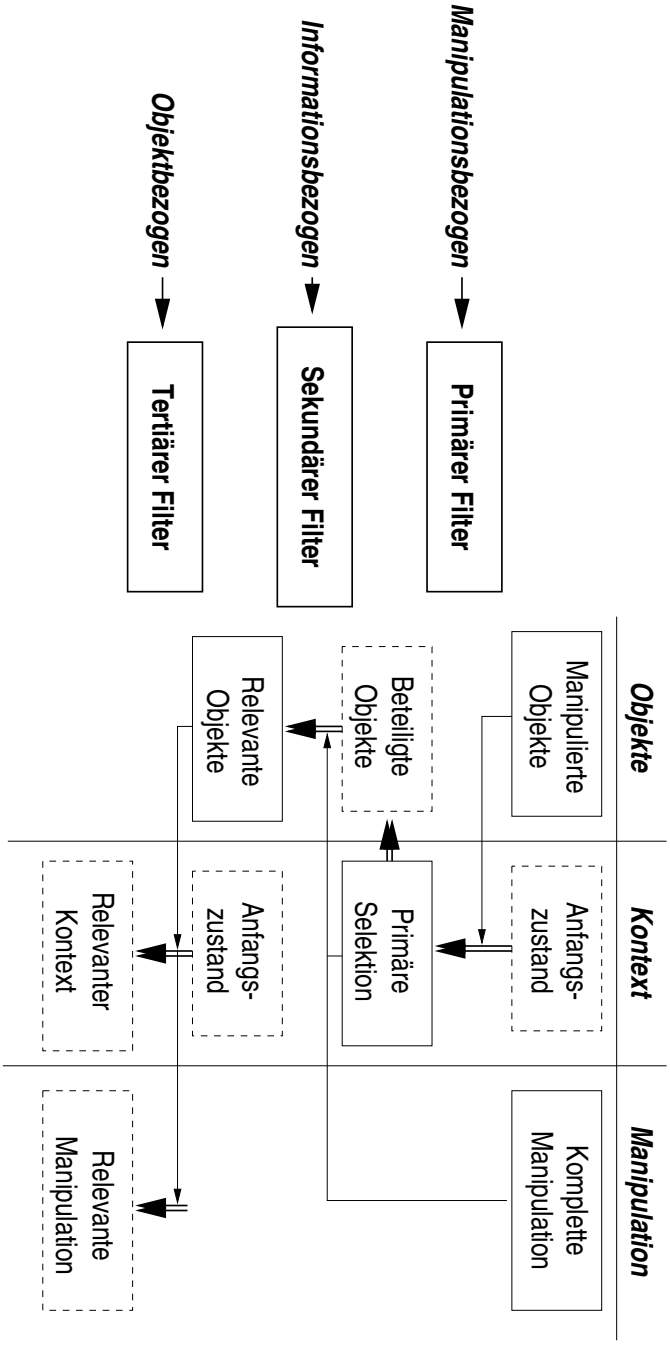


Abbildung 4.12: Schritte des Dreiphasenfilters

Eingabe: $objects, state, manipulation$
objects: An Manipulation beteiligte Objekte; *state*: Anfangszustand
manipulation: Manipulation des geschlossenen Segmentes
Ausgabe: $generalstate$

1: $selection \leftarrow getstatesig(state, objects)$ {**Primäre Selektion bestimmen**}
2: $objects \leftarrow \emptyset$ {**Beteiligte Objekte bestimmen**}
3: **for each** $relation$ **in** $selection$ **do**
4: **store** $relation.object1$ **in** $objects$; **store** $relation.object2$ **in** $objects$
5: **end for**
6: $selection \leftarrow getstatesig(state, objects)$ {**Sekundäre Selektion bestimmen**}
7: **for each** $object$ **in** $objects$ **do**
8: $stateinfo\,[object] \leftarrow 0.0$; $maninfo\,[object] \leftarrow 0.0$
9: **end for**
10: $totalstateinfo \leftarrow 0.0$ {**Objektweise Zustandsinformation**}
11: **for each** $relation$ **in** $selection$ **do**
12: $stateinfo\,[relation.object1] \leftarrow stateinfo\,[relation.object1] + information(relation)$
13: $stateinfo\,[relation.object2] \leftarrow stateinfo\,[relation.object2] + information(relation)$
14: $totalstateinfo \leftarrow totalstateinfo + information(relation)$
15: **end for**
16: $totalmaninfo \leftarrow 0.0$ {**Objektweise Manipulationsinformation**}
17: **for each** $relation$ **in** $selection$ **do**
18: $maninfo\,[relation.object1] \leftarrow maninfo\,[relation.object1] + information(relation)$
19: $maninfo\,[relation.object2] \leftarrow maninfo\,[relation.object2] + information(relation)$
20: $totalmaninfo \leftarrow totalmaninfo + information(relation)$
21: **end for**
22: $envolvedobjects \leftarrow \emptyset$ {**Relevante Objekte selektieren**}
23: **for each** $object$ **in** $objects$ **do**
24: **if** $(maninfo\,[object] > (percentage * totalmaninfo)) \vee (stateinfo\,[object] > (percentage * totalstateinfo))$ **then**
25: **collect** $object$ **into** $envolvedobjects$
26: **end if**
27: **end for**
28: $generalstate \leftarrow \emptyset$ {**Relevanten Kontext bestimmen**}
29: **for each** r **in** $selection$ **do**
30: **if** $(r, object1$ **in** $objects) \wedge (r, object2$ **in** $objects)$ **then**
31: **collect** r **into** $generalstate$
32: **end if**
33: **end for**
34: $generalmanipulation \leftarrow \emptyset$ {**Relevante Manipulationenbestimmen**}
35: **for each** r **in** $manipulation$ **do**
36: **if** $(r, object1$ **in** $objects) \wedge (r, object2$ **in** $objects)$ **then**
37: **collect** r **into** $generalmanipulation$
38: **collect** $[r.relation, r.object1, r.object2, \neg r.value]$ **into** $generalmanipulation$
39: **end if**
40: **end for**
41: **return** $generalstate, generalmanipulation$

Algorithmus 4.11: Dreiphasen Informationsfilter (*hypothesis*).

Informationsbasierte Hypothesenfilterung

Um die generalisierten Ausführkontexte und die generalisierten Manipulationen zu bestimmen, die letztlich die Benutzerintention wiedergeben, wurde eine spezielles Verfahren entwickelt, das in drei Schritten abläuft (siehe Algorithmus 4.11):

1. **Manipulationsbezogener Filter.** Im ersten Schritt werden alle Objekte betrachtet, die an einer Manipulation beteiligt waren; das ist einerseits das manipulierte Objekt selbst andererseits aber auch das Objekt relativ zu dem der Ablagepunkt definiert ist. Auf Basis dieser Objekte werden Relationen aus dem initialen Weltzustand extrahiert, welche sich auf die beteiligten Objekte beziehen. Diese resultierende Liste von Relationen wird nach deren Informationsgehalt sortiert. Dann werden solange die signifikantesten Relationen ausgewählt bis die Summe ihrer Informationsgehalte einen bestimmten Anteil an der Summe des Informationsgehaltes aller ausgewählten Relationen übersteigt. Es wird also eine minimale Auswahl an Relationen getroffen, die einen möglichst großen Anteil am Informationsgehalt aus dem initialen Weltzustand konserviert. Diese Menge wird durch den Algorithmus 4.10 bestimmt und bestimmt die *primäre Selektion*. Aus der Menge dieser Relationen werden alle beteiligten Objekte extrahiert. Dabei ist es durchaus möglich und auch gewollt, daß sich die Menge der Objekte erweitert, da Relationen auch vorher unbeteiligte Objekte involvieren können. Das ergibt die Menge der *beteiligten Objekte*.

2. **Informationbasierter Filter.** Im zweiten Schritt werden aus den *beteiligten Objekten* all diejenigen herausgefiltert, die zu einem Gewissen Anteil zu dem Informationsgehalt insgesamt beitragen. Dazu werden zunächst erneut mit dem Informationsfilter aus Algorithmus 4.10 die relevanten Relationen aus dem initialen Weltzustand extrahiert; in diesem Schritt allerdings mit den beteiligten Objekten, was die *Sekundäre Selektion* ergibt. Danach wird die Summe des Informationsgehaltes insgesamt und für jedes relevante Objekt einzeln berechnet. Diese Werte werden sowohl für den initialen Zustand als auch für die Manipulationen getrennt bestimmt. Wenn entweder manipulationsbezogen oder zustandsbezogen ein Objekt einen bestimmten Informationanteil übersteigt, dann wird das in die Menge der *relevanten Objekte* übernommen.

3. **Objektbezogener Filter.** Im letzten Schritt werden schließlich alle Relationen aus der Sekundären Selektion bzw. aus der Manipulation herausgefiltert, die ausschließlich relevante Objekten involvieren. Alle Relationen der generalisierten Manipulation werden zusätzlich auch mit negiertem Wahrheitswert in den generalisierten initialen Zustand gesichert, da es in der Regal plausibel ist, daß Relationen der Benutzerintention vor der Ausführung eines Makroprogramms noch nicht erfüllt sein sollten. Diese werden als Ergebnisse für die Hypothesen des generalisierten Ausführungskontextes bzw. der generalisierten Manipulation, also der Benutzerintention zurückgegeben. Sinn dieser Methode ist es, keine unterbestimmten Objekte in der resultierenden Generalisierung zu erzeugen.

Die in dem Verfahren angegebenen Schwellwerte wurden anhand verschiedener Beispielvorführungen empirisch bestimmt.

4.4 Inhaltliche Analyse

Im letzten Abschnitt 4.3 wurde gezeigt, wie sich Generalisierung der Ausführungskontexte und der Manipulation der Benutzervorführung bestimmen lassen. Die Generalisierung der Manipulation der Sequenz ist dabei mit der Benutzerintention gleichzusetzen, die in dem vorgestellten Analyseverfahren eine zentrale Rolle einnimmt.

Grundgedanke des Verfahrend ist es, von der Benutzerintention und den generalisierten Kontexten der geschlossenen Segmente Rückschlüsse auf folgende Parameter zu ziehen:

- **Verzweigungsbedingungen einzelner Segmente.** Zwischen geschlossenen Segmenten ist es möglich Verzweigungsbedingung zu generieren, die festlegen, unter welchen Umständen ein Segment auszuführen ist.
- **Redundante Segmente bestimmen.** Ergibt sich in der Verzweigungsanalyse, daß keine Bedingung zur Ausführung eines Segmentes führt, so kann dieses im Makrooperator herausgelassen werden, da es für die Intention des Benutzers keinen Beitrag liefert.
- **Gesamtkontext des Makrooperators.** Der Gesamtkontext des Makrooperators dient zur Bestimmung, ob ein Operator auf einer bestimmten Umwelt ausführbar ist und mit welchen Parametern. Über den Gesamtkontext des Makrooperators ist es also möglich diesen zu instantiieren (siehe Abschnitt 4.5.1).

4.4.1 Motivation

Abbildung 4.13 zeigt eine exemplarische Episode und das darauf angewandte Analyseverfahren. Auf der linken Seite sind die Greif- und Loslaßpunkte einer Benutzervorführung in der Simulation dargestellt. Die Vorführung besteht aus drei Segmenten, in denen zunächst die *Spacebar* von den beiden Löchern *hole1* entfernt und *hole2* und auf dem Tisch abgelegt wird. Danach werden die beiden *Spacer*, *spacer1* und *spacer2* in die entsprechenden Löcher gefügt. Die Spalte direkt neben den Grafiken zeigt mit eckiger Umrahmung, die nach Abschnitt 4.3 bestimmten Hypothesen für die Generalisierung der Ausführkontexte der einzelnen Segmente. Mit runder Umrandung werden in derselben Spalte Ausschnitte aus den Manipulationen der Segmente gegeben, wie sie sich nach Abschnitt 4.2.4 bestimmen. Die Spalte auf der rechten Seite zeigt die in jedem Schritt aktuelle Hypothese für den Gesamtkontext der Benutzervorführung, die mittlere die Beiträge der einzelnen Segmente zur Gesamtintention (s.u.)

Die inhaltliche Analyse verläuft rückwärtsgewand, also dem zeitlichen Ablauf der Benutzervorführung entgegengesetzt. Sie wird mit zwei Werten vorinitialisiert: der Benutzerintention und der leeren Hypothese für den Gesamtkontext des zu generierenden Makrooperators. Von Hinten beginnend wird nun die Benutzerintention sukzessive mit den Manipulationen der einzelnen Segmente bzw. der aktuellen Hypothese für den Gesamtkontext verglichen. Ergeben sich Übereinstimmungen so werden die übereinstimmenden Relationen als *Beiträge* zur Intention des Verfahrens gewertet. Ergibt sich dabei die Übereinstimmung der Relationen in der Manipulation eines Segmentes spricht man von einem

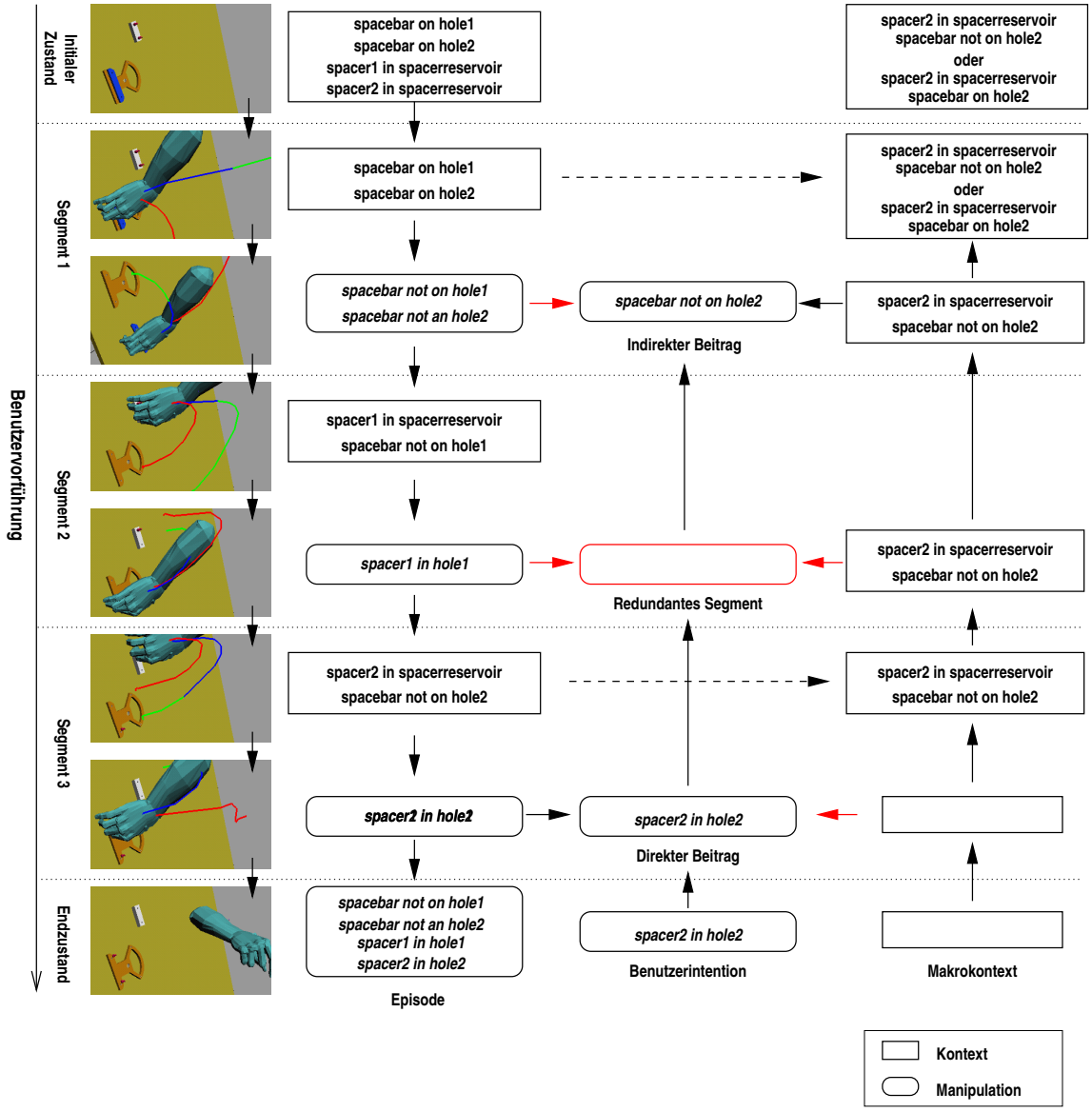


Abbildung 4.13: Schematische Darstellung der inhaltlichen Analyse

direkten Beitrag des Segmentes, da das Segment mindestens eine Relation in von die Benutzerintention vorgeschriebene Ausprägung überführt. Dies ist etwa im dritten Segment in der Abbildung 4.13 der Fall.

Es kann aber auch vorkommen, daß Manipulationen von Segmenten zwar nicht direkt zur Benutzerintention beitragen, aber dennoch Effekte hervorrufen, die in den Kontexten von späteren Segmenten Bedingung sind. Vergleicht man also die Manipulation eines Segmentes auch mit der aktuellen Hypothese für den Gesamtkontext der Benutzervorführung, so erhält man *indirekte Beiträge*. Abbildung 4.13 zeigt einen solchen Fall für das erste Segment. Weist die Manipulation eines Segmentes direkte oder indirekte Beiträge aus, so handelt es sich um ein benötigtes Segment. In diesem Fall ist dieses Segment des zu generierenden Makrooperators dann auszuführen, wenn noch nicht alle direkten und indirekten Beiträge des Segmentes vorliegen, so daß sichergestellt ist, daß am Ende des Segmentes die entsprechenden Beiträge erbracht sind. Andererseits ist es aber nicht nötig das Segment auszuführen, wenn bereits alle Beiträge vor der Ausführung erfüllt sind. Man sieht also, daß die Konjunktion von direkten und indirekten Beiträgen direkt als Verzweigungsbedingung genutzt werden kann.

Für den Kontext der Gesamtvorführung bedeutet dies, daß entweder der Operator ausgeführt werden und dessen Kontext erfüllt sein muß, oder aber er wird nicht ausgeführt, was aber voraussetzt, daß schon alle Beiträge erfüllt sind. Im Falle eines nicht-redundanten Segmentes ist die Hypothese für den Gesamtkontext also wie folgt fortzuschreiben:

$$\begin{aligned} Gesamtkontext_{vorher} \leftarrow \quad & (Gesamtkontext_{nachher} \wedge Kontext_{Segment} \setminus Beitraege) \\ & \vee (Gesamtkontext_{nachher} \wedge Beitraege) \end{aligned} \tag{4.5}$$

Liegt kein Beitrag vor, dann ist ein Segment *redundant*. Dann bleibt die Hypothese für den Gesamtkontext unverändert. Im zweiten Segment von 4.13 zeigt einen solchen Fall.

Anhand dieses kleinen Beispiels sieht man allerdings auch, wie kritisch eine korrekte Ableitung sowohl von Benutzerintention als auch der Generalisierten Kontexte ist. Wäre etwa dem System ein Fehler bei der Erkennung der Benutzerintention unterlaufen, und wäre die Relation *spacer1 in hole1* nur fälschlicherweise nicht in die Benutzerintention aufgenommen worden, dann wäre das zweite Segment in der Abbildung fälschlicherweise als redundant gekennzeichnet worden. Gleiches gilt für die korrekte Generalisierung der Kontexte. Hätte etwa das System nicht erkannt, daß beim Fügen eines *Spacers* die Loch, in das er gefügt werden soll, nicht verdeckt sein darf - wäre also die Relation *spacebar not on hole2* nicht in den Kontext des dritten Segmentes aufgenommen worden, dann würde auch das erste Segment als redundant markiert, da der indirekte Beiträg nicht aufgetreten wäre.

Betrachtet man sich aber Abbildung 4.6, so wird offenkundig, daß eine inhaltliche Analysen aber nicht nur auf einer Ebene stattfinden muß, sondern sich vielmehr auf alle Ebenen der Erklärungshierachie. Tabelle 4.1 zeigt die Zuordnung von Erklärungs- und Segmentierungsebene zu den Algorithmen 4.12, 4.13 und 4.14.

Eingabe: $globalcontext, intention, segment$
$globalcontext$**:** Kontext der Analyse
$intention$**:** Nichterfüllte Benutzerintention
$segment$**:** Zu analysierendes Segment
Ausgabe: $intention, direct, indirect, operator$

```
 1: localcontext ← ∅
 2: direct ← ∅
 3: indirect ← ∅
 4: sequence ← ∅
 5: {Rückwärts Untersegmente bearbeiten}
 6: subsegments ← segment.subsegments
 7: for each s in reverse subsegments do
 8:   {Untersegmente analysieren}
 9:   [int, dir, ind, op] ←
10:           call segment.algorithm with localcontext ∧ globalcontext, intention, s
11:   {Nicht-redundantes Segment?}
12:   if dir ∨ ind then
13:     {Operatorsequenz, direkten und indirekten Beitrag aktualisieren}
14:     add op to sequence
15:     direct ← direkt plus dir
16:     indirect ← indirekt plus ind
17:     {Neuen Kontext bestimmen}
18:     newcontext ← ∅
19:     for each conjuncion1 in localcontext do
20:       collect (conjunction1 ∧ dir) into newcontext
21:       contrib ← ind ∩ conjunction1
22:       for each conjunktion2 in op.context do
23:         if contrib then
24:           collect (conjunction1 ∧ conjunktion2 \ contrib) into newcontext
25:         end if
26:       end for
27:     end for
28:     context ← newcontext
29:     intention ← int
30:   end if
31: end for
32: {Makrooperator erzeugen}
33: contribution ← direct plus indirect
34: create operator with sequenz, localcontext, contribution
35: return intention, direct, indirect, operator
```

Algorithmus 4.12: Inhaltliche Analyse für geschlossene Segmente (*analyzeclosed*).

4.4.2 Verzweigungs- und Kontextanalyse

Geschlossene Segmente

Die inhaltliche Analyse erfolgt im wesentlichen, wie in der Motivation beschrieben. Algorithmus 4.12 zeigt, wie zunächst für die Untersegmente des zu analysierenden Segmentes die während der Segmentierung (siehe Algorithmus 4.5) festgelegte Methode zur inhaltlichen Analyse aufgerufen wird. Dies entspricht einem rekursiven Aufruf der Methoden analog zur Segmentierungshierarchie.

Der zu bestimmende Gesamtkontext (*localcontext*) des Segmentes wir zunächst leer initialisiet. Auch die direkten und indirekten Beiträge des Segmentes werden zurückgesetzt. Jetzt beginnt der rekursive Aufruf des Analyseverfahrens mit dem letzten Untersegment des Segmentes mit der aktuellen Hypothese für den Gesamtkontext dieses Untersegmentes, das sich aus dem lokalen Kontext des Segmentes und einem evtl. übergeordneten globalen Kontext zusammensetzt. Weitere Parameter sind die noch nicht erfüllte Benutzerintention sowie das zu analysierende Untersegment selbst. Als Ergebnisse des Aufrufs erhält das Verfahren die durch das Untersegment noch nicht erfüllte Intention, die direkten und indirekten Beiträge des Untersegmentes sowie einen instantiierten Makrooperator, der das Untersegment beschreibt.

Nun beginnt die eigentliche Verzweigungs- und Redundanzanalyse. Liegt ein direkter oder indirekter Beitrag des Untersegmentes vor, so wird handelt es sich um eine nichtredundantes Untersegment. Nach Gleichung 4.5 wird die neue Hypothese für den Gesamtkontext des Segmentes bestimmt. Dabei wird zwischen dem Fall unterschieden, daß schon alle Beiträge bei einer möglichen Ausführung erfüllt sind, und dem, daß der Operator für das Untersegment ausgeführt werden muß. Im ersten Fall ist eine Optimierung möglich, da nur indirekte Beiträge eine Rolle spielen, die in den einzelnen Konjunktionen ohnehin schon auftreten; nur dann wäre der indirekte Beitrag ja ein solcher für den Fall einer speziellen Konjunktion, ansonsten wäre der indirekte Beitrag für den Fall, der in dieser speziellen Konjunktion beschrieben wird, ohnehin überfällig. Also wird der erste Fall korrekt beschrieben, wenn die betreffende Konjunktion und der direkte Beitrag konjunktiv verknüpft werden. Der zweite Fall berechnet sich aus der Konjunktion der gegenwärtigen Hypothese für den Gesamtkontext und dem lokalen Kontext des Untersegmentes, wobei allerdings alle indirekten Beiträge in diesen nicht mehr mit einfließen.

In der Regel sollte in einer Erklärungshierarchie zumindest die oberste Ebene durch eine geschlossene Segmentierung entstanden sein. Wenn dem so ist, die der Algorithmus 4.12 direkt den Gesamtkontext der Episode sowie einen instantiierten Makrooperator zurück, der die Episode repräsentiert. Die Aufgabe ist also an dieser Stelle gelöst und der erhaltene Operator kann unmittelbar auf anderen Welten initialisiert und ausgeführt werden.

Offene Segmente

Die inhaltlich Analyse für offene Segmente erfolgt im wesentlichen Analog zu der für geschlossene in Algorithmus 4.12. der Algorithmus 4.13 weicht nur dadurch ab, daß keine Verzweigungsanalyse durchgeführt wird, keine redundanten Segmente bestimmt und keine Verzweigungen erzeugt werden.

Eingabe: $globalcontext, intention, segment$
$globalcontext$**:** Kontext der Analyse
$intention$**:** Nichterfüllte Benutzerintention
$segment$**:** Zu analysierendes Segment
Ausgabe: $intention, direct, indirect, operator$

```
1:  localcontext ← ∅
2:  direct ← ∅
3:  indirect ← ∅
4:  sequence ← ∅
5:  {Rückwärts Untersegmente bearbeiten}
6:  subsegments ← segment.subsegments
7:  for each s in reverse subsegments do
8:     {Untersegmente analysieren}
9:     [int, dir, ind, op] ←
10:              call segment.algorithm with localcontext ∧ globalcontext, intention, s
11:    {Operatorsequenz, direkten und indirekten Beitrag bestimmen}
12:    add op to sequence
13:    direct ← direkt plus dir
14:    indirect ← indirekt plus ind
15:    {Neuen Kontext bestimmen}
16:    newcontext ← ∅
17:    for each conjuncion1 in localcontext do
18:       for each conjunktion2 in op.context do
19:          collect (conjunction1 ∧ conjunktion2) into newcontext
20:       end for
21:    end for
22:    context ← newcontext
23:    intention ← int
24: end for
25: {Makrooperator erzeugen}
26: create operator with sequenz, localcontext, ∅
27: return intention, direct, indirect, operator
```

Algorithmus 4.13: Inhaltliche Analyse für offene Segmente (*analyzeopen*).

Eingabe: $globalcontext, intention, segment$
$globalcontext$**:** Kontext der Analyse
$intention$**:** Nichterfüllte Benutzerintention
$segment$**:** Zu analysierendes Segment
Ausgabe: $intention, direct, indirect, operator$
1: {**Direkte und indirekte Beiträge bestimmen**}
2: $changes \leftarrow delta(S[segment.start], S[segment.end])$
3: $direct \leftarrow changes \cap intention$
4: $indirect \leftarrow \emptyset$
5: **for each** $conjunction$ **in** $globalcontext$ **do**
6: **add** $changes \cap conjunction$ **to** $indirect$
7: **end for**
8: {**Direkte Beiträge aus Intention löschen**}
9: $intention \leftarrow intention \setminus direct$
10: {**Elementaroperator instantiieren**}
11: **instantiate elementary** $operator$ **with data from** $segment.start$ **till** $segment.end$
12: **return** $intention, direct, indirect, operator$

Algorithmus 4.14: Analyse für elementare Segmente (*analyzeelementary*).

Elementaroperatoren

Auf der Ebene der Elementaroperatoren läßt sich keine einheitliche Vorgehensweise angeben. Im Grunde findet auf dieser Ebene keine echte Analyse mehr statt. Vielmehr werden auf Grundlage der in Vorverarbeitung und Generalisierungsschritt bestimmten Daten, wie etwas die generalisierte Trajektorie, gegriffene Objekte, Greif- und Loslaßpunkte, der generalisierten Ausführungskontexte etc., Elementaroperatoren instantiiert, welche diesen Daten entsprechen. Ein wichtiger Unterschied zu den höheren Ebenen besteht darin, daß hier keine neue Operatoren definiert werden, vielmehr werden generisch instantiierbare Elementaroperatoren lediglich instantiiert. Auf der Manipulation zwischen Anfang und Ende des elementaren Segmente werden direkte und indirekte Beiträge bestimmt und von der noch nicht erfüllten Benutzerintention abgezogen. Die Resultate werden an die höheren Segmentierungsebenen weitergegeben.

4.4.3 Makrogenese

Nach der erfolgreichen Ableitung der generalisierten Trajektorie, der Segmentkontexte sowie des Gesamtkontextes gilt einen entsprechenden Makrooperator zu erzeugen. Analog zur Vorgehensweise im erklärungsbasierten Lernen (siehe Abschnitt 2.1.3) gilt es nun alle involvierten *freien Parameter* der einzelnen Komponenten der Erklärungsstruktur zu extrahieren. Diese Parameter bilden dann letztlich die freien Parameter des generisch zu extrahierenden Makrooperators.

In erster Linie werden diese freien Parameter durch die an den einzelnen Komponenten der Erklärungsstruktur beteiligten Objekte gegeben. Das sind einerseits diejenigen Objekte, die in der generalisierten Trajektorie als Referenzobjekt in einem Bewegungskontext auftauchen. Andererseits fließen aber auch diejenigen Objekte als freie Parameter ein, die

in den Relationen der Verzweigungsbedingungen und im Gesamtkontext der Episode vorkommen. Zusammen bilden sie die *Objektliste* des zu erzeugenden Makrooperators. Bei der Ausführung der Makrooperatoren können die Objekte der Objektliste dann durch Objekte ersetzt werden, die den Bedingung des Gesamtkontextes des Operators genügen (siehe Abschnitt 4.5). Neben den Objekten bietet das implementierte Operatorkonzept aber auch die Möglichkeit andere Parameter, wie etwa Kräfte und Geschwindigkeiten als freie Parameter dem Makrooperator bei der Ausführung zu übergeben. Diese werden in der *Parameterliste* des Makrooperators gesichert. Die Parameter der Parameterliste sind weitgehend generisch angelegt; sie werden über ein Schlüsselwort indiziert und der zugeordnete Wert darüber abgerufen, etwa *speed = 200*. Während die Parameterliste eines Makrooperators üblicherweise mit Defaultwerten belegt werden, die sich aus der Benutzervorführung ableiten, während die Objekte in der Objektliste in der Regel tatsächlich im Rahmen einer *Instantiierung* des Makrooperators für jede Ausführumgebung spezifisch zu bestimmen.

Damit die freien Parameter der Objektliste und der Parameterliste korrekt auf die entsprechenden Eingabewerte der nachfolgenden Operatoren abgebildet werden könne, ist es notwendig sog. *Verteilungslisten* zu definieren. Diese legen fest welcher Parameter des Makrooperators auf welchen Parameter der Der genaue Vorgang der Instantiierung sowie der Verteilung der freien Parameter auf die Nachfolgeoperatoren wird näher in Abschnitt 4.5 beschrieben.

4.4.4 Analyse ikonischer Vorführungen

Wie im Kapitel 3 beschrieben, bestand die Aufgabe auch darin andere Eingabedaten als direkte Benutzervorführungen zuzulassen. Mithilfe des modularen Segmentierungsverfahrens nach Algorithmus 4.5 gestaltet sich dieses relativ problemlos. Mit einem angepaßten Satz von Segmentierern ist es unmittelbar möglich, eine analoge Erklärungsstruktur zu erzeugen und mit den in diesem Abschnitt vorgestellten Verfahren inhaltlich zu analysieren. Im speziellen Fall der ikonischen Programmierung fällt das Problem besonders einfach aus, da die Erklärungshierarchie nur aus einer Ebene besteht. Jeder Makrooperator stellt per Definition für sich ein geschlossenes Segment dar. Auch sind die Kontexte der Makrooperatoren bereits bekannt, so daß der Generalisierungsschritt wegfallen kann. Man kann also direkt den Algorithmus 4.12 anwenden und erhält als Ergebnis ein Makroprogramm, das alle Schritte er einzelnen Makrooperatoren umfaßt.

4.4.5 Laufzeitverhalten

Betrachtet man die Algorithmen 4.12 und 4.13, so stellte man fest, daß die Anzahl der Konjunktionen in dem Gesamtkontext eines Makrooperators zwischen zwei Segmenten um $O(m * n)$ zunimmt, wobei m und n die jeweilige Anzahl an Konjunktion in den Untersegmenten bestimmt. Im Algorithmus 4.12 ist wegen der Verzweigungen sogar mit einem Aufwand von $O(m * n + n)$ zu rechnen. In einer Sequenz von k Segmenten, deren Kontext je n Konjunktionen enthält, läßt sich also die Anzahl der Konjunktionen im Gesamtkontext mit $O(n^k)$ annähern. Man sieht schnell ein, daß es bei komplexeren Vorführungen zu unakzeptabel komplexen Kontexten kommt. Um dieses Wachstum wenigstens geringfügig

zu beschränken bietet das implementierte System die Möglichkeit nur bei indirekten Beiträgen Verzweigungen einzufügen. Dies stellt zwar theoretisch eine fehlerhafte Vorgehensweise dar und führt dazu, daß gültige Instantiierungen nicht gefunden werden, dämpft aber das Wachstum der Konjunktionen im Gesamtkontext spürbar.

4.5 Anwendung von Makrooperatoren

Der Analyseschritt in Abschnitt4.4 dient mitunter dazu, *Gesamtkontexte* und *Verzweigungsbedingungen* des Makrooperators zu bestimmen. Die beiden Größen werden dazu benötigt einen Makrooperator zu *instantiieren* und anschließend *auszuführen*. Wie Makrooperatoren auszuführen sind wird in diesem Abschnitt beschrieben.

4.5.1 Instantiierung von Makrooperatoren

In den Abschnitten 4.3 und 4.4 hatten wir gesehen, wie aus einer konkreten Vorführung ein allgemein ausführbarer Makrooperator erzeugt wurde. Dabei wurde sowohl die Trajektorie als auch die Ausführungskontexte über die Objekte und andere freie Parameter generalisiert. Diese gilt es nun bei der Ausführung in anderen Umwelten sinnvoll zu belegen und somit den Makrooperator zu instantiieren. Während die Parameter der *Parameterliste* eines Makrooperators in der Regel mit den aus der Benutzervorführung abgeleiteten Default-Werten belegt werden können, die nur in Ausnahmefällen modifiziert werden müssen, gilt es die Objekte in der *Objektliste* in jedem Falle anzupassen, da die Objekte in der Vorführumwelt nicht in der Ausführumgebung als vorhanden angenommen werden können.

Zur korrekten Belegung der Objektliste mit zulässigen Objekten aus der Ausführumwelt dient der Gesamtkontext des Makrooperators. Dieser Gesamtkontext wurde in Abschnitt 4.4.2 derart konstruiert, daß er die schwächste Bedingung darstellt, unter welcher der Makrooperator noch korrekt ausgeführt wird. Damit gilt es in der Instantiierung des Makrooperators diejenigen Objektbelegungen für die Objektlist zu finden, unter denen der Gesamtkontext des Makrooperators erfüllt ist. Algorithmus 4.15 beschreibt das Vorgehen des Systems bei der Initialisierung eines Makrooperators. Diese Vorgehensweise erfolgt in drei Schritten:

1. **Vorfilter.** Da es bei k Objekten in der Objektliste und n Objekten in einer potentiellen Ausführumgebung n^k Objektbelegungen möglich sind, gilt es den Hypothesenraum sinnvoll zu beschränken. Hierzu werden alle unären Relationen extrahiert, die jeder Konjunktion des Gesamtkontextes gemeinsam sind. Daraus wird ein Filter konstruiert, der mögliche Instantiierungshypothesen vorfiltert.

2. **Gültige Hypothesen.** Anschließend wird der Makrooperator mit allen nach dem Vorfilter zulässigen Instantiierungshypothesen instantiiert und geprüft, ob der Gesamtkontext unter dieser Belegung erfüllt ist. Daraus werden alle gültigen Hypothesen abgeleitet.

3. **Hypothesenbewertung- und auswahl.** Im letzten Schritt werden mit dem in [Riepp 97] vorgestellten und implementierten Verfahren einzelne Instantiierungshy-

Eingabe: *operator*, *world*
operator: Makrooperator mir Kontext (*context*) und Objektliste (*objects*).
world: Ausführumgebung
Ausgabe: *instantiations*

```
1: {Vorfilter festlegen.}
2: conjunction ← first conjuntion of operator.context
3: for each position in objectlist(operator) do
4:    filter [position] ← ∅
5:    objects [position] ← ∅
6: end for
7: for each term in conjunction do
8:    if isunary(term) then
9:       if isinall(term, operator.context) then
10:         add term to filter [term.object.position]
11:      end if
12:   end if
13: end for
14: {Vorfilter anwenden.}
15: for each position in objectlist(operator) do
16:   for each object in O(world) do
17:      if object fits filter [position] then
18:         collect object into objects [position]
19:      end if
20:   end for
21: end for
22: {Instantiierungen bestimmen.}
23: instantiations ← ∅
24: for any combination from objects [∗] do
25:   if evaluate operator.context with combination then
26:      collect combiniation into instantiations
27:   end if
28: end for
29: return instantiations
```

Algorithmus 4.15: Makrooperator auf Umwelt initialisieren (*instantiate*).

pothesen bewertet und nach der Bewertung sortiert. Die Hypothese mit der besten Bewertung wird schließlich ausgewählt. Alternativ kann dieser letzte Schritt auch durch den Benutzer vorgenommen werden.

4.5.2 Ausführung von Makrooperatoren

Das Grundkonzept der implementierten Makrooperatoren beruht darauf, daß mit der erfolgreichen Initialisierung eines Makrooperators dieser unmittelbar ausführbar sein sollte. Zu diesem Zweck werden die Parameter der Parameterliste und die Objekte aus der Objektliste auf die Nachfolger des Operators verteilt. Gleichzeitig ist jeder Makrooperator nur bedingt auszuführen; d.h. ein Makrooperator ist nur dann auszuführen, wenn seine Beiträge noch nicht im gegenwärtigen Umweltzustand erfüllt sind. Im Abschnitt 4.4 wurde gezeigt, wie für einzelne Segmente und damit für die daraus generierten Nachfolgeoperatoren die direkten und indirekten Beiträge bestimmt werden. Diese Beiträge waren derart motiviert, daß sie wiederspiegeln sollten, zu welchen Zweck, also mit welcher Intention die entsprechenden Segmente demonstriert wurden. Sind nun aber alle diese direkten oder indirekten Beiträge zur Intention des Benutzers bereits vor der Ausführung des Makrooperators erfüllt, ist es nicht nötig, den Operator auszuführen, weswegen seine Ausführung übersprungen wird.

Eingabe: *operator*, *world*
operator: Makrooperator mir Kontext (*context*) und Objektliste (*objects*).
world: Ausführumgebung
Ausgabe: *instantiations*

```
1:  {1. Ausführbedingung prüfen.}
2:  if not evaluate operator.contribution with operator.objectlist then
3:    {2. Nachfolgeoperatoren instantiieren.}
4:    for each dist in operator.objectdist do
5:      collect operator.objectlist[dist.from] into successor[dist.to].objectlist
6:    end for
7:    for each dist in operator.parameterdist do
8:      collect operator.parlist[dist.from] into successor[dist.to].parlist
9:    end for
10:   {3. Nachfolgeoperatoren ausführen.}
11:   for each successor in operator.successors do
12:     execute successor
13:   end for
14: end if
15: return instantiations
```

Algorithmus 4.16: Makrooperator auf Umwelt ausführen (*execute*).

Erweist sich die Ausführung des Makrooperators als notwendig, so gilt es die Nachfolgeoperatoren des Makrooperators mit den Werten des Makrooperators zu initialisieren. Zu diesem Zweck werden die Parameter der Parameterliste und die Objekte aus der Objektliste über Verteilungslisten für Parameter und Objekte auf die Nachfolger des Operators

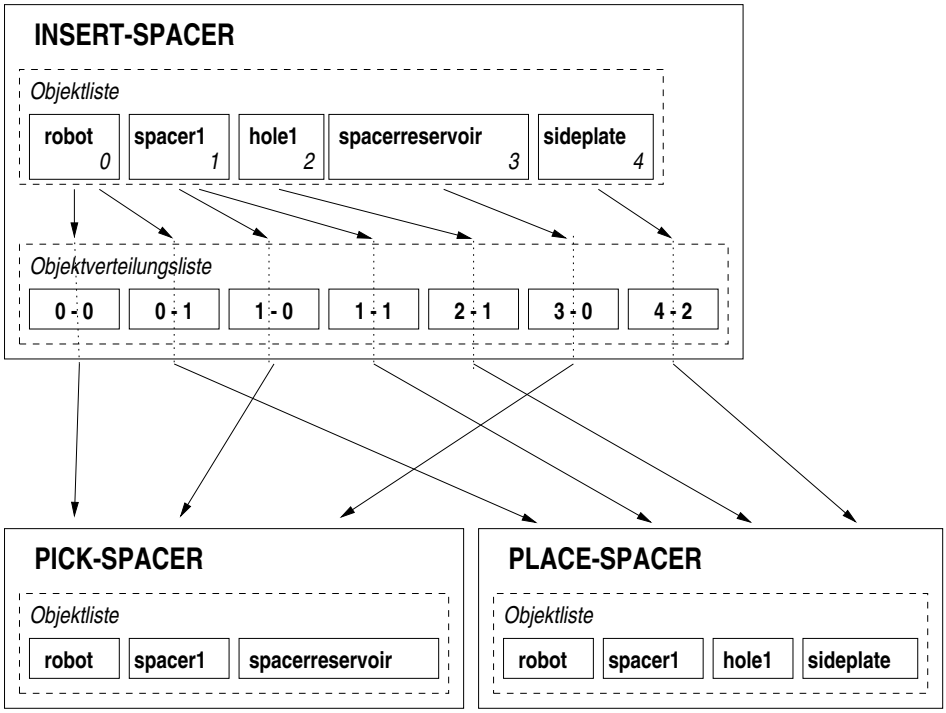


Abbildung 4.14: Propagierung der Objekte an Nachfolger eines Makrooperators

verteilt. Der Verteilungsprozeß wird in Abbildung 4.14 exemplarisch für Objekte veranschaulicht. In der Verteilungsliste referenziert der erste Wert (*from*-Slot), die Position eines Objektes in der Objektliste des Makrooperators. Der zweite Wert (*to*-Slot) bestimmt, auf welchen Nachfolgeoperator das Objekt zu übertragen ist. Durch die korrekte Reihenfolge in der Verteilungsliste wird die korrekte Position des verteilten Objektes in der Objektliste des Nachfolgers sichergestellt. Ist der Verteilungsprozeß abgeschlossen, werden sukzessive alle Nachfolger ausgeführt. Jeder Nachfolger wird in gleicher Weise ausgeführt, wie der übergeordnete Makrooperator. Für einzelne Elementaroperatoren sind separate Ausführroutinen zu implementieren, die auch vom jeweiligen Effektor abhängen können.

Damit ergeben sich für die Ausführung eines Makrooperators folgende Schritte, die in Algorithmus 4.16 verdeutlicht werden.

1. **Ausführbedingung prüfen.** Makrooperatoren werden nur ausgeführt, wenn deren Beitrag noch nicht vollständig in der Ausführumwelt erfüllt ist.

2. **Nachfolgeoperatoren instantiieren.** Die Liste der Nachfolgeoperatoren wird über der Verteilerlisten für die Parameter- und Objektliste des Makrooperators initialisiert.

3. **Ausführung der Nachfolgeoperatoren.** Danach werden die Nachfolgeoperatoren sukzessive ausgeführt.

4.5.3 Verwaltung von Makrooperatoren

Die erzeugten Makrooperatoren werden separat auf die Festplatte gesichert. Dabei ist wichtig, daß für jedes Segment in der Erklärungshierarchie jeweils ein weiteren spezifischer Makrooperator generiert wird, so daß eine Benutzervorführung stets mehrere Makrooperatoren erzeugt. Diese Feststellung ist wichtig, da bei der Ausführung von Makrooperatoren nur diejenigen Operatoren nachgeladen werden, die zur Ausführung tatsächlich benötigt werden. Werden etwa Unterbäume des Operatorbaumes wegen der Verzweigungslogik nicht benötigt, so werden diese auch weder auf der Umwelt instantiiert noch von der Festplatte geladen. Das System versucht an dieser Stelle also möglichst ressourcensparend vorzugehen.

Kapitel 5

Implementierung

Nachdem im letzten Kapitel ein Überblick über die theoretischen Grundlagen der Implementierung sowie eine Abstraktion der entworfenen Verfahren dargestellt wurde, soll dieses Kapitel nun aufzeigen, wie diese Funktionalität in einer konkreten Implementierung realisiert wurde. Das Kapitel beginnt mit der strukturellen Organisation der Komponenten sowie ihrer Zusammenarbeit im Gesamtsystem. Im zweiten Abschnitt wird aufgezeigt, welche Programmierwerkzeuge zur Implementierung verwendet wurden, und wie diese die Gesamtarchitektur beeinflußten bzw. erst möglich machen.

5.1 Systemarchitektur

Es ist einsichtig, daß die Struktur ein komplexes Programming-by-Demonstration-Systems nicht durch eine einheitliche und monolithische Struktur realisiert werden sollten. Selbst wenn eine derartige Implementierung möglich wäre, so ist diese dennoch nicht wünschenswert. Einerseits würde eine derartige Implementierung eine modulare Erweiterung des Gesamtsystems erschweren. Gerade dies ist in einem PbD-System aber essentiell, da zusätzliche Sensorsysteme oder funktionale Komponenten bei der Weiterentwicklung des Systems notwendigerweise hinzukommen. Die Option einer verteilten Ausführung des Programmes auf verschiedenen Rechnern ist für die spezielle Aufgabe des PbD in der Robotik ohnehin kaum vermeidbar, da ein einzelnes System mit der parallelen Ausführung aller notwendigen Schritte einerseits überlastet wäre, andererseits es nicht realistisch wäre, daß Sensorik, Benutzerschnittstelle und Schnittstellen zu möglichen Effektoren auf einen einzelnen Rechner realisiert sind. Des weiteren lagen bereits verschiedene Systemkomponenten vor, mit denen sich die gewünschte Funktionalität einfacher realisieren ließ. Dies ließ es wünschenswert erscheinen für jede Aufgabe, das optimale Werkzeug zur Implementierung zu verwenden. Damit ergaben sich für die Implementierung eines PbD-Systems in der Robotik folgende Anforderungen an die Architektur:

- **Modularität.** Das System sollte mit möglichst geringem Aufwand modular erweiterbar sein. Die Einbindung eines neuen Moduls soll möglichst ohne großen Aufwand zur Anpassung der existierenden Komponenten möglich sein.
- **Parallelität.** Viele Arbeitsschritte im PbD in der Robotik, etwa in der Vorverar-

beitung der Daten, sind parallelisierbar. Die Architektur sollte es zulassen, daß diese Schritte auch tatsächlich parallel ausführt werden können.

- **Heterogenität.** Verschiedene bereits vorliegende Module und Programmierwerkzeuge sollten weiterverwendbar sein, ohne sie auf eine einheitliche Programmiersprache oder eine einheitliche Systemplattform adaptieren zu müssen.

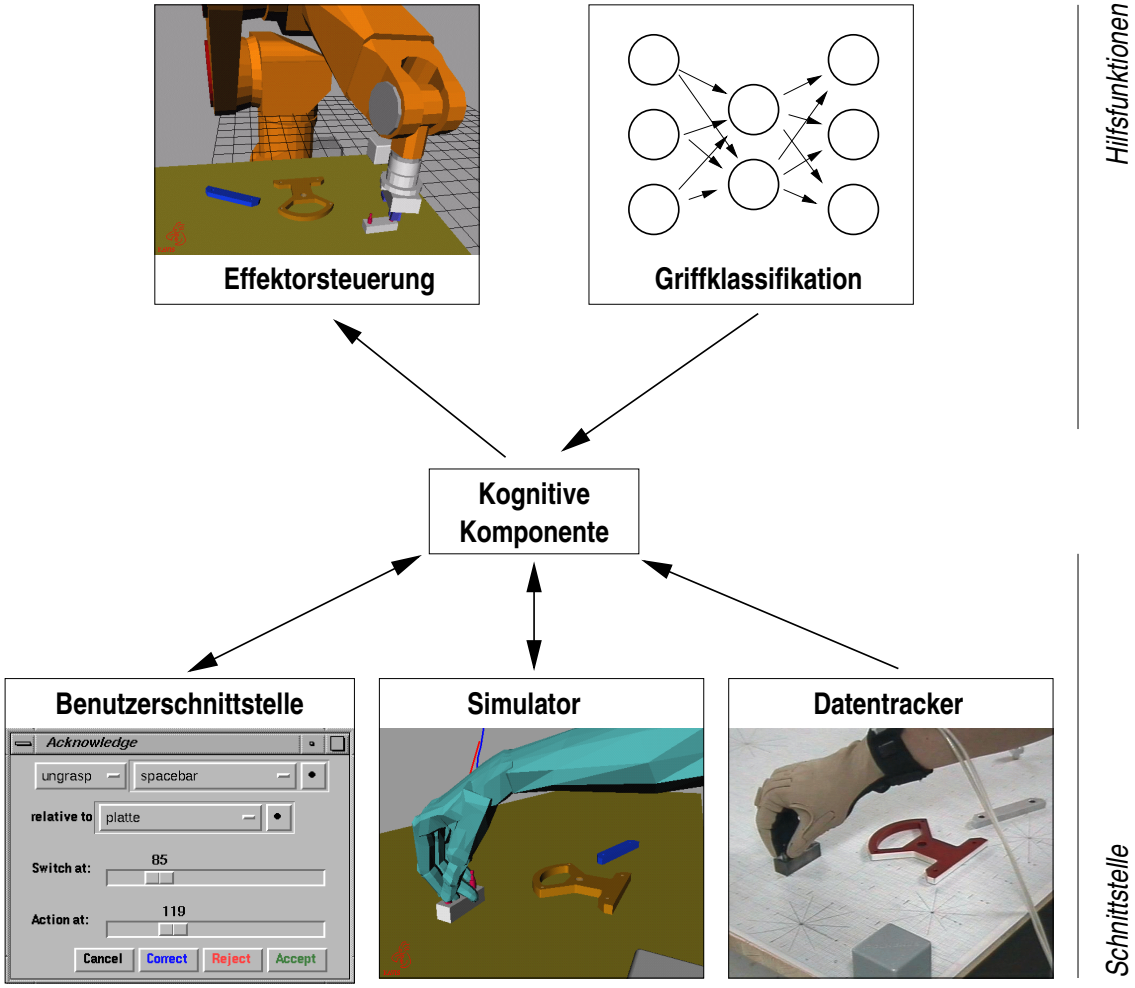


Abbildung 5.1: Architektur des Systems

5.1.1 Multiagentenstruktur

Zur Realisierung der in der Einleitung zu diesem Kapitel beschriebenen Anforderungen an das System, wurde auf das Paket PVM zur Durchführung von Remote Procedure Calls (RPC) zurückgegeriffen. Das Paket ist auf unterschiedlichen Plattformen und für verschiedene Programmiersprachen verfügbar. Es ermöglicht einen Austausch von Daten zwischen

verteilten Prozessen, die auf unterschiedlichen Rechnern und Plattformen ablaufen können. Basierend auf diesem Paket wurde eine *Master-Slave-Architektur* aufgebaut, in dem die *kognitive Komponente* als *Master* die übrigen Komponenten als *Slave* aufruft. Diese Architektur wurde weitestgehend von den Vorarbeiten übernommen (siehe [Holle 97], [Holle 98] und [Riepp 97]). Dabei teilen sich die einzelnen Komponenten die Aufgaben wie folgt (siehe Abbildung 5.1):

Kognitive Komponente.

Alle wesentlichen Schritte, die im letzten Kapitel vorgestellt wurden, sind Bestandteil der kognitiven Komponente; etwa Segmentierung, inhaltiche Erschließung, Generalisierung und inhaltliche Analyse, werden in dieser Komponente durchgeführt. Implementiert wurde die kognitive Komponente in der funktionalen Programmierspache *CLOS*. Auf Grund der Vielfältigkeit der Aufgaben der Komponente und der Tatsache, daß der Schwerpunkt dieser Arbeit sich auf diese Komponente bezog, wurde dieser Komponente ein eingener Abschnitt gewidmet, und sie wird im nächsten Abschnitt 5.1.2 näher beschrieben. Trotzdem seien die Aufgaben der kognitiven Komponente hier noch einmal Kurz aufgelistet:

- **Segmentierung** der Benutzervorführung.
- **Semantische Erschließung** der Benutzervorführung.
- **Generalisierung** von Trajektorie, Ausführungskontexten und Benutzerintention.
- **Inhaltliche Analyse** der Vorführung und Ableitung des Gesamtkontextes und der Verzweigungslogik und Bestimmung redundanter Segmente.
- **Makrogenese.** Erzeugung des Makrooperators.
- **Ausführung.** Ausfürhung erzeugter Makrooperatoren.
- **Datensicherung und -verwaltung.** Permanente Sicherung und Zugriff auf Daten aus Benutzervorführungen, auf Makrooperatoren und auf sochastischer Daten.

Simulator.

Die Simulationskomponente baute maßgeblich auf dem *Karlsruher Visualisierer* (*KaVis*) auf (siehe [Schaude et al.]). Der Komponente kommt innerhalb des Systems eine Sonderstellung zu. Einerseits dient sie natürlich zur Visualisierung von Abläufen, Hypothesen und sementatischen Beschreibungen gegenüber dem Benutzer. Zustätzlich kann der Benutzer in der Simulation Objekte manipulieren, um fehlerhafte Sensordaten zu korrigieren. Die Simulation dient weiter dazu, Objekte und Relationen auszuwählen. Damit ist sie als ein zentrales Element der Schnittstelle zum Benutzer anzusehen. Andererseits dient sie aber auch der Simulation von Benutzervorführungen zum Zweck der Analyse derselben. Tatsächlich greift die kognitive Komponente sehr häufig auf die Simulation zurück, um etwa Relationen zu evaluieren.[1] Damit kommen dem Simulator folgende Aufgaben zu:

[1] Da die Visualisierung im Vergleich zur reinen Simulation einen aufwendigeren Prozeß darstellt, existieren Überlegungen beide Funktionen auch in getrennte Module einzubetten.

- **Visualisierung** von Abläufen, Situationen und Relationen.
- **Manipulation** von Objekten zur Korrektur der Trajektorie aufgrund falscher Sensordaten.
- **Selektion** von Objekten als Korrektur von automatisch gewählten Objekten, etwa des gegriffenen Objektes, aber auch Unterstützung des Benutzers innerhalb anderer Werkzeuge, etwa der Evaluation bestimmer Relationen zwischen zwei zu wählenden Objekten.
- **Evaluation** von Relationen zwischen Objekten.

Benutzerschnittstelle.

Mit der Benutzerschnittstelle ist an dieser Stelle eine konventionelle dialog- und fensterbasierte Benutzerschnittstelle gemeint. Diese wird zur Verwaltung des Gesamtsystems und zur Entgegennahme von Kommentierungen durch den Benutzer benötigt. Sie dient aber auch dazu, dem Benutzer Rückmeldung über den Zustand des Systems, wie etwa auch über Fehlerzustände, zu informieren. Diese Komponete wurde Mithilfe der Skriptsprache *TCL* implementiert, wobei das Paket *TK* und dessen Erweiterung *TIX* verwendet wurde. Im wesentlichen arbeitet das System derart, daß ein Host auf der TCL-Seite ständig auf Ausführung von TCL-Kommandos wartet. Diese Kommendos werden durch die kognitive Komponente in TCL-Syntax erzeugt, wodurch sie direkt unter TCL ausführbar werden. Die eigentliche Programmierung der TCL-Schnittstelle findet als nicht unter TCL selbst statt, vielmehr werden CLOS-Kommandos auf TCL-Aufrufe abgebildet und unter der TCL-Shell ausgeführt. Damit wird eine optimale Flexibilität des Gesamtsystems erreicht. Insgesamt ergeben sich damit für die dialogbasierte Benutzerschnittstelle folgende Aufgaben:

- **Administration und Zugang** zum System.
- **Statusanzeige.** Anzeige des Zustandes des Gesamtsystems sowie von Fehlermeldungen.
- **Kommentierung.** Entgegennahme von Kommentierungen durch den Benutzer.
- **Hilfsmittel.** Bereitstellung von Hilfsmitteln, die beim PbD benötigt werden, etwa ein Evaluator für Relationen in einer simulierten Szene.
- **Ikonische Programmierumgebung.** An dieser Stelle wird auch eine Möglichkeit ikonisch Programme zu definieren.

Datentracker.

Der Datentracker hat die Aufgabe, die Sensordaten während einer Benutzervorführung aufzuzeichnen und an die kognitive Komponente weiterzuleiten. Dazu genügt es, daß der Datentracker einmal von der kognitiven Komponente vorinitialisiert wird. Mit der Initialisierung liefert der Datentracker kontinuierliche die sechs Freiheitsgrade der Handposition sowie 22 Werte, welche die Fingerstellung der Hand ckarakterisieren.

Effektorsteuerung.

Ein erzeugter Makrooperator kann unmittelbar mit einem Effektor, also einem Roboter, ausgeführt werden. Dazu ist allerdings eine Umsetzung der Transformationsmatrizen in Gelenkstellungen der Roboters notwendig.[2] Es gilt aber auch die Dynamik der Bewegung, etwa Geschwindigkeiten und Beschleunigungen, zu kontrollieren. Außerdem muß geprüft werden, ob die Punkte der Trajaktorie der Ausführung des Makrooperators tatsächlich im Arbeitsbereich des Roboters liegen. Diese Funktionalität wurde von [Pohl 98] implementiert. Damit fallen diesem Modul folgenden Aufgaben zu:

- **Inverse Kinematik.** Umsetzung der Positionstransformationen in Gelenkstellungen des Roboters.
- **Kontrolle der Dynamik.** Kontrolliertes Beschleunigen und Abbremsen des Effektors.
- **Kontrolle des Arbeitsraumes.** Testen der Trajektorie auf Abfahrbarkeit durch den Roboter

Griffklassifikator.

Die letzte verwendete Komponente bildet der Griffklassifikator (siehe [Grossmann 98]). Dieser bestimmt auf Grundlage der Fingerstellungen eine bestimmte Griffklasse nach der Cutkosky-Hierarchie (siehe [Cutkosky 89]). Hierzu wird er von der kognitiven Komponente des Systems mit der Fingerwinkeln versorgt, aus denen die entsprechende Griffklasse abgeleitet wird.

5.1.2 Kognitive Komponente

Die kognitive Komponente wurde in der Programmiersprache *LISP* bzw. deren objektorientierte Erweiterung *CLOS* implementiert. Eine genauere Beschreibung dieser Sprache sowie eine Darstellung ihrer Vorteile für die Aufgabe findet man in Abschnitt 5.2.2. Im letzten Abschnitt 5.1.1 hatten wir verschiedene Aufgaben dieser Komponente beschrieben. Um diese Aufgaben erfüllen zu können, gliedert sie sich ihrerseits in folgende Subkomponenten:

- **Episodenmanager.** Der Episodenmanager verwaltet Umwelten, deren Zustände sowie einzelne Benutzervorführungen als Episoden. Er ermöglich die semantische Erschließung von Episoden. Zudem besteht die Möglichkeit, Episoden auf Basis der Sensordaten aufzuzeichnen und wieder abzuspielen. Er übernimmt somit auch die Steuerung der Visualisierung einzelner Zustände in Episoden. Dabei wird immer sichergestellt, daß die interne Repräsentation des Umweltzustandes mit der des Simulators übereinstimmt.
- **Segmentierer.** Das Segmentierungsmodul erzeugt die in Abschnitt 4.2.3 beschriebene Erklärungsstruktur nach Abschnitt 2.1.3 zur Benutzervorführung. Der Segmentierer selbst ist modular aufgebaut, so daß er dynamisch erweiterbar ist; d.h. es

[2] Diesen Vorgang nennt man in der Regel die Berechnung der *Inversen Kinematik.*

können neue Regeln hinzugefügt werden, die zur Erzeugung einer Erklärungsstruktur dienen. Mit Zunahme solcher Regeln wird die Bereichstheorie des Systems erweitert. Im Grunde bildet der Segmentierer die deduktive Komponente des Lernsystems.

- **Sochastische Wissenskomponente.** Die stochastische Wissenskomponente ermittelt unüberwacht und vollkommen automatisch, alle benötigten sochastischen Informationen für typische Probleme. Dies geschieht in derart, daß nach jeder Aufzeichnung einer Episode und der anschließenden semantischen Erschließung derselben die entsprechenden Parameter aktualisiert werden. Dabei werden sowohl Zustandswahrscheinlichkeiten nach Gleichung 2.2 als auch Zustandsübergangswahrscheinlichkeiten nach Gleichung 2.4 bestimmt. Diese Daten werden asynchron und zyklisch permanent auf der Festplatte gesichert, so daß sich die Qualität der Schätzungen sukzessive verbessert. Das erworbene stochastische Wissen dient zur Hypothesenauswahl bei der Synthese von Makrooperatoren.

- **Prozedurale Wissenskomponente.** Die prozedurale Wissenskomponente dient zur Erzeugung, Verwaltung, Instanziierung und Ausführung von Makrooperatoren. Nach den Ergebnissen der inhaltlichen Analyse werden Makrooperatoren erzeugt. Diese lassen sich auf der Festplatte permanent zur späteren Ausführung und Weiterverwendung sichern. Bei der Ausführung eines Makrooperators ist dieser zunächst zu instantiieren. Danach werden die abzufahrenden Trajektorienpunkte sowie die Greiferstellungen bestimmt und an die Effektorsteuerung weitergegeben.

- **Analytische Komponente.** Die analytische Komponente dient zur Generalisierung und inhaltlichen Analyse von Benutzervorführungen. In der analystischen Komponenten wird echtes neues Wissen in Form von Makrooperatoren synthetisiert. Die analytische Komponente dient damit zur Induktion von Wissen.

5.2 Programmiertechniken und Paradigmen

Die modulare, verteilte Architektur ließ es zu, das Gesamtsystem in unterschiedlichen Programmiersprachen zu implementieren. Etwa wurden verwendet:

- **C/C++.** Alle zeitkritischen Operationen im Gesamtsystem wurden in dieser sehr effizienten Programmiersprache programmiert. Dies gilt etwa für die Effektorkontrolle und die Griffklassifikation.

- **TCL.** Wie bereits erwähnt, wurden die dialogbasierten Komponenten des System über die Skriptsprache TCL implementiert.

- **CLOS.** Alle Programmierarbeiten an dem Kern des Systems, der kognitiven Komponente, wurden in dieser Sprache verfaßt. Die nächsten Abschnitte werden sich mit der Frage beschäftigen, warum dies vorteilhaft war.

5.2.1 Selbstmodifizierenden Code und Knowledge Compilation

Ein Designprinzip des Systems lag darin, das erworbene Wissen möglichst effizient zu sichern und weiterzuverarbeiten. Ein bekanntes Verfahren, um erworbene Wissen effizienter auswerten und anwenden zu können, wird durch *Knowledge Compilation* gegeben. Das bedeutet, daß erworbenes Wissen nicht in Form von deklarativen Strukturen gespeichert wird, sondern vielmehr direkt in Programmcode der jeweiligen Srache übersetzt wird und damit direkt durch neue Funktionsaufrufe anwendbar ist. Dies setzt voraus, daß ein System seinen eigenen Programmcode modifizieren und erweitern kann (*Selbstmodifierzierende Programme*). Knowledge Compilation wird auf folgenden Ebenen eingesetzt:

- **Makrooperatoren.** Die Makrooperatoren werden bei ihrer Erzeugung direkt in Programmcode übersetzt und können dadurch optimiert ausgeführt werden.
- **Segmentierer.** Die Segmentierer bilden die Regeln zur Erklärung von Benutzervorführungen. Diese Regeln werden aber nicht deklarativ repräsentiert, sondern als Prozeduren kompiliert.

5.2.2 Motivation von CLOS als zentralem Werkzeug

Das *Common LISP Object System (CLOS)* bietet gute Voraussetzungen, Knowledge Compilation zu implementieren. Bei CLOS handelt es sich um eine dynamische Programmiersprache, die in der Lage ist, ihren eigenen Programmiercode zu modifizieren. In der Tat ist es so, daß sowohl Interpreter als auch Compiler selbst CLOS-Programme sind, die frei aufgerufen werden können. Das integrierte Makrokonzept von Common LISP macht es sehr einfach neues Wissen zu kompilieren, da mittels Makrofunktionen aus verschiedenen Parametern sehr vereinfacht neue Klassen und Methoden definiert werden können. Dieser Eigenschaft wurde genutzt, um einfache LISP-Makros zur Verfügung zu stellen, welche die Erstellung von neuen Makrooperatorklassen, Segmentierern und Algorithmen zur inhaltlichen Analyse stark vereinfachen.

Kapitel 6

Experimentelle Auswertung

Dieses Kapitel wird sich damit beschäftigen, inwieweit sich die Implementierung der in Kapitel 4 beschriebenen Verfahren am konkreten Problem bewährt hat. Dabei wird vor allem auf konzeptionell und zeitlich kritischen Komponenten eingegangen. Abschließend wird ein Beispiel eines fertigen Makrooperators gegeben, der mit Hilfe von physikalischen Vorführungen und ikonischer Programmierung interaktiv erzeugt wurde.

6.1 Bewertung des Laufzeitverhaltens

Beim Design des Gesamtsystems waren zwei Kriterien maßgeblich:

1. **Echtzeitfähigkeit.** Alle Komponenten sollten möglichst echtzeitfähig sein; d.h. daß alle Verarbeitungsschritte prinzipiell in derselben Geschwindigkeit durchgeführt werden können, die der Benutzer durch seine Vorführung vorgibt. Auch wenn wegen der notwendigen Kommentierungen durch den Benutzer die eigentliche Verarbeitung der Vorführung nicht parallel zur Vorführung stattfand, sollte die Verarbeitungsgeschwindigkeit doch analog gestaltet sein.

2. **Automatisiertes Lernen.** Alle intelligenten Komponente sollten derart autonom arbeitet, daß sie auch ohne Eingriff des Benutzers sinnvolle Hypothesen und verwertbares Wissen generieren.

Diese beiden Kriterien bestimmen die folgende Bewertung des Systems. Dabei soll deutlich gemacht werden, an welchen Stellen prinzipielle Probleme vorliegen, die eine weitere Optimierung des Systems unmöglich machen, und welche Engpässe durch eine Verbesserung der technischen und konzeptionellen Implementierung behoben werden können.

6.1.1 Datenhandschuh

Der Tracker für die Fingerstellungen des Datenhandschuhs und der Positionssensor für die Lokation des Handrückens bildeten die einzigen Sensoren des Systems. Damit ist klar, daß die Qualität der von ihm gelieferten Daten stark die Performanz des Gesamtsystems beeinflußt. Es ergab sich während der Arbeit, daß der Datenhandschuh als einzige Quelle

von Sensordaten starke Unzulänglichkeiten aufweist. Insbesondere ergaben sich folgende Probleme:

- **Ungenauigkeit der Sensordaten.** Das Sensorsystem des Datenhandschuhs arbeitet im wesentlichen magnetfeldbasiert, was das System gegeben über solchen Störeinflüssen, die von jedem elektrischen Gerät verursacht werden, anfällig macht. [Stasch 96] führte eine systematische Untersuchung des Sensorsystems in Bezug auf die euklidsche Position des Handrückens im Raum durch. Dabei ergab sich, daß bei dem Sensorsystem nicht-systematische Ungenauigkeiten im zentimeterbereich üblich waren. Im Extremfall wurden Abweichung von bis zu 5cm beobachtet. Hinzu kommt der Umstand, daß selbst diese große Ungenauigkeit nur innerhalb eines Würfels mit einer Kantenlänge von 60cm um des Sensor zu erreichen ist. Außerhalb dieses Würfels sind keine verwertbaren Ergebnisse zu erwarten. Auch wenn Abweichungen in der Rotation des Handrückens nicht systematisch untersucht wurden, steht auf Grund der analogen Auswertung der Sensordaten zu vermuten, daß sich auch hier nicht unerhebliche Abweichungen ergeben.

- **Skalierungsproblem.** Des weiteren zeigte sich, daß die Erfassung der Fingerstellungen, also der Gelenkwinkel der Finger, stark mit der Geometrie der Hand des Trägers des Datenhandschuhs variierte. Versuche eine benutzerspezifische Kalibrierung des Sensors automatisch zu erstellen, waren nicht erfolgreich. Auch das Geometriemodell des Simulators war aus demselben Grund, in der Regel unzulänglich, da die Längen von Fingern und Handrücken zwischen Benutzer signifikante Abweichungen aufweisen.

- **Aufzeichnungsproblem.** Zur zyklischen Abfrage der Daten wurde ein Silicon Graphics Workstation verwendet. Diese sollte in einer festen Samplerate Daten vom eigentlichen Tracker abgefragn. Die Triggerung der Aufzeichnungspunkte wurde durch interne Timer der Workstation bewerkstelligt. Es stellte sich aber heraus, daß diese Timer für zeitkritische Echtzeitanwendungen nicht vollkommen geeignet waren, so daß Faktoren, wie die Auslastung des Rechners, einen Einfluß auf die absoluten Zeitabstände zwischen den einzelnen Aufzeichnungszeitpunkten hatten. Durch diesen Umstand werden die relativ großen Schwankungen in der Geschwindigkeit in Abbildung 4.4 erklärt.

Während die ersten beiden Faktoren technologisch kaum noch Optimierungen zulassen, ist das dritte Problem, das allerdings die geringsten Schwierigkeiten verursachte, evtl. durch den Einsatz eines Echzeitsystems zu beheben.

Die Daten über die Fingerstellungen ließen eine relativ zuverlässige Erkennung von Greif- und Loslaßpunkten zu. Bei der Bestimmung der Trajektorie und damit der Festlegung der Position der Hand an der Greif- und Ablagepunkten und damit das gegriffene Objekt waren die gewährleistete Genauigkeit der Daten generell unzulänglich. In der Tat war es so, daß die Nachbesserung und Korrektur der Trajektorie den Hauptaufwand der Benutzerinteraktion verursachten. Ohne eine manuelle bzw. halbautomatische Nachbesserung der Sensordaten, waren überhaupt keine brauchbaren Ergebnisse zu erwarten. Generell muß festgehalten werden, daß der Datenhandschuh als alleinige Quelle für Sensordaten nicht ausreichend ist.

6.1.2 Simulation und Umweltauswertung

Die Simulation erwies sich als sehr zuverlässig und war für die gewählte Aufgabe prinzipiell gut geeignet. Es ergaben sich bei der Verwendung der Simulation nur zwei Probleme:

- **Übertragungskosten.** Während der Implementierung zeigte es sich, daß sich die zeitlich Verzögerung einer Anfrage unabhängig von deren Komplexität stets im Bereich von 50ms lag. Auch Anfragen, die nur einzelne fest kodierte Werte auslasen, waren nicht performanter. Hierbei muß noch einmal erwähnt werden, daß die Steuerung der Simulation über das RPC-Paket PVM erfolgte; d.h. jede Anfrage bzw. jedes Kommando wurde über das Netz an die Simulation übergeben, die auf demselben Weg antwortete. Prinzipiell können derartige bestätigte Anfragen von PVM erheblich schneller bewältigt werden, bei niedriger Netzlast zuverlässig unter 5ms.

  Die um einen Faktor 10 höhere Verzögerung läßt sich durch den internen Scheduler des Open Inventor Schedulers, auf dem KaVis basiert, erklären. Da die internen Datenstrukturen von KaVis nicht *threadsave* sind, also nicht von mehreren Prozessen gleichzeitig bearbeitet werden können, müssen alle Operationen, die Daten abfragen oder modifizieren sequentiell über den Scheduler aufgerufen werden. Dies bedeutet, daß auch PVM-Anfragen über diesen Scheduler abgehandelt werden müssen. Prinzipiell stellt dies kein Problem dar. Allerdings zeigte sich, daß der Scheduler im wesentlichen synchron zu der Framerate der Bildschirmdarstellung arbeitete. Open Inventor baut hierbei 20 mal in der Sekunde die Szene neu aus den aktuellen Szenedaten auf. Als Folge wird auch die Routine, die mögliche PVM-Anfragen abhandelt nur etwa 20 mal in der Sekunde aufgerufen, die dann genau eine Anfrage bearbeitete. Als Resultat ergab sich die durchschnittliche Verzögerung der Antwort als $\frac{1}{20}s = 50ms$.

- **Boundingboxen.** Eine wesentliche Aufgabe der Simulation bestand darin, Relationen zwischen den Objekten der aktuellen Szene auszuwerten. Diese Relationen werden in der gegenwärtigen Implementierung mit Hilfe sog. Boundingboxen bestimmt. Dieses sind achsenparallele Quader, welche die Objekte minimal aber vollständig umschließen. Diese Darstellung war gut geeignet, Relationen grob zu bestimmen. Detailliertere Anfrage, wie etwa die nach Kollisionen zwischen Objekten, konnten damit aber nur mit ungenügender Genauigkeit beantwortet werden.

Die relativ großen Verzögerungen bei der Beantworten von Anfragen an die Simulation setzten Beschränkungen bei der Implementierung. So mußte jede Interaktion der kognitiven Komponente mit der Simulation in Bezug auf die Anfragen minimiert werden. Ein wesentlicher Beitrag zu dieser Optimierung stellt die regelbasierte Auswertungslogik des aktuellen Umweltzustandes nach Abschnitt 4.2.4 dar. Damit ließen sich in der Regel n Relationen mit weniger als $0.05 \cdot n$ Aufrufen auswerten. Aber selbst diese Optimierung erwies sich als nicht ausreichend. Bei komplexeren Umwelten mit mehr als zehn Objekten nahm die inkrementelle Fortschreibung des Zustandes der Welt in einem Schritt mehrere Sekunden in Anspruch. Dieser Teil des Systems konnte dem Anspruch der Echtzeitfähigkeit nicht gerecht werden.

6.1.3 Lernverfahren

Bei der Bewertung der Lernverfahren war weniger die Performanz in Bezug auch die Geschwindigkeit der Verarbeitung relevant, sondern vielmehr deren Akkuratheit bei der Induktion neuen Wissens. Bei der Bewertung sollen die Analyse und die Generalisierung getrennt betrachtet werden.

Generalisierung

Wesentlicher Bestandteil des Systems zur Auswahl von Generalisierungshypothesen bildete der Dreipaseninformationsfilter aus Abschnitt 4.3. Dieser basiert wesentlich auf der korrekten Bestimmung der Zustands- und Übergangswahrscheinlichkeiten. Die Anzahl der freien Parameter ergibt sich bei den Zustandswahrscheinlichkeiten als $O(|\mathcal{R}| \cdot |\mathcal{V}|)$, also dem Produkt aus der Anzahl der Relationen und der Anzahl der Werteausprägungen. Für die Anzahl der beobachteten Zustandsübergänge ergibt sich nach Axiom 2.3 lediglich die Anzahl $O(|\mathcal{R}|)$, wenn $|\mathcal{V}| = 2$ gilt. Aufgrund der relativ kleinen Anzahl freier Parameter, die das gewählte Verfahren benötigt, ergibt sich sehr schnell eine zuverlässige Schätzung derselben. Tatsächlich war es so, daß bereits die stochastische Analyse einer einzigen Vorführung ausreichte, um brauchbare Schätzungen der Parameter zu erhalten.

Der Dreiphaseninformationsfilter liefert relativ zuverlässig Kontexthypothesen, die in der Regel direkt eine Erzeugung von brauchbaren Makrooperatoren zuläßt. Allerdings waren diese Hypothesen zu sehr auf die konkret beobachtete Manipulation fixiert, was dazu führte, daß übergeordnete Zusammenhänge nicht erkannt wurden. Das System war auch nicht flexibel gegenüber bestimmten Vorzügen von Benutzern, die bestimmte Erklärungen in der Regel systematisch bevorzugen. Um hier eine größere Adaptivität und gleichzeitig eine größere Verläßlichkeit zu erreichen, ist das bestehende Verfahren noch zu erweitern.

Analytische Verfahren

Die analytischen Verfahren selbst erwiesen sich als geeignet. Der größte Nachteil des Verfahrens lag in der quadratisch wachsenden Komplexität der Konjunktionen des Gesamtkontextes (siehe Abschnitt 4.4). Dieser Umstand kann bei dem gewählten Verfahren prinzipiell nicht behoben werden. Ein weiteres Problem ergab sich dadurch, daß das Verfahren streng systematisch arbeitet. Dies hat zur Folge, daß falsche Benutzerangaben unmittelbar in unbrauchbare Makrooperatoren übersetzt werden. Jedwede automatische Plausibilitätsprüfungen des Endresultates fehlen noch vollständig.

6.2 Anwendungsbeispiel: Cranfield-Benchmark

In diesem Abschnitt soll ausführlich ein konkretes Beispiel gegeben werden, das alle Verarbeitungsschritte des Verfahrens in der konkreten Implementierung und einen resultiereden Makrooperator beschreibt. Als Beispiel wurde der *Cranfield-Benchmark* gewählt, eine Montageaufgabe mit Manipulationsaufgaben unterschiedlicher Komplexität. Die einzelnen Bauteile der Montageaufgabe werden in Abbildung 6.1 dargestellt.

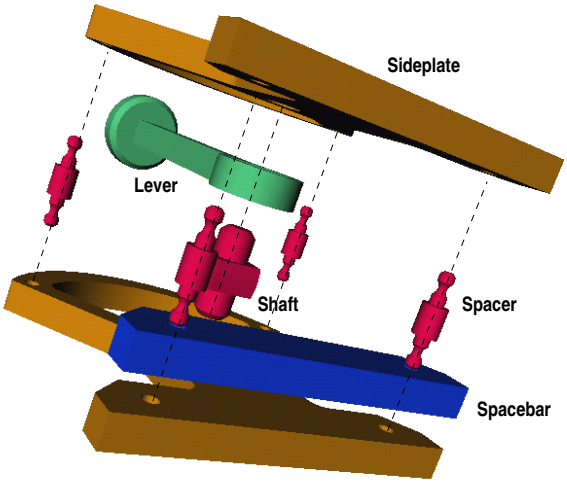


Abbildung 6.1: Montageplan des Cranfield-Benchmarks

6.2.1 Makrogenese

Zur Verdeutlichung der einzelnen Verarbeitungsschritte zur Erzeugung eines Makrooperators greifen wir auf die Episode zurück, die in Abbildung 4.11 abgebildet wurde. Die Erzeugung einer Makrooperators erfolgt nach folgenden Schritten:

1. **Schlüsselereignisse.** Zunächst müssen die Schlüsselereignisse der Vorführung, wie Greif- und Loslaßpunkte oder Kontextwechsel festgelegt werden. Dies geschieht vollautomatisch durch das System. Der Benutzer erhält aber, wie in Abbildung 6.2 gezeigt, die Möglichkeit, alle Hypothesen über Lage dieser Schlüsselereignisse sowie die Hypothesen über die an der Manipulation beteiligten Objekte zu korrigieren.

2. **Segmentierung.** Danach wird vollautomatisch eine Segmentierung und damit eine Erklärung der Vorführung erzeugt. Abbildung 6.3 zeigt einen Ausschnitt einer Erklärungsstruktur.

3. **Ausführungskontexte.** Nachdem die Segmentierung erzeugt wurde, werden nun die Hypothesen für die generalisierten Ausführungskontexte erzeugt. Auch diese lassen sich manuell bearbeiten, wie Abbildung 6.4 zeigt. Die in der Abbildung gezeigte Hypothese auf der rechten Seite, entspricht direkt der Hypothese des Systems. Dabei wurden zur Schätzung der freien Parameter der Wahrscheinlichkeiten lediglich die gerade analysierte Episode verwendet.

4. **Benutzerintention.** Auch die Generalisierung der Manipulation der gesamten Episode und damit die Festlegung der Benutzerintention wird vom System vollautomatisch durchgeführt, kann aber durch den Benutzer weiter kommentiert werden. Abbildung 6.5 zeigt aus der rechten Seite die Hypothese für unsere Beispielepisode

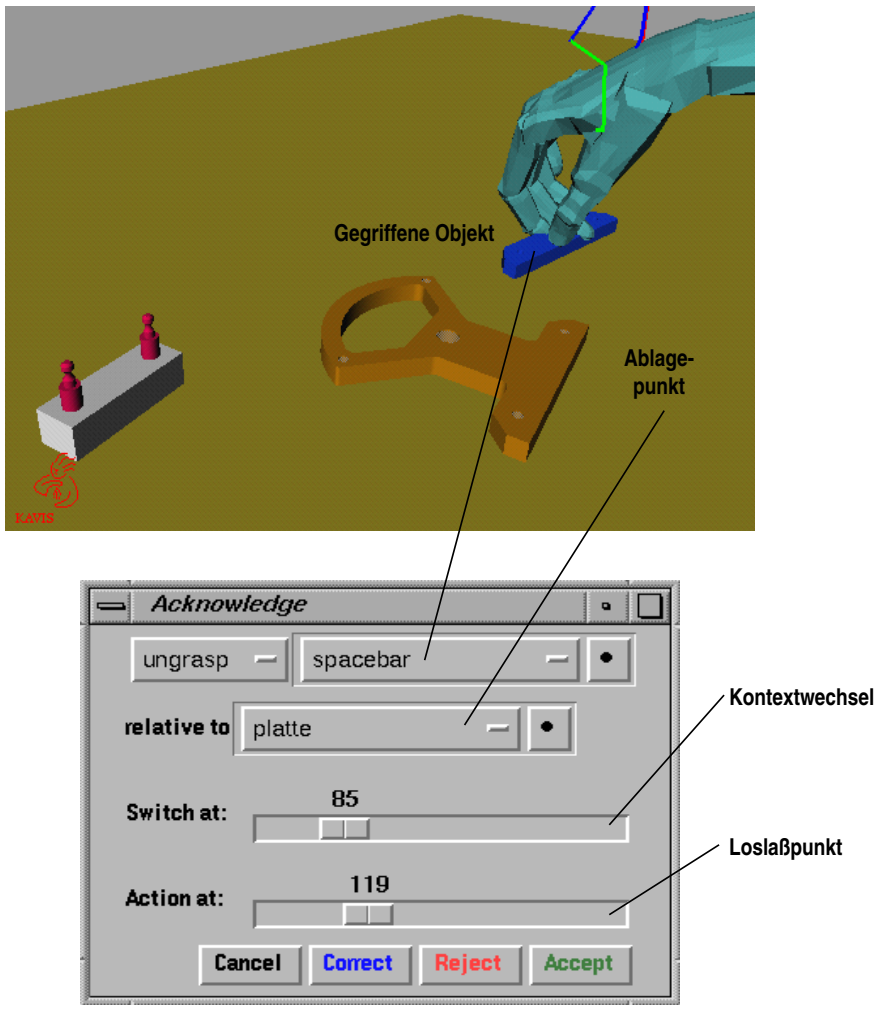


Abbildung 6.2: Korrekturen des Benutzers zur Segmentierung

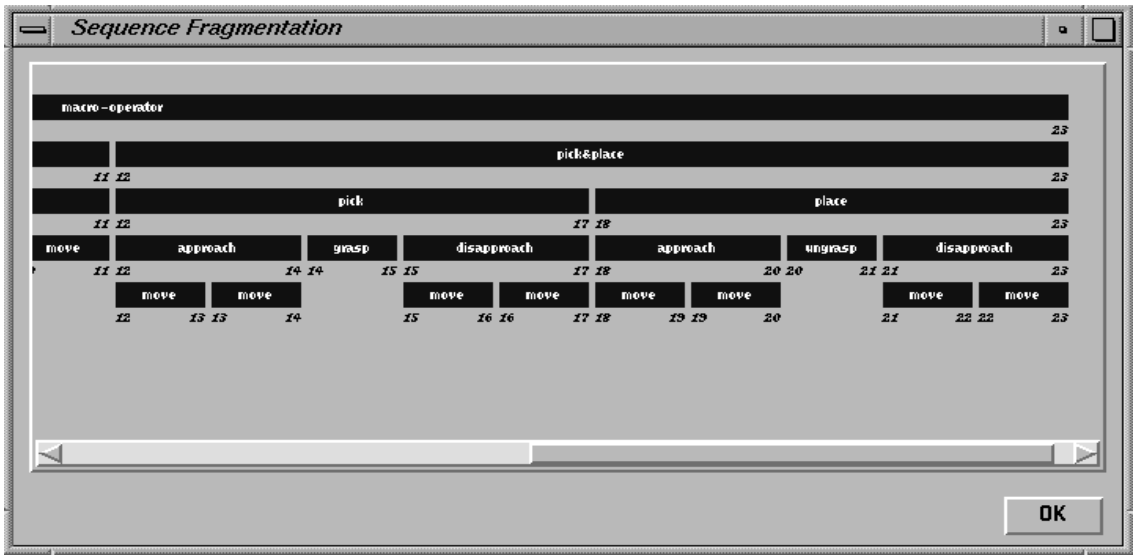


Abbildung 6.3: Ausschnitt der Erklärungsstruktur einer Vorführung

und die Möglichkeiten zu deren Modifikation. Auch diese Hypothese kam unter den Umständen zustande, die unter dem letzten Punkt geschildert wurden.

5. **Makrogenese.** Auf Basis der Erklärungsstruktur und der Hypothesen über die generalisierten Ausführungskontexte und die Benutzerintention läßt sich direkt ein Makrooperator aus der Episode erzeugen.

6. **Ikonische Programmierung.** Wie in den letzten Schritten beschrieben, lassen sich einzelne Teilaufgaben eines komplexen Problems programmieren. Diese lassen sich dann zu komplexeren Makrooperatoren mit Hilfe der ikonischen Programmierung zusammenfassen. Abbildung 6.6 zeigt die komplette Struktur des Makrooperators, der in der Lage ist, den Cranfield-Benchmark komplett zu montieren. Interessant dabei ist, daß der Makrooperator *INSERT-SPACER* nur einmal vorgeführt werden mußte, aber für das Einfügen aller vier *Spacer* verwendet werden konnte. Dies führt den Vorteile ikonischer Programmierung vor Augen, in komplexen Aufgaben bereits bekannte Lösungen für Teilaufgaben wieder- und mehrfach verwerten zu können.

6.2.2 Beispielmakro

In Abbildung 6.7 wird noch einmal den Schritt von der Benutzervorführung über die Simulation bis zur simulierten Ausführung mit dem Roboter gezeigt. Abbildung 6.6 zeigt den kompletten Makrooperator, wie er aus einzelnen Lösungen für Teilprobleme mittels ikonischer Programmierung konstruiert wurde. Dieser wird in Abbildung 6.8 auf einer Veränderten Umwelt mit einem Roboter in der Simulation unter KaVis ausgeführt.

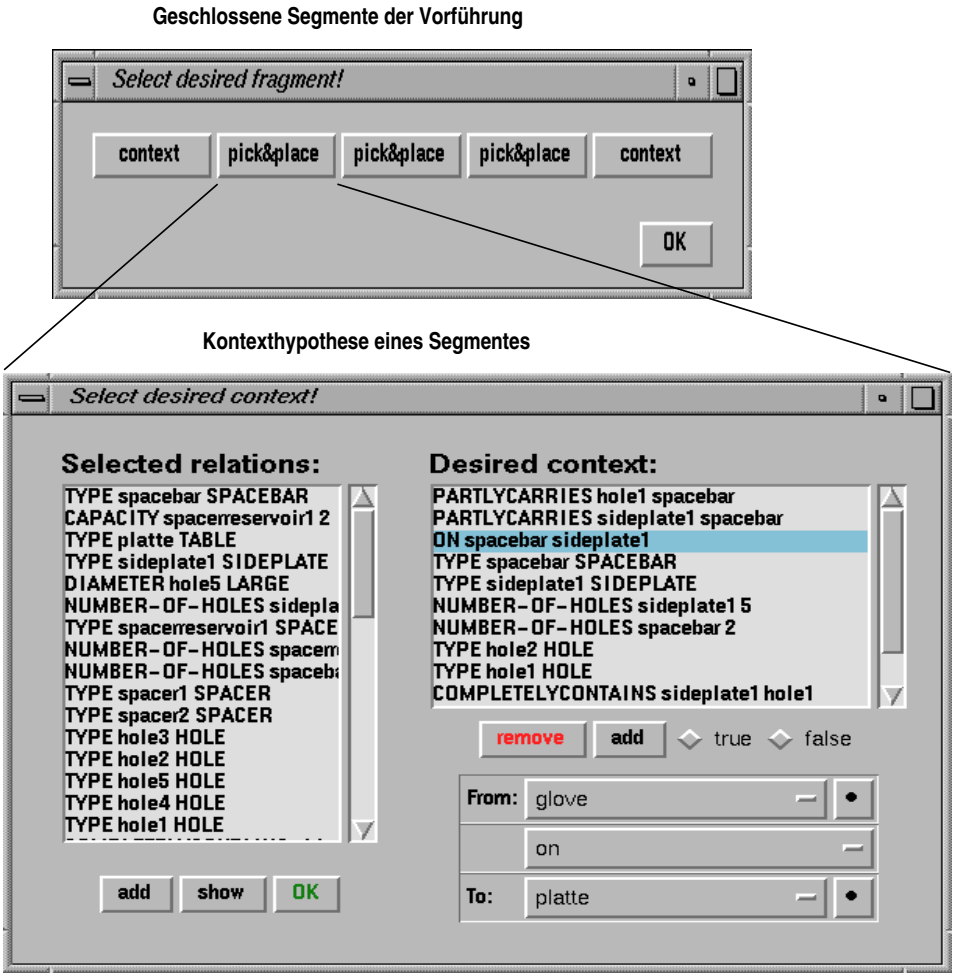


Abbildung 6.4: Nachbearbeitung der Hypothesen für die Ausführungskontexte

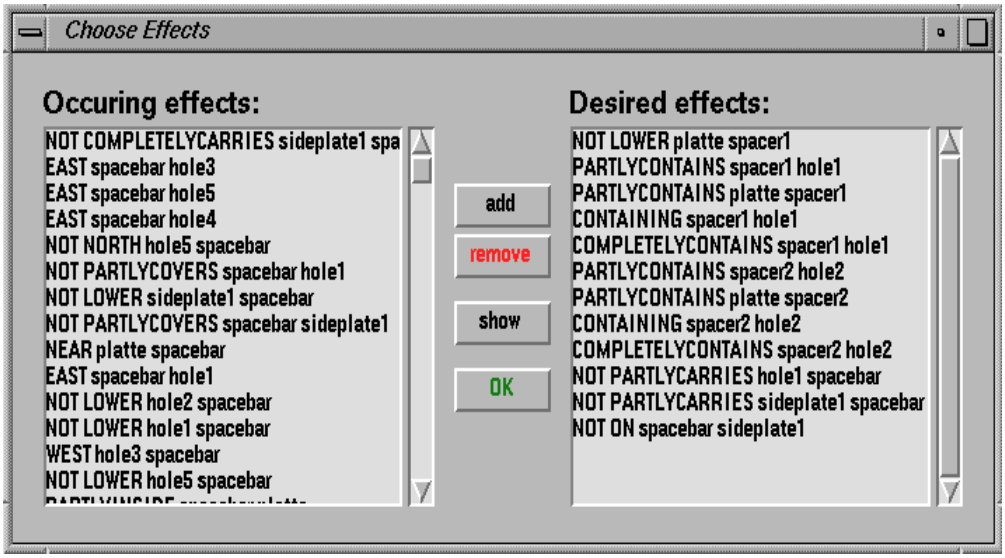


Abbildung 6.5: Korrektur der Hypothese über Manipulation

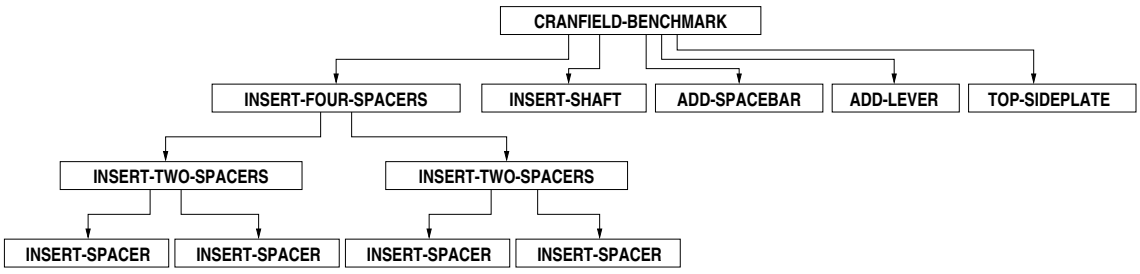


Abbildung 6.6: Struktur des Makros *CRANFIELD-BENCHMARK*

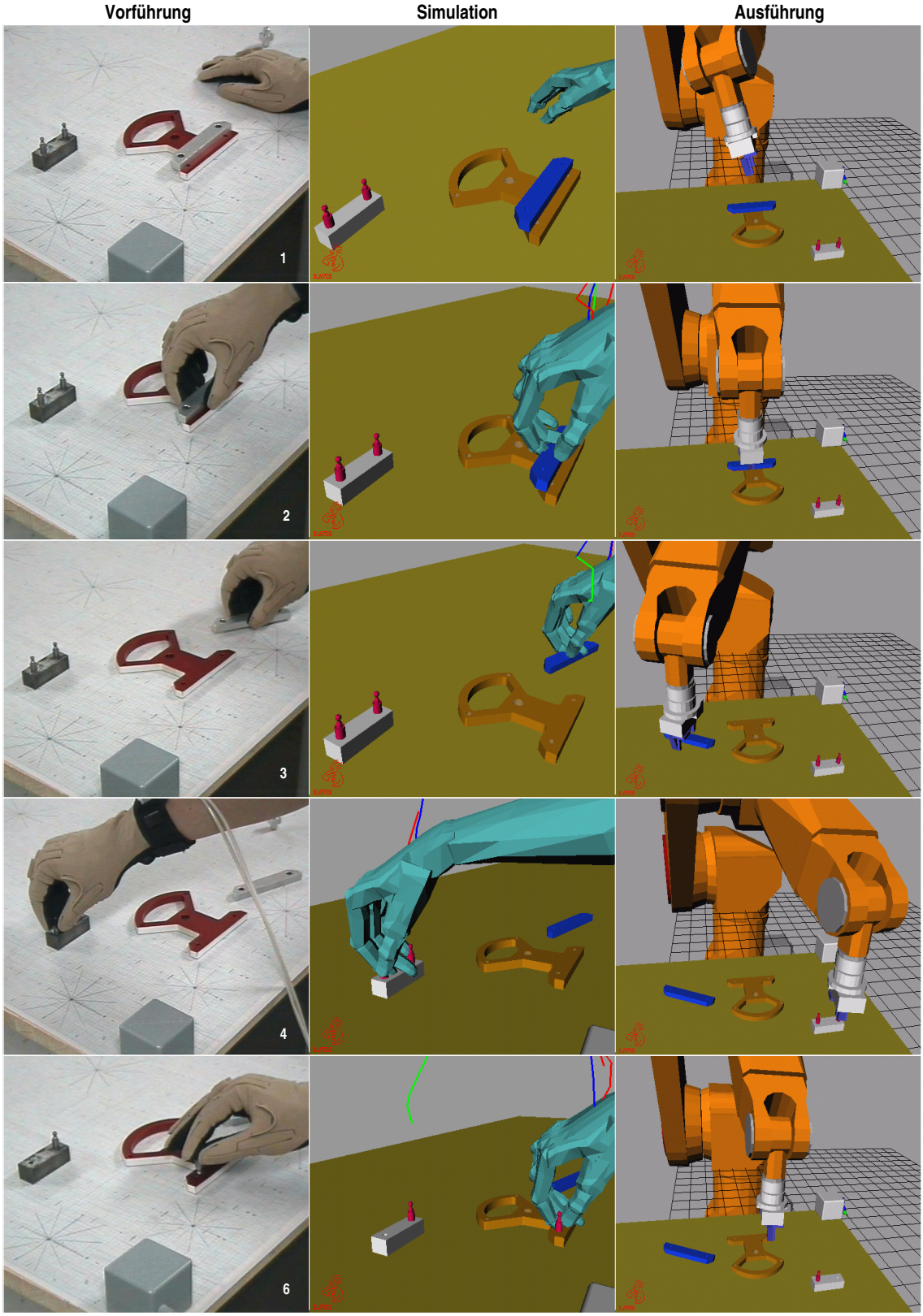


Abbildung 6.7: Vorführung, Simulation und Ausführung einer Episode bzw. eines Makros

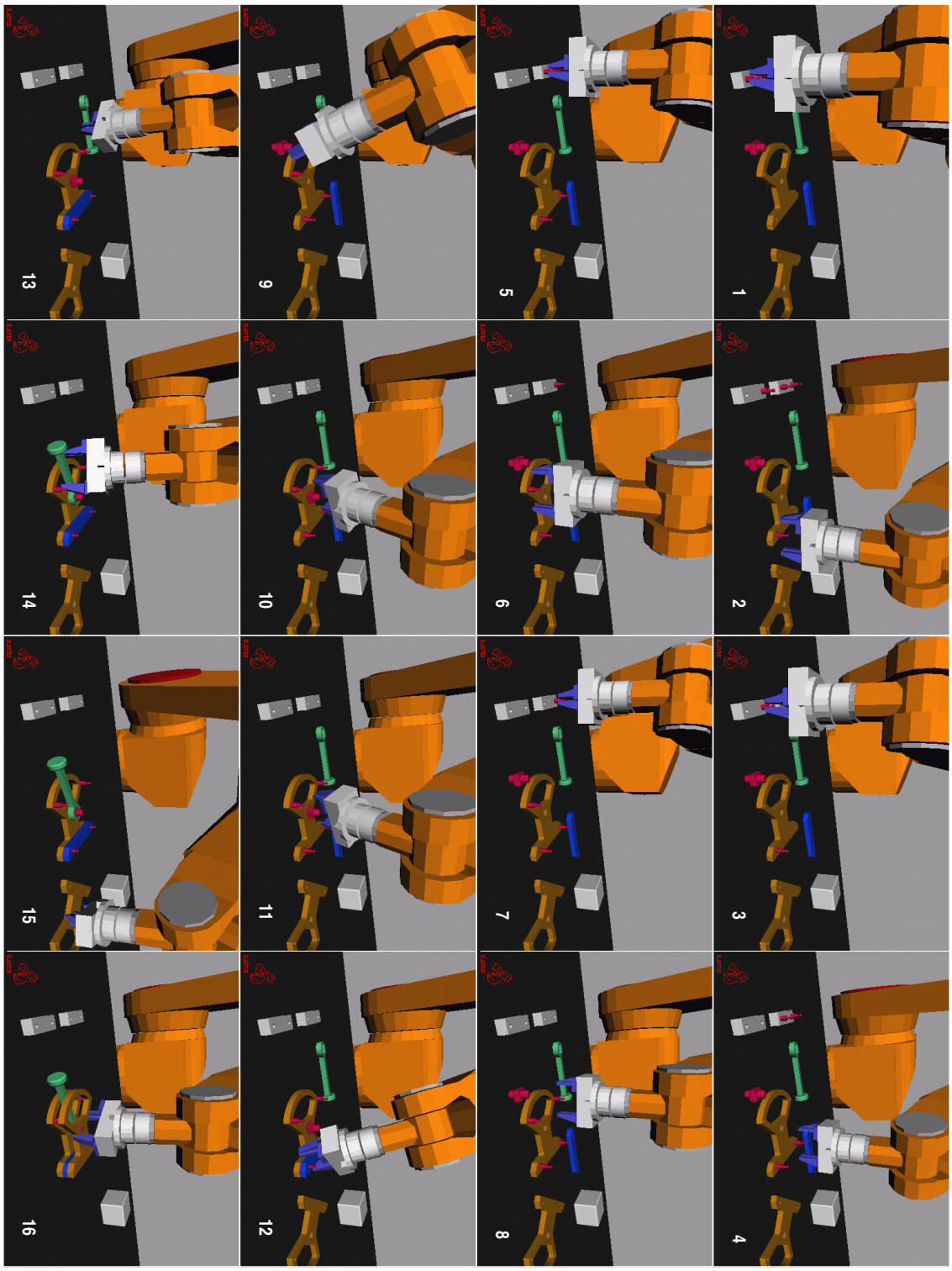


Abbildung 6.8: Ausführung des Cranfield-Benchmark-Makros mit Roboter

Kapitel 7

Zusammenfassung und Ausblicke

Mit der vorliegenden Arbeit wurde ein Verfahren zur Generierung von Makrooperatoren aus Benutzervorführungen vorgestellt. Das Verfahren läßt die Generalisierung der Benutzervorführung über die Vorführumwelt zu. Des weiteren werden redundante Segmente entdeckt. Für nicht-redundante Segmente werden Verzweigungsbedingungen erzeugt. Die gewonnenen Makrooperatoren sind damit auf veränderten Umwelten ausführbar und können auf diese adaptiv reagieren. Obwohl das implementierte Systeme weitgehend vollautomatisch arbeitet, läßt es Kommentierung des Benutzers zu, um inkorrekte Hypothesen des Systems zu korrigieren und zu verbessern. Anhand konkreter Beispiel konnte die Funktionalität des Systems demonstriert werden.

Gleichzeitig wurde aber auch auf signifikante Schwachstellen des bestehenden Systems hingewiesen, die eine Benutzerinteraktion notwendig machen. Der größte Schwachpunkt des Systems lag in den Ungenauigkeiten des Sensorsystems. Die technische Funktionsweise des Sensors läßt aber keine exakteren Positionsbestimmungen erwarten. So erscheint es unausweichlich, daß in dem System zusätzliche Sensoren eingeführt werden. Der Einsatz eines Stereokamerasystems könnte Ungenauigkeiten des Sensors ausgleichen. Um aber verschiedene Datenquellen für denselben Parameter verwerten zu können, wären adäquate Methoden zur *Sensor-Fusion* zu implementieren. Konventionelle Ansätze auf diesem Gebiet arbeiten auf der Datenebene. Im konkreten Ansatz ist aber zu überdenken, ob nicht auf der kognitiven Ebene eine Verschmelzung der Daten vorgenommen werden sollte, die intelligent entscheidet, welche Datenwerte plausibler und damit zu bevorzugen sind.

Aber auch die Selektion der Generalisierungshypothesen für Ausführkontexte und die Benutzerintention wurden nicht immer optimal vom System selektiert, so daß auch an dieser Stelle häufig die Kommentierung durch den Benutzer nötigt wurde. An dieser Stelle scheinen rein statistische Verfahren nicht ausreichend. Der Einsatz von intelligenten regelbasierten Verfahren oder anderen unüberwachten Lernverfahren sind denkbar. Ein möglicher nächster Schritt wäre etwa die Integration Fallbasierten Schließens (engl. *Case-based Reasoning*). Damit ließen sich die Kommentierungen für bereits bekannte Fälle, die der vorliegenden Benutzervorführung ähnlich sind, analog übertragen, so daß sich bei einer inkrementell wachsenden Fallbasis, die Hypothesen für die Generalisierung sukzessive verbessern und an die Gewohnheiten des Benutzers anpassen würden.

Zuletzt erwies sich das Konzept zur Repräsentation der Trajektorie als zu statisch. An dieser Stelle wären Konzepte denkbar, die eine intelligente Planungsstrategie einer Bewe-

gung lernen, anstatt nur die Benutzervorführung punktweise abzufahren. Die Idee ließe sich derart extrapolieren, daß die Makrooperatoren weiter zu sog. *kognitiven Operatoren* abstrahiert werden. Damit ließen sich auch Greif- und Ablagepunkte intelligent planen, die bislang in einer festen Relation zu einem bestimmten Objekt angenommen werden. Dies hätte den Vorteil, daß etwa bei der Ablage eines Objektes auf einem Tisch, die Operation nicht dadurch vereitelt würde, daß der konkrete Ablagepunkt bereits belegt ist. Würde der Ablagepunkte *kognitiv* über Relationen, etwa *auf dem Tisch*, beschrieben, wäre über zu implementierende Regeln ein alternativer Ablagepunkt bestimmbar.

Die gemachten Vorschläge würden die automatische Verarbeitung einer Benutzervorführung vereinfachen und die Makrooperatoren insgesamt noch flexibler gegenüber veränderten Umweltbedingungen machen.

Literaturverzeichnis

[Archibald 93] C. Archibald, E. Petriu. *Computational paradigm for creating and executing sensorbased robot skills.* In *24th International Symposium on Industrial Robots*, Seiten 401 – 406, Tokyo, 1993.

[Bocionek 94] S. Bocionek, M. Sassin. *Programmieren durch Vormachen.* Informatik Spektrum, Ausgabe 17, Seiten 309 – 311, 1994.

[Cutkosky 89] M.R. Cutkosky. *On grasp choice, grasp models, and the design of hands for manufacturing tasks. IEEE Trans. Robotics and Automation*, 5. Jahrgang, Ausgabe 3, Seiten 269–279, 1989.

[Cypher 91] Allen Cypher. *Eager: Programming Repetitive Tasks by Example.* In *Proceedings of the International Conference on Computer-Human Interaction (CHI'91)*, Seiten 33–39, New Orleans, USA, 1991. ACM.

[Cypher 93] A. I. Cypher. *Watch what I do – Programming by Demonstration.* MIT Press, Cambridge, Massachusetts, 1993.

[DeJong 86] G. DeJong, R. Mooney. *Explanation-based Learning - An alternative view. Machine Learning*, 1. Jahrgang, Seiten 145 – 176, 1986.

[Delson 94a] N. Delson, H. West. *Robot Programming by Human Demonstration: The Use of Human Variation in Identifying Obstacle Free Trajectories.* In *IEEE International Conference on Robotics and Automation*, Band 1, Seiten 564 – 571, San Diego, 1994.

[Delson 94b] N. Delson, H. West. *The use of human inconsistency in improving 3D robot trajectories.* In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Seiten 1248 – 1255, München, 1994.

[Dillmann 95a] R. Dillmann, H. Friedrich, M. Kaiser. *Integration of subsymbolic and symbolic learning to support robot programming by human demonstration.* In *International Symposium on Robotics Research*, Munich, Germany, 1995.

[Dillmann 95b] R. Dillmann, M. Kaiser, V. Klingspor, K. Morik, F. Wallner. *Teaching and understanding intelligent service robots: A Machine Learning Approach.* In *19. Deutsche Jahrestagung für Künstliche Intelligenz (KI '95 Workshop: Informationsverarbeitung in Servicerobotern)*, Bielefeld, 1995.

[Duda 73] R.O. Duda, P.E. Hart. *Pattern Classification and Scene Analysis.* J. Wiley, 1973.

[Dufay 84a] B. Dufay, J.-C. Latombe. *An Approach to Automatic Robot Programming Based on Inductive Learning.* In *Proceedings of Robotics Research: The 1st International Symposium*, M. Brady, R. Paul, Editoren, Seiten 97–115. The MIT Press, 1984.

[Dufay 84b] B. Dufay, J.-C. Latombe. *An approach to automatic robot programming based on inductive learning.* In *1st International Symposium on Robotics Research*, Seiten 97 – 115, 1984.

[Fikes 72] R.E. Fikes, P.E. Hart, N.J. Nilsson. *Learning and executing generalized robot plans. Artificial Intelligence*, 3. Jahrgang, Seiten 251 – 288, 1972.

[Friedrich 95a] H. Friedrich, Editor. *Proceedings of the Programming by Demonstration Workshop at the International Conference on Machine Learning*, Lake Tahoe, California, July 1995.

[Friedrich 95b] H. Friedrich, R. Dillmann. *Robot Programming Based On A Single Demonstration and User Intentions.* In *Proceedings of the 3rd European Workshop on Learning Robots at ECML'95*, Heraklion, Crete, Greece, April 1995.

[Friedrich 96a] H. Friedrich, M. Kaiser, R. Dillmann. *PbD - The Key to Service Robot Programming.* In *Acquisition, Learning, and Demonstration: Automating Tasks for Users*, AAAI Spring Symposium Series, Seiten 23–29, Palo Alto, CA, USA, March 25-27 1996. AAAI Press, Menlo Park, California.

[Friedrich 96b] H. Friedrich, S. Münch, R. Dillmann, S. Bocionek, M. Sassin. *Robot Programming by Demonstration: Supporting the Induction by Human Interaction. Machine Learning*, 23. Jahrgang, Seiten 163 – 189, 1996.

[Friedrich 97] H. Friedrich, M. Kaiser, R. Dillmann. *What can Robots learn from Humans?* In *Annual Reviews in Control*, J. Gertler, Editor, Band 20, Kapitel Robotics, Seiten 167–172. Elsevier, 1997.

[Friedrich 98] H. Friedrich, J. Holle, R. Dillmann. *Interactive Generation of Flexible Robot Programs.* In *IEEE International Conference on Robotics and Automation*, Leuven, B, 1998. submitted.

[Grossmann 98] Volker Grossmann. *Eekennen menschlicher Griffarten mittels neuronaler Netze.* Diplomarbeit, Universität Karlsruhe, Fakultät für Informatik, Institut für Prozeßrechentechnik und Robotik, 1998.

[Group 94] Open Inventor Architecture Group. *Open Inventor^{TM} C^{++} Reference Manual, The Official Reference Document for Open Inventor.* Addison-Wesley Publishing Company, 1994. Release 2.

[Heise 89a] R. Heise. *Demonstration instead of programming: Focussing attention in robot task acquisition.* Technischer Bericht 89/360/22, Department of Computer Science, University of Calgary, 1989.

[Heise 89b] Rosanna Heise, Bruce A. MacDonald. *Robots acquiring tasks from examples.* In $2^n d$ *International Symposium on Artificial Intelligence*, Monterrey, Mexiko, October 1989.

[Heise 92] Rosanna Heise. *Programming Robots by Example.* Technischer Bericht 92/476/14, Department of Computer Science, The University of Calgary, Canada, 1992.

[Holle 97] Jörg Holle. *Räumliche Relationen in der Virtuellen Realität.* Studienarbeit, Universität Karlsruhe, Fakultät für Informatik, Institut für Prozeßrechentechnik und Robotik, 1997.

[Holle 98] Jörg Holle. *Interactive Roboterprogrammierung mittels Datenhandschuh.* Diplomarbeit, Universität Karlsruhe, Fakultät für Informatik, Institut für Prozeßrechentechnik und Robotik, 1998.

[Kaiser 95a] M. Kaiser, H. Friedrich. *Transferring Human Knowledge to Robots.* In *Meeting GI Special Interest Group Machine Learning*, Dortmund, Germany, 1995. Also available via anonymous ftp from ftpipr.ira.uka.de.

[Kaiser 95b] M. Kaiser, H. Friedrich, R. Dillmann. *Obtaining good performance from a bad teacher.* In *International Conference on Machine Learning, Workshop on Programming by Demonstration*, Tahoe City, California, 1995.

[Kang 92] Sing Bing Kang, Katsushi Ikeuchi. *Grasp recognition using the contact web.* In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'92)*, 1992.

[Kang 93a] Sing Bang Kang, Katsushi Ikeuchi. *Toward automatic robot instruction from perception – Recognizing a grasp from observation. IEEE Transactions on Robotics and Automation*, 9. Jahrgang, Ausgabe 4, August 1993.

[Kang 93b] Sing Bing Kang, Katsushi Ikeuchi. *A Grasp Abstraction Hierarchy for Recognition of Grasping Tasks from Observation.* In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*, Band 1, Seiten 621–628, Yokohama, Japan, July 26-30 1993.

[Kang 94a] Sing Bang Kang. *Robot Instruction by Human Demonstration.* Dissertation, Carnegie Mellon University, Pittsburgh, PA, 1994.

[Kang 94b] Sing Bing Kang, Katsushi Ikeuchi. *Determination of Motion Breakpoints in a Task Sequence from Human Hand Motion.* In *Proceedings of the IEEE International Conference on Robotics and Automation (ICORA'94)*, Band 1, Seiten 551–556, San Diego, CA, USA, 1994.

[Kang 94c] Sing Bing Kang, Katsushi Ikeuchi. *Grasp Recognition and Manipulative Motion Characterization from Human Hand Motion Sequences.* In *Proceedings of the IEEE International Conference on Robotics and Automation (ICORA'94)*, Band 2, Seiten 1759–1764, San Diego, CA, USA, 1994.

[Kang 95] Sing Bang Kang, Katsushi Ikeuchi. *Toward automatic robot instruction from perception – Temporal segmentation of tasks from human hand motion. IEEE Transactions on Robotics and Automation*, 11. Jahrgang, Ausgabe 5, October 1995.

[Kang 97] Sing Bang Kang, Katsushi Ikeuchi. *Toward automatic robot instruction from perception – Mapping human grasps to manipulator grasps. IEEE Transactions on Robotics and Automation*, 13. Jahrgang, Ausgabe 1, February 1997.

[Kreuziger 94] J. Kreuziger. *An architecture for the application of symbolic learning in robotics.* Dissertation, University of Karlsruhe, Department of Computer Science, 1994. In German.

[Kuniyoshi 94] Y. Kuniyoshi, M. Inaba, H. Inoue. *Learning by watching: Extracting reusable task knowledge from visual observation of human performance. IEEE Transactions on Robotics and Automation*, 10. Jahrgang, Ausgabe 6, Seiten 799 – 822, 1994.

[Laird 90a] J.E. Laird, M. Hucka, E.S. Yager, C.M. Tuck. *Correcting and Extending Domain Knowledge using Outside Guidance.* In *Proceedings of the 7th International Conference on Machine Learning, Austin*, Seiten 235–243, 1990.

[Laird 90b] John E. Laird, Paul S. Rosenbloom. *Integrating Execution, Planning, and Learning in Soar for External Environments.* In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston*, Seiten 1022–1029, 1990.

[Langley 89] Pat Langley, Kevin Thompson, Wayne Iba, John H. Gennari, John A. Allen. *An Integrated Cognitive Architecture for Autonomous Agents.* Technischer Bericht 89-28, University of California, Irvine, Department of Information and Computer Science, Sept. 1989.

[Lebowitz 86] M. Lebowitz. *Concept learning in a rich input domain: Generalization-based memory.* In *Machine Learning: An artificial intelligence approach*, R. S. Michalski, J. C. Carbonell, T. M. Mitchell, Editoren, Band 2. Morgan Kaufmann, Los Altos, Ca, 1986.

[Lieberman 93a] Henry Lieberman. *Watch what I do*, Kapitel MONDRIAN: A teachable graphical editor. MIT Press, Cambridge, MA, USA, 1993.

[Lieberman 93b] Henry Lieberman. *Watch What I Do: Programming by Demonstration*, Kapitel Tinker: A Programming by Demonstration System for Beginning Programmers, Seiten 50–64. MIT Press, Cambridge, MA, USA, 1993.

[MacKenzie 94] Christine L. MacKenzie, Thea Iberall. *Advances in psychology*, Kapitel The grasping hand, Seiten 15 –46. Band 104. Amsterdam, NL, 1994.

[Matsui 89] T. Matsui, M. Tsukamoto. *An integrated robot teleoperation method using multi-media display.* In *International Symposium on Robotics Research (ISRR'89)*, Seiten 145 – 152, 1989.

[Maulsby 93] D. Maulsby, I.H. Witten. *Watch What I Do: Programming by Demonstration*, Kapitel Metamouse: An Instructible Agent for Programming by Demonstration, Seiten 156–182. MIT Press, Cambridge, MA, USA, 1993.

[Mitchell 82] T. M. Mitchell. *Generalisation as search. Artificial Intelligence*, 18. Jahrgang, Seiten 203 – 226, 1982.

[Mitchell 86] T. M. Mitchell, J. G. Carbonell. *Explanation-based Generalisation - A unifying view. Machine Learning*, 1. Jahrgang, Seiten 47 – 80, 1986.

[Myers 90] Brad A. Myers. *Creating User Interfaces Using Programming-by-Example, Visual Programming and Constraints. ACM Transactions on Programming Languages and Systems*, 12. Jahrgang, Ausgabe 2, Seiten 143 – 177, April 1990.

[Myers 92] Brad A. Myers. *Demonstrational Interfaces: A Step Beyond Direct Manipulation. IEEE Computer*, 25. Jahrgang, Ausgabe 8, Seiten 61 – 73, August 1992.

[Myers 93] Brad A. Myers. *Garnet: Uses of Demonstrational Techniques.* A. I. Cypher. In *Cypher (ed.) Watch what I do – Programming by Demonstration.* MIT Press, Cambridge, Massachusetts, 1993.

[Onda 97] Hiromu Onda, Hirohisa Hirukawa, Fumiaki Tomita, Takashi Suehiro, Kunikatsu Takase. *Assembly Motion Teaching System using Position/Force Simulator – Generating Control Program –.* In 10^th *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'97)*, Grenoble, France, September 7-11 1997.

[Pohl 98] Kilian Pohl. *Modellierung von Manipulationssystemen.* Diplomarbeit, Universität Karlsruhe, Fakultät für Informatik, Institut für Prozeßrechentechnik und Robotik, 1998.

[Riepp 97] Markus Riepp. *Wissensbasierte Parametrisierung von Operatorsequenzen.* Diplomarbeit, Universität Karlsruhe, Fakultät für Informatik, Institut für Prozeßrechentechnik und Robotik, 1997.

[Sassin 92] M. Sassin, S. Bocionek. *Programming by Demonstration: A basis for autocustomizable workstation software.* In *Workshop Intelligent Workstations for Professionals*, München, 1992.

[Schaude et al.] Universität Karlsruhe (TH), Fakultät für Informatik, Institut für Prozeßrechentechnik und Robotik, Horst F. Schaude u.a., *Dokumentation zu KAVIS (Karlsruher Visualisierer) Version 2.4.8*

[Shepherd 93] Barry Shepherd. *Applying visual Programming to Robotics.* In *Proceedings of the IEEE International Conference on Robotics and Automation (ICORA'93)*, Band 2, Seiten 707–712, 1993.

[Stasch 96] Martin Stasch. *Programmieren durch Vormachen.* Diplomarbeit, Universität Karlsruhe, Fakultät für Informatik, Institut für Prozeßrechentechnik und Robotik, 1996.

[Takahashi 92] T. Takahashi, H. Ogata. *Robotic Assembly Operation based on Task-Level Teaching in Virtual Reality.* In *Proceedings of the IEEE International Conference on Robotics and Automation (ICORA'92)*, Seiten 1083–1088, Nice, France, May 1992.

[Takahashi 93] T. Takahashi, H. Ogata, S. Muto. *A Method for Analyzing Human Assembly Operations for use in automatically Generating Robot Commands.* In *IEEE International Conference on Robotics and Automation*, Seiten 695 – 700, Atlanta, 1993.

[Takahashi 96] Tomoichi Takahashi. *Time Normalization and Analysis Method in Robot Programming from Human demonstration Data.* In *Proceedings of the IEEE International Conference on Robotics and Automation (ICORA'96)*, Band 1, Seiten 37–42, Minneapolis, Minnesota, USA, April 1996.

[Tso 95] S. K. Tso, K. P. Liu. *Automatic Generation of Robot Program Codes from Perception of Human Demonstration.* In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'95)*, Band 1, Seiten 23–28, Pittsburgh, PA, USA, Aug. 5-9 1995.

[Tung 95] C. P. Tung, A. C. Kak. *Automatic Learning of Assembly Tasks Using a DataGlove System.* In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'95)*, Band 1, Seiten 1–8, Pittsburgh, PA, USA, Aug. 5-9, 1995.

[Vir 95] Virtual Technologies Inc., Palo Alto, CA, USA. *CyberGlove^{TM} User's Manual*, Oktober 1995.