Untersuchung von Planungssystemen

S T U D I E N A R B E I T

von

cand. inform. Friedemann Kienzler

Universität Karlsruhe
Fakultät für Informatik
Institut für Prozeßrechentechnik und Robotik
Prof. Dr.-Ing. U. Rembold
Prof. Dr.-Ing. R. Dillmann

Mai 1988

Referent:	Prof. Dr.-Ing. U. Rembold
Korreferent:	Prof. Dr.-Ing. R. Dillmann
Betreuer:	Dipl.-Inform. B. Stocker

Ich erkläre hiermit eidesstattlich, daß ich die vorliegende Studienarbeit selbständig und ohne unzulässige Hilfe angefertigt habe. Die verwendeten Quellen sind im Literaturverzeichnis vollständig angegeben.

Karlsruhe, den 05. Mai 1988.

- Friedemann Kienzler -

INHALTSVERZEICHNIS
==========

Kapitel 1 : Einleitung

Diese Studienarbeit soll einen Überblick über bereits existierende Planungssysteme geben.

Im folgenden Kapitel werden zunächst die verschiedenen Planerstellungsmethoden der Künstlichen Intelligenz (KI) aufgezählt und kurz beschrieben. In den darauffolgenden Kapiteln folgt dann die Analyse von einigen bereits existierenden Planungssystemen. Dabei wird jeweils so vorgegangen, daß wesentliche Punkte wie Wissensrepräsentation, Architektur, Kontrollstruktur u.dgl. näher untersucht, die verwendeten Planerstellungsmethoden angegeben und abschließend eine Beurteilung etwa hinsichtlich der Verwendbarkeit in der Planung in der Robotik und eventuell Vergleiche mit anderen Planungssystemen angeführt werden.

Folgende Planungssysteme werden betrachtet:

1. MOLGEN,
2. SIPE,
3. TWEAK,
4. TWAIN,
5. PLAKON.

Die Untersuchungen wurden ausschließlich an Hand von Literatur vorgenommen, praktische Erprobung an Rechnern war nicht möglich. Allerdings sind in der Regel Beispiele angeführt, so daß die Vorgehensweise der einzelnen Systeme nachvollzogen werden kann.

Noch ein Wort zur Aktualität des Themas. Die untersuchten Systeme liegen zum Teil erst als unvollständige Prototypen (z.B. Planungssystem-Shell PLAKON), zum Teil bereits seit Anfang der 80er Jahre als erprobte Planer (z.B. MOLGEN) vor. In den vergangenen Jahren wurden die Forschungen auf dem Gebiet der Künstlichen Intelligenz immer intensiver betrieben. Dieser Zweig der Informatik, der sich damit auseinandersetzt, Computern - vorwiegend per Software, teilweise aus Effizienzgründen auch mit besonderer Hardware - zu Leistungen zu verhelfen, die ein Mensch als "intelligent" bezeichnen würde, erweist sich als dasjenige Forschungsgebiet, aus dem neue zukunftsweisende Denkansätze bei der systematischen effizienten Verarbeitung von Information hervorgehen. Zentraler Punkt ist die Darstellung und die Verarbeitung von Wissen. Expertensystem ist einer der in

diesem Zusammenhang auftretenden Begriffe. Es handelt sich dabei um ein Computerprogramm(-system), das auf einem speziellen Wissensgebiet die Kompetenz von menschlichen Experten besitzt und als Beratungs- und Problemlösungssystem eingesetzt wird.

Beim Planen, ein weiteres wichtiges Schlagwort der KI, geht es vor allem darum, menschliches Problemlösungsverhalten nachzuvollziehen bzw. zu simulieren. Diese Studienarbeit auf dem Gebiet der Planerstellungsmethoden in der KI soll dazu dienen, die verschiedenen Kontrollstrukturen und Wissensrepräsentationen in bereits existierenden Planern aufzuzeigen und somit notwendiges Hintergrundwissen für das Erstellen einer sog. Planungssystem-Shell zu liefern, die über verschiedene Methoden zur Planerzeugung verfügt und durch Hinzugabe von speziellem Bereichswissen und Auswahl der anzuwendenden Planungsmethoden zu einem Planer in einem konkreten Bereich wird.

Kapitel 2 : Planerstellungsmethoden in der Künstlichen Intelligenz

Bevor nun im folgenden ein Überblick über die Methoden des Planens, die in der KI entwickelt wurden, gegeben wird, seien zunächst noch einige grundlegende Begriffe erklärt.

Laut DUDEN-Wörterbuch bedeutet "Plan" die Vorstellung von der Art und Weise, in der ein bestimmtes Ziel verfolgt, ein bestimmtes Vorhaben verwirklicht werden soll. In der KI ist die Bedeutung dieses Wortes demgegenüber eingeengt: Plan meint hier eine Repräsentation einer Folge von Handlungen. Pläne werden entworfen, um die Lösung eines Problems zu steuern; den Anstoß zur Entwicklung eines Planes gibt also stets das Vorliegen einer Problem-Situation. Planen ist eine der grundlegenden Techniken in der Künstlichen Intelligenz. Es bezeichnet die Tätigkeit, einen Plan zu erstellen, also zur Lösung eines Problems Aktionen zusammenzustellen, die, wenn man sie ausführt, das Problem lösen.

Dieses Kapitel soll die in der KI entwickelten allgemeinen Methoden zur Erstellung von Plänen darstellen. Viele Arbeiten in der KI beschäftigen sich mit Planen, und es wurden schon zahlreiche Planungssysteme entworfen.

Die Untersuchung des Planens als KI-Technik begann mit den frühen Arbeiten von Newell/Simon (z.B. [10]). Darin findet sich die in der Literatur bis heute vorherrschende Sicht des Planens als Arbeiten in einem Situationskalkül: die Welt zerfällt für einen Planer in zwei Klassen von Objekten, Operatoren (beim eigentlichen Problemlösen tatsächlich auszuführende Handlungen) und Situationen (Modelle der Welt mit den verschiedenen planungsrelevanten Merkmalen in verschiedenen Stadien der Planausführung). Planen ist dann das Suchen einer Folge von Operatoren, die das Modell des aktuellen Weltzustandes in ein solches Weltmodell überführt, das bestimmte erwünschte Eigenschaften aufweist. Das Ziel bei jeglichem Planen liegt darin, den gefundenen Plan anschließend, also nach Erstellen des Planes, auszuführen. Der Hintergrund in der KI ist dabei immer der, daß ein autonomer Roboter den Plan ausführen soll. Beim Planen möchte man die Lösung des Problems auf Anwendung der atomaren Operatoren zurückführen, das sind

solche Operatoren, die als unmittelbar ausführbar anzusehen sind.

Grundsätzlich lassen sich zwei mögliche Ansätze beim Planen innerhalb des Situationskalküls unterscheiden:

(1) Einstufiges Planen:
Man geht von den Operatoren aus und versucht, aus ihnen Folgen zu bilden, bis man eine gefunden hat, die das Problem löst (vgl. Bottom-Up-Vorgehen).

(2) Mehrstufiges Planen:
Man geht vom globalen Problem aus, löst dieses erst im groben und verfeinert die Lösung schrittweise, bis man bei den als elementar betrachteten Operatoren angelangt ist (vgl. Top-Down-Vorgehen).

2.1 Einstufiges Planen

Man unterscheidet zwei Varianten bei der Realisierung dieses Planungsansatzes:

(1) Planen als Inferenz,
(2) Planen als Suche.

2.1.1 Planen als Inferenz

Dieser Ansatz zum einstufigen Planen entspricht einem Ableiten im Kalkül: der Plan stellt einen Beweis dar, daß aus der Problemsituation durch Anwenden der verfügbaren Operatoren eine Situation herbeigeführt werden kann, die bestimmten Zielbedingungen genügt; der Plan ergibt sich durch Zurückverfolgen des Beweises und Aneinanderhängen der in der Ableitung beteiligten Operatoren.

Es ist offensichtlich, daß die Effizienz des Planens identisch ist mit der des Beweisens. Was beim Beweisen interessiert, ist nicht nur, ob das Problem lösbar ist oder nicht, sondern vor allem der Lösungsweg, also die Substitutionen der (Situations-)Variablen.

Was sich beim Planen als Inferenz als nachteilig herausstellt, ist die Tatsache, daß zusätzlich zu der Beschreibung dessen, was die Operatoren des Problembereichs bewirken, jeweils auch noch angegeben werden muß, was eben diese nicht verändern; man muß also den konstanten Rahmen angeben, vor dem die Operatoren

Änderungen bewirken. Auf diese Weise erreichen die Axiomatisierungen der Operatoren, welche die Operatoren durch Beschreibung ihrer Wirkungen repräsentieren, schnell einen nicht mehr handhabbaren Umfang. Das Problem der Rahmenaxiome (siehe [9]) läßt sich mit dem zweiten genannten Ansatz zum einstufigen Planen umgehen.

2.1.2 Planen als Suche

Beim Planen als Suche wird die Beschreibung des Problembereichs in verschiedene Situationen unterteilt. Die Datenbasis der möglichen Zustände wird nicht mehr wie beim Planen als Inferenz als globales einheitliches Ganzes betrachtet, sondern in bestimmte Situationen partitioniert. Diese Zustandsraum-Darstellung resultiert in einem Baum, dessen Knoten Situationen entsprechen und dessen Kanten als Anwendung von Operatoren zu interpretieren sind.

Das Problem der Rahmenaxiome kann nun dadurch gemeistert werden, daß die Beschreibungen der Operatoren interpretiert werden, als enthielten sie zusätzlich zu der Axiomatisierung der Operator-Effekte noch Angaben, daß alle anderen Formeln in der Situation, in der der Operator angewandt wird, wahr blieben (sog. "STRIPS assumption", siehe [6]).

Planen in dieser Sichtweise ist identisch mit der Suche in dem Zustandsraum nach einem Weg von der Start- zu einer Zielsituation. Die Effizienz des Planens ist folglich identisch mit der Effizienz der Suche. Als besonders sinnvoll erweist sich als Suchstrategie die Rückwärtssuche (Suche vom Ziel zum Start). Dies ist naheliegend, da die Zielsituation im allgemeinen nicht vollständig beschrieben ist, sondern vielmehr eine Klasse von Situationen darstellt. Ausgangspunkt der Suche bildet eine mögliche Zielsituation aus einer solchen Klasse. Gesucht ist also der Weg von einer eindeutig bestimmten Startsituation in eine Klasse von Zielsituationen.

Beim Planen als Suche bekommt man zwar das Problem der Rahmenaxiome in den Griff, es ergibt sich allerdings eine andere Schwierigkeit. Planen nach dieser Art ist der sukzessive Aufbau einer Folge von atomaren Operatoren zur Lösung des gegebenen Problems. Bei einem komplexeren Problem wird so vorgegangen, daß die Lösungen der dieses Gesamtproblem bildenden Teilprobleme einzeln bestimmt und anschliessend zusammengesetzt werden. Beim Versuch, das Gesamtproblem in einem Zug zu lösen, wird die Suche durch die große Zahl der zu untersuchenden möglichen Operatorfolgen schnell äußerst

ineffizient. Zusätzliche Probleme gibt es dann, wenn die Teilprobleme bzw. ihre Lösungen voneinander abhängen. Ein Beispiel aus der Blockwelt, dem Standard-Beispiel-Bereich der KI, der aus einer beliebig großen, ebenen, waagrechten Fläche, z.B. einem großen Tisch, einer Menge von gleichgroßen Blöcken und einem Robotergreifarm besteht, illustriert einen derartigen Fall ganz gut:

Gegeben sei folgende Anfangssituation:

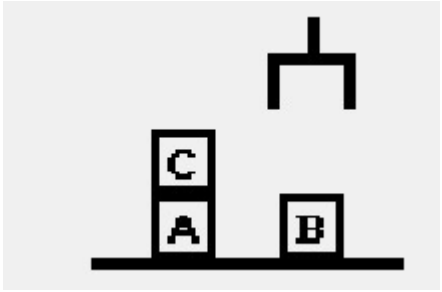


Abb. 2.1.2-1 Situation in der Blockwelt

mit der Zielbeschreibung, daß (1) Block A auf Block B und (2) Block B auf Block C liegt.

Die Lösung von Teilproblem (1) lautet:

- nehme C von A mit dem Greifarm herunter,
- lege C auf den Tisch,
- ergreife A mit dem Greifarm,
- setze A auf B.

Die Lösung von Teilproblem (2) lautet:

- ergreife B mit dem Greifarm,
- setze B auf C.

Hängt man nun beide Pläne einfach aneinander, so erhält man unausführbare Pläne, denn um den einen Teilplan ausführen zu können, muß man genau die Ergebnisse des anderen vorher rückgängig machen. Ein "Verzahnen" der beiden Teilpläne ist also erforderlich.

Die Strategie beim Erstellen eines Planes muß nun so abgestimmt sein, daß Abhängigkeiten von Teilproblemen (sog. subgoal interactions) berücksichtigt werden, möglichst

1. ohne die schon gefundenen Teilprobleme zu zerstören,

2. ohne sämtliche Operator-Permutationen durchsuchen zu müssen.

Bei den bisher erstellten Planungssystemen sind, wie auch die folgenden Kapitel belegen werden, unterschiedliche Lösungsansätze zum Beheben der subgoal interactions verwendet worden. Es können z.B. im Anschluß an den Lösungsversuch des Gesamtproblems gegebenenfalls zerstörte Teilziele wiederhergestellt werden (STRIPS, siehe [6]). Ein anderes Planungssystem arbeitet dahingehend, daß versucht wird, Abhängigkeiten in erzeugten Teilproblemen zu erkennen und durch Umordnen der Reihenfolge der Teilprobleme aufzulösen (HACKER). Diese bisher aufgezeigten Strategien werden als lineares Planen bezeichnet, bei dem zunächst ein Gesamtplan aus linear geordneten Teilplänen erstellt und dann versucht wird, Interaktionen zu beheben. Eleganter ist die Sichtweise, Pläne von vorneherein als partiell geordnete Operatorfolgen anzusehen, wobei sich die Ordnung durch Interaktion der Operatoren bzw. der Teilziele ergibt. Man spricht von nichtlinearem Planen. Das Vorgehen hierbei läßt sich so charakterisieren, daß zunächst (partiell geordnete) Teilpläne erzeugt und anschließend die Ordnung so erweitert wird, daß ein interaktionsfreier Aktionsplan entsteht.

Das Erzeugen einer partiellen Ordnung zur interaktionsfreien Integration von Teilplänen erfordert als Teilaufgaben:

1. Erkennen von Interaktionen,
2. Ordnen von Operatoren,
3. evtl. Identifizieren desselben Operators in verschiedenen Teilplänen, der nicht textuell derselbe sein muß,
4. Einfügen weiterer Planstücke an "möglichst günstigen" Stellen, falls es ansonsten unmöglich ist, eine interaktionsfreie Ordnung herzustellen.

Um am Ende dieses Abschnittes noch eine Wertung anzuführen, sei bemerkt, daß einstufiges Planen als Suche zur Auflösung von Interaktionen in Mini-Welten erfolgreich einsetzbar ist. Größere, d.h. realitätsbezogenere Welten mit komplexeren Problemen bedürfen eines anderen Ansatzes; ein größeres Maß an Abstraktion ist notwendig, um die riesigen Suchräume in den Griff zu bekommen.

2.2 Mehrstufiges Planen

Wie bereits erwähnt, wird im Bereich der KI versucht, menschliches Problemlöseverhalten nachzuvollziehen. In

natürlichen Umgebungen werden Pläne zuerst im groben entworfen und bei Bedarf immer weiter verfeinert, bis man auf elementare Handlungen stößt. Es wird versucht, den umfangreichen Raum der möglichen Zustände zunächst so übersichtlich und klein wie möglich zu halten und dann nur die "vielversprechenderen" Pfade im Suchbaum näher zu untersuchen.

Grundsätzlich lassen sich zwei Möglichkeiten dieser Art der Problemzerlegung unterscheiden:

(1) Situations-Abstraktion,
(2) Operator-Abstraktion.

2.2.1 Situations-Abstraktion

Bei diesem Ansatz des mehrstufigen Planens werden Operatoren bzw. noch allgemeiner Merkmale in der Repräsentation des Problembereiches mit unterschiedlichen Wertigkeiten versehen, die den jeweiligen "Wichtigkeiten" entsprechen sollen. Auf diese aufbauend wird der Suchraum für die möglichen Operatorfolgen untergliedert.

Zunächst wird ein Plan erst unter Berücksichtigung lediglich der wichtigsten Merkmale erstellt. Dieses "Plangerüst" wird auf die nächstuntere Stufe heruntergegeben und weiter modifiziert, d.h. die nächst weniger wichtigen Merkmale werden herangezogen, bis schließlich der Plan auf unterster atomarer Stufe angelangt ist. Scheitert die Planung auf einer Stufe, wird auf der nächsthöheren versucht, eine Alternative zu finden. Man erhält auf diese Weise verschiedene Abstraktionsstufen der Repräsentation des Problemraums. Zu bemerken ist noch, daß das Planen auf den einzelnen Stufen mit den Techniken der einstufigen Planung realisiert werden kann.

Dieser Planungsansatz empfiehlt sich vor allem bei Problemen, zu deren Lösung lange Pläne erforderlich sind. Allerdings ist dabei zu beachten, daß es vor allem auf den höheren Abstraktionsstufen einer sehr sorgfältigen Planung bedarf, um zu gewährleisten, daß der vorliegende abstrakte Planentwurf zu einem Plan auf unterster, ausführbarer Ebene ergänzbar ist.

2.2.2 Operator-Abstraktion

Eine andere Möglichkeit, den komplexen Problemraum durch Abstraktion in den Griff zu bekommen, besteht darin, abstrakte Operatoren einzusetzen, also solche Operatoren, die gar nicht tatsächlich ausführbar vorhanden sind, sondern zu deren Ausführung in Wirklichkeit eine Folge (konkreter) Operatoren nötig ist (sozusagen Makro-Operatoren).

Das Vorgehen dabei sieht so aus, daß zunächst ein Plan mit Operatoren der abstraktesten Ebene erzeugt wird. Dieser Plan wird dann durch Expansion der Operatoren in die nächste Abstraktionsebene konkretisiert. Auf diese Weise wird solange die Plankonkretisierung fortgesetzt, bis die Ebene der Elementaroperatoren erreicht ist.

Der entscheidende Vorteil dieser Methode liegt in der Fähigkeit des Planers, schon auf der Ebene der abstrakten Operatoren mögliche Reihenfolge-Restriktionen erkennen zu können; die wesentliche Arbeit kann also bereits auf der weniger komplexen abstrakten Ebene getan werden. Einen weiteren Vorteil liefert die Möglichkeit, Operatoren auf verschiedenen Abstraktionsstufen als elementar zu betrachten. Der Plan kann somit dem "Erfahrungsgrad" der Akteure angepaßt und auf unterschiedlichen Abstraktionsebenen beschrieben werden.

2.3 Meta-Planen

Den bisher beschriebenen Planungstechniken gemeinsam ist die negative Eigenschaft der Starrheit. Ein derartiges Planungssystem kann Pläne naheliegenderweise nur gemäß seiner eigenen Strukur erstellen, z.B. linear, auch wenn in dem einen oder anderen Fall ein anderes Vorgehen angebrachter wäre. Die Idee, diesem abzuhelfen, ist nun die, um solche Planungsprogramme eine weitere Schicht, eine Shell, zu legen, auf der unabhängig von aller Planung im aktuellen Problembereich "strategische", also die Kontrolle des Planerstellungsprozesses betreffende Entscheidungen getroffen werden können, nach welchem abstrakten Muster in einer gegebenen Situation (weiter-)geplant werden soll.

Man kann bei diesem Ansatz von einem übergeordneten Planungsprozeß sprechen: die verschiedenen Planungsmuster werden durch (Planungs-)Operatoren repräsentiert. Das Auswählen auf Meta-Ebene, welcher Plan-Operator in einer gegebenen Situation der geeignetste ist, entspricht dem Bestimmen eines Operators im Problembereich

auf der "unteren" Ebene. Man plant, wie man einen Plan entwickelt.

Den entscheidenden Vorteil beim Meta-Planen liefert die Möglichkeit, nicht nur Wissen über den eigentlichen Problembereich verfügbar zu machen, sondern auch Wissen über den Planungsprozeß als solchem zu sammeln und zu verwenden. Die Arbeitsstruktur über dem Objektbereich des Planers wird auf seine Kontrollstruktur übertragen.

Ein Planungssystem wird auf diese Weise in doppeltem oder besser gesagt, da der Ansatz des Meta-Planens rekursiv über mehrere Stufen realisiert werden kann, in mehrfachem Sinne genutzt: mit ein und derselben Architektur werden auf der einen Seite Entscheidungen im Problembereich gefällt und auf der anderen Seite plan-spezifische Festlegungen getroffen.

2.4 Planerstellungsmethoden in der Übersicht

Zum Abschluß dieses Kapitels hier nochmals in der Übersicht die beschriebenen Planerstellungsmethoden in der KI (Abb. 2.4-1):

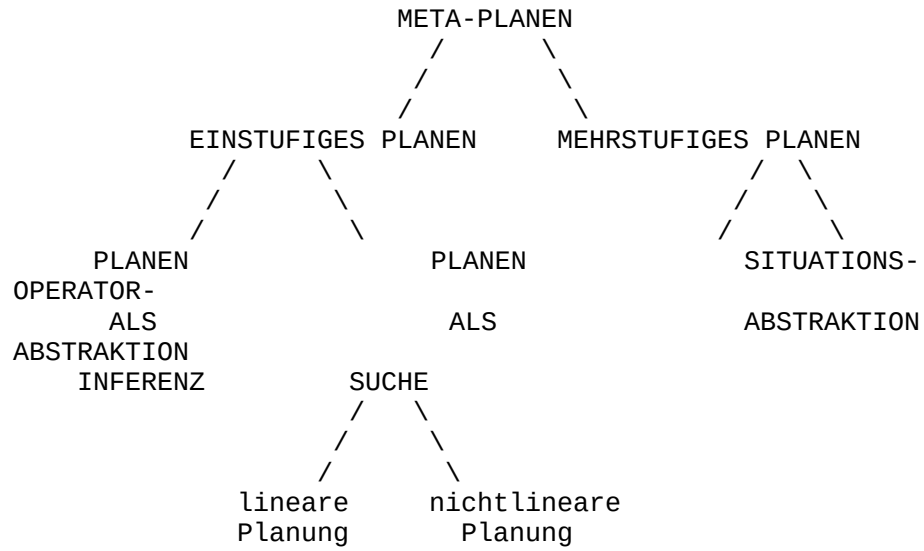


Abb. 2.4-1 Planerstellungsmethoden in der KI

Kapitel 3 : Planungssystem MOLGEN

MOLGEN [14, 15] ist ein bereits erprobtes Planungssystem, das zum Planen molekulargenetischer Experimente zur Gen-Klonung konzipiert worden ist.

MOLGEN ist aus mehreren Gründen einer näheren Betrachtung wert. Zum einen ist es das System, an dem Meta-Planen erstmals ausführlich untersucht wurde. Zum anderen läßt sich an diesem Planungssystem die Vorgehensweise des hierarchischen Planens aufzeigen, bezeichnet als Constraint Posting, das Constraints dazu einsetzt, die Interaktion zwischen Teilproblemen darzustellen. Basis von MOLGEN ist ein hierarchisch-nichtlinearer Planer.

3.1 In MOLGEN verwendete Planungstechniken

Wie bereits im vorigen Kapitel erläutert wurde, ergeben sich beim Erstellen von komplexeren Plänen zusätzliche Probleme durch Abhängigkeiten unter den Teilproblemen (subgoal interactions). Das Gesamtproblem läßt sich in der Regel in mehrere Teilprobleme untergliedern; es gilt eine Operatorfolge zu finden, die alle Teilprobleme löst. Die Frage dabei ist, wie man erkannte Abhängigkeiten von Teilproblemen behandeln kann.

Bei der nichtlinearen Planung werden zunächst Teilpläne erzeugt und die Ordnung anschließend so erweitert, daß ein möglichst interaktionsfreier Gesamtplan entsteht (Vorgehen: Unterbeschränkung bezüglich der Operator-Ordnung). Es bietet sich an, Planvariable erst so spät wie möglich zu instantiieren, also so lange es geht mit variablen Argumenten der Operatoren im bisher erstellten Teilplan zu arbeiten und auch auf diese Weise den Plan zusätzlich unterzubeschränken.

Besonders ausgiebig wird diese Technik benutzt in MOLGEN, wo im Laufe der Planung bei Verfolgung verschiedener Teilziele immer mehr einschränkende Bedingungen (Constraints) für die anfangs unbestimmten Planvariablen gesammelt werden, so daß Interaktionen aus sich widersprechenden Constraints abgelesen und aufgelöst werden können. Die wesentliche Technik, die MOLGEN

beim Erstellen von Plänen einsetzt, basiert auf Inferenzen über Constraints der Planvariablen.

Wie bereits erwähnt, ist MOLGEN ein hierarchischer Planer. Ein derartiges Planungssystem erstellt Beschreibungen von abstrakten Zuständen und untergliedert seine Planungsaufgabe in Teilprobleme, um die abstrakten Zustände zu verfeinern. Die abstrakten Zustände erlauben es, sich auf die wesentlichen Punkte zu konzentrieren, indem es vermieden wird, sich mit allen Einzelheiten sofort auseinandersetzen zu müssen.

MOLGEN ist ein Planer, der ausdrücklich beide Sichtweisen der Abstraktion, also Situations- und Operator-Abstraktion verwendet. Die wesentlichen Vorteile liegen klar auf der Hand. Reihenfolge-Restriktionen bezüglich der anzuwendenden Operatoren können bereits auf abstrakter, noch überschaubarer Ebene erkannt werden. Mit Hilfe der Abstraktionen wird der komplexe Problembereich der möglichen Operatorfolgen überschaubarer, es lassen sich die "erfolgversprechenden" Pfade im Situationssuchbaum erkennen, bevor unzählige Fehlversuche durchprobiert werden müssen.

Schließlich setzt MOLGEN auch die Technik des Meta-Planens ein, bei der die Arbeitsstruktur über dem Objektbereich des Planers auf seine Kontrollstruktur übertragen wird; die primäre Sicht des Planens wird zur Suche im Raum der Planteile. In MOLGEN sieht das Meta-Planen im wesentlichen so aus, daß stets nur ein einziger Meta-Operator ausgesucht und sofort ausgeführt wird; man kann also eigentlich nur von eingeschränktem Meta-Problemlösen mit heuristisch gesteuertem trial-and-error sprechen.

3.2 Hierarchisches Planen mit Constraints

Ein Problem aufzuteilen in möglichst viele Teilprobleme führt nicht unbedingt zu einer Erleichterung beim Bestimmen der Lösung(en). Es kommt darauf an, nach welchen Kriterien aufgeteilt wird. Feststeht, daß Teilprobleme meist interagieren, d.h. voneinander abhängen, sich gegenseitig beeinflussen. Zusätzliche Probleme entstehen dann, wenn die Interaktionen nicht rechtzeitig erkannt werden können. In solchen Fällen müssen aufwendige Änderungen vorgenommen werden. Je komplexer das Problem ist, desto schwieriger wird das Auflösen derartiger Interaktionen.

MOLGEN geht gemäß der sog. Constraint Posting-Methode (Setzen von Constraints) vor. Dabei werden Constraints dazu verwendet, die Interaktionen zwischen Teilproblemen darzustellen. Constraints (= einschränkende Bedingungen) werden dynamisch aufgestellt und an Planvariablen gebunden. Während der hierarchischen Planung werden die Constraints weitergereicht, d.h. ihr Wirkungsbereich erstreckt sich über das jeweilige Teilproblem hinaus. Auf diese Weise werden die Lösungsansätze von nahezu unabhängigen Teilproblemen miteinander verknüpft.

Die zugrundeliegende Idee bei der Constraint Posting-Methode liegt bei der Betrachtung eines Systems als Aggregat lose gekoppelter Subsysteme. Die Kopplung eines Systems [13] gibt die Stärke der Verbindungen zwischen einzelnen Komponenten an. Lose gekoppelte Systeme bestehen aus nahezu unabhängigen Einheiten. Simon [12], der sich mit dem Entwurf komplexer Systeme auseinandersetzte, prägte den Begriff des "nahezu zerlegbaren Systems": ein komplexes System kann aus lose gekoppelten Subsystemen aufgebaut werden. Diese Feststellung läßt sich auf einen Planerstellungsprozeß übertragen. Entsprechend spricht man dann von "nahezu unabhängigen Teilproblemen".

Beim hierarchischen Planen wird eine Lösung zunächst auf abstrakter Ebene zu bestimmen versucht. Darauf folgen Verfeinerungsschritte. Die so entstehenden Teilprobleme sind bestenfalls nur nahezu unabhängig, da die jeweiligen Zwischenergebnisse aufeinander abgestimmt sein müssen. Hierfür wird nun das Hilfsmittel der Constraints eingesetzt, um die Interaktionen zwischen den Teilproblemen zu repräsentieren. Falls auf abstrakter Ebene eine Aufteilung des Problems in nahezu unabhängige Teilprobleme gelingt, so nehmen die Interaktionen keinen übermäßig großen Stellenwert im Planungsprozeß ein, aber effizientes Planen verlangt gewöhnlich dennoch, auch die schwachen Interaktionen in den Planungsprozeß miteinzubeziehen.

3.2.1 Die verschiedenen Bedeutungen von Constraints

Ein Constraint drückt die Abhängigkeiten unter Planvariablen aus. Constraints werden als Prädikate dargestellt und sind direkt mit Planvariablen verknüpft. Sie können jeweils mehr als zwei Variablen miteinbeziehen, sogar auf ganze Mengen von Variablen angewendet werden.

Im Rahmen des Planerstellungsprozesses lassen sich drei verschiedene Interpretationsmöglichkeiten von Constraints unterscheiden:

(1) Constraints als Eliminationsregeln:

Bei MOLGEN ist ein Constraint mit einer Menge von Planvariablen verknüpft, die sich in der Regel auf Labor-Objekte beziehen (Labor-Objekte sind Objekte auf der Ebene des konkreten Problembereichs; weitere Ebenen der Objekte und Operatoren werden in folgenden Abschnitten aufgeführt). Sind die Variablen noch nicht an spezielle Werte gebunden, dann stellt ein Constraint eine Bedingung dar, die es zu erfüllen gilt. Es schränkt die möglichen Werte für die Variable ein, somit wird dadurch eine Selektion der erfolgversprechenden Werte für die Variablen aus allen möglichen ermöglicht.

(2) Constraints als partielle Beschreibungen und Verbindlichkeiten:

Während des Planungsprozesses stellt sich immer wieder das Problem, welcher Teil des Planes es "wert" ist, verfeinert zu werden. Ein wie auch bei MOLGEN gewählter Unterbeschränkungsansatz arbeitet dahingehend, daß Entscheidungen so lange wie möglich hinausgeschoben werden, um die Optionen noch offen zu halten. Ein Constraint stellt eine partielle Beschreibung eines möglichen Objektes dar; die Auswahl eines Objektes liefert, so gesehen, eine vollständige, und somit endgültige Beschreibung eines Objektes. Somit unterstützt der Einsatz von Constraints eine Vorgehensweise gemäß der Unterbeschränkungsstrategie; man versucht, möglichst viele Objekt-Details noch offen, also die Objekt-Beschreibung unterbeschränkt zu lassen, um einerseits mögliche Lösungen nicht frühzeitig zu "verbauen" und andererseits bereits erstellte Teilpläne nicht nachträglich "aufweichen" zu müssen.

(3) Constraints als Kommunikationsmedium:

Constraints dienen als Ausdrucksmittel für Interaktionen unter Teilproblemen, oder - anders ausgedrückt - Constraints repräsentieren Beziehungen zwischen Planvariablen. Diese Beziehungen können oftmals sehr verstrickt und weitreichend sein, abhängig davon, um welches Objekt es sich handelt. Aus bereits existierenden Constraints können neue abgeleitet werden, indem die ersteren innerhalb des Planes von einem Teilast in einen anderen "weitergereicht" werden. Dadurch wird eine Koordination der Lösungen der Teilprobleme ermöglicht.

3.2.2 Constraint-Operationen

Unter der Bezeichnung Constraint Posting-Methode verbirgt sich eine Technik, die sich zusammensetzt aus dem hierarchischen Planen und der Erfüllung von Constraints.

Es lassen sich bei diesem Ansatz drei Operationen unterscheiden, die mit Constraints durchgeführt werden können:

(1) Constraint-Formulierung,
(2) Constraint-Propagierung,
(3) Constraint-Erfüllung.

Bei der Constraint-Formulierung werden im Laufe des Planungsprozesses neue Constraints hinzugefügt, um zusätzlich zu erfüllende Bedingungen und Abhängigkeiten zu formulieren. Im Gegensatz zum reinen Ansatz gemäß der Constraint-Erfüllung, bei der von Anfang an mit einer von vorneherein bekannten festen Anzahl von Constraints gearbeitet wird, ist ein Problemlöser wie MOLGEN, der Constraints dynamisch während des Planungsprozesses erzeugen kann, nicht dazu "verurteilt", sofort mit allen Einzelheiten konfrontiert zu sein. MOLGEN kann sich auf das zunächst Wesentliche konzentrieren und "erstickt" nicht im Wust der Details.

Mit Hilfe der Constraint-Propagierung ist es möglich, neue Constraints an Hand bereits existierender in einem Plan zu formulieren. Der Sichtbarkeitsbereich eines einmal erzeugten Constraints erstreckt sich also nicht nur auf das jeweilige Teilproblem, sondern mehr oder weniger über den ganzen Plan. Das bildet die Voraussetzung für eine zwischen den Verfeinerungs-Teilproblemen stattfindende Kommunikation, wie bereits im vorigen Abschnitt erwähnt wurde. Mit dem Weiterreichen von einmal erzeugten Constraints im Plan wird auch die Vorgehensweise unterstützt, Entscheidungen bzw. Planvariableninstantiierungen während des Planentwurfs solange hinauszuschieben wie möglich.

Bei der Constraint-Erfüllung schließlich werden Werte für Variablen gesucht und eventuell eingesetzt, so daß die Menge mit dieser Variablen verknüpften Constraints erfüllt wird. Constraints dienen bei MOLGEN dazu, spezielle Forderungen an ein Labor-Objekt, also an ein Objekt im aktuellen Problembereich zu beschreiben. Damit verbunden ist bei der Constraint-Erfüllung ein "Kaufe oder Erzeuge"-Entscheidungsprozeß ('buy or build'): zuerst versucht MOLGEN, ein Constraint durch Auswahl eines verfügbaren Objekts zu erfüllen, indem es die Wissensbasis nach einem geeigneten Objekt durchsucht. Falls kein derartiges Objekt vorhanden ist, markiert

MOLGEN das entsprechende Constraint als unerfüllt; ein späterer Planschritt hat dann zum Ziel, ein geeignetes Objekt zu planen.

3.3 Wissensrepräsentation und Architektur von MOLGEN

MOLGENs Wissen wird durch eine hierarchische Wissensbasis repräsentiert, die Informationen über Objekte und Operatoren bereitstellt. Die Wissensbasis beschreibt die ausführbaren Schritte auf verschiedenen Abstraktionsebenen.

Bei Problemstellungen im Bereich des Planens sind in der Regel viele Ziele in einer ganz bestimmten Reihenfolge zu erreichen. Die Ziele beeinflussen sich gegenseitig. Diese Abhängigkeiten ergeben sich einerseits aus der Reihenfolge, in der die Ziele erreicht werden, und andererseits aus den jeweiligen Operatoren, die verwendet werden, um den Zielen näher zu kommen. Wie schon mehrmals angeführt, ist es beim Erstellen eines Planes notwendig, die Optionen möglichst offen zu halten, da Entscheidungen einen Teil eines Planes betreffend häufig Auswirkungen auf andere Teile haben.

Durch eine Vorgehensweise beim Planen, die sich auf Unterbeschränkung und Heuristik stützt, wird es ermöglicht, begrenzte Information in der Wissensbasis durch Schlußfolgern und Probieren auszugleichen. Das dafür zugrundeliegende Wissen weist eine Schichtenstruktur auf, in der "Fakten"-Wissen für Entscheidungen bezüglich des Planungsproblems vom "strategischen" Wissen für Entscheidungen den Planungsprozeß selbst betreffend abgegrenzt ist. Durch eine zusätzliche Planungsebene, die über "eigenes" Wissen verfügt, wird beim Meta-Planen die Leistung von MOLGEN dadurch erhöht, daß auf dieser neuen Schicht unabhängig vom eigentlichen Planprozeß im Problembereich Entscheidungen darüber getroffen werden können, nach welchem abstrakten Muster in einer gegebenen Situation (weiter-)geplant werden soll.

3.3.1 Kontrollstruktur

Problemlösungs-Programme müssen beim Ablauf immer wieder entscheiden, was zu tun ist. Gegeben sind verschiedene Ziele und eine Menge möglicher Aktionen. Beim Problemlösen wird entschieden, welche Aktionen wann angewendet und wie kombiniert werden können. Diese Entscheidungen bezüglich der Aktionen werden unter dem Begriff Kontrolle zusammengefaßt. Den Rahmen, diese Entscheidungen zu treffen, bildet die Kontrollstruktur.

Eine ausgefeilte Kontrollstruktur sollte ein notwendiges Maß an Flexibilität in den Prozeß des Entscheidungfällens bringen. Es sollte ein Planungsprogramm in die Lage versetzen zu erkennen, ob ein Lösungsansatz zu einem gewünschten Ergebnis führt oder in einer "Sackgasse" endet. Bei realen Planungsproblemen gibt es zu viele Möglichkeiten, als daß sie alle ausprobiert werden könnten. So muß ein Planer die nötige Kontrolle darüber ausüben, welche Möglichkeiten näher untersucht werden sollen. Die Auswahl darüber sollte auf abstrakter, noch überschaubarer Ebene stattfinden. Beim Meta-Planen bestimmt das Planungssystem, so auch MOLGEN, welche Strategie es beim jeweiligen Problem heranziehen soll.

3.3.2 Kontrollhierarchie

Der Planungsansatz mit einer zusätzlichen Meta-Ebene liefert ein "Gerüst", um Kontrollwissen in Schichten aufzuteilen. Bei MOLGEN wird das Wissen, auf das zum Lösen von Problemen zugegriffen werden kann, als Kontrollhierarchie organsisiert.

In MOLGEN findet sich die Idee des Aufbaus eines Problemlösers gemäß einer Agenda (= Aneinanderreihung konkurrierender Prozesse, Arbeitsvorschrift für die noch zu bearbeitenden Aufgaben). Die Agenda-Kontrollstruktur ist eine Verallgemeinerung des "Hole und führe aus"-Iterationsprozesses (engl.: fetch-execute-cycle), bei dem die Ausführung von Anweisungen durch einen Prozessor Veränderungen im Speicher verursacht und (nicht unbedingt) das System der Problemlösung ein Stück näher bringt. Bei Agenda-Systemen entsprechen den einfachen Instruktionen unter Umständen umfassende Tasks, die Hol- und Auswahl-Kriterien sind umfassender; der prinzipielle Aufbau ist allerdings derselbe wie beim fetch-execute-cycle. Bei der Agenda-Organisation verfügt der sog. Interpretierer (siehe Abschnitt 3.4.3) das Wissen zur Taskauswahl. Alle Auswahlmöglichkeiten und Auswertungskriterien, die sich auf den Problemlösungsprozeß beziehen, können als Überlegungen auf Meta-Ebene

zusammengefaßt werden. Auch auf diesem Level ließe sich die Technik der Constraints einsetzen, was allerdings bei MOLGEN nicht der Fall ist.

Die Schichtenidee ist nicht auf zwei Ebenen begrenzt, sie kann rekursiv angewendet werden. Um die augenscheinliche Komplexität eines Systems zu reduzieren, können solange neue Schichten eingeführt werden, bis das notwendige Wissen in der obersten Schicht "leicht handhabbar" wird. Bei MOLGEN gibt es drei Schichten.

3.4 Das Schichtenmodell

Das Schichtenmodell, das sich auch in anderen Bereichen der Informatik wie z.B. beim Entwurf komplexer Software-Systeme bewährt hat, eignet sich in besonderem Maße dafür, das Kontrollwissen innerhalb einer Problemlösers zu strukturieren.

MOLGEN arbeitet auf drei Ebenen und mit einem übergeordneten Interpreter. Die Ebenen lauten:

(1) Laborebene (unterste, konkreteste Ebene),
(2) Plan- oder Entwurfebene,
(3) Meta-Plan- oder Strategieebene.

Plan- und Laborebene verfügen über Operatoren und Objekte, auf Meta-Planebene gibt es nur Operatoren.

Bevor nun die einzelnen Ebenen und der Interpreter näher beschrieben werden, soll die folgende Abbildung die Planungsebenen von MOLGEN veranschaulichen:

siehe Abb. 3.4-1 auf der nächsten Seite !

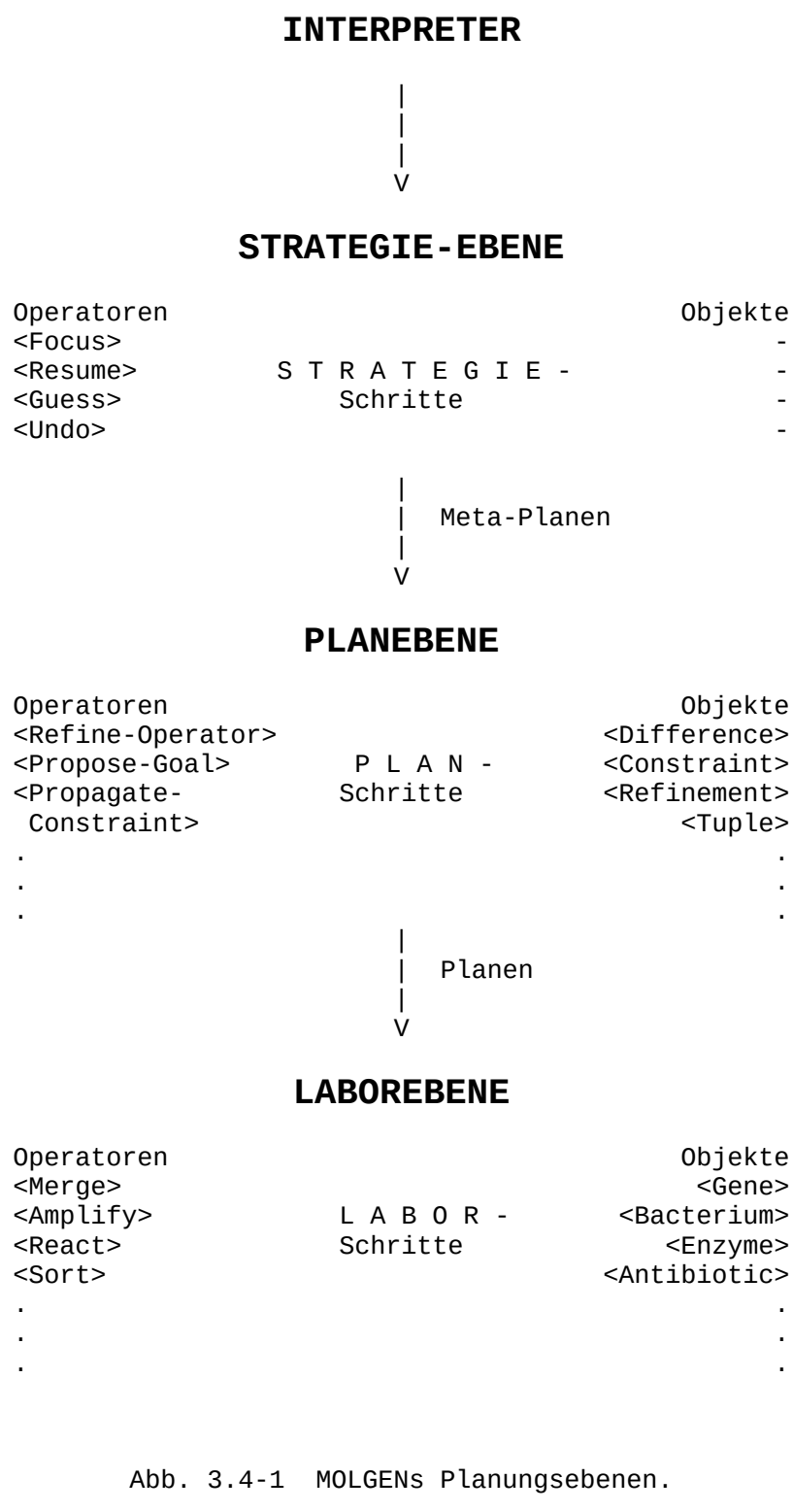


Abb. 3.4-1 MOLGENs Planungsebenen.

Jede Ebene hat die Kontrolle über Festlegung und Ausführung eines Planschrittes (Strategie-, Entwurf- oder Laborschritt) auf der nächst niederen Ebene: die Entwurfebene plant durch Selektieren und Ausführen von Laborschritten, die Strategieebene meta-plant durch Selektieren und Ausführen von Entwurfschritten.

Im folgenden wird nun näher auf die einzelnen Ebenen eingegangen, die MOLGENs Kontrollhierarchie aufweist. Zunächst sei kurz umschrieben, was die einzelnen Ebenen für eine Funktion haben:

1) Laborebene: Diese Ebene repräsentiert Wissen über Laborobjekte und -operatoren, die auf letztere angewendet werden können, um gewünschte Laborziele zu erreichen.

2) Entwurfebene (Planebene): Diese Ebene stellt eine Menge von Operatoren zur Verfügung, die hierarchisches Planen ermöglichen. Die Idee, die dahintersteckt, ist die, daß hierarchisches Planen durch Operationen auf Constraints realisiert werden kann.

3) Strategieebene (Meta-Planebene): Die dieser Ebene zugrundeliegende Idee liegt in der Trennung von unterbeschränkenden und heuristischen Vorgehensweisen beim Suchen nach Lösungen. Die Strategie ist dahingehend ausgerichtet, daß, solange wie möglich, nach dem Unterbeschränkungsansatz vorgegangen und nur im Bedarfsfall auf heuristische Techniken zurückgegriffen werden soll.

3.4.1 Die Laborebene

Die Laborebene spiegelt die reale Welt der Gen-Klonungs-Experimente wider. Mögliche Laborexperimente werden durch Laborobjekte und -operatoren beschrieben. Diese Ebene verfügt über Wissen von konkreten genetischen Objekten und Operatoren (und deren Abstraktionen). Objekte sind Gene, Bakterien, Enzyme ... , also alles, was in ein genetisches Experiment miteinbezogen werden kann; Operatoren auf dieser Ebene (z.B. Verschmelze, Töte_ab) repräsentieren atomare, also vom Laborpersonal direkt ausführbare Handlungen.

Schritte (Tasks) auf dieser Ebene werden ausgeführt, um reale genetische Experimente zu simulieren. Diese unterste Ebene bildet also keine Kontrollebene, vielmehr dient sie als "genetischer Wissensspeicher". Auf diesem

Level wird nur beschrieben, was im Labor durchgeführt werden kann, es wird jedoch nichts darüber ausgesagt, zu welchem Zeitpunkt dies geschehen soll.

Schritte auf der Laborebene beschreiben die Anwendung von möglicherweise abstrakten "genetischen" Operatoren auf genetische Objekte. MOLGEN ist nicht in der Lage, diese Schritte direkt im Labor durchzuführen; vielmehr beschreibt es eine Simulation der auf dieser Ebene auszuführenden Schritte.

Die Laborebene verfügt über kein Wissen, wie ein Experiment möglichst effektiv geplant werden kann. Wissen dafür ist auf den höheren Ebenen angesiedelt.

3.4.1.1 Laborobjekte

MOLGENs Laborobjekte sind in einer hierarchischen Struktur in der Wissensbasis auf Laborebene gespeichert. Das abstrakteste Laborobjekt ist das sog. Lab-Objekt. Auf der nächsten Abstraktionsstufe finden sich Objekte wie z.B. Antibiotic, Culture, Enzyme, Organism, Sample. Diese Hierarchie besitzt sechs Ebenen, auf der untersten Ebene stehen insgesamt 74 verschiedene Laborobjekte zur Verfügung. Bis ein Objekt vollständig spezifiziert ist, bedarf es also diverser Verfeinerungsschritte.

3.4.1.2 Laboroperatoren

MOLGEN betrachtet molekulargenetische Experimente als Kompositionen von vier abstrakten Operatoren, den sogenannten MARS-Operatoren:

(1) <Merge>: Zusammenfügen einzelner verschiedener Teile zu einem Ganzen.

(2) <Amplify>: Vergrößern des Betrages bzw. Umfangs von einem gegebenen Objekt.

(3) <React>: Verändern der Eigenschaften eines gegebenen Objektes.

(4) <Sort>: Aufteilen eines Ganzen in verschiedene Teile gemäß bestimmter Eigenschaften.

Der abstrakteste Operator auf Laborebene ist der sog. Lab-Operator. Die Operatoren, die physikalische Prozesse beschreiben, welche im Labor durchgeführt werden können, sind in vier Klassen aufgeteilt, für die die vier abstrakten MARS-Operatoren als Repräsentanten stehen. Insgesamt stehen 13 spezifische Operatoren auf Laborebene zur Verfügung, die Spezialisierungen der MARS-Operatoren sind. Beispiele für Laboroperatoren sind:

<Ligate>: Zusammenfügen von DNA-Strukturen (Spezialisierung von MARS-Operator <Merge>),

<Transform>: Hinzufügen eines Bakterienüberträgers an bzw. in einen Organismus (Spezialisierung von MARS-Operator <Merge>),

<Incubate>: Bebrüten einer Bakterienkultur unter idealen Bedingungen (Spezialisierung von MARS-Operator <Amplify>),

<Cleave>: Spalten von DNA mit Hilfe eines Enzyms (Spezialisierung von MARS-Operator <React>),

<Screen>: Abtöten von Organismen, die gegen ein gegebenes Antibiotikum nicht resistent sind (Spezialisierung von MARS-Operator <Sort>).

Insgesamt gibt es noch weitere acht Laboroperatoren.

3.4.2 Die Entwurf- oder Planebene

Auf dieser Ebene ist für den Entwurf eines Planes relevantes Wissen verfügbar. Eine Menge von Operatoren (z.B. Verfeinere_Operator, Propagiere_Constraint) wird definiert, um Pläne in abstrakter Form zu skizzieren und Constraints bei einem Verfeinerungsschritt auf Laborebene weiterzureichen. Mit diesen Operatoren wird ein Experiment entworfen. Schritte auf dieser Ebene werden ausgeführt, um einen Laborplan zu entwickeln und zu verfeinern.

Die Entwurfebene ist MOLGENs erste Kontrollebene. Sie enthält Plan-Operatoren, mit denen ein Festlegen und Arrangieren von Laborschritten ermöglicht wird. Der Grundgedanke, daß Planen als Ausführen von Operationen mit Constraints angesehen werden kann, spiegelt sich in

der Struktur der Entwurfebene wider. Drei Constraint-Operationen sind zu unterscheiden, wie schon in Abschnitt 3.2.2 aufgezeigt wurde. Unter Constraint-Formulierung versteht man das dynamische Erstellen von Constraints, wodurch die Menge der möglichen Lösungen eingeschränkt wird. Wenn MOLGEN Constraints an abstrakte Objekte (Variablen) bindet, erstellt es eine partielle Beschreibung des betreffenden Objekts und verschiebt die vollständige Instantiierung auf später. Constraint-Propagierung ermöglicht die Kommunikation zwischen den Teilproblemen, indem Informationen zwischen nahezu unabhängigen Teilproblemen weitergereicht werden. Bei der Constraint-Erfüllung wird ein Verfeinerungsschritt auf das jeweilige Objekt durchgeführt (Schritt von Abstraktion zu Spezifikation). Es werden bei der Constraint-Erfüllung alle Constraints der nahezu unabhängigen Teilprobleme herangezogen und berücksichtigt, um korrekte Lösungen auszuarbeiten.

Diese Operationen, die auf Constraints angewendet werden, bilden eine wichtige Teilmenge der Planoperatoren, die MOLGEN auf dieser Ebene zur Verfügung stehen, und ermöglichen hierarchisches Planen.

3.4.2.1 Entwurf- oder Planobjekte

Auf der Planebene gibt es nur vier mögliche Objekte: Constraint (Bedingung), Difference (Unterschied), Refinement (Verfeinerung) und Tuple (Tupel). Im Gegensatz zu den Objekten auf Laborebene, welche physikalische Objekte repräsentieren, stellen die Objekte auf Entwurfebene konzeptuelle Objekte dar.

Die Bedeutung der einzelnen Objekte wird bei der Beschreibung der in MOLGEN zur Verfügung stehenden Planoperatoren im folgenden Kapitel erläutert.

3.4.2.2 Entwurf- oder Planoperatoren

MOLGEN unterscheidet drei Kategorien von Planoperatoren:

(1) Vergleichsoperatoren: vergleichen Ziele und bestimmen Unterschiede.

MOLGEN vergleicht Ziele, findet Unterschiede und wählt Operatoren aus, um diese Unterschiede zu reduzieren. Vergleichen ist eine grundlegende Planungsoperation. Die Vergleichsergebnisse werden in Form von Objekten (als

Differences) dargestellt. MOLGEN verfügt über zwei Vergleichsoperatoren:

(a) <Find-Unusual-Features>
(Entdecke_ungewöhnliche_Eigenschaften):

Manchmal erscheint es als sinnvoll, bei der Auswahl von abstrakten Operatoren nach Eigenschaften zu suchen, in denen sich Objekte unterscheiden, von anderen abheben, und dann Operatoren zu bestimmen, die genau an diesen Punkten ansetzen. <Find-Unusual-Features> geht genau so vor, indem es Objekte mit ihren Prototypen in der Wissenbasis vergleicht.

(b) <Check-Prediction>
(Untersuche_die_Vorhersage):

Dieser Entwurfoperator vergleicht die durch Simulation eines Laborschritts erhaltenen Voraussagen mit dem für einen Schritt angestrebten Ziel. Dies erweist sich als nützlich, um in solchen Fällen, in denen die vorhergesagten Ergebnisse eines Laborschritts nicht mit den erwünschten Zielen übereinstimmen, den Plan entsprechend frühzeitig zu berichtigen.

MOLGEN entdeckt einen solchen Unterschied nach Simulation eines Laborschritts, wenn das Wissen, das dem Simulationsmodell zugrundeliegt, vollständiger ist als jenes auf Laborebene, das zur Auswahl des Laboroperators herangezogen werden konnte.

(2) Erweiterungsoperatoren: dienen dazu, einen Plan zu erweitern.

MOLGEN verfügt über drei derartige Operatoren:

(a) <Propose-Operator>
(Schlage_einen_Operator_zum Einbau_in_Plan_vor):

Wenn neue Unterschiede im Plan auftauchen, wird dieser Planoperator angewendet, um abstrakte Laboroperatoren vorzuschlagen, die diese Unterschiede reduzieren sollen. <Propose-Operator> muß feststellen, welcher der MARS-Operatoren anwendbar ist. Die hierarchische Gliederung der Laboroperatoren ausnützend, schickt <Propose-Operator> an jeden abstrakten Laboroperator eine Nachricht, besser gesagt eine Frage ("Apply?"), ob er anwendbar ist. Jeder Laboroperator verfügt über eine eigens dafür vorgesehene "Antwortprozedur", die an Hand einer Liste von Unterschieden und Constraints die Entscheidung fällen kann. Falls mehr als ein Laboroperator in Frage kommt, werden diese Kandidaten in einem Verfeinerungsobjekt zusammengefaßt; <Propose-Operator>

wartet daraufhin auf eine entsprechende Mitteilung von der Strategieebene, wie fortzufahren ist.

(b) <Propose-Goal>
(Schlage_ein_zu_erreichendes_Ziel_vor):

Um neue Ziele zu definieren, schickt <Propose-Goal> eine Nachricht an bestimmte Laboroperatoren, die lokale Prozeduren auf Laborebene dazu veranlaßt, neue Ziele bezüglich der entsprechenden Objekte zu formulieren. Beispiel: <Propose-Goal> schickt an <Merge> die Message "MAKEGOALS"; daraufhin definiert eine lokale Prozedur die Ziele für jedes Teilobjekt, das im Rahmen der Merge-Operation miteinbezogen werden soll.

(c) <Predict-Results>
(Sage_eintretende_Ergebnisse_voraus):

Dieser Entwurfoperator wird zur Simulation der Ergebnisse eines vorgeschlagenen Laboroperators eingesetzt. Beim Aufruf von <Predict-Results> wird ein Simulationsmodell aktiviert, das jeden Laboroperator berücksichtigt. Falls die Information auf Laborebene zu unvollständig zur Simulation in dem jeweiligen Planungsstadium ist, suspendiert <Predict-Results> die weitere Ausführung, bis eine entsprechende Mitteilung von höherer Ebene, also von der Strategieebene kommt.

(3) Spezialisierungsoperatoren: nehmen eine (partielle) Spezifizierung des abstrakten Planes vor.

Beim hierarchischen Planen wird zunächst ein Plan auf abstrakter Ebene erstellt und dann schrittweise durch Hinzufügen von Details verfeinert und konkretisiert. MOLGEN unterscheidet drei Spezialisierungsoperatoren, denen allen gemeinsam ist, daß sie Details an mehr oder weniger spezifizierte Pläne hinzufügen:

(a) <Refine-Operator>
(Verfeinere_Operator):

Dieser Planoperator ersetzt abstrakte Laboroperatoren (= MARS-Operatoren) durch spezifische in einem Laborschritt, z.B. <Merge> wird durch <Transform> ersetzt. In der Regel sind die Eingangsgrössen für einen Laborschritt unvollständig spezifiziert, wenn ein Operator ausgewählt wird. Da aber Laboroperatoren oft spezifische Anforderungen stellen, schließt der Verfeinerungsprozeß die Einführung spezifischer Constraints an die Eingangsgrössen mit ein. Damit wird den Anforderungen der Laboroperatoren genüge getan, ohne gleichzeitig die Eingangsobjekte entsprechend festgelegt haben zu müssen.

Wie die anderen Operatoren auf der Entwurfebene setzt <Refine-Operator> Nachrichten über die Schnittstelle Planebene/Laborebene ein, um Informationen über spezifische Laboroperatoren zu erhalten oder Anweisungen an selbige zu erteilen.

(b) <Propagate-Constraint>
(Reiche_Constraint_weiter):

<Propagate-Constraint> wird immer dann aktiviert, wenn neue Constraints im Plan auftauchen. Es werden dann neue Constraints aus bereits existierenden bestimmt. Prinzipiell können Constraints in beide "Richtungen" im Plan, vorwärts (von den Eingangsgrössen zu den Zielen des jeweiligen Laborschritts), also zu den Objekten, die noch nicht näher spezifiziert sind, oder rückwärts (von den Zielen zu den Eingangsgrössen des jeweiligen Laborschritts), also zu den Objekten, die zuvor schon näher spezifiziert wurden, weitergereicht werden; bei MOLGEN ist allerdings nur Rückwärtspropagierung implementiert.

(c) <Refine-Object>
(Verfeinere_Objekt):

Mit diesem Planoperator wird die Vorgehensweise der Constraint-Erfüllung bewerkstelligt. Sobald neue Constraints im Plan auftauchen, wertet <Refine-Object> diese aus, wobei Information aus der Wissensbasis herangezogen wird.

MOLGEN versucht zunächst, ein geeignetes Objekt zu finden, das den Constraints genügt. Falls dieses Vorgehen ohne Erfolg bleibt, wird das betreffende Constraint als noch nicht erfüllt markiert und das Fortschreiten der Verfeinerung suspendiert. Falls dieses Constraint im Fortschreiten des Planens nicht durch ein anderes Constraint ersetzt wird, wird ein neues Teilziel definiert, ein passendes Objekt "aufzubauen". Man spricht von einer "Kaufe oder Erzeuge"-Entscheidung ('buy or build').

Die erzielten Lösungen werden zusammen mit den aus anderen Constraints sich ergebenden Lösungen in Entwurfobjekten, sog. Tuples (Tupel) zusammengetragen. Manchmal kommt es vor, daß sich neue Constraints auf Objekte beziehen, die in unterschiedlichen Tupeln verzeichnet sind. In solchen Fällen wird ein neues Tupel eingerichtet, das als Einträge den Durchschnitt der Lösungen aus den verschiedenen Tupeln enthält.

Erst wenn genügend Constraints gefunden worden sind und damit die Lösung für eine abstrakte Variable eindeutig ist, wird die Variable instantiiert und an die Problemlösung gebunden.

3.4.2.3 Schnittstelle Planebene/Laborebene

Den Entwurfoperatoren stehen insgesamt sieben Messages zur Verfügung ("APPLY?", "REFINE", "MAKEGOALS?", "MAKEGOALS", "SIMULATE?", "SIMULATE", "BKWD-PROPAGATE"), mit denen sie ihre Kontrollfunktion ausüben und Information aus der Laborebene erhalten können. Jeder Laboroperator verfügt über eine eigene Prozedur, die auf jede dieser Meldungen eine entsprechende Antwort zurückmeldet.

3.4.3 Die Meta-Plan- oder Strategieebene und der Interpreter

Die Differenzierung zwischen Unterbeschränkungs- und heuristischem Ansatz zum Lösen von Problemen spiegelt sich in der Wissensstruktur von MOLGEN auf der Strategieebene wider. Schritte auf dieser Ebene werden ausgeführt, um die Schritte auf der Planebene zu bestimmen und ausführen zu lassen.

Beim Unterbeschränkungsansatz werden Entscheidungen, solange es geht, d.h. solange eine noch nicht vollständige Einschränkung der möglichen Lösungen besteht, aufgeschoben. Constraints aus unterschiedlichen Teilproblemen können miteinander kombiniert werden, bevor endgültige Entscheidungen gefällt werden. Auf diese Weise wird verhindert, daß bereits erstellte Teilpläne beim Entdecken von störenden Interaktionen mindestens zu einem Teil wieder rückgängig gemacht, d.h. spezifizierte Größen wieder abstrahiert werden müssen, also bereits aufgebrachte Planarbeit umsonst war.

Beim heuristischen Ansatz werden nicht alle möglichen Operator-Konstellationen, derer es in der Regel sehr viele gibt, betrachtet, sondern es wird eine erfolgsversprechende Auswahl näher untersucht. Beim weiteren Fortschreiten werden vorläufige Entscheidungen in Situationen gefällt, in denen nicht ausreichend Information vorliegt, um eine eindeutige und endgültige Aussage fällen zu können. Ganz allgemein wird beim Lösen von Problemen meist in solchen Fällen heuristisch vorgegangen, in denen für die Lösung keine eindeutige Lösungsstrategie bekannt ist.

Die Operatoren auf Meta-Planebene stehen unter der Kontrolle des Interpreters, der obersten Kontrollinstanz, der in der Organisationsform eines Sekretärs [5] die Regie über zwei Prozesse führt: Unterbeschränkungs- und Heuristikprozeß. Beim Starten eines Planerstellungsprozesses wird zunächst nach der Unterbeschränkungsstrategie vorgegangen. Falls nichts mehr geht, wenn also der Planerstellungsprozeß suspendiert, z.B. wegen ungünstig gewählter oder zuviel vorhandener Constraints, wird vorübergehend nach der heuristischen Methode weitergemacht.

Wie bereits erwähnt, gibt es bei MOLGEN keine speziellen Objekte auf Meta-Planebene.

3.4.3.1 Meta-Plan- oder Strategieoperatoren

MOLGEN verfügt über vier Meta-Planoperatoren:

(1) <Focus> (Richte_Augenmerk_auf): Versuche einen Entwurfoperator auf den bisher entwickelten Plan anzuwenden.

Dieser Strategieoperator wird angewendet, um neue Plan-Tasks ins Leben zu rufen und ausführen zu lassen. Dazu schickt <Focus> an jeden Planoperator die Nachricht "FIND-TASKS", wodurch mit den Planoperatoren verknüpfte Prozeduren nach einem Ansatzpunkt für den jeweiligen Planoperator im aktuellen Plan suchen und zur Meta-Planebene zurückmelden, an welcher Stelle bzw. an welchen Stellen dies der Fall ist.

Die verschiedenen möglichen Planschritte, die sich daraus ergeben, werden in eine Agenda eingetragen. Da in der Regel mehrere Planschritte (an verschiedenen Stellen im Plan) zum gleichen Zeitpunkt ausgeführt werden könnten, muß <Focus> auswählen, welcher Schritt als erster durchgeführt werden soll. Zu diesem Zweck sind bei MOLGEN den Planoperatoren Prioritäten zugeordnet.

Die Prioritäten der Planoperatoren wurden bei MOLGEN folgendermaßen festgelegt: die höchste Priorität haben die Vergleichsoperatoren, dann kommen die Erweiterungsoperatoren, und schließlich haben die Spezialisierungsoperatoren die niedrigste Priorität. Genau sieht dies wie folgt aus:

Operatorklasse	Planoperator	Priorität
VERGLEICHsop.:	<Check-Prediction>	9
	<Find-Unusual-Features>	9
ERWEITERUNGsop.:	<Propose-Goal>	7
	<Propose-Operator>	6
	<Predict-Results>	5
SPEZIALISIERUNGsop.:	<Refine-Operator>	4
	<Propagate-Constraint>	3
	<Refine-Object>	2

Tab. 3.4.3.1-1 Prioritäten der Planoperatoren.

Die Wahl der Prioritäten wurde ganz gezielt so getroffen, um MOLGEN bei einem Planerstellungsprozeß zunächst dazu zu veranlassen, nach Unterschieden zu suchen, darauf dann einen abstrakten Plan zu skizzieren und schließlich die abstrakten Objekte und Operatoren zu verfeinern.

<Focus> führt einen Planschritt aus, indem es die Nachricht "SIMULATE" abschickt. Dieses Spielchen wird solange wiederholt (also: (1) "SIMULATE"-Message abschicken, (2) Task-Ausführung, (3) erneut "FIND-TASKS"-Message abschicken, (4) eventuell weitere Einträge in die Agenda, (5) nächsten Eintrag aus der Agenda holen, (6) wiederholen (1)-(5) bis ...), bis die Agenda abgearbeitet ist oder ein Planschritt überbeschränkt wird. In letzterem Fall hält <Focus> die Ausführung an und gibt an den Interpreter die Überbeschränkt-Meldung weiter, wodurch der <Undo>-Strategieoperator (s.u.) aktiviert wird.

(2) <Resume> (Nimm_Arbeit_wieder_auf): Nimm unterbrochene Arbeit an Planoperator wieder auf.

<Resume> ist ein Strategieoperator, der unterbrochene Arbeit an Planoperatoren wiederaufnimmt, d.h. suspendierte Planschritte oder -Tasks wieder startet. Wie <Focus>, so legt auch <Resume> eine Agenda an und wählt an Hand der Prioritäten Aufgaben aus, falls mehr als einer wiederausführbar ist.

Im Gegensatz zu <Focus> sucht <Resume> nicht nach neuer, sondern nach zu früheren Zeitpunkten bereits aufgenommener, aber zwischenzeitliche unterbrochener Arbeit.

Auf die Message "RESUME?" antwortet jeder suspendierte Planschritt, ob er wieder ausführbereit ist. Durch Senden der Nachricht "RESUME" (ausgehend von <Resume>) wird ein unterbrochener Planschritt wieder aufgenommen.

Noch ein paar Beispiele für Suspendierungsfälle:

- <Refine-Object> wird aktiviert; es wird aber kein Objekt in der Wissenbasis gefunden, das die Constraints erfüllt.

- <Predict-Results> soll eine Vorhersage liefern, doch es liegt nicht ausreichend Information auf Laborebene vor, um die Simulation zu diesem Zeitpunkt korrekt durchführen zu können.

- Ein neues Constraint wird formuliert, <Propagate-Constraint> wird aktiviert; das Constraint soll an ein erst teilweise spezifiziertes Objekt auf Laborebene weitergereicht werden; auf dieses Objekt muß aber erst noch ein Operator angewendet werden, um für das weiterzureichende Constraint "greifbar" zu werden.

(3) <Guess> (Vermute): Ersetze eine Planvariable durch ein konstantes Objekt.

Gelegentlich kommt es während der Planerstellung vor, daß zum Fällen einer endgültigen Entscheidung nicht ausreichend Information zur Verfügung steht. MOLGEN erkennt derartige Situationen daran, daß <Focus> und <Resume> nichts zu tun haben. In solchen Fällen schickt <Guess> eine "GUESS?"-Message an jeden Operator eines suspendierten Planschritts. Daraufhin geben die suspendierten Tasks einen Schätzwert an, dessen Wert den zu erwartenden Nutzen die Festlegung einer möglichen Option durch einen konkreten Wert ausdrücken soll. <Guess> aktiviert den Task mit der größten Angabe durch Abschicken einer "GUESS"-Nachricht. Danach wird, ausgehend vom Interpreter, wieder <Focus> aktiviert.

(4) <Undo> (Mache_Rückgängig): Nimm die Ersetzung einer Planvariablen durch ein konstantes Objekt zurück.

Dieser Strategieoperator ermöglicht Backtracking, wenn ein Plan überbeschränkt geworden ist.

Dabei geht <Undo> so vor, daß es zunächst eine Liste der per <Guess>-Operator ermittelten Planschritte erstellt und diese Liste dann nach einem Schritt durchsucht, dessen Ausgabe die Eingabe für den überbeschränkten Planschritt bildet. Falls ein solcher Schritt gefunden

wird, veranlaßt eine "UNDO"-Message diesen Planschritt, die durch seine Ausführung entstandenen Auswirkungen auf den Planerstellungsprozeß zurückzunehmen. Der Schritt wird daraufhin als unausgeführt gekennzeichnet. Falls aber die Suche erfolglos bleibt, gibt <Undo> eine "entschuldigende" Mitteilung aus und beendet seine Ausführung.

Zu bemerken ist noch, daß <Undo> nicht untersucht, ob das Rückgängigmachen eines Schritts die überbeschränkte Plansituation überhaupt verbessert.

3.4.3.2 Schnittstelle Strategieebene/Planebene

Die Schnittstelle zur Planebene ermöglicht eine Kommunikation zwischen Meta-Plan- und Planoperatoren. Meta-Planoperatoren können sich auf Prozeduren der Planebene beziehen, ohne deren Namen zu kennen. Die verwendeten Nachrichten lauten: "FIND-TASKS", "SIMULATE", "RESUME?", "RESUME", "GUESS?", "GUESS", "UNDO".

3.4.3.3 Die Bedeutung der Strategieebene

Dadurch, daß MOLGEN den Unterbeschränkungsansatz mit heuristischen Elementen vereint, wird die Notwendigkeit unterstrichen, daß ein Planer in bestimmten Situationen während des Planerstellungsprozesses Entscheidungen treffen muß, die sich nicht auf Wissen stützen, sondern vielmehr einem Raten oder Abwägen gleichzusetzen sind.

MOLGEN kann diese heuristische Vorgehensweise einsetzen, muß es aber nicht. Sie bildet in gewissem Sinne eine Rettung in kritischen Situationen, in denen das Fortschreiten des Planens gefährdet ist. Beschränktes Wissen eines Problemlösers muß durch heuristische Methoden ausgeglichen werden. Es gibt keine Wissensbasis, die ausreichend vollständig ist, als daß auf diese Strategie verzichtet werden könnte.

3.5 Ein Planungsbeispiel mit MOLGEN

MOLGEN ist eingesetzt worden, um Experimente hinsichtlich einer Klasse von Synthetisierungsproblemen, bekannt als Gen-Klonungs-Experimente, zu planen. In solchen Experimenten geht es darum, Bakterien zur Synthetisierung eines gewünschten Eiweißstoffes einzusetzen.

Das Ratten-Insulin-Experiment ist eines von wenigen Gen-Klonungs-Experimenten, die mit MOLGEN geplant worden sind. (Bereits bei diesen wenigen Experimenten brachte MOLGEN in gewissen Punkten kein zufriedenstellendes Ergebnis !).

Bevor das Fortschreiten des Planerstellens mit MOLGEN im folgenden illustriert wird, sei zunächst noch erklärt, wie das angestrebte Ziel lautet. In MOLGENs Notation sieht das gesteckte Ziel so aus (Abb. 3.5-1):

```
Prototype is SYNTHESIS-PROBLEM

GOAL: [CULTURE-1 with
       ORGANISMS: [BACTERIUM-1 with
        EXOSOMES: [VECTOR-1 with
           GENES: [RAT-INSULIN]]]]
TO-PLAN:     META-PLAN-INTERPRETER
PLAN-NUMBER: 1
DESCR:       This synthesis problem is to
             clone the gene for rat-
             insulin.
```

Abb. 3.5-1 Zielbeschreibung des Ratten-Insulin-Experiments bei MOLGEN

Es geht also darum, eine Kultur von Bakterien mit virusähnlichen Kleinstlebewesen (engl. vector), z.B. Bakteriophagen, welche die spezifischen Gene für Ratten-Insulin in die Bakterien "befördern" sollen, zu verschmelzen. Als Zweck des Ganzen verspricht man sich die synthetische Gewinnung von Ratteninsulin, welches durch diese "modifizierte" Bakterienkultur produziert wird.

Bei der Erläuterung der Vorgehensweise von MOLGEN werden aus Platzgründen nicht die gesamten Ausgaben, die während des Planerstellens produziert werden, wiedergegeben. Nur die wesentlichen Angaben werden gezeigt. Es wird dabei auch nicht die ursprüngliche Notation verwendet.

Erstellen des Planes:
- - - - - - - - - - -

MOLGEN deutet das gegebene Ziel (Abb. 3.5-1) - u.a. festzulegen, welche Bakterie und welcher Überträger einzusetzen sind - als eine Forderung, einen Laborplan zu entwerfen, um die beschriebene Bakterie zu gewinnen. Für die zu spezifizierenden Größen werden zunächst Planvariablen eingesetzt (BACTERIUM-1, VECTOR-1, CULTURE-1). Der Interpreter startet den Prozeß durch Aktivierung von <Focus>. Auf Grund der hohen Priorität der Vergleichsoperatoren auf der Planebene geht MOLGEN nun zunächst so vor, daß es die Unterschiede zwischen der in der Zielbeschreibung gegebenen Bakterienkultur und den Angaben über die "Standardbakterienkultur" in der genetischen Wissensbasis unter die Lupe nimmt (Abb. 3.5-2).

```
Interpreter
-->   1. Strategieschritt:  <Focus>
-->   1. Planschritt: <Find-Unusual-Features>
                       INPUT: 1.Zielbeschreibung (s.o.)
-->   1. Planschritt erfolgreich abgeschlossen:
                       OUTPUT: Difference-1
      Unterschied erkannt zwischen BACTERIUM-1 und
      Bakterienprototyp in der Wissensbasis;
      Eintrag des erkannten Unterschieds (VECTOR-1
      als EXOSOME) in Planobjekt Difference-1.
```

Abb. 3.5-2 1. Strategieschritt,
1. Planschritt

MOLGEN beginnt nun, ausgehend von Difference-1, einen abstrakten Plan zu erstellen, in dem es teilweise einen Laborschritt instantiiert. Es sind Unterschiede aufgetreten: ein Signal für MOLGEN, den Planoperator <Propose-Operator> zu aktivieren, um einen zunächst abstrakten Laboroperator auszuwählen, diese Unterschiede zu reduzieren (Abb. 3.5-3).

```
Planebene
-->   2. Planschritt: <Propose-Operator>
                       INPUT: Difference-1
-->   2. Planschritt erfolgreich abgeschlossen:
                       OUTPUT: 1.Laborschritt
      <Propose-Operator> hat den 1. Laborschritt teil-
      weise instantiiert; als erster abstrakter Operator
      auf Laborebene wurde <Merge> bestimmt, da dieser
      Operator die Anfrage "APPLY?" von <Propose-Operator>
      als einziger Laboroperator bestätigt hat.
```

Abb. 3.5-3 2. Planschritt, 1. Laborschritt

Nun ist also der 1. Laborschritt teilweise instantiiert. Es ist jedoch erst ein abstrakter Operator eingesetzt und auch noch nichts darüber ausgesagt, welche Objekte miteinander durch <Merge> fusioniert werden sollen.

Die nun folgenden Planschritte (Vorschlagen von Teilzielen für die Objekte von <Merge>; Vorschlagen weiterer Schritte) werden jetzt nicht ausführlich dargelegt, um das Beispiel nicht zu langatmig werden zu lassen. Danach ergibt sich folgender, noch abstrakter Laborplan (Abb. 3.5-4):

```
Planebene
--> 6. Planschritt: <Refine-Operator>
                    INPUT: 1.Laborschritt
--> 6. Planschritt erfolgreich abgeschlossen:
                    OUTPUT: 1.Laborschritt (näher spe-
                                            zifiziert)
                            Refinement-1
                            Constraint-1
  <Refine-Operator> ersetzt den abstrakten  Operator <Merge> durch den
  spezifischen  Operator <Transform>. Dies ist mit der  Einführung von
  Constraint-1 verbunden, da  <Transform> an  die  Argumentobjekte ge-
  wisse Forderungen stellt.
  In Refinement-1 sind verzeichnet: <Merge>, <Transform>, Constraint-1
  In Constraint-1 ist festgehalten, daß die beiden Argumentobjekte von
  <Transform>, BACTERIUM-3 und  VECTOR-1, zueinander  verträglich sein
  müssen.
```

Abb. 3.5-4 Verfeinerung von 1. Laborschritt, erstes Constraint eingeführt.

MOLGEN hat also mittlerweile seinen abstrakten Plan verfeinert: der abstrakte Operator <Merge> wurde durch den konkreten Operator <Transform> ersetzt, die Eingabegrössen von <Transform> sind beschrieben und Constraint-1 wurde eingeführt. Anstatt die Planvariablen BACTERIUM-3 und VECTOR-1 mit Werten zu instantiieren,

die den Anforderungen genügen würden, hält MOLGEN die Optionen offen und bindet ein Constraint an die Eingabegrössen (Variable) von <Transform>.

Im folgenden wird von der Planebene ausgehend <Transform> durch Aktivierung des Planoperators <Check-Prediction> simuliert. MOLGEN verfügt zu jedem Laboroperator über ein Simulationsmodell, wodurch es ermöglicht wird, Ergebnisse eines Laborschrittes vorherzusagen. Die Simulation von <Transform> bringt das Ergebnis, daß bei einem solchen Verschmelzungsprozeß nicht alle Bakterien den Überträger aufnehmen; ein Teil der Bakteriophagen kann die Bakterienmembran nicht durchdringen. Folglich liefert der 1. Laborschritt als Ausgabegrössen BACTERIUM-4, das den Überträger nicht aufgenommen hat, und BACTERIUM-3, dessen Membran von den Überträgern durchdrungen werden konnte.

Es stellt sich also heraus, daß ein Unterschied besteht zwischen angestrebtem Ziel und tatsächlichem Ergebnis bei Durchführen des Laboroperators. Folglich wird nun, um die Unterschiede möglichst zu reduzieren, ein weiterer Planschritt gestartet mit der Eingabe der festgestellten Unterschiede, wobei <Propose-Operator> aktiviert wird. Als abstrakter Laboroperator wird <Sort> bestimmt. Nach einigen Zwischenschritten (wie gehabt) liefert dann die Anwendung des Plan-Operators <Refine-Operator> als einzigen möglichen spezifizierten Laboroperator <Screen>, der die Verfeinerung des MARS-Operators <Sort> darstellt. <Screen> tötet Bakterien mittels eines Antibiotikums ab. Eine neue Größe für das Laborobjekt Antibiotic muß eingeführt werden. MOLGEN sieht zunächst wieder von einer sofortigen, passenden Instantiierung ab, da eventuell in weiteren Planschritten Informationen zur Auswahl des geeignetsten Antibiotikums ermittelt werden. Damit sich MOLGEN nicht auf ein spezielles Antibiotikum aus der Wissensbasis beziehen muß, führt es eine neue Variable, ANTIBIOTIC-1, dafür ein und bindet an die Variablen BACTERIUM-3 und ANTIBIOTIC-1 das CONSTRAINT-2, das besagt, daß BACTERIUM-3 gegen ANTIBIOTIC-1 resistent sein muß, und an die Variablen BACTERIUM-4 und ANTIBIOTIC-1 das CONSTRAINT-3, das besagt, daß BACTERIUM-4 gegen ANTIBIOTIC-1 nicht resistent sein darf.

Als Zwischenstand sind mittlerweile folgende Objekte und Variablen eingeführt (Abb. 3.5-5):

BACTERIUM-1: Bakterien, bevor sie im Transform-Prozeß mit VECTOR-1 verschmolzen werden. Untergliedern sich nach dem Verschmelzungsprozeß in BACTERIUM-3 und BACTERIUM-4.

BACTERIUM-3: Bakterien, die gegen ANTIBIOTIC-1 resistent sind und deren Membran für die Überträger VECTOR-1 durchlässig ist.

BACTERIUM-4: Bakterien, die gegen ANTIBIOTIC-1 nicht resistent sind Und deren Membran für VECTOR-1 nicht durchlässig ist.

ANTIBIOTIC-1: Antibiotikum, das die Bakterien BACTERIUM-4, die den Überträger nicht aufgenommen haben, abtöten.

VECTOR-1: Virusähnliches Kleinstlebewesen, das als Überträger die für Ratten-Insulin spezifischen Gene trägt.

VECTOR-2: Virusähnliches Kleinstlebewesen, das als Überträger noch keine für Ratten-Insulin spezifischen Gene trägt.

CONSTRAINT-1: Forderung an BACTERIUM-1 und VECTOR-1, daß beide zueinan der verträglich sein müssen.

CONSTRAINT-2: Forderung an BACTERIUM-3, daß sie gegen ANTIBIOTIC-1 resistent sein müssen.

CONSTRAINT-3: Forderung an BACTERIUM-4, daß sie gegen ANTIBIOTIC-1 Nicht resistent sein dürfen.

Abb. 3.5-5 Objekte und Variablen während des Planerstellens

Die nun folgenden Schritte illustrieren die Propagierung von Constraints.

Eine Möglichkeit, Interaktionen möglichst frühzeitig zu erkennen, besteht darin, Constraints zwischen Teilproblemen weiterzureichen. In solchen Fällen werden neue Constraints aus bereits existierenden abgeleitet.

Nach Einführung neuer Constraints (CONSTRAINT-2 und CONSTRAINT-3) wird der Planoperator <Propagate-Constraint> aktiviert, um die Information der neuen Constraints im Plan zu verbreiten. Dieser Propagierungsprozess spiegelt die folgenden Zusammenhänge in gegebenem Planbeispiel wider:

Gemäß CONSTRAINT-2 sind die Bakterien BACTERIUM-3 gegen ANTIBIOTIC-1 resistent, gemäß CONSTRAINT-3 sind die Bakterien BACTERIUM-4 gegen ANTIBIOTIC-1 nicht resistent.

Die Widerstandsfähigkeit einer Bakterie gegen ein Antibiotikum wird durch dafür spezifische Gene verliehen, die entweder in den Bakterienchromosomen oder außerhalb derselben, also extrachromosomal lokalisisert sind. Die beiden im Plan vorkommenden Bakterienformen BACTERIUM-3 und BACTERIUM-4 stammen von denselben Bakterien BACTERIUM-1 ab, sind folglich vom gleichen Typ und haben dieselben Chromosomen. Deshalb kann die Widerstandsfähigkeit der Bakterien BACTERIUM-3 nicht von den eigenen bakteriellen Chromosomen herrühren, sondern muß von extrachromosomalen Körpern stammen. VECTOR-1 ist der einzige extrachromosomale Körper, der in BACTERIUM-3, aber nicht in BACTERIUM-4 vorhanden ist. Daher muß VECTOR-1 ein für die Widerstandsfähigkeit gegen ANTIBIOTIC-1 spezifisches Gen tragen, das BACTERIUM-3 die Widerstansfähigkeit verleiht. VECTOR-1 entstand aus VECTOR-2 zusammen mit dem Ratten-Insulin-Gen. Und da das Ratten-Insulin-Gen kein für irgendein Antibiotikum widerstandsspezifisches Gen enthält, muß VECTOR-2 ein für die Widerstandsfähigkeit gegen ANTIBIOTIC-1 spezifisches Gen tragen. Letztere Aussage wird durch die Formulierung von CONSTRAINT-7, gebunden an die Planvariable VECTOR-2, festgehalten.

So konnte nun durch eine Aneinanderreihung von Argumenten eine Aussage über ein bereits im Plan existierendes Objekt, nämlich VECTOR-2, getroffen werden, sobald CONSTRAINT-3 und CONSTRAINT-4 formuliert waren, ermöglicht durch die Technik der Constraint-Propagierung.

An dieser Stelle sei bemerkt, daß MOLGEN unterschiedliche Variablennamen verwendet, z.B. BACTERIUM-1 und BACTERIUM-4, um sich auf Objekte im Plan zu unterschiedlichen Zeiten, d.h. in unterschiedlichen Planungsstadien zu beziehen. Solche Variablen stehen in einer "Gleicher_Typ"-Relation. Somit implizieren unter Umständen Lösungen für eine Variable Lösungen für andere Variablen desselben Typs.

Die Technik, die bisher im Beispiel noch nicht verwendet wurde, ist die der Constraint-Erfüllung: Constraints in MOLGEN beschreiben Einschränkungen für Laborobjekte in Plänen. Das Erfüllen von Constraints ist lediglich ein Durchsuchen des Wissensbasis nach in ihr gespeicherten, geeigneten Objekten, die die Constraints, welche an die Planvariablen gebunden sind, erfüllen. <Refine-Object> ist der Planoperator zur Constraint-Erfüllung von Laborobjekten. Er veranlaßt eine Suche durch die Wissensbasis nach möglichen Bindungen und legt sie in einer speziellen Datenstruktur, einem Tupel, ab. Tupel-Datenstrukturen beinhalten bei MOLGEN die möglichen Lösungen für Constraints.

Beim Beispiel sieht dies wie folgt aus (Abb. 3.5-6):

```
Planebene
--> 15. Planschritt: <Refine-Object>
                      INPUT: CONSTRAINT-1
--> 15. Planschritt erfolgreich abgeschlossen:
                      OUTPUT: TUPLE-1

TUPLE-1:
  Constraints:      CONSTRAINT-1
  Variablen:        BACTERIUM-3, VECTOR-1
  mögl. Bindungen:  BACTERIUM-3:  E.COLI
                    VECTOR-1:     PSC101,
                                  PBR322,
                                  PMB9
```

Abb. 3.5-6 Constraint-Erfüllung am Beispiel

An dieser Stelle soll das Beispiel abgebrochen werden. Alle bei MOLGEN eingesetzten Techniken mit Constraints sind bereits vorgekommen. Beim restlichen Planerstellungsprozeß wird "im gleichen Stil" vorgegangen.

Dieses Planbeispiel sollte die in den ersten Abschnitten dieses Kapitels aufgezeigten Methoden beim konkreten Einsatz erläutern und die theoretischen Beschreibungen plausibel veranschaulichen.

3.6 Beurteilung und Kritik

MOLGENs hierarchische Kontrollstruktur ermöglicht, dem eigentlichen Planprozeß im Problembereich eine Meta-Ebene überzuordnen, welche die den vielen Planungssystemen eigene Starrheit insofern etwas auflockert, daß MOLGEN zwischen verschiedenen Planstrategien umschalten kann, je nach Bedarf. Allerdings kann nicht von Meta-Planen im ursprünglichen Sinn gesprochen werden, da immer nur ein einziger Strategieoperator bestimmt und gleich ausgeführt wird. Um das Umschalten der Operatoren auf Strategieebene noch auszufeilen, könnte man noch eine Meta-Meta-Ebene einführen.

Die wenigen Experimente, die mit MOLGEN geplant wurden, haben bereits gezeigt, daß die erzielten Planergebnisse nicht immer zufriedenstellend sind: die erzeugten Planprotokolle sind sehr umfangreich, mögliche Lösungen wurden unterschlagen. Eine Möglichkeit, den Planerstellungsprozeß zu beschleunigen, bestünde darin, bereits existierende Pläne durchzusehen, falls nötig, zu modifizieren (z.B. mit <Undo>) und, wenn die Ziele zum aktuellen Problem übertragbar wären, in den neuen Plan zu übernehmen und auf diese Weise aufwendige Planarbeit zu sparen. Die Tatsache, daß mögliche Lösungen von MOLGEN manchmal nicht entdeckt werden, liegt am nicht ausreichend verfügbaren Wissen. Wenn MOLGEN auf ein detaillierteres genetisches Modell zugreifen könnte, wäre dieser Planer in der Lage, noch mehr Constraints während des Planerstellungsprozesses zu formulieren und dadurch die Optionen noch länger offen zu halten.

Die Kombination der Operationen Formulieren, Propagieren und Erfüllen von Constraints ist sehr vorteilhaft bei MOLGEN. Auf diese Weise müssen die bei der Planung verwendeten Constraints nicht bereits zu Beginn des Planungsprozesses festgelegt, sondern sie können dynamisch formuliert werden. Hierarchisches Planen wird dadurch ermöglicht. Nachteilig allerdings ist, daß Constraints nur auf Laborebene verwendet werden können. Meta-Constraints könnten sich auf den Planungsprozeß beziehen und gewisse Einschränkungen ausdrücken; z.B. könnte ein solches Meta-Constraint "vorschreiben", daß der Plan aus maximal 12 Schritten aufgebaut sein darf. Constraints können bei MOLGEN also nur zur Objektspezifizierung, nicht aber zur Prozeßbeschreibung eingesetzt werden. Ein weiteres Manko beim Arbeiten mit Constraints liegt bei MOLGEN darin, daß Constraints in den einzelnen Teilproblemen nur vom Ziel zu den Eingangsgrössen weitergereicht werden können und nicht umgekehrt. Die durch Constraint-Propagierung mögliche Kommunikation zwischen den Teilproblemen ist also nur beschränkt möglich.

Durch die Aufteilung von Wissen in Schichten kann die Struktur eines Problemlösers überschaubarer und besser handhabbar gestaltet werden. Dadurch, daß die Strategieüberlegungen auf einer Metaebene angesiedelt sind, kann die Komplexität der möglichen Interaktionen zwischen den Planschritten bereits frühzeitig eingedämmt werden. Reihenfolge-Restriktionen können schon auf abstrakter, noch überschaubarer Ebene erkannt werden. Als vorteilhaft erweist sich der <Undo>-Strategieoperator, der aufgetretene Überbeschränkungen durch Rückgängigmachen von Variableninstantiierungen wieder beheben kann. <Undo> liefert allerdings nicht in allen derartigen Situationen das gewünschte Ergebnis und prüft auch nicht nach, ob die vorgenommenen Modifikationen überhaupt eine Verbesserung gebracht haben.

Bei nicht ausreichend vorhandenem Wissen ist MOLGEN immer noch in der Lage, erfolgreich zu planen, aber nur nach aufwendigem zusätzlichem Backtracking. Fehlendes Wissen bei der Constraint-Erfüllung hat negative Auswirkungen insofern, daß falsche Planungsentscheidungen unentdeckt bleiben, bis viel unnötige Arbeit aufgewendet werden mußte.

Ein Verbesserungsvorschlag für MOLGEN wäre, bei nicht herausgebrachtem Plan für ein Experiment, also bei einem Fehlversuch, den Grund für das Scheitern genauer anzugeben, ob es etwa daran liegt, daß während des Problemlösevorgangs nicht aufeinander abstimmbare Constraints eingeführt wurden, oder daran, daß das Wissen bezüglich der Objekte auf Laborebene zu unvollständig ist, oder daran, daß das Wissen darüber, wie überhaupt ein Experiment im entsprechenden Fall zu planen ist, nicht ausreichend vorhanden ist. Um diesen Mangel bei MOLGEN zu verbessern, müßte Wissen über mögliche Abbruchbedingungen und über den (Un-)Vollständigkeitsgrad der Wissenbasis verfügbar gemacht werden.

Schließlich sei noch angeführt, daß bei MOLGEN die Repräsentation des Faktors Zeit nicht zufriedenstellend gelöst ist. Um die zeitlichen Änderungen von Objekten in die Planung miteinzubeziehen, gebraucht MOLGEN unterschiedliche Namen, die sich entweder auf dasselbe Objekt oder zumindest auf ein Objekt desselben Typs beziehen. Indirekt können daraus Rückschlüsse auf den Zeitpunkt von Operationen, z.B. Constraint-Erfüllungen, gezogen werden, Aussagen über ganze Zeiträume können jedoch nicht daraus abgeleitet werden.

Kapitel 4 : Planungssystem SIPE

SIPE (System for Interactive Planning and Execution Monitoring) [17, 18] ist ein domänen-unabhängiges Planungsprogramm, welches die automatische und interaktive Erstellung von Plänen ermöglicht.

Bei der Implementierung von SIPE wurde auf folgende Punkte besonders geachtet:

- In erster Linie sollte das Planungssystem nicht auf eine spezielle Domäne fixiert sein.
- Großer Wert wurde auf die Verwendung vielfältiger und ausdrucksstarker Repräsentation von verfügbarem Wissen und erstellten Daten gelegt.
- SIPE sollte neue ausgefeiltere Techniken zum Erkennen und Beheben störender Interaktionen zwischen Teilproblemen einsetzen.
- Parallele Bearbeitung von Teilproblemen innerhalb eines Planes sollte durch SIPE zum Erstellen effizienter Pläne unterstützt werden.
- Verbesserte Kontrollmöglichkeiten des Benutzers über den Planungsprozeß und die Ausführung eines erstellten Planes sollten SIPE zu einem interaktiven Planer machen.

Inwieweit es gelungen ist, mit SIPE ein Planungssystem entwickelt zu haben, das obige Fähigkeiten aufweist, soll die nun folgende Beschreibung und Analyse von SIPE in diesem Kapitel aufzeigen.

4.1 In SIPE verwendete Planungstechniken

SIPE baut einen Plan als Baumhierarchie auf, in der die zueinander in Beziehung stehenden Knoten nicht nur die einzelnen Planschritte repräsentieren, sondern sämtliche planrelevanten Grössen wie Planungsobjekte, -ziele und -operatoren. Somit läßt sich der Plan auf verschiedenen Abstraktionsstufen beschreiben.

Mehrstufiges Planen wird dadurch ermöglicht, daß abstrakte Operatoren im Fortschreiten des Planens nach und nach verfeinert, d.h. auf konkretere, umfassendere Detail-Ebenen expandiert werden. Zur Beschreibung der größtenteils domänen-spezifischen Operatoren ist eine

Sprache gewählt, die das Formulieren von Operatoren für verschiedene Domänen zuläßt.

Die zusätzliche Verwendung deduktiver Operatoren befreit den Wissensingenieur bei der Formulierung von Operatoren davon, sämtliche, durch Anwendung eines Operators zu erwartenden Änderungen des Welt-Modells explizit aufzählen zu müssen. Das System ist in der Lage, aus gegebenen Fakten weitere Schlußfolgerungen zu ziehen.

Die Strategie, nach der SIPE vorgeht, läßt sich in zweifacher Hinsicht als hierarchisches Planen bezeichnen. Einerseits wird ein Plan hierarchisch aufgebaut: ausgehend von einem sehr abstrakten Entwurf wird ein detaillierter, ausführbarer Plan, repräsentiert als Netzwerk von durch verschiedene Attribute genau spezifizierten Planschritten, erstellt. Andererseits sind alle statischen, konstanten Planobjekte bzw. Objekteigenschaften in einer speziellen Objekthierarchie verwaltet - die dynamischen Plangrößen werden als Planvariablen formuliert. Dadurch ist ein schnelleres Auffinden von korrekten Instantiierungswerten für Variablen und eine prägnantere Formulierung des Domänenwissens ohne unnötige redundante Angaben über "ähnliche" Objekte möglich.

Zur Bestimmung eines in einer gegebenen Situation anzuwendenden Operators stellt SIPE einen Mechanismus zur Verfügung, einstufiges Planen als Inferenz auf einer Planebene zu ermöglichen. Die Auswahl eines geeigneten Operators erfolgt dann dadurch, daß alle möglichen Operatoren daraufhin untersucht werden, ob ihre Anwendung die gewünschten (Teil-)Zielbedingungen herbeiführt. Jede Operatorbeschreibung enthält unter anderen die Attribute, in welchen Situationen der betreffende Operator angewendet werden kann und welche Effekte sich durch die Anwendung ergeben. Dadurch gestaltet sich die Suche nach einem geeigneten Operator einfacher. Der verwendete Suchalgorithmus ist allerdings nicht sehr effizient; es wird kein Wissen zur möglichst schnellen Operatorauswahl eingesetzt, sondern stur Operator für Operator untersucht: ein Merkmal dafür, daß SIPE für interaktives Planen konzipiert wurde. Der Benutzer kann (oder soll) während der Planerstellung "Hilfestellungen" z.B. in Form von auszuführenden Operationen geben, die dann das System sogleich auswertet.

Die Verwendung von Constraints ermöglicht in SIPE nicht nur eine umfassendere und detailliertere Repräsentation von Domänenwissen - dadurch können nicht nur Objekteigenschaften, sondern auch -beziehungen spezifiziert werden -, und ein effizienteres Planen insofern, daß Randbedingungen in den Planungsprozeß miteingehen können, sondern auch die Formulierung

quantifizierter Grössen durch Bindung von möglichen Variablenwerten an eine bestimmte Klasse. Dies unterstützt die Vorgehensweise des Planens als Inferenz, bei der die Darstellungsweise der Prädikatenlogik (1. Stufe) bei SIPE verwendet wird.

Die Idee des Meta-Planens ist nur sehr beschränkt verwirklicht. Operatoren auf der abstraktesten Ebene können als Meta-Operatoren aufgefaßt werden. Sie werden zum einen beim ersten Entwurf eines Planes angewendet, zum anderen finden sie bei den bei SIPE sehr fortgeschrittenen Möglichkeiten zu Ausführungsüberwachung und Umplanung Anwendung.

4.2 Das Handhaben paralleler Aktionen

Bei der Erstellung von Plänen in realistischen Domänen muß beachtet werden, daß bei der Planausführung bestimmte Aktionen parallel ausgeführt werden können. Ein Beispiel: ein Roboter mit zwei Greifarmen als Manipulatoren stehe zur Verfügung, um eine Konstruktion aus einzelnen Teilen aufzubauen; dann können gewisse Aktionen von den Greifarmen parallel, also gleichzeitig durchgeführt werden, andere Aktionen hingegen beeinflussen sich gegenseitig und erfordern eine sequentielle Ausführung.

Zwei Teile eines Planes stehen in paralleler Beziehung, falls die partielle Ordnung des Planes die Durchführungsreihenfolge dieser Teile nicht festlegt. Die Beschreibungssprache für Operatoren ermöglicht bei SIPE die explizite Formulierung paralleler Aktionen (als Beispiel siehe Abb. 4.3.2-1).

Die Strategie bei der Planung mit SIPE besteht nun darin, möglichst viele Parallelitäten in den Plan mitaufzunehmen, um eine schnelle Planausführung zu gewährleisten und eine "Zwangssequentialisierung" der einzelnen Aktionen zunächst zu vermeiden. Interaktionen zwischen Planästen müssen dann als solche erkannt werden: störende Interaktionen sind zu korrigieren oder zu beheben und nützliche Interaktionen müssen ausgenützt werden.

4.2.1 Verschiedene Arten von Interaktionen ... ... und wie sie von SIPE bewältigt werden

Von einer Interaktion zwischen zwei Teilplänen oder Planästen spricht man dann, wenn ein Ziel, das in dem einen Teilplan auf irgendeiner Hierarchieebene zu erreichen versucht wird, durch Ausführung einer Aktion aus dem anderen, zum ersteren parallelen Teilplan erfüllt (nützliche Interaktion) oder verhindert (schädliche Interaktion) wird. Auftretende Interaktionen können schon vor Anwendung von Operatoren vorhergesagt werden, da jeder Operator die zu erwartenden Effekte explizit angibt.

Das Planungssystem muß aus einer nützlichen Interaktion entsprechende Vorteile ziehen können. Dies kann dadurch erreicht werden, daß durch Einführung von zusätzlichen Constraints eine Reihenfolge der durchzuführenden Schritte festgelegt wird, damit aus der gemachten Feststellung Nutzen gezogen werden kann. Diese Reihenfolgefestlegung kann interaktiv vom Benutzer kontrolliert werden. Es können aber auch andere Maßnahmen angebrachter sein.

Ein Beispiel für eine nützliche Interaktion: ein in späteren Planschritten zu erfüllender Zweck oder zu erfüllendes Ziel wird bereits durch Anwendung eines Operators in einem früheren Planschritt als Seiteneffekt erzielt; folglich wird in einigen Schritten unnötige Planarbeit geleistet, diese Schritte können aus dem Plan bzw. Planentwurf gestrichen werden.

Die optimale Entscheidung, was bei der Erkennung einer Interaktion zu tun ist, könnte erst dann getroffen werden, wenn alle aus einer bestimmten (Reihenfolge-)Festlegung resultierenden Konsequenzen ausgewertet werden würden. Da diese Vorgehensweise aus Effizienzgründen nicht praktizierbar ist, bedarf es der Verwendung von Heuristiken.

Vor allem bei der Beseitigung von störenden Interaktionen kann dieser Ansatz bei einem domänenunabhängigen Planer wie SIPE Schwierigkeiten bereiten, da viele mögliche Typen solcher Interaktionen domänenspezifische heuristische Lösungsstrategien notwendig machen. Zur Aufhebung von störenden Interaktionen reicht in der Regel eine bloße Reihenfolgeumordnung nicht aus. Die Anwendung weiterer Operatoren auf vielleicht übergeordneter Ebene, Instantiierung einer Variablen mit einem anderen Wert oder andere Tricks werden unter Umständen erforderlich. Das Problem liegt darin, daß störende Interaktionen zum Erreichen des Planzieles beseitigt werden müssen, wohingegen nützliche Inter-

aktionen nur aus Effizienzgründen und nicht aus einer Notwendigkeit heraus berücksichtigt werden sollten.

Für das Beheben von störenden Interaktionen zwischen parallelen Teilplänen läßt sich keine pauschale Lösungsstrategie angeben, um immer das beste Ergebnis zu erzielen. Für verschiedene Fälle sind verschiedene Vorgehensweisen anzuwenden. Um den Algorithmus, nach welchem SIPE zur Beseitigung solcher Konflikte vorgeht, kurz aufzuzeigen, sind im folgenden einige mögliche Situationen beschrieben, in denen SIPE entsprechende Konfliktlösungsmaßnahmen ergreifen muß.

Angenommen, ein bestimmtes gültiges Prädikat erhielte bei Erreichen irgendeines Knotens innerhalb eines Planastes den Wahrheitswert FALSCH und auf irgendeinem Knoten innerhalb eines zum erstgenannten parallelen Planastes den Wahrheitswert WAHR. Diese Konfliktsituation kann verschiedenen Charakter haben, je nachdem, ob es sich bei dem konflikt-aufzeigenden Prädikat um einen unerheblichen Seiteneffekt handelt, der für den Plan nicht relevant ist, oder um den Zweck des Planes, also permanent den Wahrheitswert WAHR behalten muß, oder um eine Vorbedingung für die Anwendung eines Operators zur Erfüllung eines späteren Zweckes, also nur vorübergehend den Wahrheitswert WAHR haben muß.

Daraus ergeben sich nun verschiedene mögliche Situationen, von denen hier nur ein paar Beispiele genannt seien:

> Drückt das betrachtete Prädikat in einem Ast eine Vorbedingung aus, besteht unabhängig von dem, was das Prädikat im parallelen Planast darstellt, eine mögliche Lösung darin, die Reihenfolge im Plan soweit festzulegen, daß das Plansegment, dessen Vorbedingung formuliert wird, als erstes ausgeführt wird. Dadurch ist sichergestellt, daß die Anwendung des betreffenden Operators erfolgen und daraus resultierend der als Attribut spezifizierte Zweck erfüllt werden kann. Welchen Wahrheitswert das betrachtete Prädikat daraufhin besitzt, spielt keine weitere Rolle mehr (es sei denn, es beschreibt die Vorbedingungen beider Planäste; dann sind weitere Maßnahmen nötig), der Konflikt ist behoben.
>
> Falls das konflikt-aufzeigende Prädikat in beiden parallelen Planästen Seiteneffekte beschreibt, besteht keine eigentliche Konfliktsituation. Es bedarf keiner Änderung.
>
> Falls das Prädikat in dem einen Planast einen Seiteneffekt, im parallelen Ast einen Zweck

formuliert, ist eine sinnvolle Lösung, zunächst den Planast mit dem Seiteneffekt im Plan einzugliedern, gefolgt von dem dazu paralllelen Planast. Dadurch wird gewährleistet, daß das Zweck-Prädikat den Wahrheitswert WAHR behält, nachdem er einmal erreicht worden ist.

Im schlimmsten Fall führt jeweils die Ausführung des einen Planastes zur Nichterfüllung des Zweckes im parallelen Planast. Dann muß ein Operator auf höherer oder gleicher Planebene angewendet werden oder größeres Umplanen wird nötig. Derartige Maßnahmen bringen mit sich, daß der gesamte Plan nach daraus möglicherweise resultierenden Auswirkungen analysiert werden muß.

4.2.2 Arbeiten mit Ressourcen - ein neues Konzept

Die Beschreibung eines Operators enthält unter anderen Angaben über alle Objekte, die an der Ausführung der spezifizierten Operation beteiligt sind. Anders als bei früher entwickelten Planungssystemen wird bei SIPE hierbei eine weitere Differenzierung vorgenommen. Solche Objekte, die für die Ausführung eines Operators zur Verfügung stehen müssen und in dieser Zeit für andere Operatoren nicht verwendbar sind, anschließend jedoch wieder "frei" sind, werden unter der Rubrik Ressourcen geführt. Alle anderen beteiligten Objekte bilden die Argumente eines Operators.

Die sich daraus ergebenden Vorteile liegen zum einen in einer detaillierteren Repräsentation und erleichterten Axiomatisierung der Domäne, zum andern in einer Verbesserung der Fähigkeiten des Systems, problematische Interaktionen zwischen parallelen Teilästen zu erkennen und damit entsprechend zu verfahren. Durch die Spezifizierung von Ressourcen kann der Wissensingenieur beim Entwurf eines Operators wichtiges domänen-spezifisches Wissen, die Wirkungsweise von Aktionen bezüglich des Welt-Modells betreffend, mit in die Operatorbeschreibung einbringen.

Wie können störende Interaktionen an Hand von Ressourcen erkannt und behoben werden ?

Das Planungssystem legt fest, daß ein Ressource-Objekt zu einem Zeitpunkt nur an einer Stelle in zueinander parallelen Teilplänen erscheinen darf. Bei sogenannten geteilten Ressourcen, deren Spezifikation in SIPE auch möglich ist, ist die Variante erlaubt, daß ein solches Objekt in einem anderen parallelen Teilast des Planes wenigstens noch als Argument auftreten darf. Bei

"gewöhnlichen", also ungeteilten Ressourcen ist auch dies nicht erlaubt und würde zu einem Ressource-Argument-Konflikt führen.

Bei einem Ressource-Argument-Konflikt sieht die heuristische Vorgehensweise von SIPE so aus, daß der Planast, welcher das konflikt-verursachende Objekt als Ressource-Objekt verwendet, vor dem dazu parallelen Planast, welcher das konflikt-verursachende Objekt als Argument-Objekt verwendet, in den Planungsablauf eingesetzt wird. Der Grund, warum sich gerade diese Konvention bei mehreren Anwendungen bereits bewährt hat, liegt an den Kriterien, nach welchen in Aktionen beteiligte Objekte als Argument- oder Ressource-Objekte spezifiziert werden. Solche Objekte, deren Zustand und/oder Position sich während der Anwendung eines Operators ändern, werden als Ressource-Objekte deklariert. Dieser Ansatz erscheint im Rahmen der gewählten Strategie als sinnvoll, da Objekte mit Positions- und/oder Zustandsänderungen während der Ausführung einer Aktion in der Tat als konflikt-kritisch aufzufassen sind. Alle anderen beteiligten Objekte werden als Argumente spezifiziert, wobei es nicht immer klar auf der Hand liegt, welcher Klasse ein Objekt zuzuordnen ist.

Falls ein Ressource-Argument-Konflikt erkannt wird, sieht die Heuristik also so aus, daß zunächst die mit dem Ressource-Objekt beteiligte Aktion ausgeführt wird, damit das betreffende Objekt "an Ort und Stelle" im Arbeitsbereich ist, um als Argument in der darauffolgenden Aktion aus dem parallelen Planast beteiligt zu sein. Diese Vorgehensweise hat sich bewährt, schließt aber ein Scheitern keineswegs aus. Der Benutzer kann durch ein entsprechendes Kommando (Setzen eines Flags) veranlassen, daß SIPE nicht nach dieser Heuristik vorgeht, wenn sie für die vorliegende Domäne unangemessen zu sein scheint.

Ressource-Ressource-Konflikte bereiten größere Probleme. Falls ein und dasselbe Objekt in parallelen Teilplänen als Ressource-Objekt verwendet wird, bedarf es unter Umständen der Anwendung weiterer Operatoren, um ein Fortschreiten des Planens unter Berücksichtigung der zu erreichenden Ziele gewährleisten zu können.

4.2.3 Constraints zum Handhaben von Interaktionen

Durch die Verwendung von Constraints wird die Zahl möglicher Ressource-Konflikte reduziert. Taucht eine noch nicht vollständig instantiierte Variable unter den Ressourcen eines Operators auf, werden Constraints an sie und andere konflikt-verdächtige Variablen aus parallelen Teilplänen gebunden, um möglichst solche Variablenwerte zu bestimmen, die störende Interaktionen vermeiden. Auf diese Weise werden beide mächtigen Werkzeuge von SIPE zur Interaktions-Reduzierung, Constraints und Ressourcen, kombiniert eingesetzt.

4.3 Wissensrepräsentation

Zur automatischen Plangenerierung durch ein Computer-system bedarf es einer Repräsentation der Welt, Aktionen und ihre Auswirkungen im Welt-Modell müssen formuliert werden; was eine Hintereinanderausführung mehrerer solcher Aktionen bewirkt und welche Aktionen sich gegenseitig wie beeinflussen, muß daraus gefolgert werden können.

Ein zentrales Problem, das sich bei der Entwicklung eines Planungssystems stellt, liegt in der formalen Darstellung von Information, die das System im Laufe einer Planung auswerten muß. Die Formulierung von Zwischenergebnissen während des Planungsprozesses durch das System muß ebenfalls derart gestaltet sein, daß diese zwischenzeitlichen Ausgabegrössen zur weiteren automatischen Plangenerierung als Eingabegrössen verarbeitet werden können.

Bei SIPE wurden verschiedene Repräsentationsformen für Domänenwissen, Planziele und Operatoren gewählt. Diese Repräsentationen sollten einerseits genügend flexibel und ausdrucksreich sein, um Wissen für unterschiedliche Problembereiche darstellen zu können, das System also weitestgehend domänen-unabhängig zu machen, andererseits genügend prägnant und ausdrucksstark, um effizientes Planen nicht an aufwendigen Suchstrategien scheitern zu lassen.

4.3.1 Repräsentation von Domänenobjekten und Planzielen

Konstante Eigenschaften von Objekten, die sich durch Ausführung von geplanten Aktionen nicht verändern, werden bei SIPE auf eine andere Weise repräsentiert als

solche Merkmale, die während der Planausführung einer Änderung unterliegen und somit (Teil-)Ziele darstellen.

Erstere, die unveränderlichen Objekteigenschaften, werden in einer hierarchischen Baum-Datenstruktur abgelegt. Dadurch können zum einen Eigenschaften innerhalb dieser Hierarchie vererbt werden, was eine effiziente und knappe Formulierung des Wissens ermöglicht, zum anderen unterstützt es die Technik des Setzens und Weiterreichens von Constraints, die an Attributwerte von Objekten gebunden sind. Die einzelnen Knoten sind verschiedenen Typs abhängig davon, ob sie "leere" Variablen, Objekte, Operatoren oder bestimmte Klassen von Objekten repräsentieren. Sämtliche plan-relevanten Teile des Systems verkörpern die Knoten innerhalb dieser Objekt-Hierarchie, wobei bei dieser Namensgebung der Begriff 'Objekt' für anwendungsbezogene Objekte (z.B. Blöcke oder Werkzeuge) und plan-spezifische Objekte (z.B. Variablen oder Operatoren) steht. Die Objekt-Hierarchie stellt sozusagen einen Baukasten dar, mit dessen Teilen SIPE einen Plan aufbaut.

Die Idee, nicht nur die eigentlichen Anwendungs-Objekte in dieser Hierarchie zu verwalten, sondern sämtliche system-relevanten Fakten, bringt die Vorteile einer einfacheren Implementierung und einer komfortableren Anwenderschnittstelle mit orthogonalem Charakter. Dieser Ansatz findet sich auch bei neueren Betriebssystemen wieder, bei denen Dateien nicht mehr nur Verwaltungseinheiten für Daten auf externen Speichern bilden, sondern auch Peripherie- oder vom Benutzer definierte logische Geräte als solche aufgefaßt werden (siehe [16]).

An jeden Knoten in der Hierarchie können Attribute gebunden werden, deren Werte aus Zahlen, Zeigern auf andere Knoten, Schlüsselwörtern oder beliebigen Zeichenketten bestehen. In der Hierarchiestruktur gilt das Vererbungsprinzip: Objekt-eigenschaften auf höherer Ebene in der Hierarchie haben auch für Knoten auf darunterliegenden Ebenen Gültigkeit, wenn sie durch Kanten miteinander verbunden sind, d.h. zueinander in Beziehung stehen.

Zur Darstellung der veränderlichen Objekt-Eigenschaften, mit welchen Ziele formuliert werden, dient eine eingeschränkte Form des Prädikatenkalküls erster Ordnung. Der Vorteil dieser Art der Spezifizierung von Zielen, die nichts anderes als zu erreichende, notwendigerweise veränderliche Objekteigenschaften und -beziehungen darstellen, besteht darin, daß eben solche Ziele gleichermaßen wie Vor- und Nachbedingungen von Operatoren beschrieben werden können.

Dadurch, daß konstante und veränderliche Merkmale auf unterschiedliche Art und Weise repräsentiert werden,

ermöglicht SIPE effizientes Schlußfolgern und somit eine schnellere Planerstellung, da die in Frage kommenden, zu manipulierenden Größen separat verzeichnet sind. Nachteilig bei diesem Ansatz ist, daß beim Planungsprozeß nicht die Möglichkeit eingeräumt wird, Objekte in der statischen Objekt-Hierarchie neu zu definieren oder aus der Hierarchie zu streichen, was in manchen Domänen als Folge der Ausführung von Aktionen nützlich sein könnte.

4.3.2 Repräsentation von Operatoren

Innerhalb eines Planes stehen Operatoren stellvertretend für konkrete bzw. abstrakte Aktionen, je nach Lokalisation in der Plan-Hierarchie, die unmittelbar bzw. nach einigen Verfeinerungsschritten in der Domäne ausgeführt werden können.

Operatoren repräsentieren dabei nicht nur Aktionen. Ihre Beschreibung enthält auch alle diejenigen Objekte, die an der betreffenden Aktion beteiligt sind und als Argumente oder notwendige Hilfsgrössen (= Ressourcen) aufgefaßt werden. Schließlich spezifiert ein Operator auch noch die zu erreichenden Ziele, die Auswirkungen auf die jeweils aktuelle Situation durch Ausführung der festgelegten Aktionen und die Vorbedingungen, die erfüllt sein müssen, bevor der Operator auf eine jeweilige Situation angewendet werden kann.

Zusätzlich zu der Beschreibung dessen, was ein Operator bewirkt, müßte der Vollständigkeit halber bei dieser Planungsweise des einstufigen Planens als Inferenz auf den einzelnen Plan-Abstraktionsebenen auch noch angegeben werden, was er nicht verändert. Dies kann ein ziemlich aufwendiges Unterfangen werden, da der konstante "Rahmen", vor dem die Operatoren Änderungen bewirken, in der Regel einer viel größeren Zahl von Angaben bedarf als die Modifikationen, die nach Operator-Aktivierung durchgeführt sein sollten und das eigentlich Wesentliche bei einem solchen Schritt darstellen.

Dieses sogenannte Frame-Problem [9] wird bei SIPE dadurch gelöst, daß die Beschreibungen der Operatoren interpretiert werden, als stünde zusätzlich zu der Formulierung der Operator-Effekte noch darin, daß alle existierenden, gültigen Relationen zwischen Objekten bestehen bleiben, unabhängig von der Ausführung sie nicht beeinflussender Operatoren, bis durch eine geplante Aktion eine Änderung spezifiziert wird. Da diese Festlegung zuerst in dem Planungssystem STRIPS [6]

verwendet wurde, wird sie auch als "STRIPS assumption" bezeichnet.

Bei SIPE ist diese Technik insofern noch ausgefeilt worden, daß nicht nur, wie bei STRIPS, explizit spezifizierte Änderungen bestehender Relationen berücksichtigt werden, sondern auch solche, die im Rahmen der Planerstellung abgeleitet werden können. Dadurch wird auch das Problem etwas abgeschwächt, wirklich alles angeben zu müssen, was ein Operator in der Welt verändert. Letzteres kann nämlich, zumal wenn man auch noch Ausnahmen und Grenzfälle beschreiben will, sehr aufwendig sein: einzelne Operatoren können soviele Änderungen bewirken, daß man sie unmöglich alle deklarativ angeben kann. SIPE stellt deduktive Operatoren zur Verfügung, um aus explizit gegebenen Fakten implizite Information "herauszuholen" (siehe nächster Abschnitt).

Bei der Beschreibung von Operatoren geht SIPE von der "Closed-World-Assumption" aus. Dabei wird angenommen, daß der beschriebene Weltausschnitt im Sinne der Aufgabenstellung vollständig ist.

Abb. 4.3.2-1 auf der nächsten Seite zeigt den abstrakten PUTON-Operator, durch dessen Anwendung eine Reihe von PUTON.DETAILED-Operatoren aktiviert werden, um die nötigen Blockbewegungen in der richtigen Reihenfolge zu planen. Der PUTON.DETAILED-Operator wiederum spezifiziert die Planschritte auf der nächsten Detaillierungsebene. Abb. 4.3.2-1 illustriert darüberhinaus die Verwendung verschiedener Attribute zur detaillierteren Beschreibung eines Operators, in diesem Fall eines Operators auf höherer, abstrakter Ebene in der Objekt-Hierarchie.

Im folgenden sind die von SIPE verwendeten Attributbezeichnungen jeweils in Klammern gesetzt.

Das VORBEDINGUNG-Attribut (PRECONDITION) - in der angegebenen PUTON-Operatorbeschreibung nicht verwendet - zeigt an, in welchen Situationen ein Operator angewendet werden kann. SIPE ist nun nicht bemüht, die spezifizierte Vorbedingung eines Operators durch entsprechende Maßnahmen zu erfüllen. Vielmehr wird in solchen Fällen, in denen keine dazu "passende" Situation vorliegt, festgestellt, daß der betrachtete Operator nicht geeignet ist, und ein anderer Operator wird daraufhin analysiert.

```
Operator:     PUTON
Arguments:    BLOCK1,
               OBJECT1 IS NOT BLOCK1,
               BLOCK2;
Purpose:      (ON BLOCK1 OBJECT1);

Plot:

   Parallel
     Branch1:
        Goals:     (CLEARTOP OBJECT1);
        Arguments: OBJECT1;
        Branch2:
        Goals:     (CLEARTOP BLOCK1);
        Arguments: BLOCK1;
   End Parallel

   Process
   Action:       PUTON.DETAILED;
   Arguments:    OBJECT1;
   Resources:    BLOCK1;
   Effects:      (ON BLOCK1 OBJECT1);

END
```

Abb. 4.3.2-1 Ein Operator in SIPE

Solche Bedingungen, zu deren Erfüllung das Planungssystem veranlaßt wird, erfolgversprechende Planschritte festzulegen, werden bei SIPE durch das ZIELE-Attribut (GOALS) formuliert. Ist ein Ziel gegeben, aber noch nicht erreicht, dann wird nicht gleich ein anderer Operator bestimmt, sondern vielmehr versucht, den betrachteten Operator weiter zu verfeinern, zu konkretisieren, um dem gestellten Ziel näher zu kommen. SIPE stellt durch die Verwendung verschiedener Typen von Vorbedingungen wenn auch nur in beschränktem Maße Wissen bereit, wie jeweils vorgegangen werden soll, um die Ziele zu erreichen.

Vorbedingungen stellen eine Verbindung her zwischen unterschiedlichen Detaillierungsebenen innerhalb der erstellten Plan-Hierarchie. Die Vorbedingung eines Operators setzt möglicherweise voraus, daß gewisse Bedingungen bereits auf höherer Ebene erfüllt worden sind. Dadurch werden zwei verschiedene Beschreibungsebenen zueinander in Beziehung gesetzt.

Überdies bieten Vorbedingungen einen zusätzlichen Schutz vor falschem Wiederaufsetzen der Planerstellung, nachdem Umplanen erforderlich wurde, bedingt vielleicht durch das Erkennen eines vom Benutzer falsch gegebenen Kommandos interaktiv während des Planungsprozesses.

Ein weiteres Attribut, das ZWECK-Attribut (PURPOSE), dient dazu zu spezifizieren, was durch Anwendung des betreffenden Operators auf der unteren Detail-Ebene erreicht werden soll (z.B. detailliertere Positionsangaben von Teilen), während durch das ZIEL-Attribut die auf höherer, abstrakterer Detail-Ebene zu erreichenden Ziele beschrieben werden (z.B. BLOCK1 auf BLOCK2). Durch die beiden Attribute ZIEL und ZWECK werden also Ziele ein und desselben Operators auf unterschiedlichen Abstraktionsstufen formuliert. Bei dem gezeigten Beispiel (Abb. 4.3.2-1) ist das ZWECK-Attribut nicht detaillierter angegeben und repräsentiert daher gleichzeitig auch das ZIEL, welches daher nicht mehr extra spezifiziert ist.

Noch ein Wort zu den übrigen verwendeten Attributen eines Operators. AKTION (ACTION) gibt eine verfeinerte Version des zugrundeliegenden abstrakten Operators an, wenn es sich nicht schon um einen elementaren Operator handelt. Planexpandierungsschritte bestehen darin, daß die in den Operatorbeschreibungen enthaltenen verfeinerten Operatoren die nächste Detail-Ebene spezifizieren. ARGUMENTE (ARGUMENTS) und RESSOURCEN (RESOURCES) sind diejenigen Objekte, die während der Planausführung eine Rolle spielen. Unter dem Operator-Merkmal RESSOURCEN sind solche Objekte aufgezählt, die für die Ausführung eines Operators zur Verfügung stehen müssen und erst danach wieder ungebunden, d.h. für andere Zwecke verfügbar sind (Näheres siehe Abschnitt 4.2.2). Das Attribut EFFEKTE (EFFECTS) beschreibt die bei der Durchführung des Operators zu erwartenden Auswirkungen auf das Welt-Modell.

Das Operator-Merkmal VORGEHENSWEISE (PLOT) spezifiziert, auf welche Art und Weise eine Operation durchgeführt werden soll. Eine derartige Beschreibung kann durch Angabe zu erfüllender ZIELE (GOALS) und/oder zu aktivierender PROZESSE (PROCESS) erfolgen. ZIELE legen dabei nicht unbedingt die einzelnen Aktionen fest, deren Durchführung die gewünschte Situation herbeiführt; dabei geht es mehr um das Endergebnis der Operation. Hingegen PROZESSE liefern eine explizite Auflistung der Aktionen; nicht nur das Ergebnis ist von Interesse, sondern auch der Weg dorthin. Zusätzlich ermöglicht SIPEs Operatorbeschreibungssprache die Angabe parallel ausführbarer Aktionen. Eine spezifizierte Vorgehensweise kann sich auch auf eine ganze Liste von Objekten beziehen, indem SIPE die Möglichkeit bietet, Variablen innerhalb einer Operatorbeschreibung nacheinander mehrfach zu instanti-

ieren: Der Operator wird iterativ auf eine Liste von Objekten angewendet.

4.3.3 Deduktive Operatoren

Bei der Beschreibung des Operators PUTON (siehe Abb. 4.3.2-1) erscheint unter der Rubrik EFFEKTE nur das Attribut (ON BLOCK1 OBJECT1). Irgendwelche sich während der Operatoranwendung ergebenden Seiteneffekte, wie z.B. das mögliche Freilegen des Objektes BLOCK2, werden nicht explizit erwähnt. Diese Information wird bei SIPE durch deduktive Operatoren abgeleitet.

SIPE ermöglicht die Spezifizierung nicht nur von "normalen", Aktionen beschreibenden, sondern auch von deduktiven Operatoren, welche aus dem aktuellen Weltzustand Fakten ableiten können. Besonders bei komplexeren Domänen können einzelne Operatoren, zumal wenn man auch noch ihre Grenzfälle und Ausnahmen beschreiben will, soviele Änderungen bewirken, daß man sie unmöglich alle deklarativ angeben kann. Durch die Verwendung von deduktiven Operatoren (Abb. 4.3.3-1 auf der nächsten Seite) braucht nicht mehr wirklich alles aufgezählt zu werden, was ein Operator in der Welt verändert, sondern es können Operator-Effekte respektive -Seiteneffekte an Hand von Axiomen über die Welt abgeleitet werden. Solche durch die Anwendung eines Operators zu erwartenden Veränderungen der Welt, die durch deduktive Operatoren abgeleitet werden und nicht explizit aufgeführt sind, werden als Seiteneffekte bezeichnet. Somit vermögen Operatoren in SIPE Haupt- und Seiteneffekte zu spezifizieren.

Dadurch, daß in SIPE die deduktiven und die anderen Operatoren in derselben Notation formuliert werden, können beide Operatorklassen mit denselben Mechanismen gehandhabt werden. Für die Anwendung müssen jeweils bestimmte Kriterien erfüllt sein.

```
Deductive.Operator: DCLEAR
Arguments:          OBJECT1,
                    OBJECT2,
                    BLOCK3 IS NOT OBJECT2,
                    OBJECT4 CLASS EXISTENTIAL
                            IS NOT OBJECT1;
Trigger:            (ON OBJECT1 OBJECT2);
Precondition:       (ON OBJECT1 BLOCK3),
                    (NOT (ON OBJECT4 BLOCK3));
Effects:            (CLEARTOP BLOCK3);
```

Abb. 4.3.3-1 Ein deduktiver Operator in SIPE

Alle möglichen Schlußfolgerungen werden während der Planerstellung immer dann gemacht, wenn ein Operator expandiert oder verfeinert wird, unmittelbar vor Operatoranwendung. Deduktive Operatoren besitzen als Auslöser (TRIGGER) bestimmte, bei der Anwendung von betrachteten Operatoren direkt zu erwartende Änderungen in der Welt.

Das Arbeiten mit deduktiven Operatoren stellt sich so dar, daß nicht wie bei anderen Operatoren irgendwelche Aktionen festgelegt werden, falls die Operator-Vorbedingung gilt, um Knoten innerhalb des Plannetz-werkes zu expandieren, vielmehr wird bei Aktivierung eines deduktiven Operators zwar zunächst auch unter-sucht, ob die Vorbedingung erfüllt ist, aber im positiven Fall erfolgt eine andere Reaktion des Planers: das aktuelle Welt-Modell wird entsprechend der spezifizierten Effekte des deduktiven Operators erweitert, ohne den existierenden Plan dabei zu verändern. Wird die Vorbedingung nicht erfüllt, bleibt die Aktivierung ohne Wirkung.

Bei Anwendung des in Abb. 4.3.3-1 gegebenen deduktiven Operators DCLEAR ergibt ein "Matchen" der Vorbedingung (d.h. Untersuchen der in der Vorbedingung verwendeten Variablen, an welche Objekte eine konsistente Bindung wie erfolgen kann), daß die Variable BLOCK3 an den Block gebunden wird, auf dem sich unmittelbar vor Ausführung des "normalen" Operators XXX (z.B. PUTON), durch dessen Expandierung die Aktivierung des deduktiven Operators DCLEAR erfolgt ist und dessen Operator-Effekte (ON OBJECT1 OBJECT2) als TRIGGER für DCLEAR formuliert sind, OBJECT1 befindet (Teil 1 der Vorbedingung (ON OBJECT1 BLOCK3)). Die Bindung des Constraints "CLASS EXISTENTIAL" an die Variable OBJECT4 ermöglicht, OBJECT4 als existentiell quantifizierte Variable zu spezifizie-ren. Durch die zusätzlich gegebene Einschränkung, daß OBJECT4 ungleich OBJECT1 ist ("OBJECT4 IS NOT OBJECT1"),

was zur Formulierung des Constraints NOT-SAME, gebunden an OBJECT4 und OBJECT1, führt (siehe nächster Abschnitt), und auf Grund der weiteren Vorbedingung (NOT (ON OBJECT4 BLOCK3)) ergibt weiteres Matchen der Vorbedingung, daß OBJECT1 das einzige Objekt repräsentiert, das sich vor Ausführung der Bewegung von OBJECT1 nach OBJECT2 auf BLOCK3 befindet. Die Begründung dafür lautet: die Aussage "es gilt nicht, daß ein Objekt OBJECT4 ungleich OBJECT1 existiert mit (ON OBJECT4 BLOCK3)" ist äquivalent zu der Aussage "für alle Objekte OBJECT4 ungleich OBJECT1 gilt nicht: (ON OBJECT4 BLOCK3)"; und daß (ON OBJECT1 BLOCK3) gilt, drückt der erste Teil der Vorbedingung von DCLEAR aus. Soweit das Auswerten der Vorbedingung.

Und da in der Argumentliste noch zusätzlich angegeben ist, daß BLOCK3 verschieden von OBJECT2 sein muß ("BLOCK3 IS NOT OBJECT2"), ist gewährleistet, daß zum Erreichen des Operator-Effektes (ON OBJECT1 OBJECT2) eine Bewegungsoperation durchgeführt werden muß, weil sich OBJECT1 noch nicht auf OBJECT2, sondern auf BLOCK3 (ungleich OBJECT2) befindet.

Somit kann abgeleitet werden, daß der Operator XXX in einer derartigen Situation als Seiteneffekt bewirkt: der Block, an den die Variable BLOCK3 gebunden wird, liegt nach Operatoranwendung XXX frei vor, d.h. es befinden sich keine anderen Objekte auf ihm. Dieser (Seiten-) Effekt braucht folglich nicht explizit angegeben zu werden.

Zum Abschluß dieses Abschnitts sei noch ein Anwendungsbeispiel aus einer realistischeren Domäne angeführt, das die Nützlichkeit des Ableitens von Seiteneffekten unterstreicht: Zur Ausführung zweier "ähnlicher" Aktionen in einer Roboterwelt sei das gleiche Werkzeug notwendig. In dem Planast I, der u.a. für die Planung der Aktion 1 zuständig ist, für die das Werkzeug zuerst benötigt wird, muß berücksichtigt werden, daß das nötige Werkzeug zuerst aus dem Werkzeugkasten herausgenommen werden muß. SIPE sollte dann in der Lage sein, in dem anderen Planast II die Information bzw. den Seiteneffekt, welcher sich nach Ausführung der Aktion 1 ergibt, folgern zu können, nämlich daß nach Ausführung von Aktion 1 das Werkzeug unbenutzt auf der Arbeitsfläche liegt und deshalb für Aktion 2, welche im Planast II spezifiziert wird und für deren Ausführung ebenfalls das Werkzeug benötigt wird, nicht mehr aus dem Werkzeugkasten geholt werden muß.

4.3.4 Constraints

Die Verwendung von Constraints bietet die Möglichkeit, partielle Beschreibungen von noch unspezifizierten Objekten zu erstellen. Dies bringt aus mehrerlei Gründen Vorteile mit sich: zum einen lassen sich Objekte dadurch mit ein und demselben Formalismus auf verschiedenen Abstraktionsebenen beschreiben, zum andern wird die Wahrscheinlichkeit erhöht, wirklich alle möglichen Lösungen für ein gegebenes Problem zu bestimmen, indem Planentscheidungen möglichst lange verzögert werden, bis ausreichend Information während der Planerstellung gesammelt ist. Ein weiterer Vorteil ergibt sich aus der Verwendung von Constraints zur Reduzierung von Ressource-Konflikten (Näheres siehe Abschnitt 4.2.3).

4.3.4.1 Darstellung von Constraints

Noch nicht instantiierte Planvariablen können dadurch partiell beschrieben werden, daß Constraints an mögliche Instantiierungswerte gebunden werden. Auf diese Weise können Einschränkungen für Objekteigenschaften und -beziehungen zu anderen Objekten formuliert werden. Das Instantiieren einer Variable kann solange hinausgeschoben werden, bis es aus der Sicht der Planfortschreitung notwendig wird oder ausreichend Information vorliegt, um keine "falschen" Variablenwerte festzulegen.

Während der Planerstellung können bei SIPE Constraints generiert werden (Constraint-Formulierung), um sich ergebende Interaktionen zwischen Teillösungen in die Planung miteinzubeziehen. Constraints werden an andere Variablen in der Plan-Hierarchie weitergereicht (Constraint-Propagierung). Variablenwerte werden so festgelegt, daß alle daran gebundenen Constraints erfüllt werden (Constraint-Erfüllung).

Mögliche Constraints an eine Variable V sind:

- CLASS ==> V muß einer spezifischen Klasse von Objekten in der Objekt-Hierarchie angehören.
- NOT-CLASS ==> V darf einer bestimmten Klasse Von Objekten nicht angehören.
- PRED ==> Bei der Instantiierung von V muß gelten, daß ein gegebenes Prädikat, das V als Argument-grösse besitzt, den Wahrheits-wert WAHR bekommt.

- NOT-PRED ==> Bei der Instantiierung von V Muß gelten, daß ein gegebenes Prädikat, das V als Argumentgrösse besitzt, den Wahrheitswert FALSCH bekommt.
- SAME ==> V muß mit demselben Wert wie Eine andere gegebene Variable, an die dieses Constraint auch gebunden ist, instantiiert werden.
- NOT-SAME ==> V darf nicht mit demselben Wert wie eine andere gegebene Variable, an die dieses Constraint auch gebunden ist, instantiiert werden.
- INSTAN ==> V muß mit einem gegebenen Wert instantiiert werden.
- NOT-INSTAN ==> V darf nicht mit einem gegebenen Wert instantiiert werden.
- OPTIONAL-SAME ==> V sollte, wenn möglich, mit demselben Wert wie eine andere gegebene Variable, an die dieses Constraint auch gebunden ist, instantiiert werden; falls dies nicht möglich ist, so ist auch ein anderer Wert akzeptabel.
- OPTIONAL-NOT-SAME ==> V sollte, wenn möglich, nicht mit demselben Wert wie eine andere gegebene Variable, an die dieses Constraint auch gebunden ist, instantiiert werden; falls dies nicht möglich ist, dann ist auch derselbe Wert akzeptabel.
- <Attributname><Attributwert> ==> V muß mit einem Objekt instantiiert werden, dessen Attribut <Attributname> den spezifischen Wert <Attributwert> hat.

4.3.4.2 Beispiele für die Verwendung von Constraints

Die Formulierung eines Constraints während des Planungsprozesses ergibt sich aus der Anwendung eines Operators. Derartige Constraints werden implizit durch die

Beschreibung eines Operators gegeben. Der in Abb. 4.3.2-1 beschriebene PUTON-Operator verdeutlicht dies als Beispiel:

- Die Formulierung "OBJECT1 IS NOT BLOCK1" führt bei weiterer Verfeinerung des Planes zum Binden eines NOT-SAME-Constraints an die Variablen OBJECT1 und BLOCK1. Damit wird sichergestellt, daß die Variablen OBJECT1 und BLOCK1 mit unterschiedlichen Werten instantiiert werden.

- Die Namensgebung BLOCK1 impliziert, daß an diese Variable ein CLASS-Constraint gebunden wird, welches verlangt, daß BLOCK1 mit einem Objekt instantiiert wird, das einen Repräsentanten der Klasse BLOCKS darstellt.

Constraints können auch bereits explizit in einer Operatorbeschreibung gegeben sein. Ein Beispiel dafür ist die Beschreibung des deduktiven Operators DCLEAR (Abb. 4.3.3-1):

- Die Spezifizierung "OBJECT4 CLASS EXISTENTIAL" im Argument-Attribut legt fest, daß OBJECT4 als existentiell quantifizierte Größe zu verstehen ist.

4.3.5 Ressourcen

Bei SIPE dienen zur näheren Beschreibung von Operatoren unter anderen die beiden Attribute ARGUMENTE und RESSOURCEN. Sie liefern eine Aufzählung all jener Objekte, eventuell repräsentiert durch eine Planvariable, die im Zusammenhang mit der Ausführung einer Operation stehen. Argumente sind "normale" Operationsobjekte, an die außer den explizit formulierten keine speziellen Forderungen geknüpft sind, während Ressourcen solche Objekte darstellen, die nur für die Ausführung eines Operators zur Verfügung stehen müssen, jedoch anschließend wieder "frei" sind.

Die Deklarierung eines Ressource-Objekts im Zusammenhang mit einer auszuführenden Aktion beschreibt eine zusätzliche Vorbedingung für die Aktion, denn die Verfügbarkeit des Ressource-Objekts bildet eine Notwendigkeit zur Ausführung der entsprechenden Aktion.

SIPE stellt Mechanismen zur Verfügung, Ressourcen zuzuweisen und wieder freizugeben und Ressource-Konflikte zu erkennen. Bereits vor der Anwendung eines Operators sind die Vorbedingungen und somit auch die notwendigen Ressourcen bekannt, so daß sich wider-

sprechende Ressource-Forderungen frühzeitig noch vor der Planverfeinerung zur nächsten Ebene ausgemacht werden können. Es können unter Erfüllung der Constraints solche Operatoren ausgewählt werden, die keine Ressource-Konflikte hervorrufen. Ressource-Konflikte lassen sich dabei auch bezüglich Objekten erkennen, die durch noch nicht instantiierte Planvariablen repräsentiert werden.

Die explizite Aufstellung der Ressourcen bringt den weiteren Vorteil, daß die Operatorbeschreibungen und somit auch ein vollständig erstellter Plan leserlicher und damit verständlicher werden.

4.3.6 Repräsentation des Planes

Bei SIPE stellt ein Plan eine Menge von partiell geordneten Zielen und Aktionen dar, welche vom Planungssystem mit Hilfe von Operatoren beschrieben werden, deren Anwendung die Lösung des gegebenen Problems hervorbringen soll.

Ein Plan wird hierarchisch Schritt für Schritt aufgebaut. Ausgehend von wenigen initialen Knoten auf abstraktester Ebene, spezifiziert durch die Aufgabenstellung, entsteht eine Baumstruktur, deren Blätter atomare Operatoren bilden. Genauer gesagt stellt ein Plan ein Netzwerk dar, da Beziehungen auch über mehrere Hierarchie- oder Abstraktionsebenen hinweg bestehen. Knoten repräsentieren nicht nur Operatoren, sondern z.B. auch Prozesse und Ziele. Die hierarchische Struktur der von SIPE aufgebauten Pläne erweist sich auch beim Weiterreichen von Constraints von einem Ast zum anderen als vorteilhaft.

Um einen Knoten weiter zu spezifizieren, d.h. eine Verbindung zu einer umfassenderen Detail-Ebene hinunter im Baum zu legen, bedarf es der Anwendung eines Operators. SIPE analysiert, welche Knoten durch welche Aktionen wann generiert wurden, um planungs-spezifische Entscheidungen fällen zu können. Dem Planungssystem ist dadurch Sinn und Zweck eines jeden Knotens bekannt.

Dieser Ansatz erweist sich als brauchbar, um das Planungssystem in die Lage zu versetzen, erklärende Kommentare zu liefern bei fehlgeschlagener Planung oder zumindest bei Stagnieren der Planerstellung oder bei Rückfragen von Seiten des Benutzers, warum gewisse Planungsschritte bis zum aktuellen Zeitpunkt durchgeführt worden sind. SIPE nutzt diese Information während des Planungsprozesses, um störende Interaktionen zwischen parallelen Teilplänen analysieren und beseitigen zu helfen. Außerdem schränkt SIPE dadurch die bei

den einzelnen Aktionen zu beachtenden Bedingungen ein. Nur das, worauf es "ankommt", muß berücksichtigt werden, andere Informationen können außer acht gelassen werden.

4.4 Architektur

Die wesentlichen Komponenten, aus denen SIPE besteht, sind eine Wissensbasis, die sich in einen statischen und dynamischen Teil untergliedert, eine Inferenzmaschine und ein für die Kontrolle zuständiger Modul, der eng mit der Anwenderschnittstelle des Planungssystems gekoppelt ist, die ihrerseits durch eine nicht mehr zum eigentlichen System gehörende Schnittstelle erweitert und noch komfortabler gestaltet worden ist.

4.4.1 Wissensbasis

Das Wissen, das SIPE benötigt, um ein für das Erstellen von Plänen notwendiges Welt-Modell aufbauen zu können, liefert die statische Wissensbasis. Alle objekt-beschreibenden Attribute - Eigenschaften, Relationen und Constraints -, sowie sämtliche Operatoren und noch nicht instantiierte Planvariablen liegen in Form einer Objekt-Hierarchie vor, in der das Vererbungsprinzip gilt. Der andere Teil der statischen Wissensbasis stellt Kontrollwissen bereit. Der Suchalgorithmus zum Auffinden von Instantiierungswerten, die den gestellten Bedingungen genügen, ist nicht optimal, da dabei auf kein domänen-spezifisches Wissen zugegriffen wird. Heuristiken werden zum Lösen von Konfliktsituationen eingesetzt und sind aus Effizienzgründen größtenteils domänen-spezifisch.

4.4.2 Inferenzmaschine

Die Komponente des Planungssystems, die die Schlußfolgerungen durchführt an Hand deklarativen Wissens, ist die Inferenzmaschine. Dabei handelt es sich um ein Modul des Planers, das z.B. dafür zuständig ist, in einer bestimmten Situation festzustellen, welcher Operator anwendbar ist, indem die Erfüllung der Vorbedingungen gezeigt oder widerlegt wird. Auch beim Ableiten von sich aus der Anwendung von Operatoren ergebenden Seiteneffekten mit Hilfe deduktiver Operatoren tritt der Inferenzmechanismus in Kraft.

4.4.3 Benutzerschnittstelle

SIPE ermöglicht dem Benutzer, interaktiv mit dem System zu arbeiten. Während der Planerstellung können von Seiten des Benutzers Planoperatoren aktiviert oder für den Planungsprozeß relevante Informationen eingegeben werden. Außerdem ist der Benutzer in der Lage, die Ausführung von Plänen zu überwachen und gegebenenfalls zu korrigieren bzw. eine Korrektur in die Wege zu leiten.

Ziel war es, mit SIPE einen interaktiven Planer zu entwikkeln, der eine Art der Darstellung verwendet, die eine verständliche Kommunikation mit dem Benutzer erlaubt. Kommandos können soweit abstrahiert und dennoch detailliert eingegeben werden, daß der Benutzer bald das System zu bedienen vermag, ohne über tiefere Kenntnisse über das "Funktionieren" verfügen zu müssen. Auch die Ausgabegrössen des Planungssystems müssen so formuliert sein, daß das Erstellen eines Planes zu einem einigermaßen transparenten Prozeß wird.

Um die Eingaben aus Benutzersicht möglichst einfach unter Zuhilfenahme von graphischen Elementen zu gestalten, wurde speziell für SIPE eine Anwenderschnittstelle implementiert, die sich über das eigentliche System legt und über die der Benutzer interaktiv mit dem Planer arbeitet. Der Benutzer kann sich wahlweise per Menütechnik oder über eine Tastatur per Namensangabe auf Planschritte (Knoten im Plan-Netzwerk) beziehen. Überdies erlaubt die Schnittstelle eine graphische Darstellung des von SIPE erstellten Planes bzw. von Teilplänen und die graphische Repräsentation der aktuellen Domänenkonfiguration oder der sich durch Ausführung von einzelnen Planschritten ergebenden Lokalisationen der Domänen-Objekte.

4.5 Kontrollstruktur

Oftmals sind solche Planungssysteme, die zur Lösung von Problemen aus einem speziellen Anwendungsbereich entworfen wurden, bezüglich ihrer Kontrollstruktur und der zur Verfügung stehenden Strategien so stark auf die spezielle Domäne fixiert, daß sie zur Planerstellung in anderen Problembereichen nicht eingesetzt werden können.

Um nicht für jede neue Domäne ein neues Planungssystem mit bereichsspezifischen Inferenzmechanismen entwickeln zu müssen, geht der Forschungstrend dahin, allgemeine Planer wie SIPE zu implementieren, die sowohl domänenunabhängige Techniken zur Planerstellung, als auch

domänen-spezifische Methoden der Repräsentation von Wissen und Heuristiken zur Verfügung stellen.

Die Suchstrategie, nach der im Rahmen von Planverfeinerungen neue Planschritte bestimmt werden, basiert auf einer simplen Tiefensuche innerhalb der gegebenen Objekt- bzw. der aufgebauten Plan-Hierarchie. Dabei wird kein strategisches oder problem-spezifisches Wissen eingesetzt, wodurch sich die automatische Suche nicht als besonders effizient erweist.

SIPE wurde ganz gezielt unter dem Gesichtspunkt entwickelt, den Benutzer in die Planerstellung miteinzubeziehen, interaktives Planen zu ermöglichen. Auf beliebigem Abstraktionslevel kann der Benutzer Planoperationen aktivieren, die daraufhin vom System durchgeführt werden und das Aufbauen von "Plan-Inseln" unterstützen, welche im Fortschreiten des Planens zusammengefügt werden. Vom Benutzer während der Planerstellung gegebene Kommandos, d.h. Aktivierungen von Planoperatoren, können etwa lauten: "Expandiere Knoten 32 mit dem PUTON-Operator", "Expandiere den gesamten bisher erstellten Plan um eine Ebene" oder "Bestimme und korrigiere störende Interaktionen".

4.5.1 Alternative Planäste

Eine weitere die interaktive Planung unterstützende Technik basiert auf der Einführung sogenannter Wahl-Knoten innerhalb des Plan-Netzwerkes. Derartige Knoten tauchen an solchen Stellen im Plan auf, an denen mehrere Alternativen zur weiteren Planexpandierung möglich sind. Constraints für Variablenwerte drücken einschränkende Bedingungen relativ zu solchen Wahl-Knoten aus. Der Benutzer kann die Auswirkungen von alternativen Vorgehensweisen nebeneinander durch Spezifizierung der unterschiedlichen Planpfade mit Hilfe von Wahl-Knoten untersuchen. Dieselben Variablen können in unterschiedlichen Kontexten mit unterschiedlichen Constraints verknüpft werden. Der Planungsfokus kann während der Planung zwischen verschiedenen Alternativen umgeschaltet werden. Dies ist bei solchen Planungssystemen, die nur Backtracking einsetzen, nicht möglich.

4.5.2 Meta-Planen

Die Idee des Meta-Planens besteht darin, Entscheidungen, die den Planungsprozeß selbst betreffen, nach demselben Schema (Erfüllen von Constraints, Beachtung bestimmter Kriterien, Operator-Spezifizierung und -Aktivierung ...) zu fällen wie solche, die auf der eigentlichen Problemebene eine Rolle spielen. Mit einer einzigen System-Architektur wird geplant, (1) welche strategischen Entscheidungen darüber getroffen werden, nach welchem abstrakten Muster (weiter-)geplant werden soll, und (2) wie die Planung, die auszuführenden Aktionen im aktuellen Problembereich aussehen soll. Die strategischen Überlegungen sind weitestgehend domänen-unabhängig.

Bei SIPE wurde dieser Planungsansatz in Anbetracht der einfachen zugrundeliegenden Kontrollstruktur nur beschränkt realisisert. Die Operatoren auf abstraktester Ebene können insofern als Meta-Operatoren aufgefaßt werden, daß sie strategie-spezifisches Wissen in ihrer Beschreibung bereitstellen und so in gewissem Sinne mit SIPE geplant werden kann, wie ein Plan entwickelt werden soll. Die Sprache, in der Domänenwissen von SIPE formuliert ist, besitzt nicht die Reichhaltigkeit, um sämtliches, für das Meta-Planen relevante Wissen über die "Eigenheiten" eines Planes auszudrücken. In SIPE liegen die Schwerpunkte auf anderen Plantechniken und -hilfsmitteln.

4.5.3 Planausführungs-Überwachung

Führt man Pläne in Bereichen der realen Welt aus, bleiben Ausführungsfehler und andere unerwartete Effekte in der Regel nicht aus. Dieses Problem kann durch Bereitstellung von ausgefeilten Techniken zur Überwachung von Planausführungen und durch verbesserte Neu- bzw. Umplanungsstrategien angegangen werden.

Die Überwachung der Planausführung gestattet es, die im Arbeitsbereich tatsächlich eingetretenen Veränderungen mit den im Plan erwarteten Effekten zu vergleichen. Es erweist sich als vorteilhaft, Planerstellung und -ausführung miteinander zu kombinieren: zunächst Erstellung eines partiellen Planes für eine oder wenige Aktionen, dann Ausführung dieses Teilplanes unter Kontrolle, dann eventuell Umplanung oder das Vornehmen von Modifikationen, schließlich Eingliederung des Planstücks in den Gesamtplan und Planung der nächsten Aktionen. Während des Planungsprozesses können immer wieder Änderungen durchgeführt und daraufhin der Plan

gleich auf den neuesten Stand gebracht werden, um mit unerwarteten Effekten fertig zu werden.

SIPE unterstützt diese Vorgehensweise durch die Fähigkeiten, den Sinn und Zweck eines jeden Planteils (= Knoten im Plan-Netzwerk) angeben und aus explizit angeführten Fakten weitere Schlußfolgerungen (Seiteneffekte) ziehen zu können.

Der Benutzer kann das Planungssystem an beliebiger Stelle im Plan darüber in Kenntnis setzen, daß eine Bedingung erfüllt worden ist, falls er diese Beobachtung gemacht hat, obwohl SIPE möglicherweise davon ausgegangen ist, daß die Erfüllung der Bedingung erst in folgenden Planschritten erfolgt. Daraufhin wird automatisch der gesamte Plan nach Ziel-Knoten durchsucht, welche die betrachtete Bedingung beeinflußt, und unter Umständen modifiziert.

Probleme, die durch unerwartete Effekte verursacht werden, können an Hand der durch das System gegebenen Information, welche Aufgaben die einzelnen Plan-Knoten zu erfüllen haben, identifiziert und daraus resultierende, notwendige Änderungen an den entsprechenden Stellen im Plan vorgenommen werden. SIPE schlägt dazu Lösungen vor, wobei vor allem drei Möglichkeiten in Frage kommen:

(1) Instantiierung einer Variablen mit einem anderen, neuen Wert.
(2) Bestimmung eines Operators, um ein nicht mehr erfülltes Ziel wieder zu erreichen, und korrekte Eingliederung des daraus resultierenden Teilplanes in den ursprünglichen Gesamtplan.
(3) In schwerwiegenden Fällen: Bestimmung einer höheren Abstraktionsebene, von der aus eine Neu- bzw. Umplanung erfolgen soll.

Die Durchführung der angegebenen Lösungsvorschläge bringt meist aufwendige Planarbeit mit sich. Beim Neu- bzw. Umplanen sollte darauf geachtet werden, daß möglichst viele Teile vom ursprünglichen Plan übernommen werden können. Das richtige "Einbetten" eines neuen Teilplanes in den Gesamtplan bedarf in gleichem Maße einer darauffolgenden Analyse des ganzen Planes nach möglichen, aus der Änderung sich ergebenden Wirkungen wie das Binden von Variablen an andere Werte.

4.6 Beurteilung von SIPE

SIPE hat bisher für vier verschiedene Domänen korrekte Pläne erstellt (Blockwelt, Kochen, Abflugplanung für Flughafen, Reiseplanung). Zum Beispiel in der Koch-Domäne erstellte Pläne erwiesen sich als effizient, da viele Aktionen (Kochen von Gerichten) parallel unter möglichst optimaler Ausnutzung der zur Verfügung stehenden Ressourcen (z.B. Kochplatten) geplant wurden. In den anderen Domänen, z.B. in der Blockwelt-Domäne, verliehen die Verwendung von Constraints und Deklarierung von Ressourcen dem Planungssystem die Fähigkeit, Interaktionen zwischen Teilproblemen recht schnell zu entdecken und zu korrigieren.

Wie den vorausgegangenen Ausführungen zu entnehmen ist, sind die Repräsentationsmöglichkeiten von planungsrelevantem Wissen bei SIPE sehr vielfältig und unterstützen effizientes Planen. Eine ausdrucksstarke Operatorbeschreibungssprache bietet die Voraussetzung für die Erstellung verständlicher Pläne. Durch die Bereitstellung deduktiver Operatoren können als Operator-Attribute "Haupt"- und Seiteneffekte dargestellt werden. Des weiteren können die Beschreibungen abstrakter Operatoren strategie-spezifische Informationen beinhalten. Die Verwendung von Ressourcen bildet ein leistungsfähiges Werkzeug, die Auswirkungen von Aktionen und das Verhalten von Objekten zu repräsentieren.

Als weiterer Schritt in Richtung Domänen-Unabhängigkeit steht bei SIPE eine Menge allgemeiner, explizit formulierter Constraints zur Verfügung, die für unterschiedliche Domänen verwendet werden können. Im Vergleich dazu werden z.B. bei MOLGEN die möglichen Constraint-Formulierungen nicht standardmäßig vorgegeben. MOLGENs Constraints zeigen eher wieder domänenspezifischen Charakter.

Dadurch, daß an den beteiligten Ressourcen mögliche Konflikte bereits vor Anwendung der Operatoren erkannt und dementsprechend durch die Verwendung von Constraints konflikt-meidende Variableninstantiierungen zu erreichen versucht werden, stellt SIPE ein Planungssystem dar, das in besonderem Maße Anhängigkeiten zwischen Teilplänen zu bewältigen verspricht. Dennoch bedarf es aus Effizienzgründen des Einsatzes domänen-spezifischer Heuristiken zur Lösung domänen-eigener Probleme.

Während der Planerstellung und -ausführung bekommt der Benutzer die Möglichkeit, in das Geschehen einzugreifen, den Ablauf zu beeinflussen. Auf diese Weise kann der Benutzer den Planungsprozeß verfolgen und gegebenenfalls steuern. SIPE zeigt an unterschiedlichen Stellen interaktives Verhalten: in Konfliktsituationen kann der Benutzer zwischen verschiedenen Heuristiken oder Lösungsvorschlägen wählen, als Kommandos können Planoperationen direkt eingegeben werden. Die Möglichkeiten bei SIPE zur Ausführungs-Überwachung und Umplanung sind schon recht fortgeschritten, sind aber immer noch verbesserungswürdig.

Bei der Entwicklung vorausgegangener Planungssysteme gesammelte Erfahrungen wurden bei SIPE verwertet. Die Vorgehensweise des hierarchischen Planens erwies sich als vorteilhaft, um zusätzliche Probleme zu vermeiden, die sich durch zu viele Detailfakten auf "unterer", konkreter Ebene gleich zu Beginn der Planerstellung ergeben können. Die Idee des Meta-Planens wurde nur beschränkt realisiert. Statt dessen wurde ein Ansatz gewählt, der in Richtung interaktives Planen geht. Das Ziel, ein Planungssystem zu entwickeln, das ohne jegliche Zusatzinformationen eine gestellte Aufgabe mit Hilfe des in einer Wissensbasis verfügbaren Domänenwissens lösen sollte, stand nicht mehr im Vordergrund. Vielmehr war ein domänen-unabhängiger Planer angestrebt, der unter Zuhilfenahme zusätzlicher Benutzereingaben während des Planungsprozesses solche Pläne erstellen sollte, die effizient, knapp formuliert, leicht verständlich und den Anforderungen entsprechend sind und auch für Anwendungen in der realen Welt eingesetzt werden können.

Letzteres, das Erstellen von Plänen für Anwendungen in der realen Welt, ist bei SIPE noch nicht möglich. Dieses System erweist sich als noch nicht robust genug, um in komplexeren Domänen zufriedenstellend zur Lösung von Problemen Pläne zu generieren. Mögliche Verbesserungsvorschläge bestünden darin, (a) die verwendeten Suchstrategien zu optimieren, z.B. durch den Einsatz von Wissen, (b) Erweiterungen zum Ressourcen-Ansatz derart zu implementieren, daß eine Berücksichtigung von anderen Ressourcen, z.B. Geld und Computerleistungen, im Plan erfolgen kann, (c) SIPEs Operatorbeschreibungssprache dahingehend zu erweitern, daß Aktionen für vorhersehbare Fehler in die Beschreibung eines Operators miteinbezogen werden können.

Die Repräsentation im Plan der Größe Zeit erfolgt nicht explizit, sondern nur implizit und in unbefriedigendem Maße durch Reihenfolgefestlegung.

Alles in allem handelt es sich bei SIPE um einen Planer, der sich bereits in einfacheren Domänen bewährt hat und neue Techniken einsetzt, die prinzipiell auch in komplexeren Anwendungsgebieten zufriedenstellende Ergebnisse liefern könnten. Gute Ansätze sind vorhanden, bedürfen jedoch noch einer Verbesserung und Ausreifung.

Kapitel 5 : Planungssystem TWEAK

TWEAK ist ein nichtlineares Planungsprogramm, dessen Vorteile gegenüber früher erstellten derartigen Planungssystemen in einem einfachen und verständlichen, der Planungstätigkeit zugrundeliegenden Algorithmus und in der Erstellung von Plänen liegen, die einen klaren Aufbau besitzen, nachvollziehbar und mit Hilfe von durch Bedingungen spezifizierten Aktionen präzise formuliert sind.

Laut seinem Verfasser David Chapman [2] stellt TWEAK eine strikte mathematische Rekonstruktion früherer, nichtlinearer, domänen-unabhängiger Planer wie z.B. SIPE dar. Es setzt "intelligentes" Backtracking ein, falls während der Planerstellung zum Erreichen eines (Teil-) Ziels keine geeigneten Aktionen oder Planschritte spezifiziert, die Ausführungsreihenfolge ungünstig festgelegt oder unangemessene Variableninstantiierungen vorgenommen worden sind. Chapman beweist sogar die Behauptung, daß TWEAK korrekt und vollständig arbeitet, d.h. falls für ein Problem eine Lösung existiert, so wird diese von TWEAK bestimmt.

5.1 In TWEAK verwendete Planungstechniken

Ein nichtlinearer, domänen-unabhängiger Planer wie TWEAK zeichnet sich dadurch aus, daß er nicht auf einen speziellen Anwendungsbereich fixiert ist und die von ihm automatisch generierten Pläne die Reihenfolge der durchzuführenden Schritte unter Umständen nur partiell festlegen.

Falls bei der Planerstellung zur Lösung eines Problems gleichzeitig mehrere Ziele zu erreichen sind, was in der Regel der Fall ist, bereiten die meistens zwischen den einzelnen Teillösungen auftretenden Interaktionen Schwierigkeiten. Wie auch bei den bisher betrachteten Planungssystemen bilden Constraints bei TWEAK die Voraussetzung zum nichtlinearen Planen. Constraints werden dynamisch während des Planungsprozesses formuliert, um Reihenfolgefestlegungen zu treffen oder Beziehungen zwischen Objekten auszudrücken. Dadurch können u.a. Variableninstantiierungen verzögert werden. Je

später Entwurfsentscheidungen getroffen werden, desto länger bestehen Möglichkeiten zur Planoptimierung.

Die zentrale Planerstellungsmethode bei TWEAK besteht im einstufigen Planen als Suche. Die Beschreibung des Problembereichs wird in verschiedene Situationen unterteilt. Die Aufgabenstellung ist derart spezifiziert, daß aus einem anfänglichen Zustand der Welt (Ausgangssituation) ein modifizierter Zustand (Finalsituation) erreicht werden soll, in dem dann bestimmte Bedingungen (Ziele) erfüllt sind. Durch die Anwendung eines Operators wird die aktuelle Situation verändert. Ziel ist es, eine Sequenz von Operationen zu bestimmen, die eine korrekte Transformation von der Ausgangs- zur Finalsituation ermöglichen. Die eigentliche Planungstätigkeit besteht also darin, einen Weg von der Start- zu einer Zielsituation zu suchen.

Eine für das Planen sinnvolle Suchstrategie, die auch bei TWEAK Anwendung findet, ist die Rückwärtssuche, d.h. die Suche vom Ziel zum Start. Der Grund dafür liegt darin, daß die Ziel- oder Finalsituation im allgemeinen nicht vollständig beschrieben ist; z.B.: die Zielbeschreibung eines Blockwelt-Problems laute (on a b) & (on b c) (siehe auch Abschnitt 5.5), über die Position möglicher anderer Blöcke im Problemraum ist nichts ausgesagt, lediglich diese beiden Bedingungen (Ziele) müssen erfüllt sein. Eine derartige Zielbeschreibung repräsentiert eine ganze Klasse von Situationen. Eine Start- oder Ausgangssituation hingegen ist meist vollständig und eindeutig spezifiziert. Der Planungsprozeß stellt sich folglich als Suche nach Aktionen dar, die eine eindeutig bestimmte Startsituation in eine Klasse von Zielsituationen überführen.

Das Kriterium für die Suche nach geeigneten Aktionen bildet eine Menge von Zielen, die es zu erreichen gilt. Dazu zählen nicht nur die eigentlichen Planziele, die durch die Zielsituation formuliert sind, sondern auch sich während der Planerstellung ergebende Teilziele, die etwa dergestalt sein können, Vorbedingungen für nachfolgend anzuwendende Operatoren zu erfüllen. Auf einer übergeordneten Kontrollebene wird jeweils ein Ziel ausgewählt. Dementsprechend werden daraufhin Planschritte zusammengestellt. Dieser Vorgang wiederholt sich solange, bis sämtliche Ziele berücksichtigt sind.

Der sukzessive Aufbau einer Folge von atomaren Operatoren zur Lösung eines gegebenen Problems erfolgt in der Art, daß gemäß den Planzielen solche Aktionen bestimmt werden, in deren Nachbedingungen unter anderen eben diese Planziele erscheinen. Meist sind dann noch nicht die Vorbedingungen erfüllt, was zur Anwendung eines Operators letztendlich notwendig ist. Bei anderen Planern wie z.B. SIPE dienen die Vorbedingungen als

Auswahlkriterien, welcher Operator in einer bestimmten Situation angewendet werden soll und kann. Sind bei einem Operator die Vorbedingungen nicht erfüllt, so wird bei SIPE nicht durch Planmodifikationen die Eingangssituation für den Operator anzupassen versucht, sondern ein anderer Operator untersucht. Anders bei TWEAK: hier erfolgt die Operatorauswahl entsprechend den durch die Nachbedingungen implizit gegebenen Effekten. Sind die durch die Problemstellung beschriebenen Planziele "von hinten her" (vom Ziel her) erreicht, müssen im Fortschreiten des Planens die noch nicht erfüllten Vorbedingungen erfüllt werden. Diese Vorbedingungen bilden dann die neuen zu erreichenden Ziele, die nach und nach auf höherer Ebene ausgewählt werden. Mit jeder Einführung eines neuen Operators wiederholt sich dieser Vorgang, bis sämtliche Vorbedingungen erfüllt sind und der Plan somit erstellt ist, da bereits von vorneherein die Aktionen so ausgewählt worden sind, daß die Planziele bei korrekter Ausführung erreicht werden.

Die Realisierung der Idee, bei TWEAK über die eigentliche Planungsebene eine Meta-Ebene zu spannen, um strategische Entscheidungen bezüglich des Planungsverlaufes fällen zu können (wie z.B. bei MOLGEN), scheitert an den beschränkten Repräsentationsmöglichkeiten von TWEAK.

5.2 Repräsentationsmöglichkeiten von TWEAK

In diesem Abschnitt soll u.a. gezeigt werden, wie TWEAK Constraints dazu einsetzt, das umfassende Domänenwissen besser in den Griff zu bekommen. Wie letzteres, das Domänenwissen (z.B. Objekteigenschaften), genau repräsentiert wird, ist aus den Ausführungen von TWEAKs Entwickler nicht ersichtlich. Die Schwerpunkte liegen darauf, wie der Plan an sich und der Problembereich als Ganzes beschrieben werden.

5.2.1 Verwendung von Constraints

Um die Idee des nichtlinearen Planens realisieren zu können, macht TWEAK regen Gebrauch von der Technik des Setzens von Constraints. Das Erstellen eines Planes sieht so aus, daß mit dem Fortschreiten des Planens immer mehr partielle Beschreibungen durch Constraints spezifiziert werden, denen die einzelnen Planteile genügen müssen. Dadurch müssen die Eigenschaften der im Plan benutzten Objekte - als Variablen repräsentiert - nicht gleich zu Beginn der Planerstellung festgelegt

werden. Mögliche Optionen können solange offen gehalten werden, bis eine ausreichend fundierte Entscheidung getroffen werden kann.

Da nicht alle möglichen Planspezifikationen bzw. -vervollständigungen auf eine Zielerfüllung hin untersucht werden können, müssen im Laufe des Planungsprozesses auf heuristischer Basis gewisse Entscheidungen getroffen werden. Erweist sich eine Festlegung im weiteren als ungünstig, setzt TWEAK Backtracking ein, um eine erfolgreiche Fortsetzung der Planerstellung zu gewährleisten. Die Verwendung von Constraints trägt dazu bei, die Zahl der Fälle, in denen Backtracking notwendig wird, zu reduzieren. Je mehr Constraints an die Variablen (--> Werte) und Operatoren (--> Reihenfolge) eines Planes gebunden werden, desto geringer wird die Zahl der vollständigen Pläne, die durch den noch unfertigen Plan repräsentiert werden.

Bis zum Abschluß der Planerstellung arbeitet TWEAK an einem in zweierlei Hinsicht unvollständigen Plan: zum einen ist die Reihenfolge, in denen die Planschritte ausgeführt werden sollen, nur teilweise festgelegt, zum andern sind die Planschritte selbst nicht vollständig spezifiziert. In jedem Planungsstadium repräsentiert der Planentwurf oder unvollständige Plan eine mehr (zu Beginn des Planungsprozesses) oder weniger (gegen Ende des Planungsprozesses) große Menge verschiedener "fertiger" Pläne. Während der Planung formulierte und an Planvariablen gebundene Constraints legen mehr und mehr fest, wie der noch unvollständige Plan konkretisiert werden soll, bis schließlich als Ergebnis ein Plan vorliegt, dessen Ausführung das gegebene Problem löst.

Allerdings hat das Setzen von Constraints auch seine Tükken. In ungünstigen Fällen können inkonsistente Constraints erzeugt werden. Dies hat dann zur Folge, daß ein Planschritt überbeschränkt wird, d.h. kein Wert kann die gestellten Bedingungen mehr erfüllen; die Planerstellung gelangt in eine Sackgasse, und Backtracking muß eingesetzt werden, um gesetzte Einschränkungen rückgängig zu machen und den Planungsprozeß an einer erfolgversprechenderen Stelle wieder fortzusetzen.

5.2.2 Repräsentation des Planes

Die Repräsentation eines Planes ist bei TWEAK an derartige Einschränkungen gebunden - Gründe dafür siehe Abschnitt 5.6 -, daß sie für die meisten Domänen zu wenig ausdrucksstark ist. Ein vollständiger Plan stellt eine (partiell) geordnete Sequenz von Planschritten dar, wobei die Reihenfolge den Zeitpunkt der Ausführung festlegt und die Planschritte die durchzuführenden Aktionen repräsentieren. Solche Planschritte werden durch Vor- und Nachbedingungen (kurz: Ausführungsbedingungen) näher beschrieben. Es wird jeweils angegeben, was vor Ausführung eines Planschrittes gelten muß (Vorbedingungen) und welche Bedingungen nach dessen Ausführung erfüllt sind (Nachbedingungen). Diese Ausführungsbedingungen werden mit Hilfe von Variablen und /oder Konstanten auf einfache Weise, d.h. ohne Verwendung von Funktionen, Operatoren (außer Negationsoperator) oder Quantoren, formuliert.

5.2.3 Beschreibung des Problembereichs

Indem Planschritte ausgeführt werden, ändert sich jeweils der aktuelle Zustand der Welt. Die Beschreibung des Problembereichs, der Welt, unterteilt sich bei TWEAK in verschiedene Situationen. Eine Situation stellt eine Menge von Ausführungsbedingungen dar. Diejenige Situation, die den Zustand der Welt unmittelbar vor Ausführung des Planes beschreibt, bildet die Ausgangs- oder Startsituation. Entsprechend versteht man unter der Final- oder Zielsituation den Zustand der Welt nach Ausführung des Planes, genauer gesagt eine Teil-beschreibung davon. Die Menge der Bedingungen, die direkt vor Ausführung eines Planschrittes erfüllt sind, ist die Eingabesituation der betreffenden Aktion. Die nach Ausführung eines Planschrittes geltenden Bedingungen beschreiben entsprechend die Ausgabe-situation der Aktion. In einem vollständig spezifizierten Plan gilt das Pipelining-Prinzip [16]: die Eingabesituation eines Planschrittes ist identisch mit der Ausgabesituation des vorausgehenden Planschrit-tes.

Ein Planschritt kann nur dann ausgeführt werden, wenn seine sämtlichen Vorbedingungen in der Eingabesituation, in der er angewendet werden soll, erfüllt sind. Ist dies der Fall, dann ist die Transformation von der Eingabe- zur Ausgabesituation dergestalt, daß gewisse Ausfüh-rungsbedingungen, die in der Eingabesituation noch erfüllt waren, den Wahrheitswert FALSCH bekommen und

andere Bedingungen, die in der Eingabesituation noch nicht galten, in der Ausgabesituation erfüllt sind; alle übrigen Ausführungsbedingungen werden von der Eingabe- zur Ausgabesituation übernommen (--> Ansatz zur Lösung des Frame-Problems [9]).

Als nachteilig bei dieser Art der Repräsentation von Planungsoperationen erweist sich der Sachverhalt, daß sämtliche von der Ausführung eines Planschrittes ausgehenden Effekte explizit angegeben werden müssen, eine in komplexeren Domänen sehr aufwendige oder gar unlösbare Aufgabe. Bei anderen Planungssystemen wie z.B. SIPE ist die implizite Formulierung von Seiteneffekten durch die Verwendung deduktiver Operatoren möglich. Bei TWEAK müssen alle Änderungen in der Welt mit Hilfe von Nachbedingungen spezifiziert werden.

Ein Problem ist durch eine Start- und eine Zielsituation gegeben, beides Mengen von Bedingungen. Zur Lösung eines Problems, d.h. zur Transformation von der gegebenen Start- oder Ausgangssituation zur Ziel- oder Finalsituation, soll TWEAK einen Plan erstellen. Ein Ziel stellt dabei eine Bedingung dar, die es in einer bestimmten Situation zu erfüllen gilt. Die Finalsituation eines Planes spezifiziert sämtliche Ziele, die nach Ausführung des letzten Planschrittes erreicht sein müssen, wenn der Plan die Lösung des Problems herbeiführen soll. Dabei wird nicht gefordert, daß TWEAK einen eindeutigen, vollständig spezifizierten Plan erstellen muß. Liefert der Planungsprozeß nur einen unvollständigen Plan, der mehrere mögliche fertige Pläne repräsentiert, muß allerdings gewährleistet sein - will man von einem korrekten Ergebnis der Planerstellung mit TWEAK sprechen -, daß die Ausführung eines jeden Repräsentanten der durch den unvollständigen Plan spezifizierten Klasse von Plänen zur Lösung des Problems führt.

5.3 Architektur

In [2] ist die Architektur von TWEAK zwar nicht explizit beschrieben, auf Grund der Wirkungsweise des Planers lassen sich aber gewisse Planungsmoduln identifizieren und der grobe Aufbau des Systems erkennen.

5.3.1 Wissensbasis

Wie SIPE so stellt auch TWEAK ein domänen-unabhängiges Planungssystem dar, das erst durch Hinzugabe von speziellem Bereichswissen zu einem einsatzfähigen Planer wird. Die Wissensbasis stellt Daten bereit, um den Problembereich modellieren zu können. Zu diesem Wissen zählen nicht nur objekt-spezifizierende Grössen, sondern auch "Operator-Schablonen", die zur Lösung eines gegebenen Problems während des Planungsprozesses von TWEAK gemäß den zu erfüllenden Bedingungen bzw. zu erreichenden Zielen ausgewählt und in darauffolgenden Schritten näher spezifiziert werden. Mit letzterem, der Operator-Spezifizierung, ist die Bindung der in der Schablone verwendeten Grössen an die aktuellen Planvariablen oder Konstanten und die Reihenfolge-festlegung gemeint.

Bei TWEAKs Strategie, Pläne zu erstellen (Näheres siehe Abschnitt 5.4), bedarf es domänen-spezifischer Heuristiken, um bestimmte Planentscheidungen zu treffen. Entscheidend ist, solche Ziele als zu erreichende Grössen aus der Menge der während der Planung zu erfüllenden Bedingungen auszuwählen, daß effizientes Planen möglich wird, d.h. Erstellung eines möglichst minimalen Planes mit geringem Aufwand. Ferner müssen zur Erfüllung bestimmter Bedingungen die "richtigen" Operatoren angewendet werden, damit die erwarteten Effekte eintreten.

Constraints stellen auch bei TWEAK Wissen dar, das dynamisch während der Planerstellung formuliert wird.

5.3.2 Planungsmoduln

Einer von mehreren Planungsmoduln bei TWEAK ist als übergeordnete Instanz über dem eigentlichen Problembereich dafür zuständig, die in den verschiedenen Planstadien für die Festlegung weiterer oder Modifizierung bestehender Planschritte entscheidenden Ziele auszuwählen. Als Schleife überspannt diese "Modultätigkeit" den gesamten Planungsprozeß. Ein voll-ständiger Plan ist erstellt, sobald sämtliche Ziele erfüllt sind.

Die Aufgabe eines weiteren Planungsmoduls besteht darin, Planmodifikationen durchzuführen, wann immer es als nötig erscheint. Falls (1) durch das Setzen zueinander inkonsistenter Constraints ein Planschritt, welcher immer eine Aktion repräsentiert, überbeschränkt wird oder falls (2) es der Umordnung im Plan bereits

bestehender Operatoren bedarf, um bestimmte noch unerfüllte Ziele zu erreichen, muß der bis dahin erstellte Plan korrigiert, umgebaut oder erweitert werden. In dem erst genannten Fall (1) ist es notwendig, Backtracking einzusetzen, um bereits getroffene Entscheidungen wieder rückgängig zu machen. Im zweiten Fall (2) müssen mit Hilfe von Constraints eine andere Reihenfolgefestlegung getroffen und sich daraus ergebende Auswirkungen auf den Plan erkannt werden. Die Notwendigkeit nach weiteren korrigierenden Maßnahmen (z.B. Anwendung neuer Operatoren) kann die Folge sein. Dies sind nur zwei Beispiele für mögliche Situationen, in denen Planmodifikationen eingeleitet werden müssen.

Die Funktion des Planungsmoduls, der die eigentliche Planungstätigkeit ausübt, wird im folgenden Abschnitt beschrieben. Es geht darum, den richitgen Weg von der Start- zur Zielsituation eines Problems zu finden. TWEAK erkennt die noch nicht erfüllten Bedingungen im Plan und geht systematisch vor, dagegen für Abhilfe zu sorgen.

5.4 Kontrollstruktur

TWEAK wird wie andere Planungssysteme dafür eingesetzt, einen Plan zur Lösung für ein spezielles gegebenes Problem zu erstellen. Dazu werden Mechanismen bereitgestellt, die domänen-unabhängiges Planen ermöglichen. Die Wissensbasis verleiht dem System die für eine spezielle Domäne relevanten Informationen, welche in erster Linie objekt- und nicht plan-spezifischer (--> Strategiewissen) Natur sind. TWEAKs Kontrollstruktur besitzt hauptsächlich eindimensionalen Charakter. Die meisten Entscheidungen werden auf einer Ebene, der Ebene des Problembereichs getroffen. Lediglich die Auswahl zu erfüllender Planziele ist dem eigentlichen Planungsprozeß übergeordnet.

5.4.1 Ein Weg von der Start- zur Zielsituation

TWEAK arbeitet im Rahmen des Planungsprozesses an einem unvollständigen Plan, einem "Plan-Gerüst", und versucht, Schritt für Schritt die Zahl der daraus resultierenden möglichen fertigen Pläne durch Detaillierungsangaben und Einführung neuer Planschritte zu reduzieren. Von Interesse sind nur solche Pläne, die die Startsituation des gegebenen Problems in eine solche Situation transformieren, welche die in der Zielsituation des Problems spezifizierten Bedingungen als Teilmenge enthält.

Zu Beginn der Planerstellung liegt ein erster unvollständiger Plan vor, dessen Startsituation identisch ist mit der durch das Problem gegebenen. Zu diesem Zeitpunkt sind noch keinerlei Planschritte oder Constraints formuliert. Das Fortschreiten des Planungsprozesses gestaltet sich dann so, daß in einer der eigentlichen Planungsaktivität übergeordneten Schleife immer wieder ein Ziel ausgewählt wird, das die Planerstellung in Richtung Erfüllung des jeweiligen Zieles unter Beachtung bereits erreichter Ziele und gesetzter Constraints lenkt. Dieser Vorgang wiederholt sich solange, bis alle Ziele erreicht und somit das Problem gelöst ist.

Die Suchstrategie dafür, welches Ziel in einer bestimmten Situation als nächstes angegangen und welche Schritte daraufhin zum Erreichen des Ziels festgelegt werden sollen, basiert auf einer abhängigkeits-gerichteten Breitensuche. Dieser Ansatz erweist sich gegenüber der chronologisch orientierten Vorgehensweise bei solchen Anwendungen als vorteilhaft, bei denen der Suchraum nahezu in voneinander unabhängige Teilräume zerlegt werden kann, so daß ein Scheitern in einem Planteil das Rückgängigmachen von nur in diesem Teil getätigter Planarbeit erforderlich macht und andere Planteile unbetroffen bleiben. Dadurch gestaltet sich die Durchführung von Backtracking im Falle einer ungünstig gewählten Festlegung als sehr effizient, "intelligentes" Backtracking wird ermöglicht. Allerdings ist die Voraussetzung für ein sinnvolles Arbeiten nach diesem Ansatz, nämlich die Zerlegbarkeit des Suchraums in möglichst unabhängige Teilbereiche, in komplexeren Problembereichen nur selten erfüllt.

Die Bestimmung von Planschritten - jeder Planschritt repräsentiert eine Aktion, die in der Domäne, in der das Problem spezifiziert wird, ausgeführt werden kann - zur Erfüllung eines auf übergeordneter Planebene gewählten Zieles stützt sich auf folgendes "Erfüllungs-Kriterium" (für den Beweis dieses Satzes sei auf die Literatur [2] verwiesen):

Eine Bedingung p ist in einer Situation s genau dann erfüllt, wenn gilt:

(1) In einer Situation t, die entweder mit der Situation s identisch ist oder eine der Situation s im Planungsverlauf vorausgehende Situation bildet, bekommt p den Wahrheitswert WAHR;

und

(2) (a) jeder Planschritt, durch dessen Ausführung die gemäß (1) erfüllte Bedingung p oder eine andere Bedingung q negiert wird, wobei q in Wahrheitswert-Äquivalenzrelation (kurz: Wert-Äquivalenzrelation) zu p steht, d.h. p besitzt den Wahrheitswert FALSCH bzw. WAHR, falls q den Wahrheitswert FALSCH bzw. WAHR besitzt, und umgekehrt, wird erst nach Erreichen der Situation s oder vor Erreichen der Situation t (siehe (1)) angewendet;

oder

(2) b) für jeden vor Erreichen von Situation s und nach Erreichen von Situation t (falls s ungleich t; sonst: Fall 2a) (siehe (1)) angesiedelten Planschritt Z ("Zerstörer-Planschritt") und jede Ausführungsbedingung q, die zu p in Wert-Äquivalenzrelation steht (bea.: p ist zu sich selbst auch wert-äquivalent wegen reflexiver Eigenschaft der Äquivalenzrelation) und die nach Ausführung von Planschritt Z den Wahrheitswert FALSCH besitzt, existiert ein Planschritt W ("Wiedergutmacher-Planschritt"), der im Plan nach Planschritt Z und vor Erreichen der Situation s lokalisiert ist und durch dessen Ausführung die Bedingung r den Wahrheitswert WAHR erhält, wobei für r gilt: r und p stehen in Wert-Äquivalenzrelation zueinander genau dann, wenn p und q zueinander wert-äquivalent sind.

Die Aussage dieses Satzes sei noch einmal in einer kleinen Skizze verdeutlicht (Abb. 5.4.1-1 auf der nächsten Seite), in der Situationen durch Kreise und Planschritte durch Rechtecke dargestellt werden. Durchgezogene Pfeile drücken notwendige, gestrichelte Pfeile mögliche Reihenfolgebeziehungen aus, wobei nur die interessanten Konstellationen aufgezeigt sind.

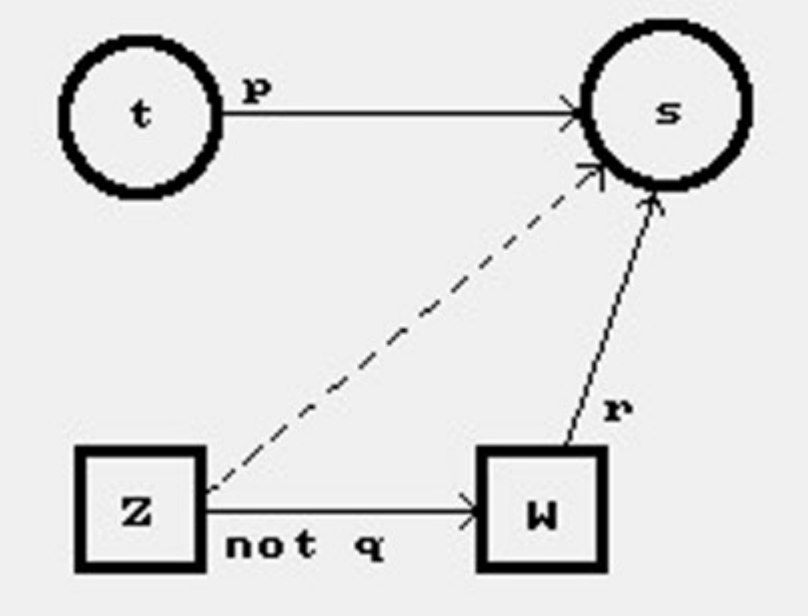


Abb. 5.4.1-1 Das Erfüllungs-Kriterium

Das Erfüllungs-Kriterium zählt alle Möglichkeiten auf, in einer bestimmten Situation (s) den Wahrheitswert WAHR für eine spezielle Ausführungsbedingung (p) (nach an irgendeiner Stelle im Plan aufgetretener Negation) wieder zu erhalten.

Um ein ausgewähltes Planziel zu erreichen, geht TWEAK so vor, auf nichtdeterministische Art und Weise eine der durch das Erfüllungs-Kriterium gegebenen Möglichkeiten aufzugreifen und den Plan entsprechend zu modifizieren. Dabei werden Constraints formuliert und an Variablen-werte gebunden, um gewisse Reihenfolgefestlegungen zu treffen und Beziehungen zwischen Ausführungsbedingungen (z.B. Wert-Äquivalenz) auszudrücken. Werden bei diesem Vorgang inkonsistente Constraints generiert, (z.B. "Situation s vor Situation t" und "Situation t vor Situation s"), erkennt dies das Planungssystem und veranlaßt Backtracking.

Während der Planerstellung können also im Plan bereits existierende Planschritte näher spezifiziert werden, oder es kann notwendig sein, neue Planschritte einzuführen. TWEAKs Strategie unterstützt die erstgenannte Vorgehensweise. Dadurch wird aus einem bestehenden Planentwurf mehr "herausgeholt", und es wird vermieden, unnötig große Pläne zu erzeugen. Zusätzliche, neue Aktionen werden nur dann in die Planung miteinbezogen, wenn es anders nicht mehr geht. Das

Einführen neuer Planschritte bringt neue Bedingungen mit in den Plan, die es zu erfüllen gilt. Darüberhinaus besteht die Gefahr entstehender Interaktionen, die bereits erreichte Ziele wieder beeinflussen könnten. TWEAK wurde konzipiert, knappe Pläne zu erstellen.

5.4.2 Mögliche Planungsausgänge

Bei der Erstellung eines Planes durch TWEAK kann nicht ausgeschlossen werden, daß keine Konvergierung auf einen das gegebene Problem lösenden Plan eintritt. Drei verschiedene Ausgänge der Planung mit TWEAK sind möglich:

(1) Erfolg:	Ein korrekter Plan wurde erstellt.
(2) Abbruch:	Ein korrekter Plan konnte nicht erstellt werden, selbst nachdem sämtliche in Frage kommenden Planmodifikationen ausprobiert wurden.
(3) Nichtterminierung:	Der Plan wächst um immer mehr Planschritte an, ohne dabei auf einen korrekten, das Problem lösenden Plan zu konvergieren.

Jeder dieser drei Ausgänge ist unabhängig von Domäne und Problem bei der Erstellung eines Planes mit TWEAK möglich.

5.5 Beispiel einer Planerstellung mit TWEAK

Um die Arbeitsweise von TWEAK zu illustrieren, sei an dieser Stelle ein Planungsbeispiel aus der Blockwelt gegeben, das bereits in einem früheren Kapitel dieser Arbeit (Kapitel 2) verwendet wurde, um mögliche Interaktionen zwischen Teilplänen aufzuzeigen.

Gegeben sei das folgende Problem, bekannt unter dem Namen "Sussman-Anomalie" (Abb. 5.5-1, Abb. 5.5-2):

Startsituation:

```
(on    c  a)
(clear c )
(clear b )
(on    a  table)
(on    b  table)
```

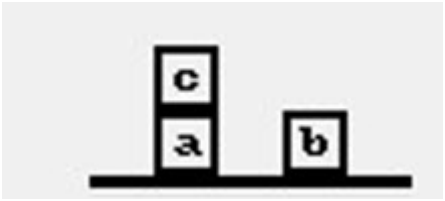


Abb. 5.5-1 Startsituation der Sussman-Anomalie

Erwünschte Zielsituation:

(on a b)
(on b c)

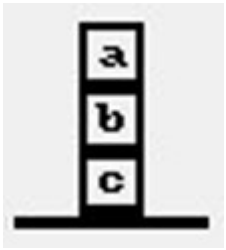


Abb. 5.5-2 Zielsituation der Sussman-Anomalie

Die zur Aufgabenformulierung verwendeten Prädikate haben folgende Bedeutung:

(on x y)	ist wahr genau dann, wenn Block x auf Block y liegt.
(on x table)	ist wahr genau dann, wenn Block x direkt auf dem Tisch liegt.
(clear x)	ist wahr genau dann, wenn kein Block auf Block x liegt.

Als zusätzliche Einschränkung zur Lösung der Aufgabe solle gelten, daß immer nur ein Block auf einmal transportiert werden kann. Ein Block x muß also das Prädikat (clear x) erfüllen, bevor er umpositioniert werden kann.

Nun zur Lösung der Aufgabe mit TWEAK:

Logischerweise ist von vorneherein nur ein Operator, der PUTON-Operator (Abb. 5.5-3), in der Blockwelt-Domäne verfügbar.

Vorbedingungen:

```
(on    x  z)
(clear x )
(clear y )
```

Aktion:

(puton x y)

Nachbedingungen:

```
     (on    x  y)
(not (on    x  z))
(not (clear y ))
     (clear z )
```

Abb. 5.5-3 PUTON-Operator

TWEAK wählt nun nacheinander die beiden Planziele (on a b) und (on b c) aus und stellt zum Erreichen von diesen Planschritte, die Aktionen repräsentieren, zusammen.

Vorgehensweise (nichtdeterministisch):

- Auswahl des Ziels (on a b) auf übergeordneter Kontrollebene.
- Bestimmung des Operators (puton x y) zum Erreichen dieses Ziels.
- Auswahl des Ziels (on b c) auf übergeordneter Kontrollebene.
- Bestimmung des Operators (puton x y) zum Erreichen dieses Ziels.

Daraus resultiert ein erster unvollständiger Plan für die Sussman-Anomalie (Abb. 5.5-4).
(Die nicht erfüllten Vorbedingungen sind im folgenden immer mit einem Stern ('*') markiert !)

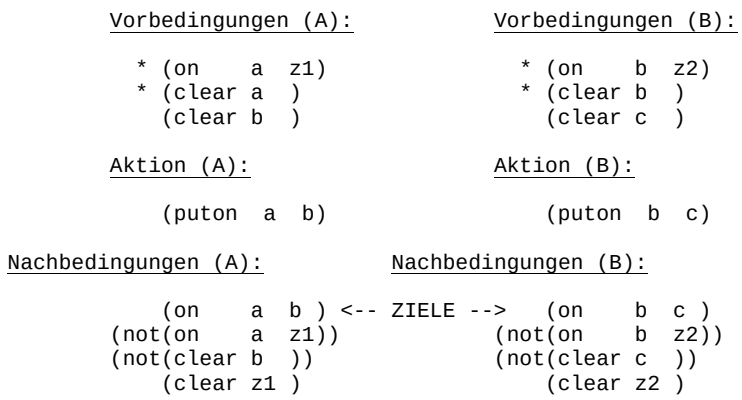


Abb. 5.5-4 Ein erster unvollständiger Plan für die Sussman-Anomalie

Erklärung für den ersten unvollständigen Plan zur Lösung der Sussman-Anomalie (Abb. 5.5-4):

Nach Auswahl des PUTON-Operators zur Erfüllung von Ziel (on a b) (--> Teilplan A) bzw. Ziel (on b c) (--> Teilplan B) müssen die beiden "Operator-Schablonen" (puton x y) für die beiden Teilpläne näher spezifiziert werden. Constraints müssen zur Einschränkung der möglichen Instantiierungswerte für x,y und z (siehe PUTON-Operatorbeschreibung Abb. 5.5-3) formuliert werden: x,y und z müssen an verschiedene Werte gebunden werden, da es sich um verschiedene Objekte handelt. Die Variable x wird an den Wert a (im Teilplan A) bzw. b (im Teilplan B) gebunden, die Variable y an den Wert b bzw. c. Diese Festlegung wurde so gewählt, damit in den Nachbedingungen die zu erreichenden Planziele in der gewünschten Form erscheinen (siehe ZIELE-Vermerk in Abb. 5.5-4).

In der PUTON-Operatorbeschreibung (Abb. 5.5-3) repräsentiert die Variable z den Block, auf welchem x vor Anwendung des PUTON-Operators liegt. In Teilplan A wird diese Variable zunächst mit z1, in Teilplan B mit z2 instantiiert, wobei es sich bei z1 und z2 um noch nicht näher spezifizierte Objekte handelt (in diesem Planungsstadium ist noch nicht bekannt, ob z1 = z2 gilt und mit welchen Werten eine Instantiierung erfolgen könnte; daher dieses Vorgehen).
Die Vorbedingungen für die Aktionen in den beiden Teilplänen sind erst teilweise erfüllt. Weitere Planarbeit ist notwendig.

TWEAKs Bestreben bei der Erstellung eines vollständigen Planes liegt in einer möglichst knappen Formulierung ohne Verwendung unnötiger Planschritte. Sind in einem

noch unvollständigen Plan Ausführungsbedingungen noch nicht erfüllt, versucht der Planer zunächst, Aktionen näher zu spezifizieren und/oder Reihenfolgefestlegungen zu treffen, bevor neue Planschritte eingeführt werden.

Vorbedingungen (A):	Vorbedingungen (B):
(on a table)	(on b table)
* (clear a)	* (clear b)
(clear b)	(clear c)
Aktion (A):	**Aktion (B):**
(puton a b)	(puton b c)
Nachbedingungen (A):	**Nachbedingungen (B):**
(on a b)	(on b c)
(not(on a table))	(not(on b table))
(not(clear b))	(not(clear c))
(clear table)	(clear table)

Abb. 5.5-5 Ein zweiter unvollständiger Plan für die Sussman-Anomalie

Erklärung für den zweiten unvollständigen Plan zur Lösung der Sussman-Anomalie (Abb. 5.5-5):

Dadurch, daß z1 und z2 (siehe Abb. 5.5-4) an den konstanten Wert 'table' gebunden werden, bekommen die Bedingungen (on a z1) und (on b z2) als Vorbedingungen zu den PUTON-Operationen in den beiden Teilplänen jeweils den Wahrheitswert WAHR, womit TWEAK dem Ziel, sämtliche Ausführungsbedingungen zu erfüllen, wieder ein Stückchen näher kommt. Die Bedingung (clear table) ist immer erfüllt, denn auf den Tisch können unabhängig von der aktuellen Belegung stets Blöcke gestellt werden. Außerdem müssen an die Variablen x und y aus der PUTON-Operatorbeschreibung (Abb. 5.5-3) Constraints gebunden werden, daß die Instantiierungswerte ungleich der Konstanten 'table' sein müssen (zum Vergleich: bei SIPE würde dies mit dem Constraint NOT-SAME ausgedrückt werden).

Die Tatsache, daß die Vorbedingung (clear b) in Teilplan A als erfüllt gilt, in Teilplan B aber nicht, ergibt sich daraus, daß die in Teilplan A spezifizierte Aktion vor der in Teilplan B spezifizierten ausgeführt werden könnte (bea.: eine Ausführungsreihenfolge ist noch nicht festgelegt !). Dadurch würde die zuvor noch geltende Bedingung (clear b) negiert werden, ohne daß darauf ein "Wiedergutmacher-Planschritt" (siehe Erfüllungs-Kriterium in Abschnitt 5.4.1) folgen

würde. Bei umgekehrter Ausführungsreihenfolge bleibt diese Vorbedingung allerdings erfüllt.

Es zeigt sich, daß der Wahrheitswert einer eine Bedingung formulierende Aussage nicht immer so ersichtlich sein muß, wie dies bisher in den anderen Fällen bei dem Beispiel der Fall war. Der für die eigentliche Planarbeit zuständige Modul muß erkennen, ob eine im Plan formulierte Bedingung erfüllt ist oder nicht und dementsprechend vorgehen, etwa andere Planungsmoduln aktivieren oder das Setzen von Constraints veranlassen.

Vorbedingungen (B):	Vorbedingungen (A):
(on b table)	(on a table)
(clear b)	* (clear a)
(clear c)	(clear b)
1. Aktion (B):	**2. Aktion (A):**
(puton b c)	(puton a b)
Nachbedingungen (B):	**Nachbedingungen (A):**
(on b c)	(on a b)
(not(on b table))	(not(on a table))
(not(clear c))	(not(clear b))
(clear table)	(clear table)

Abb. 5.5-6 Ein dritter unvollständiger Plan für die Sussman-Anomalie

Erklärung für den dritten unvollständigen Plan zur Lösung der Sussman-Anomalie (Abb. 5.5-6):

Damit in beiden Teilplänen die Vorbedingungen erfüllt sind, legt TWEAK gemäß dem Erfüllungs-Kriterium (siehe Abschnitt 5.4.1) durch Formulierung von Constraints die Ausführungsreihenfolge so fest, daß die in Teilplan B spezifizierte Aktion (puton b c) vor der in Teilplan A spezifizierten (puton a b) im Gesamtplan erfolgt. Auf diese Weise wird die im Erfüllungs-Kriterium als "Zerstörungs-Schritt" bezeichnete Aktion (puton a b) - "zerstörend" im Hinblick auf Erfüllung der Bedingung (clear b) - an eine Stelle im Plan verlagert, an der der Wahrheitswert von (clear b) keine Auswirkung mehr hat auf die Anwendung bzw. Nichtanwendung weiterer Operatoren (Fälle 1 und 2a im Erfüllungs-Kriterium).

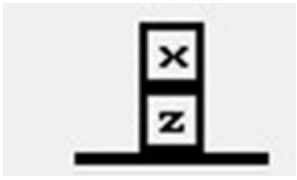
Die Vorbedingung (clear a) vor Anwendung der Operation (puton a b) ist noch nicht erfüllt. Eine noch detailliertere Spezifizierung der bestehenden Aktionen

ist nicht mehr möglich. Auch eine andere Festlegung der Reihenfolge bietet sich nicht an, da bereits in der vorausgegangenen Planphase die nun bestehende Reihenfolge als notwendig erkannt wurde. Folglich muß in den Plan ein weiterer Planschritt eingeführt werden. Die zusätzliche Anwendung des bereits eingesetzten PUTON-Operators (Abb. 5.5-3) brächte keinen Vorteil, da die in den Nachbedingungen formulierten Effekte nicht zum gewünschten Ziel führen, sondern einen Konfliktfall hervorrufen würden: die notwendige PUTON-Operation (puton a table) vor (puton a b) hätte die Nachbedingung (not(clear table)), die eine Kontradiktion, eine unerfüllbare Bedingung, darstellt. Daher bedarf es der Einführung eines neuen Operators (Abb. 5.5-7).

Die Anwendung des Operators NEWTOWER (Abb. 5.5-7) bewirkt, daß der als Argument angegebene Block direkt auf den Tisch gesetzt wird und somit den Grundbaustein für einen neuen Turm bildet.

Vorbedingungen:

(on x z)
(clear x)



Aktion:

(newtower x)

Nachbedingungen:

(on x table)
(not (on x z))
(clear z)

Abb. 5.5-7 NEWTOWER-Operator

TWEAK wählt nun die Ausführungsreihenfolge zunächst so, daß der Planschritt (newtower c) vor (puton a b) ausgeführt werden soll, damit der auf Block a

befindliche Block c weggeschafft und die Vorbedingung (clear a) der Aktion (puton a b) erfüllt ist.

```
Vorbedingungen (B):     Vorbedingungen (A):     Vorbedingungen (C):

 (on    b  table)        (on    a  table)        (on    c  a)
 (clear b )              (clear a )            * (clear c )
 (clear c )              (clear b )

1./2. Aktion (B):       3. Aktion (A):          2./1. Aktion (C):

 (puton  b  c)           (puton  a  b)           (newtower c )

Nachbedingungen (B):    Nachbedingungen (A):    Nachbedingungen (C):

  (on    b  c)            (on    a  b)           (on    c  table)
(not(on    b  table))   (not(on    a  table))  (not(on    c  a))
(not(clear c ))         (not(clear b ))           (clear a)
  (clear table)         (clear table)
```

Abb. 5.5-8 Ein vierter unvollständiger Plan für die Sussman-Anomalie

Erklärung für den vierten unvollständigen Plan zur Lösung der Sussman-Anomalie (Abb. 5.5-8):

> Nach Auswahl des NEWTOWER-Operators zur Erfüllung der Vorbedingung (clear a) von Aktion (puton a b) muß die Operator-Schablone (newtower x) näher spezifiziert werden. Die Variable x wird an den Wert c gebunden. Diese Festlegung wurde so gewählt, damit in den Nachbedingungen (C) die zu erreichende Vorbedingung (clear a) für (puton a b) explizit erscheint.

Im weiteren Planungsverlauf wird durch weitere Reihenfolgefestlegungen schließlich erreicht, daß auch alle Vorbedingungen der Operation (newtower c) erfüllt sind.

```
Vorbedingungen (C): Vorbedingungen (B): Vorbedingungen (A):

  (on    c  a)        (on    b  table)     (on    a  table)
  (clear c )          (clear b )           (clear a )
                      (clear c )           (clear b )

1. Aktion (C):      2. Aktion (B):      3. Aktion (A):

  (newtower c )       (puton  b  c)       (puton  a  b)

Nachbedingungen(C): Nachbedingungen(B): Nachbedingungen(A):

  (on    c  table)    (on    b  c)         (on    a  b)
(not(on    c  a))   (not(on    b  table)) (not(on    a  table))
  (clear a )        (not(clear c ))       (not(clear b ))
                      (clear table)         (clear table)
```

Abb. 5.5-9 Ein vollständiger Plan zur Lösung der Sussman-Anomalie

Erklärung zum letzten Schritt der Planungsphase, dessen Resultat ein vollständiger Plan ist, durch dessen Ausführung das gegebene Problem gelöst wird:

> Im vorausgegangenen Planentwurf (Abb. 5.5-8) war die Vorbedingung (clear c) von Operation (newtower c) nicht erfüllt, da möglicherweise der Planschritt (puton b c) vor dem Planschritt (newtower c) hätte ausgeführt werden können - die Reihenfolge war noch nicht vollständig festgelegt -, was die Negation der zuvor noch erfüllten Bedingung (clear c) zur Folge gehabt hätte. Die Reihenfolgefestlegung durch Constraints derart, daß (newtower c) vor (puton b c) in der Ausführungsreihenfolge steht, also insgesamt (newtower c) den 1., (puton b c) den 2. und (puton a b) den 3. Planschritt darstellen, verhindert diesen störenden Effekt. Gemäß dem Erfüllungs-Kriterium wird wieder der "Zerstörungs-Schritt", diesmal (puton b c), so verlagert, daß sich durch dessen Ausführung ergebende Effekte keinen störenden Einfluß auf andere Aktionen haben. Durch die Nachbedingungen von (newtower c) werden keine Vorbedingungen von dem folgenden Planschritt (puton b c) negativ beeinflußt.

Da nun sämtliche Vorbedingungen der verwendeten Operatoren erfüllt sind und bereits zu Beginn der Planerstellung die Operatorfestlegung so vorgenommen wurde, daß die zu erreichenden Planziele, die durch die Zielsituation beschrieben werden, in der Liste der Operator-Nachbedingungen erscheinen, ist somit ein vollständiger Plan erstellt, der die Transformation von der durch die Problemstellung gegebenen Startsituation in die Zielsituation genau spezifiziert.

5.6 Beurteilung von TWEAK

Allen vor TWEAK (1984) entwickelten domänen-unabhängigen Planungssystemen ist gemeinsam, daß sie eine Art und Weise der Repräsentation von Aktionen verwenden, die für Anwendungsbereiche aus einer komplexen, realen Welt noch nicht ausreichend sind. Die Erweiterung der Repräsentationsmöglichkeiten von Planern ermöglicht zwar eine größere Anwendungsbreite, bringt aber auch mit sich, daß der Planungsaufwand exponentiell ansteigt.

Die Repräsentationsmöglichkeiten von Domänenwissen, von Planschritten oder von der Größe Zeit sind auch bei TWEAK sehr beschränkt - eher noch mehr als bei anderen Planern -, so daß dieser Planer für realistischere Anwendungsaufgaben, die über den Horizont der Blockwelt

hinausgehen, ungeeignet ist. Einschränkungen bei der Repräsentation von Aktionen liegen unter anderem darin, daß implizite Seiteneffekte nicht aus gegebenen Operator-Effekten ableitbar sind und alle durch die Anwendung eines Operators zu erwartenden Veränderungen in der Welt explizit als Nachbedingungen spezifiziert sein müssen.

Das bereits vor TWEAK 1983 entwickelte Planungssystem SIPE bietet die Möglichkeit, aus explizit formulierten Fakten Schlußfolgerungen zu ziehen. Dieser Ansatz erweist sich in bestimmten einfacheren Domänen als ganz brauchbar, ist jedoch nicht vollständig. Bei der Entwicklung von TWEAK wurde auf Korrektheit und Vollständigkeit großen Wert gelegt. Hätte man die Repräsentationsmöglichkeiten vielfältiger gestaltet etwa in der Weise, daß deduktive Operatoren bereitgestellt würden, hätte das Erfolgs-Kriterium, das die Grundlage bei der Erfüllung von (Teil-)Zielen durch TWEAK bildet, keine Gültigkeit mehr.

Constraints bilden bei TWEAK ein nicht so mächtiges Werkzeug zur Darstellung von Abhängigkeiten und einschränkenden Bedingungen, wie dies z.B. bei MOLGEN der Fall ist. Zum einen werden Constraints bei TWEAK eingesetzt, gewisse Reihenfolgerestriktionen festzulegen, zum anderen beschreiben sie Bindungen zwischen Variablen und deren Instantiierungswerten. Die Propagierung von Constraints ist nur indirekt über die sich durch das Setzen von Constraints ergebenden Bedingungskonstellationen möglich. Die Angabe von Wertebereichseinschränkungen für Variablen ist nicht so komfortabel wie etwa bei MOLGEN oder PLAKON (siehe Kap. 7). Laut Chapman, dem Entwickler von TWEAK, sollte dadurch gewährleistet sein, daß sich die Bestimmung von Constraints einfacher gestaltet und die Vorgehensweise, basierend auf dem Erfüllungs-Kriterium, korrekt ist.

Es stellt sich die Frage, warum bei TWEAK alle diese Einschränkungen bei der Planerstellung in Kauf genommen wurden: die Repräsentationsmöglichkeiten sind dürftig, die Constraints nicht so ausdrucksstark wie bei anderen Planern. Ein bereits angesprochener Vorteil liegt in der Korrektheit und Vollständigkeit bei der Vorgehensweise dieses Planers. Chapman beweist in seinen Ausführungen [2] das Korrektheits-/Vollständigkeitstheorem: Terminiert TWEAK bei der Erstellung eines Planes, dann wird bei Ausführung dieses Planes das Problem tatsächlich gelöst; bricht TWEAK den Planungsprozeß mit entsprechender negativer Mitteilung ab oder erfolgt keine Terminierung, dann existiert zum gegebenen Problem keine Lösung. Der Beweis für diese Behauptung erstreckt sich über nur wenige Zeilen und folgt unmittelbar aus der verwendeten Suchstrategie und der Vorgehensweise von TWEAK bei der Erfüllung von Zielen und Änderung von noch

unvollständigen Plänen gemäß dem Erfüllungs-Kriterium. Chapmans Aussage muß vor Aufkommen zu großer Euphorie gleich relativiert werden. Erstens wird vorausgesetzt, daß die Wissensbasis zur Lösung von Problemen in der jeweiligen Domäne sämtliche möglicherweise notwendigen Operatoren korrekt spezifiziert bereithält - eine nahezu unlösbare Aufgabe für den Wissensingenieur, der die Wissensbasis erstellt. Zweitens wird davon ausgegangen, daß jedes Problem für das Planungssystem verständlich, d.h. in seiner Repräsentationssprache, formuliert werden kann. Die eingeschränkten Repräsentationsmöglichkeiten von TWEAK lassen dies in den meisten Domänen nicht zu. Drittens ist es ein unentscheidbares Problem, ob eine noch nicht eingetretene Terminierung bei der Erstellung eines Planes noch erfolgt oder nicht. Trotz dieser drei abschwächenden Punkte ist die in obigem Theorem gemachte Aussage aus theoretischer Sicht sehr mächtig. Über keinen vor TWEAK entwickelten Planer kann Derartiges ausgesagt werden. Dennoch zeigt die Praxis, daß ein nicht so perfekt, gemeint ist hier korrekt und vollständig, arbeitendes Planungssystem, das über umfassendere Repräsentationsmöglichkeiten und vielleicht noch über interaktive Elemente verfügt, für komplexere Problembereiche besser geeignet wäre.

Wenn TWEAK (z.B. in der Blockwelt-Domäne) eingesetzt werden kann und der Planungsprozeß terminiert, dann zeichnen sich die erstellten Pläne durch prägnante Formulierung, gute Leserlichkeit, Nachvollziehbarkeit und minimalen Umfang aus.

Ein weiterer Vorteil bei der Vorgehensweise von TWEAK besteht darin, daß mit polynomialem Zeitaufwand bezüglich der Zahl der Planschritte das Erfolgs-Kriterium ausgewertet werden kann. Schnelle, effiziente Planerstellung ist die Folge. Allerdings kann dem Planer nicht durch Bereitstellung von Strategiewissen Arbeit abgenommen werden; dies stünde im Widerspruch zur prinzipiellen Vorgehensweise und würde schon an TWEAKs eingeschränkten Repräsentationsmöglichkeiten von Wissen scheitern.

Die Planerstellung mit TWEAK ist, wie eben gesagt, recht effizient gestaltet - von nicht vorhandenem bzw. nicht eingesetztem Meta-Wissen einmal abgesehen -, hingegen wird mögliches paralleles Ausführen von im Plan spezifizierten Aktionen (wie z.B. bei SIPE) nicht unterstützt. Mit Hilfe von Constraints wird die Planausführung sequentialisiert und zumindest partiell eine Reihenfolge festgelegt. Gerade in Anwendungsbereichen der Robotik, in denen eine schnelle Ausführung von Plänen unter möglichst optimaler Ausnutzung vorhandener Manipulatoren angestrebt wird, sind derartige Planer wie TWEAK ungeeignet.

Kapitel 6 : Planungssystem TWAIN

TWAIN [1] stellt ein sogenanntes Task-Level-Planungssystem dar, das einen Ansatz zur automatischen Roboter-Programmierung liefert.

Die Idee bei der Task-Level-Programmierung besteht darin, daß der Benutzer nur durch Angabe der physikalischen Beziehungen von Objekten die Programmierung eines Roboters spezifizieren kann, wobei diese Beziehungen die zu erreichenden Planziele bilden. Dies bedeutet insofern eine Vereinfachung, daß es im Gegensatz zum hier verwendeten Ansatz des impliziten Programmierens - d.h. das Programm ist implizit durch die Spezifikation der Aufgabe und der Umwelt gegeben - nicht mehr einer expliziten Auflistung der einzelnen Schritte durch den Benutzer bedarf, die zum Erreichen der Ziele durchgeführt werden müssen. Als Konsequenz dieser Vorgehensweise ergibt sich, daß die Informationen, die vom Benutzer in den Planungsprozeß eingebracht werden müssen, roboter-unabhängig sind; es müssen von Seiten des Benutzers keinerlei Positionen oder Zugriffswege spezifiziert werden, die von gegebenen Geometrie- oder Bewegungsdaten des Roboters abhängen. Möglich wird dieser Ansatz erst durch eine umfangreiche Wissensbasis, die ein einigermaßen wirklichkeitsnahes Weltmodell aufzubauen erlaubt.

6.1 In TWAIN verwendete Planungstechniken

Die in Kapitel 2 vorgestellten Planerstellungsmethoden finden in TWAIN wie auch in anderen Planungssystemen in Mischformen Verwendung, um möglichst effizientes Planen zu ermöglichen.

Die Aufgabenstellung wird als Aneinanderreihung abstrakter Operatoren interpretiert, die es in darauffolgenden Planschritten zu konkretisieren gilt. Gemäß dem Vorgehen beim mehrstufigen Planen wird ein gegebenes Problem als abstrakte Einheit ohne Details über die konkrete Lösung dieser Einheit betrachtet.

Zunächst wird die gestellte Aufgabe in Teilaufgaben zerlegt; dabei wird top-down vorgegangen, d.h. die Teilprobleme werden schrittweise verfeinert bis man bei

den als elementar betrachteten Operatoren angelangt ist. Entscheidender Vorteil bei dieser Vorgehensweise ist, daß schon auf der Ebene abstrakter Operatoren Reihenfolgerestriktionen erkannt werden können, was dazu führt, daß auftretende Interaktionen frühzeitig erkannt und behebende Maßnahmen gegen diese eingeleitet werden können, ohne bereits getätigte Planarbeit rückgängig machen zu müssen.

Die Technik der Situations-Abstraktion ist bei TWAIN insofern eingesetzt worden, daß den zum Erreichen eines Teilziels durchzuführenden Roboter-Aktionen Prioritäten zugeordnet sind: Feinbewegungsoperationen werden vor Greifoperationen im Plan instantiiert, und Grobbewegungen schließlich werden zuletzt spezifiziert. Die Gründe für diese Art der Festlegung liegen in bestehenden Abhängigkeiten zwischen den verschiedenen Operationsarten, was in den folgenden Abschnitten dieses Kapitels noch näher erläutert wird.

Das Planen in dieser Abstraktions-Hierarchie funktioniert nun so, daß die Aufgabenstellung den Plan auf abstraktester Ebene darstellt. Für die im aktuellen Planungszustand erkennbaren Planschritte werden "Skelette" eingeführt, die auf nächstuntere Stufen heruntergereicht und dort ausgefüllt, d.h. instantiiert, werden. Auf diese Weise wird solange vorgegangen (Planschritte verfeinern, Skelette einführen und instantiieren), bis man bei einem Plan auf unterster Stufe, der Stufe der ausführbaren Roboter-Operationen, angelangt ist.

Die Verwendung von Plan-Skeletten, genauer gesagt von Planschritt-Skeletten, ermöglicht wissensbasiertes Planen, d.h. in Problemgebieten bereits existierende abstrakte Standard-Lösungsstrategien können eingesetzt werden. Die Plan-Skelette können im Sinne der Operator-Abstraktion auf mehreren Abstraktionsstufen vorliegen. Die eigentliche Planungstätigkeit besteht nun darin, für eine gestellte Aufgabe aus der Wissensbasis "passende" Plan-Skelette herauszusuchen und sie für das Problem im Rahmen der Planerstellung zu instantiieren. Scheitert die Planung auf einer Stufe, wird auf der nächsthöheren versucht, eine Alternative zu finden. Hierarchisches, nichtlineares Planen wird praktiziert.

Auf den einzelnen Abstraktionsstufen findet sich die Technik des einstufigen Planens als Suche wieder, um eine von möglichen Aktionen im Rahmen einer Planschrittverfeinerung auszuwählen, die nach bestimmten Kriterien die wohl erfolgversprechendste ist. Dabei werden die bei Ausführung einer Aktion sich ergebenden Situationen miteinander verglichen. Diejenige Aktion, die bei Ausführung die bestehende Situation in eine solche überführt, die der (Teil-)Zielbeschreibung unter

Einhaltung gewisser Bedingungen am nächsten kommt, wird weiter spezifiziert.

Schließlich werden auch bei TWAIN Constraints eingesetzt, im Rahmen einer Propagierung die Kommunikation zwischen den Teilproblemen zu ermöglichen und auf diese Weise bestehende gegenseitige Beeinflussungen in den Griff zu bekommen. Während der Planerstellung zu berücksichtigende Bedingungen werden durch Constraints dargestellt.

6.2 Task-Level-Programmierung mit TWAIN

Die Technik der Task-Level-Programmierung oder Programmierung auf Aufgabenebene unterstützt die Abstraktion der Kontrolle und der Daten. Die Möglichkeiten zur Daten-Abstraktion befreien den Programmierer von dem Wissen um detaillierte geometrische und kinematische Angaben bezüglich der Objekte und des Roboters, die Kontroll-Abstraktion erlaubt es, den Kontrollfluß im Programm bequem zu steuern.

6.2.1 Motivation zur Task-Level-Programmierung

Die meisten Roboter werden nach der sogenannten "Teaching-by-Showing"- oder "Teaching-by-Guiding"-Methode programmiert: der Roboter wird gemäß eines vorgegebenen Positionsplanes geführt; dabei werden Gelenkstellungen des Roboters und die in den jeweiligen Positionen durchzuführenden Operationen wie z.B. das Schließen einer Greifhand aufgezeichnet. Als Ergebnis erhält man ein Programm, das vereinfacht gesagt eine Folge von Gelenkkoordinaten und Aktivierungssignalen für Manipulatoren darstellt.

Der Vorteil dieser Vorgehensweise liegt in der einfachen Durchführ- und Implementierbarkeit.

Als großer Nachteil erweist sich jedoch, daß dieser Programmierungsansatz wichtigen Einschränkungen unterliegt, vor allem beim zusätzlichen Einsatz von Sensoren. Der Roboter kann seine Arbeit nur nach der einmal festgelegten Ausführungssequenz durchführen. Die wiederholte oder bedingte Ausführung bestimmter Aktionen läßt sich nicht spezifizieren, Miteinbeziehung der von Sensoren aufgenommenen Daten in den Ausführungsprozeß ist nicht möglich.

Die Roboter-Programmierung bei realistischeren und damit komplexeren Anwendungen bedarf deshalb der Konstrukte einer Programmiersprache, um das Hilfsmittel der Kontroll-Abstraktion einsetzen zu können.

Durch den Einsatz einer traditionellen höheren Programmiersprache, die noch um Kommandos zur Aufnahme von Sensordaten und zur Spezifizierung von Roboter-Bewegungen erweitert wird, können mit einem einmal erstellten Programm verschiedene verwandte Aufgaben gelöst werden, die Zahl der möglichen Anwendungen vergrößert sich. Als nachteilig erweist sich hier allerdings die Forderung, daß ein Programmierer von Robotern zum einen die Programmierung von Computern zu beherrschen hat, zum andern auch im Entwurf von sensor-basierten Handlungsstrategien vertraut sein muß.

Bei der Task-Level-Programmierung wird der Roboter-Programmierungsprozeß insoweit abstrahiert, daß der Benutzer nur noch die zu erfüllenden Beziehungen zwischen Objekten als Ziele anzugeben braucht. Die auf Task-Ebene erstellte Spezifikation ist roboter-unabhängig. Um nun an Hand dieser Eingaben einen Plan ausführbarer Aktionen erstellen zu können, muß TWAIN, wenn nicht vom Benutzer, von einer Wissensbasis die notwendigen Informationen beziehen können. Ein Task-Level-Planungssystem erfordert daher ein vollständiges geometrisches und physikalisches Modell des Roboters und der Anwendungsumgebung.

6.2.2 Beispiel einer (Teil-)Aufgabenstellung

Eine Task könnte folgendermaßen formuliert sein (Abb. 6.2.2-1):

```
POSITIONIERE  WÜRFEL A  so, daß:

   (Fläche A.4 gegen Tisch gerichtet)            und
   (Fläche A.1 gegen Fläche F.1 gerichtet)       und
   (Fläche A.2 gegen Fläche F.2 gerichtet),


POSITIONIERE  WÜRFEL B  so, daß:

   (Fläche B.4 gegen Fläche A.3 gerichtet)       und
   (Fläche A.1 und Fläche B.1 bilden eine Ebene) und
   (Fläche A.2 und Fläche B.2 bilden eine Ebene).
```

Abb. 6.2.2-1 Beispiel einer Taskbeschreibung

Ohne die einzelnen Schritte explizit angegeben zu haben, wird durch obige Aufgabenformulierung (Abb. 6.2.2-1) die Sequenz der durchzuführenden Aktionen vollständig spezifiziert. Erstellt der Benutzer eine unvollständige Taskbeschreibung, muß ein zusätzlicher Planungsmodul, der sog. Assembler-Planer, dem eigentlichen Planungsprozeß vorgeschaltet werden, um eine vollständige Beschreibung zu liefern. Welche Fläche eines Würfels mit '.4' oder '.1' in Abb. 6.2.2-1 gemeint ist oder um welches Teil es sich bei 'F' handelt, kann vom Planer in der Wissensbasis nachgeschlagen werden. "POSITIONIERE" stellt einen abstrakten Operator dar, der nach ganz bestimmten Regeln und unter Einhaltung gegebener Bedingungen durch eine Sequenz elementarer Operatoren konkretisiert wird. Die in der Aufgabenstellung formulierten Beziehungen spezifizieren eine Zielsituation, die es nach Durchführung bestimmter Planschritte zu erreichen gilt.

Abb. 6.2.2-2 illustriert noch die in Abb. 6.2.2-1 spezifizierte, erwünschte Situation.

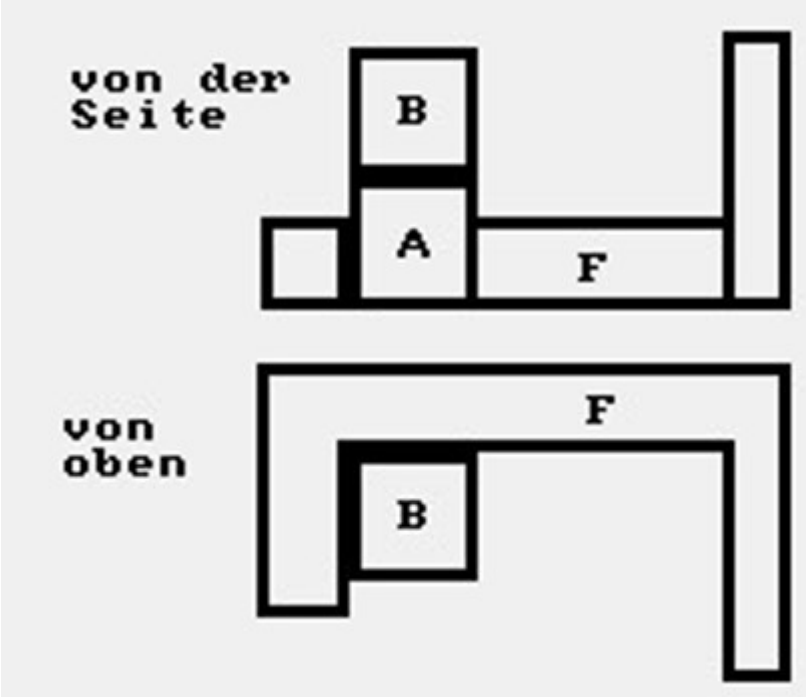


Abb. 6.2.2-2 Blockwelt-Situation

Im Rahmen eines ersten Verfeinerungsschrittes könnte folgendes noch nicht ausführbare Programm zu der obigen Aufgabenformulierung generiert werden (Abb. 6.2.2-3 auf der nächsten Seite):

1. ÖFFNE_FINGER bis <Kantenlänge von A + epsilon>;
2. BEWEGE_GREIFHAND nach <Position der Greifstellen von A> über <Zugriffsweg>;
3. SCHLIESSE_FINGER bis <Kantenlänge von A - epsilon>;
4. BEWEGE_GREIFHAND nach <Position von F> über <Zugriffsweg>;
5. ÖFFNE_FINGER bis <Kantenlänge von A + epsilon>;
6. ÖFFNE_FINGER bis <Kantenlänge von B + epsilon>;
7. BEWEGE_GREIFHAND nach <Position der Greifstellen von B> über <Zugriffsweg>;
8. SCHLIESSE_FINGER bis <Kantenlänge von B - epsilon>;
9. BEWEGE_GREIFHAND nach <Position von A> über <Zugriffsweg>;
10. ÖFFNE_FINGER bis <Kantenlänge von B + epsilon>.

Abb. 6.2.2-3 Erster Verfeinerungsschritt einer gegebenen Taskbeschreibung

Die Transformation einer Taskbeschreibung in ein ausführbares Programm bedarf vieler Schritte. Auf die sich dabei ergebenden Probleme und zu klärenden Details wird im nächsten Abschnitt eingegangen.

6.2.3 Von der Aufgabenstellung zum Programm

Der eigentliche Planungsprozeß für eine gestellte Aufgabe, der von einem Task-Planer wie TWAIN vollzogen wird, besteht darin, die abstrakte, vom Benutzer gegebene Spezifikation, die die zu erreichenden Ziele enthält, in ein Programm umzusetzen, das konkrete, von einem Roboter durchführbare Aktionen spezifiziert, um die Aufgabe zu erfüllen. Der Task-Planer muß also die roboter-spezifischen Operationen planen.

Die Überführung einer vom Benutzer gegebenen Aufgabenbeschreibung auf Task-Ebene in ein Programm auf Roboter-Ebene bringt verschiedene Probleme mit sich:

(1) Beschaffung der Teile:
Für jedes Teil, das im Rahmen der Aufgabenbearbeitung benötigt wird, muß festgelegt werden, auf welche Art und Weise es möglichst schnell und sicher in den Arbeitsbereich eingebracht werden kann.

(2) Layout:
Der Planer muß festlegen, wo genau im Arbeitsbereich jede einzelne Operation durchgeführt werden soll. Ziel ist es dabei, die Zahl der dazu nötigen

Greifoperationen, mögliche Positionierungs- und Erfassungsfehler und schließlich die aufzubringende Arbeitszeit zu minimieren. Um dieser Aufgabe gerecht zu werden, erfolgt zunächst die Skizzierung eines groben Layouts, das auf Durchführbarkeit gegebener Aktionen unter Wahrung der Ziele getestet wird. Ein detailliertes Layout kann erst gegen Ende der Planerstellung bestimmt werden.

(3) Feinbewegung:
Der Planer muß bezüglich Situationserfassungen über Sensoren und durchzuführender Bewegungsabläufe eine Strategie auswählen, die zuverlässiges Zusammenfügen von Teilen gewährleistet. Etwaige Fehler in der Steuerung des Roboters oder bei der Positionierung von Teilen auf Grund von Lücken in der Wissensbasis, unvermeidbarer Ausführungsungenauigkeiten oder unerwarteter Einflüsse aus der Umgebung werden dabei möglichst in vollem Umfang berücksichtigt und korrigiert.

(4) Fixierung:
Einspannvorrichtungen und fest positionierte Gegenstände im Arbeitsbereich müssen bestimmt werden, welche die zu bearbeitenden oder zu montierenden Teile fixieren sollen, um trotz gewisser Krafteinwirkungen, z.B. beim Durchführen von Bohrungen, das geforderte Maß an Genauigkeit einhalten zu können.

(5) Greifen:
Für jedes Teil müssen die Stellen bestimmt werden, an denen zugegriffen werden kann und zwar so, daß das betreffende Teil stabil genug und positionsgünstig erfaßt werden kann, um die darauffolgenden Aktionen auszuführen und mögliche Kollisionen der Greifhand mit Objekten während des Greifens oder Plazierens zu vermeiden.

(6) Grobbewegung:
Um effizientes Arbeiten zu gewährleisten, muß der Planer bei der Positionierung und Ergreifung von Teilen möglichst kurze kollisionsfreie Bahnen ermitteln.

Die größten Probleme ergeben sich dadurch, daß mit Fehlern bei der Ausführung von Plänen in Bereichen der realen Welt zu rechnen ist. Es bedarf des Einsatzes von Sensoren, derartige Fehler zu erkennen und zu berichtigen. Der Planer muß dementsprechende Aktionen einkalkulieren. Des weiteren können unerwartete Ereignisse zu einem Scheitern der Ausführung einer Operation führen. Der Planer kann derartige Vorfälle durch Miteinbeziehung der Wahrscheinlichkeiten für ihr Eintreten in den Plan zu einem Teil berücksichtigen. Daher wundert es nicht, daß bereits einfache Aufgabenbeschreibungen auf Task-Ebene zu umfangreichen, von Robotern ausführbaren Programmen führen. Dies

unterstreicht die Nützlichkeit von Task-Level-Planungssystemen wie TWAIN, die eine komfortable Programmierung auf abstrakter Ebene ermöglichen.

Im Fortschreiten der Planerstellung gilt es, eine Fülle von Zahlenwerten zu verarbeiten: Positionsangaben (z.B. Ergreifungspunkte, Objektkonfigurationen), Größenangaben (z.B. Öffnungsweiten für Greiffinger, um bestimmte Teile zu erfassen), Richtungsangaben (z.B. bei "vorsichtigen" Bewegungen mit Sensoreinsatz) und Bahnangaben für Positionier- und Greifbewegungen. Auf der abstrakten Task-Ebene ist der Benutzer beim Einsatz von TWAIN mit derartigen spezifischen Fakten nicht konfrontiert. Das Planungssystem holt sich diese Informationen aus der Wissensbasis.

6.3 Wissensrepräsentation

Zentrales Thema der künstlichen Intelligenz ist die Darstellung und Verarbeitung von Wissen. In TWAIN sind verschiedene Ansätze verwirklicht, Domänen- und Strategiewissen in den Planungsprozeß miteinzubringen. Physikalische und geometrische Fakten über Roboter und Objekte ermöglichen den Aufbau eines Weltmodells. Mit Hilfe von Constraints können Randbedingungen in die Planerstellung integriert werden. Die Verwendung von Plan-Skeletten hat den Vorteil, Bereichswissen in strukturierter Form verfügbar zu machen und bereits existierende Lösungsschemata für Teilprobleme einzusetzen, um noch effizienter zu planen.

6.3.1 Das Weltmodell

Jede beim Planen verwendete Repräsentation eines komplexeren Weltbereiches enthält Verkürzungen, da entweder das Wissen über die Welt von vorneherein unsicher ist oder weil aus Effizienzgründen nur Standardfälle repräsentiert werden können. Die Repräsentation eines Bereiches der realen Welt ist folglich nur als Näherung zu verstehen.

Um abstrakte Operatoren wie z.B. GREIFE_OBJEKT oder POSITIONIERE in für den Roboter ausführbare Aktionen zu transformieren, muß bei dem benutzerfreundlichen Ansatz des Planens auf Task-Ebene, also auf der Ebene abstrakter Operationen, ein Zugriff auf Detailinformation möglich sein. Bei Roboter-Operationen sind eine Vielzahl von Parameterangaben nötig, um erfolgreich und effizient ausgeführt werden zu können.

Der Benutzer genießt den Komfort, sich um derartige Dinge bei der Aufgabenformulierung nicht kümmern zu müssen, sondern sich auf das Wesentliche, die zu erreichenden Planziele, konzentrieren zu können.

Durchführbare Bewegungen eines Objekts innerhalb des Arbeitsbereiches werden durch das Vorhandensein anderer Objekte eingeschränkt. Des weiteren spielen die einzelnen Objektformen eine wichtige Rolle. TWAIN benötigt daher vollständige geometrische Beschreibungen der Objekte, die manipuliert werden, und Details über das Verhalten des Roboters bei Bewegungsabläufen (Geschwindigkeiten, Beschleunigungsfähigkeiten, Freiheitsgrade, Trägheitsverhalten usw.). Zusätzliche Angaben über physikalische Eigenschaften von den Objekten erlauben eine Vorhersage des Verhaltens eines Objekts während der Ausführung einer Operation. So kann an der Masse, der Trägheit, dem Oberflächen-reibungskoeffizienten und der Lokalisation des Schwer-punktes eines Körpers abgelesen werden, wie schnell er von einer Stelle zu einer anderen bewegt werden kann und mit welcher Kraft der Greifarm diesen Körper festhalten muß, um die gewünschte Operation durchführen zu können.

Zur vollständigen Angabe der Position eines Objekts ist eine Menge von Parametern notwendig, um die Lage sämtlicher "markanter" Punkte eines Objekts zu spezifizieren. Diese Menge wird als Konfiguration eines Objekts bezeichnet.

Da in der Praxis nicht jegliches Verhalten auf Grund theoretischer Gesetzmäßigkeiten vorherbestimmt werden kann, sollten Roboter-Systeme über die Möglichkeiten verfügen, nicht nur Aktionen auszuführen, sondern auch während einer Operation eingetretene, möglicherweise unerwartete Ereignisse zu erkennen. Durch optische Wahrnehmungen können nicht ganz korrekt ausgeführte Positionierbewegungen korrigiert oder zum Beispiel durch unerwartete äußere Einwirkungen umgefallene Teile erkannt werden. Auf Druck ansprechende Sensoren erlauben etwa das Bewegen von Teilen, bis ein anderes Teil berührt wird, wodurch Ungenauigkeiten bei Positionsangaben ausgeglichen werden können. Das Planungssystem muß auch diese möglichen Erfassungen von Daten bei der Roboter-Programmgenerierung berücksichtigen und die durchzuführenden Schritte in den jeweiligen Situationen darauf abstimmen.

6.3.2 Verwendung von Constraints

Constraints werden in TWAIN dazu benutzt, verschiedene gegebene geometrische oder physikalische Eigenschaften von Objekten zueinander in Beziehung zu setzen, um auf diese Weise gewisse Bedingungen zu formulieren, die sich aus der Aufgabenstellung oder während des Planens aus Durchführbarkeitsüberlegungen ergeben.

In TWAIN ist es möglich, Constraints sowohl vorwärts (von den Eingangsgrössen eines durchzuführenden Schritts zu den Zielgrössen) als auch rückwärts (von den Ziel- zu den Eingangsgrössen) weiterzureichen. Diese Aufgabe wird von einem eigens dafür zuständigen Modul, dem Constraint-Propagierer, übernommen. An irgendeiner Stelle im Plan erzeugte Constraints werden an sämtliche Teilpläne propagiert. Bei anderen Planungssystemen wie z.B. bei MOLGEN ist dieser Kommunikationsmechanismus zwischen den Teilästen eines Planers nur in einer Richtung implementiert.

Constraints werden bei TWAIN algebraisch durch (Un-) Gleichungen mit folgenden Variablen formuliert:

- Variablen für physikalische Grössen, deren Werte konkrete Sachverhalte wiedergeben (z.B. Masse eines Objektes);
- Planvariablen, deren Werte während des Planungsprozesses zu bestimmende Grössen darstellen (z.B. Konfiguration eines Teils vor Ausüben einer Greifoperation) und die durch möglichst spätes Instantiieren ermöglichen, Entscheidungen während des Planens aufzuschieben (--> Unterbeschränkungsansatz beim nichtlinearen Planen);
- Variablen für unsichere Information, deren Werte Unsicherheiten im Plan widerspiegeln, die als Folge einer unvollständigen Modellvorstellung des Planers von der realen Welt bei nichttrivialen Aufgabenstellungen unvermeidbar sind und zu keinem Zeitpunkt während des Planens exakt bestimmt werden können. Lediglich durch Bereichsangaben können solche Grössen spezifiziert werden. Üblicherweise repräsentieren die Unsicherheitsvariablen den Unterschied zwischen einer konkreten physikalischen Größe und dem dafür vom Planer eingestellten Nominalwert.

6.3.2.1 Beispiel zur Constraint-Formulierung

Ein Beispiel verdeutlicht am besten das Erstellen von und Arbeiten mit Constraints.

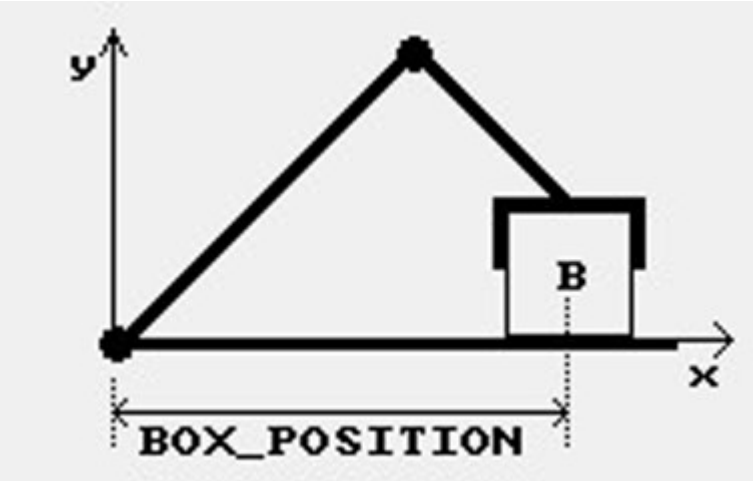


Abb. 6.3.2.1-1 Definition der Variablen BOX_POSITION

Es geht um folgende, in Abb. 6.3.2.1-1 abgebildete zweidimensionale Situation:

> Das Objekt B soll mit Hilfe eines Greifarms mit zwei regulierbaren Gelenken an eine bestimmte Stelle auf der Arbeitsfläche positioniert werden.

Es kann davon ausgegangen werden, daß der Greifarm mit einem Fehler von e Längeneinheiten (LE) in x-Richtung positionieren kann. Empirisch sei für e folgende Ungleichung ermittelt worden:

(1) -0.2 LE + 0.005 * Px <= e <= 0.2 LE - 0.005 * Px,

wobei Px die gewünschte x-Koordinate der Objektposition in LE angibt (der Einfachheit halber sei hier nur Py = 0 von Interesse, wobei Py entsprechend die gewünschte y-Koordinate repräsentiert).

Der Greifarm habe nur eine beschränkte Reichweite, so daß für die einstellbaren x-Koordinaten Px gelte:

(2) 5 LE <= Px <= 20 LE.

Für die möglichen Situationen nach Ausführung der Bewegungsoperation von B mit dem Greifarm lassen sich daraus Bedingungen ableiten, die als Constraints mit folgenden drei Variablen formuliert werden:

BOX_POSITION: Variable für physikalische Größe (siehe Abb. 6.3.2.1-1), deren Wert die tatsächliche aktuelle Position nach Ausführung der Bewegungsoperation repräsentiert; genauer: BOX_POSITION ist die tatsächliche x-Koordinate des Schwerpunktes von B;

BOX_NOM: Planvariable, deren Wert die während der Planung bestimmte Nominalgrösse der Position repräsentiert, zu der das Objekt B mit dem Greifarm hinbefördert werden soll (gemeint ist wieder die x-Koordinate des Schwerpunktes von B, die dem festgelegten Nominalwert anzugleichen ist);

BOX_UNC: Variable, deren Wert die unsichere Größe möglicher Positionierungsfehler in x-Richtung durch den Greifarm repräsentiert (Bereichsangabe: [- ... ; + ...]).

Damit lassen sich nun folgende Constraints formulieren:

(3) BOX_POSITION = BOX_NOM + BOX_UNC
(folgt unmittelbar aus obiger Def.)
(4) 5 LE <= BOX_NOM <= 20 LE (vgl. (2))
(5) -0.2 LE + 0.005 * BOX_NOM <= BOX_UNC und
0.2 LE - 0.005 * BOX_NOM >= BOX_UNC (vgl. (3))

Aus (4) und (5) kann durch den Inferenzmechanismus während der Planerstellung für die Variable BOX_UNC eine Wertbereichseinschränkung abgeleitet werden:

(6) -0.175 LE <= BOX_UNC <= +0.175 LE.

Angenommen, es ergäbe sich nun während der Planung auf Grund der Aufgabenstellung und/oder propagierter Constraints aus anderen Teilplänen die Notwendigkeit oder Forderung, daß der mögliche Positionierfehler in x-Richtung auf das Intervall [-0.15 LE;+0.15 LE] beschränkt bleibe. Erfüllung von (6) würde also nicht ausreichen. Aus (5) und der gestellten Forderung

bezüglich der Positionierfehler könnte dann abgeleitet werden:

(7) BOX_NOM >= 10 LE,

denn:
(5) ist äquivalent zur folgenden Betragsungleichung:
| BOX_UNC | <= 0.2 LE - 0.005 * BOX_NOM (I).
Ferner läßt sich auch die zusätzliche Forderung als Betragsungleichung formulieren:
| BOX_UNC | <= 0.15 LE (II).
Aus (I) und (II) folgt:
0.2 LE - 0.005 * BOX_NOM <= 0.15 LE
<==> - 0.005 * BOX_NOM <= -0.05 LE
<==> BOX_NOM >= 10 LE q.e.d.

TWAIN könnte also das Constraint (7) dynamisch während der Planerstellung formulieren und an die Planvariable BOX_NOM binden, um im Fortschreiten des Planens zu gewährleisten, daß die Bedingung -0.150 LE <= BOX_UNC <= +0.150 LE erfüllt wird.

6.3.3 Verwendung von Plan-Skeletten

Bei der Planerstellung werden die einzelnen Planschritte zunächst als "Skelette", als "Black-Box"-Grössen eingeführt. Für den Rest des Planes interessieren zunächst nur die Ein- und Ausgabegrössen eines Planschrittes; wie die Transformation aussieht, wird dann erst in weiteren Verfeinerungsschritten wichtig, wenn Interaktionen zwischen den Planschritten oder Teilplänen auftreten und durch Umplanen oder Constraint-Formulierungen berücksichtigt werden müssen.

Durch die Wahl eines bestimmten Skeletts wird die Vorgehensweise zur Erstellung des betreffenden Teilplanes festgelegt. Die Planverfeinerung sieht dann so aus, daß dieser Rahmen angemessen instantiiert wird. Der Vorteil liegt klar auf der Hand: bereits existierende Standardlösungsstrategien können auf diese Weise eingesetzt werden. Dabei muß allerdings gewährleistet sein, daß eine genügend große Zahl von Plan-Skeletten verfügbar ist, um auf das jeweilige Teilproblem "zugeschnittene" Strategien anwenden zu können. Stellt sich im Fortschreiten des Planens heraus, daß keine Lösung unter den gegebenen Bedingungen ermittelt werden kann, wird ein anderes Plan-Skelett ausgewählt.

In der Wissensbasis verfügt TWAIN über eine Vielzahl von Planschritt-Skeletten, passend für die jeweiligen Situationen, welche durch eine geometrische Beschreibung

von Objekten und algebraischen Constraints spezifiziert sind. Der Spezifizierungsgrad ist dabei unterschiedlich: es gibt ein Skelett für allgemeine Greifoperationen und solche für ganz spezielle Feinbewegungen wie z.B. "Bolzen_in_Bohrung" (siehe Abschnitt 6.3.3.1).

Mit Hilfe solcher Skelette lassen sich Planschritte auf verschiedenen Detaillierungsebenen beschreiben. Im Laufe der Planschrittverfeinerung, d.h. während der Plan-Skelett-Instantiierung, können weitere Aktionen wiederum durch Skelette repräsentiert und in weiteren Schritten instantiiert werden.

Die Auswahl des richtigen Skeletts wird durch den Rahmen-Ermittler-Modul vorgenommen, wobei Vergleiche der zwischen jedem Skelett zugrundeliegenden geometrischen Beschreibung und der aktuellen Situation gemacht werden.

Als weiteres Mittel zur Spezifikation eines Planschritt-Skeletts dienen Constraints. Propagierungs-Constraints legen die möglichen Situationen fest, die sich aus einer gegebenen Situation durch Ausführung des betrachteten Planschritts ergeben können, und Anwendbarkeits-Constraints schränken diejenigen Situationen ein, in denen die nötigen Voraussetzungen gegeben sind, den jeweiligen Planschritt auszuführen. Bei der Auswahl eines geeigneten Plan-Skeletts wird für die möglichen Skelette an Hand der Propagierungs-Constraints untersucht, ob die sich durch Instantiierung und Ausführung ergebenden Situationen dem Teilplan-Ziel näher kommen. Die Anwendbarkeits-Constraints zeigen an, welche Skelette überhaupt erst in Frage kommen.

Die Verwendung von Skeletten für Planschritte, welche nach und nach instantiiert werden, - man könnte auch von "Rahmen" oder "Gerüsten" sprechen - ermöglicht nicht nur hierarchisches Top-Down-Vorgehen bei der Planerstellung, sondern bietet die zusätzliche Möglichkeit, durch kaum einschränkende Constraints gleichzeitig eine ganze Klasse möglicher Aktionen für einen Planschritt zunächst in Betracht zu ziehen, um dann erst zu einem späteren Zeitpunkt sich auf einen detaillierteren Plan festlegen zu müssen, der mit Hilfe der speziellen Planungsmoduln (siehe Abschnitt 6.4) erzeugt wird. Dadurch wird eher gewährleistet, keine potentiellen Lösungen schon frühzeitig während des Planens zu unterschlagen. Dafür muß auch anfangs kein unüberschaubares Maß an Möglichkeiten verwaltet werden, da die Aktionsbeschreibungen zunächst abstrakt formuliert werden.

6.3.3.1 Beispiel eines Plan-Skeletts

Zur Illustration hier nun das vereinfachte Plan-Skelett zur Feinbewegungsoperation "Bolzen_in_Bohrung".

Der Einfachheit halber sei eine zwei-dimensionale Welt angenommen. Ferner seien hier nur die Bedingungen formuliert, die bei den in x-Richtung, also parallel zur Arbeitsfläche vorzunehmenden Bewegungen zu erfüllen sind.

Die Ausgangssituation sei die in Abb. 6.3.3.1-1 gezeigte:

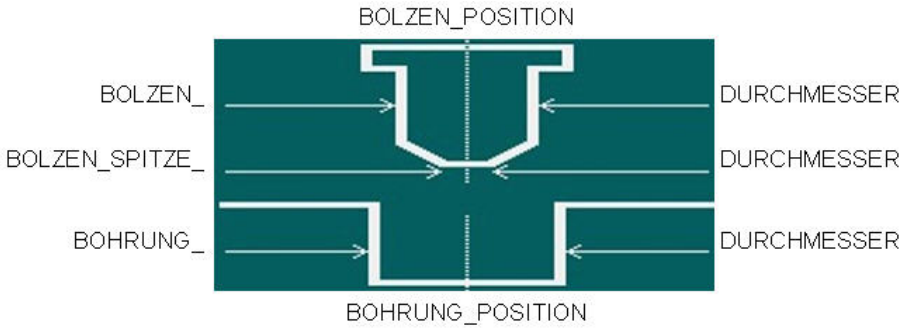


Abb. 6.3.3.1-1 Situation: Bolzen vor der Bohrung
(Zielsituation: Bolzen in der Bohrung)

Das in dieser Situation anzuwendende Plan-Skelett "Bolzen_in_Bohrung" wird folgendermaßen durch Constraints spezifiziert:

Anwendungsconstraints:

(1) BOLZEN_DURCHMESSER <= BOHRUNG_DURCHMESSER

(2) (BOLZEN_SPITZE_DURCHMESSER - BOHRUNG_DURCHMESSER)/2
<= BOHRUNG_POSITION - BOLZEN_POSITION
<= (BOHRUNG_DURCHMESSER - BOLZEN_SPITZE_DURCHMESSER)/2

(3) nominal(BOHRUNG_POSITION) = nominal(BOLZEN_POSITION)

Die zu erfüllenden Vorbedingungen lauten in Worten:

zu (1): Der Durchmesser der Bohrung muß mindestens so groß sein wie der Durchmesser des Bolzens;

zu (2): Die Differenz zwischen tatsächlicher Bolzenposition und tatsächlicher Position der Bohrung muß betragsmäßig kleiner gleich sein als die Differenz Bohrungsradius minus Bolzenspitzeradius;

zu (3): Nach korrekter Ausführung vorausgegangener Positionier- bzw. Bewegungsoperationen müßte die Position des Bolzens identisch sein mit der der Bohrung (nur x-Koordinaten betrachtet). Unter Zuhilfenahme von Sensoren und Erfüllung der Constraints kann also vor Ausführung der Operation "führe Bolzen in Bohrung" davon ausgegangen werden, daß Bolzen und Bohrung gleich positioniert sind.

Propagierungsconstraints:

(4) (BOLZEN_DURCHMESSER - BOHRUNG_DURCHMESSER)/2
<= BOHRUNG_POSITION - RESULTIERENDE_BOLZEN_POSITION
<= (BOHRUNG_DURCHMESSER - BOLZEN_DURCHMESSER)/2

(5) nominal(BOHRUNG_POSITION) =
nominal(RESULTIERENDE_BOLZEN_POSITION)

Die Bedingungen, die nach Ausführung der Operation "führe Bolzen in Bohrung" (kurz: Bolzen_in_Bohrung) erfüllt sein müssen, lauten in Worten:

zu (4): Die Differenz zwischen tatsächlicher Position der Bohrung und tatsächlicher Bolzenposition nach Bewegungsausführung muß betragsmäßig kleiner gleich sein als die Differenz Bohrungsradius minus Bolzenradius;

zu (5): Von möglichen Ausführungsfehlern abgesehen sind die Positionen von Bohrung und Bolzen (wieder nur x-Koordinaten betrachtet) identisch.

Zusätzliche Informationen, wie etwa BOLZEN_DURCHMESSER >= BOLZEN_SPITZE_DURCHMESSER, kann der Planer der geometrischen Beschreibung der einzelnen Objekte entnehmen.

Im Laufe der Plan-Skelett-Instantiierung werden die in den Constraints verwendeten Variablen mit angemessenen Werten belegt.

6.4 Architektur

TWAIN besteht grob aus einer Wissensbasis und den folgenden Moduln, die die eigentliche Planungstätigkeit ausüben:

- Constraint-Propagierer
- Rahmen-Ermittler
- Feinbewegung-Planer \
- Greif-Planer - TWAINs Planungsmoduln
- Grobbewegung-Planer /

Die Funktionen der ersten beiden Moduln ist bereits in den vorigen Abschnitten aufgezeigt worden. Die drei Planungsmoduln sollen im folgenden etwas näher untersucht werden.

Den Ausgangspunkt für das Erstellen eines Planes bilden eine Reihe von Plan-Skeletten, welche der Rahmen-Ermittler als Lösungsschemata für verschiedene Standard-Lösungsstrategien bestimmt.

Der Constraint-Propagierer veranlaßt dann ein Weiterreichen von Information im Plan, wenn neue Constraints bestimmt, Planentscheidungen gefällt oder Variablen an einen (neuen) Wert gebunden worden sind. Constraint-Propagierung findet bei TWAIN allerdings nicht nach jedem Verfeinerungsschritt statt. Vielmehr weist die Kontrollstruktur (siehe Abschnitt 6.5) spezielle Phasen für das Weiterreichen von Constraints auf.

Die eigentliche Planungstätigkeit besteht darin, die Plan-Skelette zu instantiieren, indem der Plan mit Hilfe der drei Planungsmoduln schrittweise verfeinert wird. Die Planungsmoduln sind so konzipiert, daß jeder Modul für eine bestimmte Klasse von Situationen zuständig ist, in denen es dann einen detallierteren Planungsschritt zu spezifizieren gilt.

Im schlimmsten Fall bricht ein Modul seine Ausführungen ohne Angabe eines Grundes ab. Es ist aber bei einem Mißerfolg auch möglich, daß eine Begründung und vielleicht auch noch ein "vernünftiger" Korrekturvorschlag für die gegebene Situation mitgeliefert werden. Nach erfolgreicher Durchführung einer Planungs-Teilaufgabe entsteht als Ergebnis ein detaillierterer Plan eventuell mit neu formulierten Constraints, wobei letztere in darauffolgenden Schritten für weitere Verfeinerungsschritte herangezogen werden müssen. Wird ein Planungsmodul zur Lösung derselben Teilaufgabe erneut aktiviert, liefert er ein anderes Ergebnis oder bricht eventuell ab.

6.4.1 Ausführliche Beschreibung der Planungsmoduln

Alle drei Planungsmoduln für Feinbewegung, Greifen und Grobbewegung können auf Daten über Objekte zugreifen, deren Position, genauer Konfiguration im Laufe der Planung festgelegt oder zumindest auf wenige mögliche Werte eingegrenzt wird. Diese Informationen sind durch geometrische Beschreibungen, physikalische Angaben und algebraische Constraints repräsentiert.

(1) Feinbewegung: Der Feinbewegung-Planungsmodul bestimmt eine Sequenz von Bewegungen, bei deren Ausführung gewährleistet ist, daß eine Konfiguration aus einem spezifizierten Bereich von Zielkonfigurationen, ausgehend von einer Konfiguration aus einem spezifizierten Bereich von Startkonfigurationen, erreicht wird.

Als Eingangsgrössen werden benötigt:

- AUSGANGSPOSITIONEN der Teile:
  Für jedes zu manipulierende Teil muß die Menge aller möglichen Ausgangspositionen bekannt sein.

- ZIELSPEZIFIKATION:
  Die Bereiche der zulässigen Endkonfigurationen der einzelnen Objekte müssen vor Ausführung einer Feinbewegung bzw. vor deren Planung angegeben werden.

- TOLERANZBEREICHE:
  Die Wahrscheinlichkeit von Ausführungsfehlern kann dadurch in die Planung miteinbezogen werden, daß die möglichen Abweichungen von Ist- zu Sollwerten (Greifarmstellungen, Objektpositionierungen ...) durch Angaben von Toleranzbereichen, eventuell durch Variablen repräsentiert (vgl. Variable BOX_UNC in Abschnitt 6.3.3.1), abschätzbar werden.

- CONSTRAINTS:
  Sich auf Bewegungsabläufe beziehende Bedingungen werden durch Constraints formuliert. Diese gilt es zu erfüllen. Beispiel: Ein zerbrechliches Teil darf während der Bewegungsausführung mit einer Kraft pro Fläche von höchstens X N/qcm von der Greiferhand belastet werden.

Ausgehend von diesen Informationen spezifiziert TWAIN, genauer gesagt der zuständige Planungsmodul mögliche Feinbewegungen. Folgende Ergebnisse ergeben sich aus diesem Schritt:

- BEWEGUNGSSTRATEGIE:
  Auf Grund unvermeidbarer Ausführungsfehler, die vor allem bei Feinbewegungen ins Gewicht fallen, werden Überwachungsvorkehrungen getroffen: ein Teil wird nicht an eine Position XYZ befördert, sondern wird solange bewegt, bis eine Zielbedingung (z.B. Fläche A berührt Fläche B) erfüllt ist. Der Planungsmodul bestimmt Bewegungsabläufe, die garantieren, daß die zu manipulierenden Teile nach Ausführung der Bewegung ihre Zielkonfiguration einnehmen; Vorraussetzung dabei ist, daß von einer zulässigen Ausgangskonfiguration gestartet wird.

- STARTPOSITIONEN:
  Gemäß den Einschränkungen, die sich bei Durchführung einer Bewegung etwa durch andere Objekte ergeben, wird die Zahl der möglichen Ausgangspositionen eines Teils, die als Eingangsgrössen geliefert werden, nochmals reduziert.

- INFORMATION für nachfolgende Planphasen:
  Zum Zeitpunkt der Feinbewegungsplanung ist noch kein detailliertes Layout des Arbeitsbereiches spezifiziert. In einer darauffolgenden Planphase, in der dann diese Festlegung erfolgt, werden die aus der Feinbewegungsplanung gewonnenen Daten benutzt, eine möglichst optimale Anordnung der Teile zu bestimmen.

(2) Greifen: Der Greif-Planungsmodul wählt für ein zu ergreifendes Objekt die Oberflächen aus, an denen die Greifhand ansetzen kann, und erstellt für diese Oberflächen eine Greifkonfigurations-Beschreibung.
Dabei müssen Kollisionen zwischen der Greifhand und sich in der Nähe befindlichen Objekten vermieden werden. Es muß weiter sichergestellt sein, daß der Griff stabil genug ist, während der Bewegung wirkende Kräfte aushalten zu können.

Der für die Greifoperationen zuständige Planungsmodul benötigt als Eingabegrössen:

- STARTPOSITION der Greifoperation:
  Die jeweilige Position der Teile zum Startzeitpunkt der Greifoperation muß, eventuell repräsentiert durch eine Planvariable, bekannt sein.

- ZIELPOSITION der Greifoperation:
  Die Position, die ein Teil jeweils unmittelbar nach Ausführung der Greifoperation einnimmt, muß für alle zu ergreifenden Teile gegeben sein.

- TOLERANZBEREICHE:
  Zusätzliche Angaben müssen Aufschluß über mögliche Fehler bezüglich der Objekt-Positionsangaben unmittelbar vor Ausführung der Greifoperation geben.

- CONSTRAINTS:
  An welchen Stellen der Oberfläche ein Teil ergriffen werden sollte und an welchen nicht, kann durch Constraints ausgedrückt werden.

Unter Berücksichtigung dieser Daten und dem Wissen aus dem Weltmodell werden folgende Ausgabegrössen ermittelt:

- GREIFKONFIGURATION:
  Der Planungsmodul legt fest, an welchen Stellen ein Teil ergriffen werden und welche Stellung die Greifhand dazu einnehmen sollte.

- GREIFBEWEGUNGEN:
  Den zu erwartenden Fehlern bei der Ausführung wird insofern entgegengewirkt, daß korrigierende Bewegungen vor Durchführung der Greifoperation eingeplant werden, um falsche Positionierungen von Teilen oder vom Manipulator auszugleichen.

- INFORMATION für nachfolgende Planphasen:
  Die Stellen auf einer Objektoberfläche, an denen ein Teil ergriffen werden sollte, hängen nicht nur vom jeweiligen Teil ab, sondern auch von benachbarten Objekten. Die sowohl vor der Greifoperation als auch danach in der Nähe des betreffenden Teils befindlichen Objekte müssen hierbei berücksichtigt werden. Dementsprechend müssen Lokalisation der Greifstellen eines Teils

und auch das detaillierte Layout des Arbeitsbereiches, das in einer späteren Planphase festgelegt wird, darauf abgestimmt werden.

(3) Grobbewegung: Die Aufgabe des für die Grobbewegung zuständigen Planungsmoduls besteht darin, kollisionsfreie Bewegungen des Manipulators zu ermitteln, um den Greifarm zur Position eines Objektes zu führen, das daraufhin ergriffen werden soll, oder neu aufgenommene Teile mit dem Greifarm von einer Stelle des Arbeitsbereiches zu einer anderen zu befördern. Einzelne zulässige Bewegungen werden dabei zu einer vollständigen Greifarmbewegung zusammengesetzt, die den gestellten Bedingungen genügt.

Der Grobbewegung-Planungsmodul benötigt als Eingangsgrössen:

- START- und ZIELKONFIGURATONEN:
  Die Mengen aller möglichen Ausgangs- und Zielkonfigurationen müssen für alle zu bewegenden Teile bekannt sein.

- TOLERANZBEREICHE:
  Es kann nicht davon ausgegangen werden, daß die angegebenen Positionen der Teile und die einzustellenden Greifarmstellungen genau den Sollwerten entsprechen. Die zu erwartenden Spielräume werden angegeben.

- CONSTRAINTS:
  Die während der Bewegung zu durchlaufende Bahn muß gewissen Bedingungen genügen, z.B. ist die Reichweite eines Greifarms begrenzt.

Die daraus ermittelten Ergebnisse lauten:

- BAHN:
  Eine spezifizierte Bahn legt fest, wie der Greifarm von einer Position zu einer anderen gesteuert werden soll. Im schlimmsten Fall kann keine kollisisonsfreie Bewegung ermittelt werden, dann muß ein neues grobes Layout gewählt werden.

- START- und ZEILKONFIGURATIONEN:
  Durch das Vorhandensein von anderen Objekten kann es notwendig sein, die möglichen Start-

und/oder Zielkonfigurationen für ein Teil einschränken zu müssen, um eine kollisionsfreie Bewegung durchführen zu können.

6.5 Kontrollstruktur

Wie bei TWAIN die bisher beschriebenen Techniken und Hilfsmittel dazu eingesetzt werden, die in einer Aufgabenstellung gegebenen Probleme durch Erstellen eines Planes zu lösen, soll in diesem Abschnitt behandelt werden.

6.5.1 Hierarchisches Zerlegen

Hierarchisches Zerlegen einer Aufgabe in einzelne, möglichst voneinander unabhängige Teilaufgaben bringt im Zusammenhang mit der Roboter-Programmierung wie auch in anderen Problembereichen Schwierigkeiten mit sich: bestehende Abhängigkeiten zwischen Operationen erlauben nur eine beschränkte Zerlegung. Jede Roboter-Operation setzt gewisse Annahmen voraus, die etwa die Position oder die Ansatzpunkte zum Greifen des zu bearbeitenden Teils betreffen. Beispielweise legt die Wahl der Greifpunkte an einem Teil fest, welche Bewegungen des Greifarms mit dem Teil möglich sind und welche nicht. So ist es etwa nicht möglich, ein Teil mit der Fläche, an der der Greifarm ansetzt, ohne Zwischenschritt auf ein anderes stellen zu lassen.

Die Möglichkeit, Daten über Sensoren aufzunehmen und dementsprechend Bewegungsabläufe zu kontrollieren und gegebenenfalls zu korrigieren, trägt dazu bei, daß bestimmte Operationen in einer größeren Zahl von Situationen erfolgreich durchgeführt werden können; die Abhängigkeiten zwischen den Operationen werden dadurch eingeschränkt. Im Extremfall könnte jede Operation die vollständige, aktuelle Arbeitssituation erfassen und dementsprechend gänzlich unabhängig von anderen Aktionen entscheiden, welche Maßnahmen getroffen werden müssen, um die jeweilige Teilaufgabe zu lösen. Dies wäre allerdings extrem aufwendig, denn ständiges Zurück- bzw. Umgreifen, Neupositionieren etc. wären die Folge.

Ganz allgemein kann man festhalten, daß Entscheidungen auf der "unteren" Ebene (z.B. Greifpunkte bestimmen, Abstand der Greiffinger festlegen) als Folge der Planung eines Schrittes auf der "höheren" Ebene, der Task-Ebene, (Layout festlegen, Ergreifen, Bahnen bestimmen, Daten erfassen) die Entscheidungen beeinflussen, die in

vorausgehenden oder nachfolgenden Planschritten gefällt werden müssen. Eine Operation hat also immer unmittelbare Auswirkungen auf ihre "Nachbaroperationen" im Plan, zeitlich gesehen.

Aus diesem Grunde wurde für TWAIN ein Ansatz gewählt, bei dem auf zwei Ebenen geplant wird. Auf einer übergeordneten Konfigurierungsebene werden in Form von Plan-Skeletten Lösungsschemata bestimmt, die auf der eigentlichen Problemebene verfeinert werden, indem die Plan-Skelette angemessen instantiiert werden.

Als Ausdrucksmittel der zwischen den Planschritten bestehenden Interaktionen werden Constraints (siehe Abschnitt 6.3.2) formuliert, die die Zahl der möglichen Verfeinerungen der einzelnen Operatoren einschränken. Bei der Fülle von möglichen Verfeinerungsschritten einer abstrakten Roboter-Operation läßt sich dennoch ein Unterbeschränkungsvorgehen (Optionen offen halten; endgültige Entscheidungen hinausschieben, solange es geht) nicht in der Weise wie z.B. bei MOLGEN praktizieren. Einige wenige Festlegungen müssen bereits auf abstrakter Ebene getroffen werden. Da dabei die Gefahr besteht, falsche Instantiierungen vorzunehmen, was sich erst in darauffolgenden Schritten herausstellen kann, stellt TWAIN ein Backtracking-Mechanismus zur Verfügung, der es ermöglicht, getroffene Entscheidungen wieder rückgängig zu machen.

Unter Berücksichtigung der Constraint-Informationen, die über den ganzen Plan propagiert werden, wird jeder Planschritt verfeinert, wobei die Reihenfolge vom jeweiligen Typ der Operation abhängt. Die unterschiedlichen Operationen werden verschieden stark von an anderer Stelle im Plan gefällten Entscheidungen beeinflußt. Die Planung von Grobbewegungsoperationen wird auf den Schluß aufgeschoben, da derartige Aktionen kaum von anderen beeinträchtigt werden. Von während der Planung gefällten Entscheidungen stärker abhängige Operationen, z.B. Feinbewegung oder Erfassen von Daten, werden im Rahmen der Verfeinerung zunächst berücksichtigt, um das in solchen Fällen sicher öfters anzuwendende Backtracking effizienter zu gestalten. Bei entdeckten Konflikten muß auf diese Weise weniger Planarbeit rückgängig gemacht werden, als wenn die Bearbeitung in umgekehrter Reihenfolge vonstatten ginge.

6.5.2 Die verschiedenen Phasen bei der Planerstellung

Wie die Erstellung eines Planes Schritt für Schritt mit TWAIN aussieht, sei im folgenden etwas näher unter die Lupe genommen (Abb. 6.5.2-1):

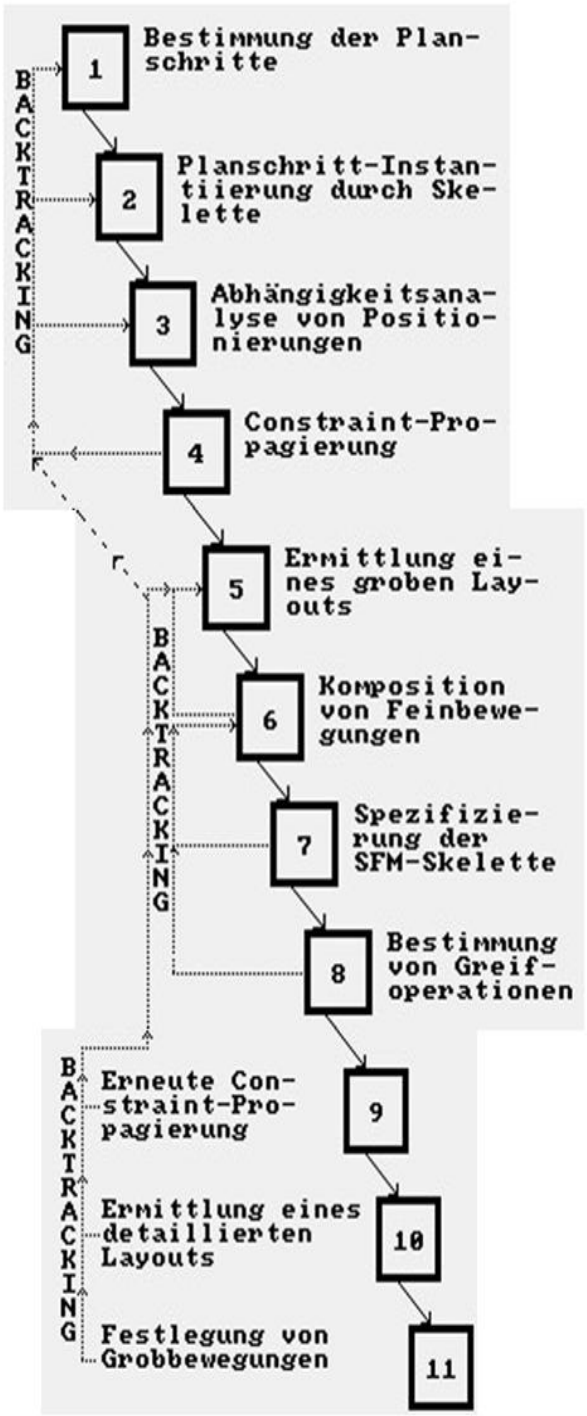


Abb. 6.5.2-1 Phasen bei der Planerstellung mit TWAIN

1. Phase: Bestimmung der Planschritte

Die auf Task-Ebene formulierte Aufgabenstellung spezifiziert eine Reihe von Positionsänderungen von Objekten innerhalb des Arbeitsbereiches. Die daraus ableitbaren durchzuführenden Roboter-Operationen (Grobbewegung, Feinbewegung, Greifen und Loslassen von Teilen) bilden die ersten Planschritte auf abstraktester Ebene.

Der Ansatz, durch die Angabe einer Folge von Operationen eine durchzuführende Task zu spezifizieren, erweist sich bei diesen Anwendungen als vorteilhafter gegenüber dem, eine Aufgabe durch eine Folge von Modellzuständen oder Situationen zu spezifizieren, da ersterer vollständig ist und keinerlei zusätzlicher Angaben von irgendwelchen Parameterwerten bedarf, um die Art und Weise des Erreichens einer Zielsituation durch Ausführen einer Operation genauer festzulegen.

Anzumerken ist noch, daß die Bearbeitung einer Task in der Regel die Planung mehrerer Bewegungsabläufe erforderlich macht.

2. Phase: Planschritt-Instantiierung durch Skelette

Für bereits bekannte Standardfälle liegen Lösungsschemata, Skelette (siehe Abschnitt 6.3.3), in der Wissensbasis von TWAIN vor. Jedes Plan-Skelett repräsentiert einen Planschritt. Die Auswahl eines passenden Skeletts übernimmt der Rahmen-Ermittler-Modul.

Für Grobbewegungs- und Greifoperationen gibt es ein allgemeineres Plan-Skelett, für Feinbewegungsoperationen liegen spezifische Skelette, wie z.B. "BOLZEN_IN_BOHRUNG" (siehe Abschnitt 6.3.3.1), vor. Der allgemeinere Rahmen SFM (synthesized fine motion oder zusammengesetzte Feinbewegung) wird bei solchen Feinbewegungsoperationen ausgewählt, für die keine spezifischeren Skelette vorhanden sind.

3. Phase: Abhängigkeitsanalyse von Positionierungen

An welchen Stellen ein Objekt positioniert werden kann, hängt u.a. davon ab, wie die Positionen im Arbeitsbereich der anderen Objekte lauten. Diese Abhängigkeiten gilt es zu analysieren und festzuhalten (--> Constraints).

4. Phase: Constraint-Propagierung

Die aus den Objekt-Lokalisationen sich ergebenden Constraints und die sog. Anwendbarkeits-Constraints, welche die Vorbedingungen zum erfolgreichen Ausführen

einer Operation spezifizieren, werden durch den gesamten Plan von einem Planschritt zum anderen weitergereicht (Aufgabe des Constraint-Propagierer-Moduls). Können bestimmte Bedingungen nicht erfüllt werden, müssen beim soweit instantiierten Plan wieder Modifikationen vorgenommen werden, Backtracking ist erforderlich. Unter Umständen ist es nötig, gewisse Teile der Planausführung mittels Sensoren zu überwachen. Derartige Maßnahmen müssen bei der Planung bereits berücksichtigt werden.

Solche Feinbewegungsoperationen, für die spezifische Plan-Skelette vorliegen, bilden insofern einen Sonderfall, daß bei diesen nach fehlgeschlagenem Backtracking der Rahmen-Ermittler erneut aktiviert wird, um ein neues Plan-Skelett auszuwählen, und daraufhin wird erneut die Constraint-Propagierung durchgeführt.

5. Phase: Ermittlung eines groben Layouts

Gemäß den bis zu diesem Zeitpunkt formulierten Constraints läßt sich bereits ein grobes Layout bestimmen. Der Arbeitsbereich wird in Zonen eingeteilt, in denen SFM-Operationen mit unterschiedlichen Unsicherheiten, die sich z.B. aus unterschiedlich gestrecktem Greifarm oder zu überwindenden Hindernissen ergeben, ausgeführt werden können. Dementsprechend werden in späteren Planphasen unterschiedliche, vom Bereich, in dem die Durchführung stattfinden soll, abhängige Strategien für SFM-Operationen bestimmt.

6. Phase: Komposition von Feinbewegungsoperationen

Aus den in Phase 5 gewonnenen Informationen wird eine Synthese von durchzuführenden Feinbewegungen berechnet.

Dabei wird so vorgegangen, daß zunächst untersucht wird, ob das jeweils betrachtete Teilziel bereits durch Ausführen einer einzigen Feinbewegungsoperation erreicht werden kann oder ob dafür eine Sequenz von Feinbewegungen nötig ist. Falls letzteres der Fall ist, wird rekursiv weitergemacht: die Menge K1 aller Konfigurationen, von denen aus eine Zielkonfiguration aus der Menge Z1 der möglichen Zielkonfigurationen nach Ausführung einer Feinbewegung F1 erreicht werden kann, wird als neue Menge Z2 der Zielkonfigurationen aufgefaßt. Nun gilt es, alle die Konfigurationen K2 zu bestimmen, von denen aus nach Ausführung einer Feinbewegung F2 eine Konfiguration aus der neu definierten Menge der Zielkonfigurationen Z2 erreicht werden kann - usw. Auf diese Weise werden solange Feinbewegungen aneinandergereiht, bis eine Konfiguration aus Ki vorliegt, von der aus eine Zielkonfiguration aus Zi nach Ausführung einer Feinbewegung Fi erreicht werden

kann, wobei die besagte Konfiguration aus Ki auch in der Menge der möglichen Startkonfigurationen liegt. Dann ist das zu erreichende Teilziel erfüllt. Als SFM (= zu Feinbewegung) ergibt sich: Fi, Fi-1, ... , F2, F1. Wird keine derartige Synthese ermittelt, muß ein neues grobes Layout (--> Phase 5) entworfen werden.

7. Phase: Spezifizierung der SFM-Skelette

Innerhalb der SFM-Skelette werden nun unter Berücksichtigung propagierter Constraints konkrete Feinbewegungsoperationen instantiiert. Das in Phase 6 ermittelte Ergebnis dient dabei als Grundlage. Ergeben sich unerfüllbare Bedingungen, so daß die Planerstellung nicht fortgesetzt werden kann, müssen in Phase 6 andere Kompositionen von Feinbewegungsoperationen ermittelt werden.

8. Phase: Bestimmung von Greifoperationen

Um ein Objekt von einem Roboter greifen zu können, bedarf es vielerlei Informationen wie Oberflächenbeschaffenheit (--> Greifdruck), äußere Abmessungen (--> Greiffingerweite), Masse (--> Greifdruck) und dergleichen mehr. An welchen Stellen eines Teils ein Greifpunkt lokalisiert sein darf, hängt des weiteren davon ab, wie die Start- und Zielkonfigurationen des zu ergreifenden Teils lauten.

Unter Berücksichtigung aller dieser Faktoren wird in dieser Planphase für jedes Objekt, das im Rahmen der Planausführung ergriffen werden soll, eine Liste aller möglichen Greifkonfigurationen erstellt. Kann bei nur einem Objekt kein geeigneter Eintrag in die zugehörige Liste erfolgen, wird die Planerstellung wieder bei Phase 6 fortgesetzt.

9. Phase: Erneute Constraint-Propagierung

Constraints werden wieder in beide Richtungen im Plan weitergereicht, was zum Einführen neuer Planvariablen führen kann. Auch kann in diesem Stadium erkannt werden, daß bei weiteren Operationen Überwachungsvorkehrungen getroffen werden müssen.

Die Instantiierungen der einzelnen Planschritte haben mittlerweile detailliertere Formen angenommen.

10. Phase: Ermittlung eines detaillierten Layouts

Die konkreten physikalischen Positionen, die die einzelnen Objekte zu Beginn der Planausführung einnehmen, sowie alle die sich nach Durchführung von Bewegungsoperationen ergebenden Nominal-Positionen eines jeden Objekts werden festgelegt. Gelingt dies nicht, so müssen ab einschließlich Phase 5 (Ermittlung eines groben Layouts) nochmals alle Phasen der bisherigen Planerstellung durchlaufen werden.

11. Phase: Festlegung von Grobbewegungen

Schließlich werden die Planvariablen instantiiert, die sich auf die gröberen Bewegungsabläufe beziehen. Wie bereits an anderer Stelle erwähnt, hängen diese Grössen kaum von anderen im Plan gefällten Entscheidungen ab und werden deshalb an den Schluß der Planerstellung "verbannt".

Es wird spezifiziert, wie Objekte von einem Ort zu einem entfernteren innerhalb des Arbeitsbereiches bewegt werden. Auch wird festgelegt, wie der Manipulator zwischen auseinanderliegenden "Arbeitsplätzen" innerhalb des Arbeitsbereiches gesteuert werden muß. Generell muß das Planungssystem dabei beachten, kollisionsfreie Bewegungen zu planen. Gelingt dies nicht, müssen je nach Ursache des Scheiterns mehr oder weniger viele vorausgegangene Planerstellungsphasen von neuem durchlaufen werden.

6.6 Beurteilung von TWAIN

Bei der Erstellung von Plänen, die von Robotern ausgeführt werden sollen, stellt sich das Problem, die Aufgabenstellung exakt genug, konsistent und vollständig zu formulieren, daß die gewünschten Ziele eindeutig definiert sind. Bei Problemen aus der realen Welt sind dafür viele Detailangaben bezüglich zu manipulierender Objekte und ausführender Manipulatoren notwendig. Der Benutzer wäre überfordert, müßte er bei jeder Problemstellung auf derartige, oftmals konstante Grössen achten.

Bei TWAIN wurde ein anwendungsfreundlicher Ansatz gewählt. Der Benutzer kann sich bei der Aufgabenstellung auf die wesentlichen Punkte konzentrieren. Lediglich durch Angabe der Planziele ist es möglich, ein Roboter-Programm zu generieren. Nötiges Detailwissen stellt die Wissensbasis zur Verfügung, der Benutzer wird damit nicht konfrontiert. Dadurch wird das Planungssystem

unabhängig vom Roboter, der die Planschritte ausführen soll. Es muß nur ein "Wissensblock" für den oder die eingesetzten Roboter in der Wissensbasis enthalten sein, der sämtliche roboter-spezifischen Daten wie z.B. Tragfähigkeit, Gelenkzahl oder Greifarmreichweiten bereitstellt. Inwieweit TWAIN beim Einsatz mehrerer Manipulatoren paralleles Arbeiten unterstützt, kann der Literatur [1] nicht entnommen werden, doch es spricht nichts dafür, warum dies bei dem gewählten Ansatz nicht gut zu lösen sei.

Eine umfangreiche Wissensbasis soll das Planungssystem "intelligent" genug machen, eine abstrakte Aufgabenformulierung zu verstehen und entsprechende konkrete Aktionen zu planen. Dies setzt allerdings voraus, daß die Wissensbasis genügend Information bereitzustellen vermag. Scheitert die Planerstellung, ohne eine entsprechende Mitteilung zu liefern, weiß der Benutzer nicht, ob der Fehler bei ihm, d.h. bei seiner Aufgabenformulierung, oder beim System liegt. Es müßte die Möglichkeit ausgeschlossen werden können, daß ein Planungsmodul im Laufe der Planverfeinerung seine Ausführungen ohne Angabe eines Grundes abbricht.

Bei der in den vorausgegangenen Abschnitten erfolgten Beschreibung und Analyse von TWAIN fällt auf, daß keine vollständige Trennung des Planungssystems in domanen-unabhängige Inferenz- und Planungsmechanismen und anwendungsbezogene Teile vollzogen werden konnte. TWAIN ist auf Grund seiner Architektur und der verwendeten Lösungsstrategien domänen-fixiert. Andere Systeme wie MOLGEN (siehe Kap. 3) und vor allem PLAKON (siehe Kap. 7) bieten die Möglichkeit, für verschiedene Anwendungsbereiche eingesetzt werden zu können, indem eine domänen-unabhängige Planungsebene über den eigentlichen Problembereich gelegt wird. Aus Effizienzgründen wurde bei TWAIN diese Möglichkeit weitestgehend eingeschränkt. Dafür ist dieses Planungssystem auf die speziellen Probleme bei der Roboter-Programmierung zugeschnitten.

Eine Technik, die in TWAIN verwendet wird und effiziente Planerstellung ermöglicht, ist die Verwendung von Plan-Skeletten. Der Vorteil liegt darin, daß nicht stur nach einer bestimmten Strategie zur Planung einer Aktion vorgegangen wird, sondern bereits existierende Lösungsschemata genutzt werden, um ohne vergrößerten Planungsaufwand individuell auf die einzelnen Teilprobleme eingehen zu können. Zusätzlich bietet sich die Möglichkeit, das System durch weitere Plan-Skelette zu erweitern und noch "intelligenter" zu machen, den gestellten Anforderungen, wenn auch mehr oder weniger domänen-fixiert, anzupassen. Das System ist somit ausbau- und verbesserungsfähig, nachdem es bereits eingesetzt worden ist.

Die Kehrseite der Medaille ist die, daß das Planen mit Hilfe solcher Skelette nur dann einen Vorteil bringt, wenn zu jeder Situation, was kaum der Fall sein wird, auch ein angemessenes Skelett vorhanden ist. Ist letzteres nicht der Fall, so erfolgt die (Teil-)Planerstellung nicht nach einer optimalen Strategie.

Auch eine effizienzsteigernde Wirkung hat die Planung in Abstraktionsstufen, wie es bei TWAIN der Fall ist. Die wesentliche Planarbeit kann bereits auf abstrakter, überschaubarer Ebene geleistet werden. Auf Grund von zu erwartenden Ausführungsfehlern oder zur Erfüllung von Constraints notwendiges Umplanen kann auf einer Ebene mit noch abstrakten Operatoren vollzogen werden; im günstigen Fall unter Beibehaltung des Planes auf den abstrakteren Stufen.

Noch eine abschließende Bemerkung zur Verwendung von Constraints: wie bei anderen Planungssystemen gehen auch bei TWAIN Constraints, statisch oder dynamisch formuliert, als zu erfüllende Randbedingungen in den Planungsprozeß mit ein. Durch Propagierung ermöglichen sie einen Informationsaustausch zwischen den einzelnen Teilplänen. Bei TWAIN kann man dabei wirklich von einem Kommunikationsmechanismus sprechen, da sowohl Vorwärts- als auch Rückwärtspropagierung im Plan möglich ist. Zum Vergleich: Bei MOLGEN ist dies nur für eine Richtung implementiert worden.

Kapitel 7 : Planungssystem-Shell PLAKON

PLAKON [3, 4, 7] soll als Planungssystem-Shell dazu dienen, verschiedene Techniken, die jede für sich in bereits existierenden Systemen einzeln eingesetzt werden, zur Bewältigung von Konstruktionsaufgaben, also Planungs- und Konfigurierungsaufgaben, verfügbar zu machen, indem es ermöglicht, auf einer übergeordneten Planungsebene die "Weichen" für die unterschiedlichen Strategien für den Konstruktionsvorgang zu stellen. Um dieser Aufgabe gerecht zu werden, kann auf entsprechendes Kontrollwissen zugegriffen werden.

PLAKON bildet ein domänen-unabhängiges Kernsystem, das durch Hinzugabe von speziellem Wissen aus einem konkreten Problembereich zum Aufbau von Anwendungsexpertensystemen im Bereich der Planungs- und Konfigurierungsaufgaben, vor allem in technischen Anwendungen dienen soll.

7.1 Mögliche Planungstechniken in PLAKON

Bei der Betrachtung möglicher Planvorhaben fällt sofort ihre hohe Komplexität ins Auge. Der Grund dafür ist, daß Problematiken betrachtet werden, in denen eine große Vielfalt von Zwecken und Einzelaspekten zu berücksichtigen ist. Im Unterschied zum Erstellen einer Diagnose, wo aus einer fest vorgegebenen und mehr oder weniger genau umschriebenen Menge nach gewissen Kriterien ein Element ausgewählt wird, bedarf es bei einem Planverfahren der Erstellung eines im Prinzip beliebig komplexen Objekts, welches auch gewissen Kriterien zu genügen hat, wobei eine Menge von Elementarbausteinen (Operatoren, Objekte) und eine Menge von Konstruktionsmöglichkeiten (Kombinationen von Operatoren und Objekten) gegeben sind.

Um nun dieser komplexen Aufgabe einer Planerstellung gewachsen zu sein, genügt es nicht, einzelne individuelle Planungsaufgaben zu beherrschen, vielmehr müssen auch die Interaktionen der einzelnen Komponenten erfaßt werden. Des weiteren erweist es sich als vorteilhaft, verschiedene Teilziele nach eigens dafür zugeschnittenen Lösungsstrategien zu erreichen und nicht starr auf einer Strategie zu beharren. Zum Beispiel

plant ein hierarchischer, nichtlinearer Planer, unabhängig vom vorliegenden Problem, immer nur hierarchisch nichtlinear.

Bei den in den vorigen Kapiteln vorgestellten Planungssystemen spiegelte zum Teil die Kontrollarchitektur gewisse Aspekte wider, die in beschränktem Maße für eine Flexibilität der Ablaufsteuerung sorgen; z.B. bei MOLGEN: Bereitstellung von vier Strategieoperatoren (siehe Kapitel 3). Mit PLAKON wird dieser Ansatz weiterentwickelt. Wie gesagt, ist PLAKON selbst ein "leeres" System, kein selbständiger Planer. Es bedarf eines Wissensingenieurs (Knowledge-Engineer), eines Spezialisten mit sowohl softwaretechnischem, als auch Anwendungswissen, eine Begriffshierarchie zu formulieren, die zum einen das Domänenwissen repräsentiert und zum anderen die Menge aller zulässigen Konstruktionen beschreibt.

Die Auswahl eines Konstruktionsschritts in einer bestimmten Situation erfolgt gemäß der mit der Begriffshierarchie zusätzlich zum Domänenwissen verfügbaren Information bezüglich bestehender Beziehungen zwischen den einzelnen Objekten in der Wissensbasis. Dabei wird Top-Down-Vorgehen (Dekomposition; --> mehrstufiges Planen) und Bottom-Up-Vorgehen (Aggregation; --> einstufiges Planen) gleichermaßen unterstützt. Es wird dadurch ermöglicht, opportunistisch immer an der erfolgversprechendsten Stelle der vorliegenden Teilkonstruktion weiterzuplanen.

Durch Angabe eines Phasenablaufplanes können die Strategien angegeben werden, nach denen PLAKON in den einzelnen Phasen des Konstruktionsvorganges jeweils planen soll. PLAKON ist also nicht auf eine spezielle Planerstellungsmethode fixiert. Ein Teil der statischen Wissenbasis liefert das nötige statische Kontrollwissen über einzuschlagende Strategien. Man kann folglich nicht sagen, PLAKON gebraucht diese oder jene Technik. Eine Methode ist für PLAKON in dem Moment verfügbar, wenn das nötige domänen-unabhängige Strategiewissen der Wissensbasis zugeführt worden ist und eine entsprechende Referenz im Phasenablaufplan erscheint. Der modulartige Aufbau, den PLAKONs Architektur aufweist, ermöglicht nachträgliche Erweiterungen z.B. puncto neuer Strategien oder neuer Konfliktlösungsmethoden.

Constraints werden bei PLAKON dazu eingesetzt, Abhängigkeiten zwischen Objekten auszudrücken und somit als Nebenbedingungen in den Konstruktionsvorgang miteinzugehen. Sowohl die Technik der Constraint-Propagierung, die u.a. dynamisches Formulieren von Constraints ermöglicht, wie auch die der Verknüpfung von Constraints zu Constraint-Netzen finden bei PLAKON Anwendung.

Schließlich unterstützt PLAKON "intelligentes" Backtracking und ein sog. Truth Maintenance System zur optimalen Fortsetzung des Konstruktionsvorganges im Konfliktfall; dazu mehr im Abschnitt 7.3.

An dieser Stelle sei noch bemerkt, daß am Ende dieses Kapitels kein ausführliches Planungsbeispiel mit PLAKON aufgezeigt wird, da der Konstruktionsvorgang wesentlich - mehr noch als bei anderen Planungssystemen - vom Domänenwissen mit den zusätzlich enthaltenen Objekt-relationen und dem jeweiligen, vom Knowledge-Engineer formulierten Phasenablaufplan abhängt. Erläuternde Beispiele für einzelne Aspekte und Techniken in PLAKON werden allerdings in den folgenden Abschnitten angeführt.

7.2 Anwendungsbereiche von PLAKON

PLAKON wird an der Universität Hamburg in Zusammenarbeit mit großen Firmen wie Philips und Siemens entwickelt. Das deutet darauf hin, daß der Einsatzbereich nicht auf wissenschaftlicher Ebene beschränkt sein soll. Vielmehr soll PLAKON für konkrete Planungs- und Konfigurierungsaufgaben eingesetzt werden. Wie eingangs schon erwähnt, bezeichnet Planen die Tätigkeit, zur Lösung eines Problems Aktionen zusammenzustellen, die, wenn man sie ausführt, das Problem lösen. Konfigurierung ist der Vorgang, bei dem aus einzelnen Komponenten ein Gesamtsystem zusammengestellt wird. Beide Aufgabenbereiche unterscheiden sich insofern, daß es zwar bei beiden darum geht, immer aus Einzelkomponenten eine Gesamtheit zusammenzustellen, nur daß bei der Erstellung eines Planes Planschritte (Operatoren, Planteile) und bei der Konfigurierung Bauteile (z.B. Schaltelemente) die genannten Komponenten, die sogenannten Konstruktionsobjekte bilden. Mit dem Begriff Konstruktion werden beide Tätigkeiten zusammengefaßt.

Anwendungsbeispiele für PLAKON wären etwa:

- Planung mechanischer Bearbeitungs- und Fertigungs-prozesse, z.B. Generierung von Arbeitsplänen in der mechanischen Teilefertigung.

- Konfigurierung von elektronischen Aggregaten aus Standardkomponenten.

Mit PLAKON soll dem Anwender eine offene, d.h. erweiterbare Systemarchitektur mit einem speziell auf die genannten Aufgabenbereiche abgestimmten Inferenz-mechanismus zur Verfügung gestellt werden.

7.3 Die Repräsentation von Wissen in PLAKON

Um die Anforderungen bei konkreten Konstruktionsaufgaben zufriedenstellend bewältigen zu können, muß PLAKON zum einen über umfassende Repräsentationsmöglichkeiten für Konstruktionsobjekte verfügen, zum anderen braucht es eine flexible Ablaufsteuerung zur Realisierung typischer Konstruktionsstrategien.

Das für einen bestimmten Problembereich nötige Domänenwissen wird deklarativ, d.h. ohne Angaben über die Entstehung und die Verwendung, durch eine Begriffshierarchie dargestellt, die zwei Ebenen aufweist:

(1) taxonomische Hierarchie,
(2) kompositionelle Hierarchie.

Konstruktionsobjekte werden durch Frames repräsentiert. Objektdeskriptoren in deren Slots geben die zulässigen Wertebereiche der betreffenden Eigenschaften oder Parameter an.

Man kann sich diese Begriffshierarchie so vorstellen, daß es sich hierbei um zwei Graphen handelt. Die Knoten bilden die Konstruktionsobjekte, also die Objekte der Anwendungsdomäne. Die Kanten bezeichnen unterschiedliche Relationen: beim "taxonomischen Graphen" stehen die Kanten für eine "ist_ein" Relation, beim "kompositionellen Graphen" für eine "ist_Teil_von"- bzw. "hat_als_Teile"-Relation. Welche Hierarchie zu welchem Zweck dient, soll gleich näher untersucht werden.

Zuerst noch das Beispiel einer Begriffshierarchie, zumindest eines Ausschnitts davon. Als Beispieldomäne sei, wie auch bei den folgenden Beispielen, die Konstruktion eines Autos gewählt (Abb. 7.3-1):

Konstruktionsobjekt	Relation	(Richtung)	[Anzahl]	Konstruktionsobjekt
Auto	ist_ein	(<->)		**Konstruktionsobjekt**
Auto	hat_als_Teile	(->)		Motor
Auto	hat_als_Teile	(->)		Karosserie
Auto	hat_als_Teile	(->)		Fahrgestell
LKW	ist_ein	(<->)		**Auto**
LKW	hat_als_Teile	(->)		Diesel-Motor
LKW	hat_als_Teile	(->)		...
PKW	ist_ein	(<->)		**Auto**
PKW	hat_als_Teile	(->)		...
Diesel-Motor	ist_ein	(<->)		**Motor**
Diesel-Motor	ist_Teil_von	(->)		LKW
Spar-Motor	ist_ein	(<->)		**Motor**
Renn-Motor	ist_ein	(<->)		**Motor**
Renn-Motor	hat_als_Teile	(->)		Einspritzung
Renn-Motor	hat_als_Teile	(->)	[4..6]	Zylinder
Renn-Motor	hat_als_Teile	(->)		Turbolader
Motor	ist_ein	(<->)		**Autoteil**
Motor	ist_Teil_von	(->)		Auto
Motor	hat_als_Teile	(->)	[2..8]	Zylinder
Motor	hat_als_Teile	(->)		Benzinaufbereitung
Motor	hat_als_Teile	(->)	[0..1]	Turbolader
Autoteil	ist_ein	(<->)		**Konstruktionsobjekt**
Karosserie	ist_ein	(<->)		**Autoteil**
Karosserie	ist_Teil_von	(->)		Auto
Karosserie	hat_als_Teile	(->)	[2..5]	Türen
Fahrgestell	ist_ein	(<->)		**Autoteil**
Fahrgestell	ist_Teil_von	(->)		Auto
Fahrgestell	hat_als_Teile	(->)	[2..4]	Achse
Fahrgestell	hat_als_Teile	(->)	[4..8]	Rad
Einspritzung	ist_ein	(<->)		**Benzinaufbereitung**
Einspritzung	ist_Teil_von	(->)		Renn-Motor
Zylinder	ist_ein	(<->)		**Motorteil**
Zylinder	ist_Teil_von	(->)	[4..6]	Renn-Motor
Zylinder	ist_Teil_von	(->)	[2..8]	Motor
Benzinaufbereitung	ist_ein	(<->)		**Motorteil**
Benzinaufbereitung	ist_Teil_von	(->)		Motor
Turbolader	ist_ein	(<->)		**Motorteil**
Turbolader	ist_Teil_von	(->)	[0..1]	Motor
Turbolader	ist_Teil_von	(->)		Renn-Motor
Motorteil	ist_ein	(<->)		**Autoteil**
Achse	ist_ein	(<->)		**Fahrwerksteil**
Achse	ist_Teil_von	(->)	[2..4]	Fahrgestell
Rad	ist_ein	(<->)		**Fahrwerksteil**
Rad	ist_Teil_von	(->)	[4..8]	Fahrgestell
Rad	hat_als_Teile	(->)		Felge
Rad	hat_als_Teile	(->)		Reifen
Felge	ist_ein	(<->)		**Fahrwerksteil**
Felge	ist_Teil_von	(->)		Rad
Reifen	ist_ein	(<->)		**Fahrwerksteil**
Reifen	ist_Teil_von	(->)		Rad

Abb. 7.3-1 Begriffshierarchie (Ausschnitt) für die Autodomäne

(Hinweis: - Richtungsangaben bei den Relationen redundant
- Konstruktionsobjekte, von denen aus die Kanten weggehen, sind fett gedruckt)

7.3.1 Taxonomische Begriffshierarchie

Die taxonomische Hierarchie innerhalb der Begriffshierarchie, die einen Teil der statischen Wissensbasis von PLAKON bildet, wird durch die "ist_ein"-Relation dargestellt. Sie ermöglicht, Objekte zu typisiseren (z.B. RENN-MOTOR ist_ein MOTOR und DIESEL-MOTOR ist_ein MOTOR) und Spezialisierungen von Objekten zu beschreiben und deren Eigenschaften festzulegen (z.B. ZYLINDER ist_ein MOTOR-TEIL, MOTOR-TEIL ist_ein AUTOTEIL, AUTOTEIL ist_ein KONSTRUKTIONSOBJEKT). Durch diesen Ansatz wird die Technik der dynamischen Spezialisierung ermöglicht, bei der während des Konstruktionsvorgangs an Hand dieser Spezialisierungshierarchie Objekte bereits nach teilweiser Parametrierung zu einem mehr spezifizierten Objekt spezialisiert werden können; ein Beispiel dazu: ein Auto(-Objekt), dem nach weiterer Bearbeitung ein Diesel-Motor(-Objekt) zugeordnet wird, kann danach zu einem LKW(-Objekt) spezialisiert werden.

7.3.2 Kompositionelle Begriffshierarchie

Die "ist_Teil_von"- bzw. "hat_als_Teile"-Relation, die die Grundlage für diese Hierarchie bildet, unterstützt das Top-Down-Vorgehen beim Konstruktionsvorgang. Damit können Objekte der Wissensbasis in ihre Komponenten zerlegt werden. Die Slots der Objekt-Frames enthalten Verweise auf die Teilkomponenten. Dadurch wird das Konzept der Abstraktion in den Konstruktionsvorgang miteinbezogen. Das nötige Wissen, wie die abstrakten Objekte näher zu spezifizieren sind, kann aus der Begriffshierarchie abgelesen werden.

7.3.3 Verwendung von Constraints

Auch bei PLAKON werden Constraints dazu verwendet, die Interaktionen zwischen den einzelnen Teilaufgaben in den Griff zu bekommen, d.h. möglichst früh zu erkennen, und daraufhin den weiteren Konstruktionsprozeß so abzuändern, daß die entdeckten Abhängigkeiten zum Beispiel durch Abändern der Reihenfolge, in der die Teilaufgaben bearbeitet werden, möglichst umgangen oder zumindest abgeschwächt werden.

Die Constraints drücken bestehende Abhängigkeiten zwischen Konstruktionsobjekten aus der Wissensbasis aus, indem sie an die betreffenden Objekte bzw. deren Slots gebunden werden. Sie gehen als Randbedingungen in den Konstruktionsvorgang ein. Ein Beispiel:

Die Beziehung

"Der Gesamthubraum eines Motors ist gleich dem Produkt aus Zylinderzahl und Hubraum der einzelnen Zylinder."

wird in PLAKON durch folgendes Constraint repräsentiert:

```
(CONSTRAINT ((?M (ein Motor))
             (?Z (ein Zylinder  (ist_Teil_von ?M))))
 (MULTIPLY (?Z Hubraum)
           (?M Anzahl)
           (?M Gesamthubraum))).
```

Bei PLAKON haben Constraints zweierlei Aspekte: zum einen sind sie ein statisches Gebilde, da sie bereits zu Beginn des Konstruktionsvorganges in der Wissensbasis, sprich Begriffshierarchie vorliegen, zum anderen zeigen sie dynamisches Verhalten, da sie wie Konstruktionsobjekte instantiiert und an die Slots anderer Objekt-

instanzen gebunden werden und vor allem das Weiterreichen von Slotwerten ermöglichen.

Constraint-Propagierung ermöglicht eine Kommunikation der Werte aus den einzelnen Teilaufgaben. An Hand bereits vorliegender Slotwerte können neue Werte bestimmt und bereits existierende auf Konsistenz bezüglich der bestehenden Abhängigkeiten überprüft werden. Die Gesamtheit der Constraints bildet ein Constraint-Netz, das sämtliche bestehenden Abhängigkeiten innerhalb der aktuellen Konstruktion widerspiegelt.

7.4 Konstruieren mit PLAKON

Die Grundidee ist die, daß eine Konstruktion aus vielen Teilobjekten zusammengesetzt ist.

Ausgangspunkt des Konstruktionsvorganges bildet eine initiale Teilkonstruktion, eine lose Menge von Objekten, die von der Aufgabenstellung gefordert sind (Beispiel: es soll ein Auto konstruiert werden, über das von vorneherein nichts ausgesagt ist außer den beiden Angaben, daß die Motorleistung durch einen Turbolader verbessert werden soll und als Bereifung Gürtelreifen zu verwenden sind). Darauf aufbauend wird schrittweise eine vollständige und korrekte Endkonstruktion zu erstellen versucht.

Der Ablauf des Konstruktionsvorganges wird durch zwei Faktoren bestimmt: Begriffshierarchie und Phasen-ablaufplan. Die Begriffshierarchie steuert den Ablauf auf Objektebene, der Phasenablaufplan legt die jeweilige Strategie zugrunde, ist also für die Kontrollebene über dem Objektbereich zuständig. Im Phasenablaufplan formuliert der Wissensingenieur die Reihenfolge der abzuarbeitenden Phasen, in die er den Konstruktions-vorgang eingeteilt hat, und das für jede Phase spezifische Kontrollwissen, für jede Phase wird die entsprechende Strategie angegeben. Des weiteren gehören zu jeder Phase sogenannte Auswahlregeln, mit denen Reihenfolgewissen innerhalb einer Phase ausgedrückt werden kann. Eine derartige Regel könnte etwa lauten: "Bearbeite bevorzugt alle Konstruktionsschritte, die den Motor betreffen !".

Von der initialen Teilkonstruktion zur vollständigen Endkonstruktion bedarf es meist vieler Schritte. Das Überführen einer Teilkonstruktion, die Instanzen von Konstruktionsobjekten enthält, in eine andere durch Ausführen eines Konstruktionsschrittes wird als Elaboration bezeichnet.

Während des Konstruktionsvorganges wiederholen sich immer wieder folgende Schritte:

(1) Zuerst wird im Phasenablaufplan "nachgeschlagen", ob eine neue Phase anbricht;

falls ja, wird die nächste Phase begonnen, wobei vorher noch die Strategie festgelegt wird, indem auf das im Phasenablaufplan formulierte spezifische Kontrollwissen und das statische Kontrollwissen in PLAKONs Wissensbasis zugegriffen wird;

falls nein, wird der Konstruktionsvorgang ohne Kontrollmodifikationen fortgesetzt.

Zu bemerken ist noch, daß solange nach einer jeweiligen Strategie vorgegangen wird, bis ein entsprechender Hinweis im Phasenablaufplan erfolgt oder ein Abbruchkriterium eintritt. Dadurch wird der Konstruktionsvorgang schneller, da nicht unbedingt in jeder Phase nach einer neuen Strategie "nachgeschaut" werden muß.

(2) Daraufhin wird die aktuelle Teilkonstruktion bestimmt. Diese ist auf Grund der zusätzlich gegebenen Information der Auswahlregeln, die die Reihenfolge der Durchführung der in einer Agenda eingetragenen Konstruktionsschritte festlegen, und der Beziehung in der Begriffshierarchie meist eindeutig. Sind allerdings etwa die Auswahlregeln unvollständig spezifiziert und mehrere Relationen in der kompositionellen Hierarchie auf selbem Level, müssen an dieser Stelle spezielle Maßnahmen getroffen werden, um den Konstruktionsvorgang fortsetzen zu können. Strategieabhängige Foci, "intelligentes Backtracking" und ein Truth Maintenance System, das die Übernahme aller Entscheidungen, welche von einer geänderten Entscheidung nicht betroffen sind, garantiert, sind die möglichen Konfliktlösungsverfahren.

Letztere beiden Methoden sind vor allem dann wichtig, wenn bei der Analyse von Teilkonstruktionen festgestellt wird, daß das angestrebte Ziel nicht erreicht werden kann. Das dafür nötige Wissen liegt zum einen in der statischen Wissensbasis vor: dort findet sich Information darüber, wie bei den einzelnen Konfliktlösungsmethoden vorgegangen wird und welche der möglichen Verfahren in welcher Situation angewendet wird.

Zusätzliche Information liefert ein dynamisch im Verlaufe des Konstruktionsvorganges aufgebautes Elaborationsnetz, das die gesamten bis zum aktuellen Zustand vorgenommenen Entscheidungen und durchgeführten Schritte repräsentiert: bei jeder Elaboration wird eine Teilkonstruktion mit den sich daraus ergebenden neuen Teilkonstruktionen durch Elaborationskanten verbunden, die Informationen über den durchgeführten Schritt enthalten.

(3) Der nun folgende Schritt besteht darin, die vorliegende Teilkonstruktion zu analysieren und dementsprechend einen geeigneten Konstruktionsschritt auszuwählen.

Dabei hat PLAKON mehrere elementare Konstruktionsschritte zur Auswahl:

- OBJEKT_ZERLEGEN
  (Dekomponieren):

  Ein Objekt wird dadurch zerlegt, daß mindestens eine Komponente, auf die eine "hat_als_Teile"-Kante, ausgehend vom betreffenden Objekt, gerichtet ist, mit einem Wert instantiiert wird (Top-Down-Vorgehen).

  Beispiel: Zerlegung des Objekts AUTO in die Objekte MOTOR, KAROSSERIE und FAHRGESTELL, die alle in der Relation "ist_Teil_von" zu AUTO stehen.

- OBJEKT_SPEZIALISIEREN:

  Soweit es geht, werden Slots, die über "ist_ein"-Kanten mit anderen Objekten verbunden sind, mit entsprechenden Werten versehen, wenn die notwendigen Voraussetzungen erfüllt sind.

  Beispiel: Das Objekt AUTO bekommt im Rahmen des Konstruktionsvorganges einen RENN-MOTOR zugeordnet und kann daraufhin zum Objekt PKW spezialisiert werden.

- KOMPONENTEN_ZUSAMMENFÜGEN
  (Aggregation, Clustering):

  Beim Clustering von Komponenten zum Aufbau eines beliebig komplexen Objektes werden Aggregate (Objekte auf höherer Ebene innerhalb der Begriffshierarchie) instantiiert, sobald ausreichend Teilkom-

ponenten (Objekte auf darunterliegenden Ebenen innerhalb der Begriffshierarchie, die zum Aggregat-Objekt in der "ist_Teil_von"-Relation stehen) vorhanden sind (Bottom-Up-Vorgehen).

Beispiel: Konstruktion eines speziellen MOTORs, ausgehend vom TURBOLADER, der in der Aufgabenstellung gefordert wurde.

- PARAMETERWERTE_FESTLEGEN (parametrieren):

  Gemäß den einschränkenden Bedingungen, die als Constraint-Information propagiert werden, werden Parameterwerte von Objekten festgelegt oder zumindest eingeschränkt. Constraints ermöglichen dadurch die Kontrolle über die Wertzuweisung an die Parameter eines Objekts.

  Beispiel: Im Laufe des Konstruktionsvorganges wird das Objekt AUTO zerlegt in MOTOR und ZYLINDER [6]. Laut Aufagenstellung war ein Auto mit mindestens 3 Liter Hubraum gefordert. Über das bereits formulierte Constraint, das die Größen Gesamthubraum, Zylinderzahl und Hubraum je Zylinder in Beziehung bringt, wird der Parameterwert Hubraum des Objektes ZYLINDER auf >= 0,5 Liter eingeschränkt.

  Zu jedem Konstruktionsschritt-Typen existiert eine Funktion, die untersucht, welche Konstruktionsschritte bei der aktuellen Teilkonstruktion noch durchgeführt werden müssen. Die notwendigen Schritte werden daraufhin in eine Agenda eingetragen.

(4) Vor der Durchführung des Konstruktionsschrittes wird die Agenda durchsucht. Die mittels Auswahlregeln festgelegte Reihenfolge der durchzuführenden Aktionen wird berücksichtigt, wodurch gewährleistet ist, daß der am erfolgversprechendste Schritt ausgewählt wird.

(5) Nun wird der ausgewählte Konstruktionsschritt durchgeführt.

Falls nach den vorausgegangenen Schritten die durchzuführende Aktion noch nicht eindeutig festliegt, kann beim Benutzer nachgefragt, ein Defaultwert eingesetzt oder ein Wert aus einer

bereits erfolgten Konstruktion übernommen werden.

Nach Durchführung des Konstruktionsschrittes wird eine neue Teilkonstruktion gewonnen, die mit einer Elaborationskante verbunden wird.

(6) Schließlich werden die bestehenden Randbedingungen an die neuen Konstruktionsäste über das Constraint-Netz weitergegeben.

7.5 Die Architektur von PLAKON im Überblick

Die folgende Übersicht (Abb. 7.5-1) soll die wesentlichen Komponenten von PLAKON aufzeigen, deren Zusammenwirken in den vorangegangenen Abschnitten bereits beschrieben wurde (genauere Übersicht siehe [3]):

siehe Abb. 7.5-1 auf der nächsten Seite !

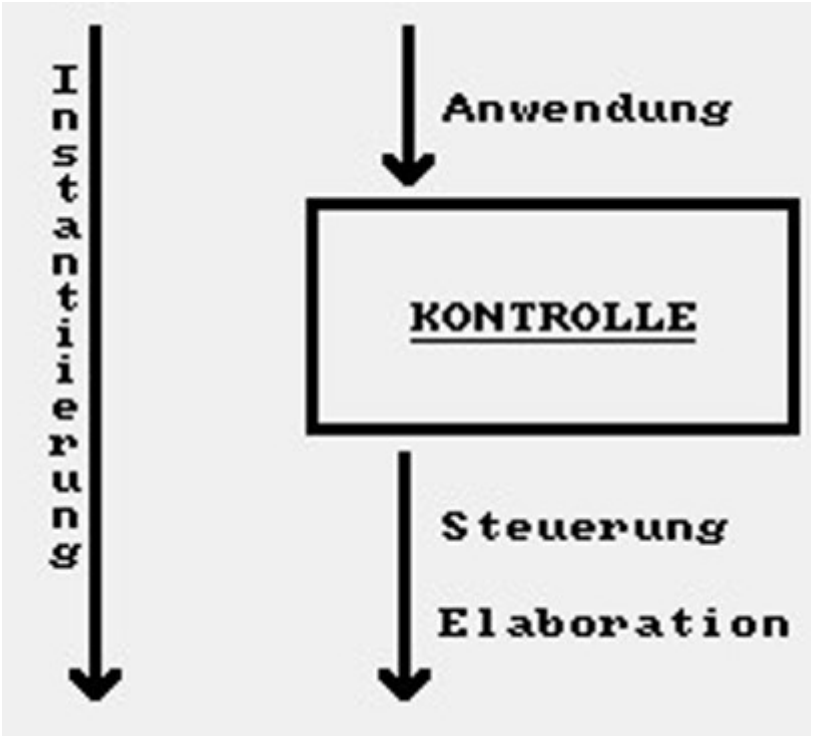
Statische Wissensbasis

BEGRIFFSHIERARCHIE

- Konzepte
- Relationen
- Constraints

STATISCHES KONTROLLWISSEN

- Strategiewissen
- Konfliktlösungsverfahren
- Phasenablaufplan mit Auswahlregeln



Dynamische Wissensbasis

ELABORATIONSNETZ

- Teilkonstruktionen
- Rechtfertigung (für Erklärungskomponente)

CONSTRAINTNETZ

- Constraint-Propagierung

AUFGABENSTELLUNG

- Initiale Teilkonstruktion

AGENDA

- Mögliche Konstruktionsschritte

Abb. 7.5-1 PLAKONs Architektur in der groben Übersicht

7.6 Beurteilung von PLAKON

PLAKON erweist sich durch die flexiblen Eigenschaften als ein System, das auf Grund der verwendeten Inferenzmechanismen am besten für Konstruktionsaufgaben geeignet ist, dabei sind nahezu keine Einschränkungen für die in Frage kommenden Domänen gegeben. Dies ergibt sich daraus, daß PLAKON für jedes Anwendungsgebiet nicht nur Fakten über die betreffenden Konstruktionsobjekte, sondern auch Kontrollwissen erhält, das für den jeweiligen Problembereich die am besten zugeschnittenen Strategien verfügbar macht.

Wie das so häufig der Fall ist, kann ein allgemeines System, wie es eine Planungssystem-Shell darstellt, nicht die Güte eines Spezial-Systems, hier also gemeint eines Planungssystems in einem speziellen Problembereich erreichen. Dies trifft für PLAKON vielleicht nicht für die Anwendung zu, was die Erfahrung erst noch zeigen wird, aber vorab läßt sich sagen, daß das Erstellen der Wissensbasis für PLAKON eine sehr aufwendige Aufgabe ist, die eines Wissensingenieurs bedarf.

Bestrebungen auf dem Gebiet der Künstlichen Intelligenz gehen in die Richtung, das Fachwissen in einem Bereich dem Rechner in möglichst einer Form beizubringen, die für den Experten der betreffenden Domäne, der in der Regel nicht über softwaretechnische Kenntnisse verfügt, leicht nachzuvollziehen und der ihm gewohnten Darstellung von Wissen ähnlich ist.

Bei PLAKON wird zusätzliche Information in die Begriffshierarchie eingebracht, die eine spezielle Aufbereitung und Strukturierung der Konstruktionsobjekte bedarf. Dies wirkt sich zwar positiv auf die Flexibilität des Systems aus, da spezifisches Kontrollwissen in das Domänenwissen integriert wird, erfordert aber größere Mühen, das System mit notwendigen Informationen zu "füttern".

Der Ansatz der Constraint-Propagierung erweist sich, wie schon bei existierenden Systemen, als hilfreich beim frühzeitigen Entdecken und Erfassen von Interaktionen zwischen Teilkonstruktionen.

Das System zeigt interaktive Ansätze, was den Bedienkomfort in jedem Fall erhöht. Falls ein durchzuführender Schritt während des Konstruktionsvorgangs noch nicht eindeutig bestimmt ist, kann der Benutzer während der Ausführung zusätzliche Angaben machen.

Der Phasenablaufplan und damit die Möglichkeit, Reihenfolgewissen in den Konstruktionsvorgang miteinzubeziehen, ermöglicht, gesammelte Erfahrungen, die den Kontrollablauf betreffen, miteinzubringen.

Das System ist in vielerlei Hinsicht ausbaufähig. Die Informationen im Elaborationsnetz ausnützend, kann als Bestandteil einer zusätzlichen Oberflächenkomponente, die die Schnittstelle zwischen dem Benutzer und dem Kern komfortabler gestalten könnte, z.B. bei Stellen der Aufgabe, eine zusätzliche Erklärungskomponente eingeführt werden, die auch bei Scheitern eines Konstruktionsvorganges wichtige Informationen über die Gründe geben könnte.

PLAKON zeigt in einigen Punkten das Verhalten eines Blackboard-Systems [11]. Dies wird zum Teil beim zentralen Zyklus des Konstruktionsvorganges deutlich. Bei diesem Ansatz können komplexe Probleme aufgeteilt und an verschiedene kleine "Unter-Planungssysteme", die als sog. Spezialisten in der statischen Wissensbasis modelliert werden, weitergereicht werden. Bei PLAKON findet sich jedoch eine starke Sequenzialisierung bei Abarbeitung der einzelnen Schritte.

Gewisse Parallelität beim Konstruktionsvorgang zeigt sich bei PLAKON beim kombinierten Top-Down- und Bottom-Up-Vorgehen. Es ist dabei gemäß der Begriffshierarchie möglich, daß, je nach Aufgabenstellung, teils - auf höherer Ebene - top-down, teils - auf unteren Ebenen in der Hierarchie - bottom-up vorgegangen wird. Auch dadurch zeigt sich PLAKON durch Hinzugabe von speziellem Bereichswissen als effizientes Planungssystem, dessen Ablaufsteuerung mehrere Konzepte aufweist und vereint.

Kapitel 8 : Zusammenstellung der betrachteten Planer

In diesem letzten Kapitel sollen noch einmal die Punkte eines jeden in den vorausgegangenen Kapiteln untersuchten Planungssystems hervorgehoben werden, welche die Vorgehensweise des jeweiligen Planers besonders charakterisieren. Um sich davon ein Bild machen zu können, welche Techniken bei den einzelnen Planern neu sind und welche nur eine Weiterentwicklung von bereits bestehenden Methoden darstellen, ist zunächst noch in einer Übersicht angegeben, wann die Systeme entwickelt wurden.

8.1 Chronologie

Die Anordnung der Beschreibungen zu den Planungssystemen MOLGEN, SIPE, TWEAK, TWAIN und PLAKON in dieser Studienarbeit erfolgte nach chronologischen Gesichtspunkten. MOLGEN stellt unter den betrachteten Systemen das älteste dar, PLAKON spiegelt die neuesten Ideen auf diesem Gebiet wider und liegt erst als Prototyp vor. Abb. 8.1-1 illustriert die Entwicklungszeitpunkte bzw. -räume der behandelten Planer.

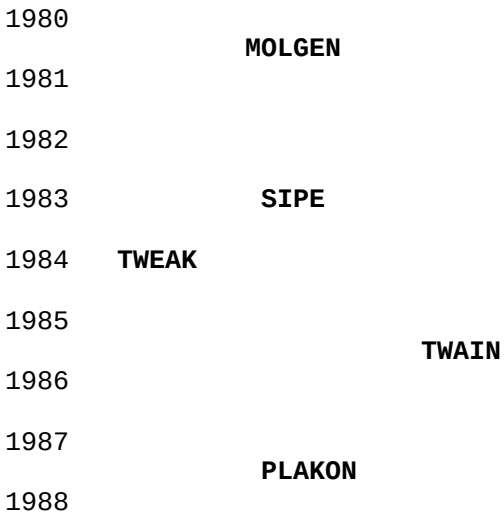


Abb. 8.1-1 Chronologie der untersuchten Planer

8.2 Charakteristisches zu den einzelnen Planern

MOLGEN, der "Klassiker" unter den analysierten Planungssystemen, zeichnet sich durch eine ausgetüftelte Kontroll- und Wissensstruktur in verschiedenen Ebenen aus. Die Idee des Meta-Planens ist verwirklicht. Mit MOLGEN erfolgreich erstellte Pläne beweisen, daß das gleiche System sowohl Aktionen zum Lösen eines Problems zusammenstellen vermag, als auch in der Lage ist zu planen, wie ein Plan entwickelt werden soll. Neu an diesem Ansatz ist auch, daß nicht nur Domänenwissen, sondern auch Strategiewissen aus gewonnenen Erfahrungen eingesetzt wird. Ausgiebig wird von der Technik des Setzens von Constraints Gebrauch gemacht, wobei auch dynamisches Formulieren von Constraints während der Planerstellung möglich ist.

Die Schwachpunkte liegen beim zu großen Umfang der erstellten Pläne und bei einer nur schwer verständlichen Repräsentationssprache, die auch Grund dafür ist, daß MOLGENs Wissen nicht umfassend genug ist, zufriedenstellende Ergebnisse mit größeren Häufigkeiten zu erzielen.

Die Schwerpunkte bei SIPE liegen bei verbesserten Repräsentationsmöglichkeiten. Eine ausdrucksstarke Operatorbeschreibungssprache erlaubt das Erstellen verständlicher, nachvollziehbarer Pläne und dient zur domänen-unabhängigen Repräsentation von Wissen. Operatoren werden mit Hilfe von Attributen näher spezifiziert. Das Operator-Merkmal Ressource soll dazu dienen, erstellte Pläne noch leserlicher und die Repräsentationsmöglichkeiten noch reichhaltiger werden zu lassen. Außerdem stellen Ressourcen mit Hilfe von Constraints ein mächtiges Werkzeug zum Beheben von Interaktionen zwischen Teilproblemen dar. Ein ausgeklügelter Inferenzmechanismus verleiht dem System die Fähigkeit, aus gegebenen Fakten Schlußfolgerungen zu ziehen und somit aus bestehendem Wissen neues Wissen abzuleiten. Um eine schnelle Planausführung zu ermöglichen, können parallele Aktionen im Plan explizit formuliert werden. Bei SIPE wird der Benutzer in den Planungsprozeß miteinbezogen, er kann die Planung kontrollieren und steuern.

Die in SIPE verwendeten Ansätze scheinen in die richtige Richtung zu gehen, sind aber noch zu wenig ausgereift.

TWEAK ist der einzige der betrachteten Planer, der nicht top-down von der Problemstellung aus vorgeht. Nach der Methode des einstufigen Planens als Suche wird eine Operatorfolge zu bestimmen versucht, die eine gegebene Start- in eine erwünschte Zielsituation überführt. Dabei sucht TWEAK rückwärts von einer möglichen Zielsituation

zur gegebenen Startsituation. Constraints sind bei TWEAK ein nicht so mächtiges Werkzeug wie bei MOLGEN, dienen aber auch hier als Ausdrucksmittel für zusätzlich während der Planerstellung zu erfüllende Bedingungen. Der Benutzer hat keinerlei Möglichkeiten, in das Geschehen einzugreifen. Falls eine Domäne nicht zu komplex ist, so daß die sehr beschränkten Repräsentationsmöglichkeiten von TWEAK ausreichen, und das verfügbare Wissen (bezüglich Objekten und Operatoren) umfassend genug ist, zeichnen sich die durch TWEAK erstellten Pläne durch knappen Umfang und prägnante Formulierung aus.

Um dem Ziel gerecht zu werden, korrekt und vollständig zu arbeiten - d.h.: (1) jedes gefundene Ergebnis stellt tatsächlich eine Lösung des gestellten Problems dar und (2), falls ein Problem lösbar ist, ermittelt TWEAK seine Lösung(en) -, sind dem Planer zu viele Einschränkungen auferlegt worden, als daß er in realen Anwendungsbereichen eingesetzt werden könnte.

Auf die Roboter-Domäne am ehesten "zugeschnitten" ist das Planungssystem TWAIN. Der Benutzer wird von der lästigen Aufgabe der umständlichen Roboter-Programmierung befreit. Es bedarf lediglich der Eingabe der zu erreichenden Planziele. Die auf Task-Level abstrakt spezifizierte Aufgabe, die roboter-unabhängig formuliert werden kann, wird dann unter Verwendung einer notwendigerweise sehr detaillierten Wissensbasis in ein von einem Roboter ausführbares Programm transformiert. Strategiewissen wird durch die Verwendung von sog. Plan-Skeletten in die Planung miteingebracht. Dadurch kann nach bekannten Lösungsschemata vorgegangen werden, was ein schnelleres Auffinden des fertigen Planes ermöglicht. Allerdings ist dadurch die Vorgehensweise auf gewisse Arbeitsmuster fixiert.

TWAINs Tauglichkeit in realen Anwendungsbereichen hängt in erster Linie von der Vollständigkeit seiner Wissensbasis (u.a. geometrische und physikalische Objektdaten, Plan-Skelette) ab. Die Repräsentationsmöglichkeiten sind auch hier nicht genügend ausdrucksstark.

PLAKON, das erst als Prototyp vorliegt, soll die bei der Entwicklung vorausgegangener Planer gesammelten Erfahrungen verwerten. Eine Begriffshierarchie stellt die für den Planungsprozeß notwendigen Informationen, die Objekte betreffend, zur Verfügung (vgl. Objekthierarchie bei SIPE). Constraints dienen als Ausdrucksmittel für Abhängigkeiten und Relationen und gehen als Randbedingungen in die Planung mit ein. Ein Elaborationsnetz spiegelt die während des Planungsprozesses gefällten Entscheidungen wider und kann als Informationsgrundlage für eine mögliche Erklärungs-

komponente von PLAKON dienen. Auch interaktive Elemente wurden aufgenommen: wenn ein Planschritt noch nicht eindeutig bestimmt werden kann, soll der Benutzer weiterhelfen. Im Konfliktfall wird "intelligentes" Backtracking eingesetzt. Die Eigenschaft, durch die sich PLAKON wohl am meisten auszeichnet, ist die flexible Anpassung der Strategie an die Problem- oder auch nur Teilaufgabenstellung. Mehrere Strategieformen (z.B. Top-Down und Bottom-Up) können auf verschiedenen Ebenen gleichzeitig angewendet werden.

Inwieweit sich dieser Ansatz als der richtige, d.h. für reale Anwendungen einsetzbare, herausstellen wird, zeigt die Zukunft. Feststeht, daß nur eine flexible Kontrollstruktur und ausdrucksstarke, aber dennoch leicht handhabbare und für unterschiedliche Domänen verwendbare Repräsentation von Wissen der Schlüssel zum Erfolg ist. Die Idee der Planungssystem-Shell zeigt dafür die besten Ansätze.

LITERATURVERZEICHNIS
==========

1. Brooks, R.A., Lozano-Perez,T., An approach to automatic robot programming, MIT Artificial Intelligence Laboratory, A.I. Memo No. 842 (1985).

2. Chapman, D., Planning for conjunctive goals, Artificial Intelligence 32 (3) (1987) 333-377.

3. Cunis, R., Günter, A. Peters, H., Syska, I., PLAKON – Ein Ansatz zur domänen-unabhängigen Konstruktion, FB Informatik, Universität Hamburg; Unterlagen zum Workshop "Planen und Konfigurieren" am 03.12.1987, veranstaltet von: Institut für Werkzeugmaschinen und Betriebstechnik, Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe.

4. Cunis, R., Günter, A., Syska, I., Planen mit PLAKON, in: Hertzberg, J. (Hrsg.), Proc. Workshop Planen 1987, GMD-Arbeitspapiere Nr. 247 (1987) 58-89.

5. Dijkstra, E. W., Hierarchical ordering of sequential processes, Acta Informatica 1 (1971) 115-138.

6. Fikes, R. E., Nilsson, N. J., STRIPS: A new approach to the application of theorem proving to problem solving, Artificial Intelligence 2 (1971) 189 (ff.).

7. Günter, A., Syska, I., Cunis, R., PLAKON Anforderungskatalog, TEX-K-Bericht (1987).

8. Hertzberg, J., Planerstellungsmethoden der Künstlichen Intelligenz, Informatik-Spektrum 9 (1986) 149-161.

9. McCarthy, J., Hayes, P. J., Some philosophical problems from the standpoint of Artificial Intelligence, Mach. Int. 4 (1969) 463 (ff.).

10. Newell, A., Shaw, J. C., Simon, H. A., Report on a general problem solving program, Proc. Int. Conf. on Information Processing (ICIP), Paris (1959).

11. Schnupp, P., Leibrandt, U., Expertensysteme, Springer Verlag (1986).

12. Simon, H. A., The science of design and architecture of complexity, in: Sciences of the Artificial (MIT press, Cambridge, 1969).

13. Sommerville, I., Software-Engineering, Addison Wesley (1987).

14. Stefik, M., Planning with Constraints (MOLGEN Part I), Artificial Intelligence 16 (1981) 111-140.

15. Stefik, M.,Planning and Meta-Planning (MOLGEN Part II), Artificial Intelligence 16 (1981) 141-170.

16. Wettstein, H., Architektur von Betriebssystemen, 2. Auflage, Hanser Verlag (1984).

17. Wilkins, D. E., Domain-independent planning: representation and plan generation, Artificial Intelligence 22 (1984) 269-301.

18. Wilkins, D. E., Hierarchical planning: definition and implementation, ECAI (1986) 466-478.

19. Winzer, T., Künstliche Intelligenz und Robotik, Franzis-Verlag (1987).