CAPlan – ein SNLP-basierter Planungsassistent*

Frank Weberskirch, Jürgen Paulokat
Universität Kaiserslautern, FB Informatik
Postfach 3049, D-67653 Kaiserslautern
{weberski | paulokat}@informatik.uni-kl.de

Zusammenfassung

Im Rahmen dieser Arbeit beschreiben wir die wesentlichen Merkmale der CAPlan-Architektur, die die interaktive Bearbeitung von Planungsproblemen ermöglichen. Anhand des SNLP-Algorithmus, der der Architektur zugrunde liegt, werden die im Laufe eines Planungsprozesses auftretenden Entscheidungspunkte charakterisiert. Mit Hilfe von frei definierbaren Kontrollkomponenten kann das Verhalten an diesen Entscheidungspunkte festgelegt werden, wodurch eine flexible Steuerung des Planungsprozesses ermöglicht wird. Planungsziele und -entscheidungen werden in einem gerichteten azyklischen Graphen verwaltet, der ihre kausalen Abhängigkeiten widerspiegelt. Im Gegensatz zu einem Stack, der typischerweise zur Verwaltung von Entscheidungen eingesetzt wird, erlaubt die graphbasierte Repräsentation die flexible Rücknahme einer Entscheidung, ohne alle zeitlich danach getroffenen Entscheidungen ebenfalls zurücknehmen zu müssen.

1 Einleitung

Arbeiten aus dem Bereich der Planung liegt meistens die Annahme zugrunde, daß eine Planungsaufgabe von einem autonomen Planungssystem gelöst werden kann. Die in diesem Zusammenhang beschriebenen Planungsalgorithmen basieren auf einem allgemeinen Suchverfahren, mit dessen Hilfe der Raum möglicher Pläne schrittweise, ohne jegliche Interaktion mit der Außenwelt abgearbeitet wird. Ohne auf die Verwaltung von Planungsentscheidungen explizit einzugehen, legen die Darstellungen der Algorithmen einen Stack zu ihrer Verwaltung nahe, der auch in vielen Implementierungen von Planungsalgorithmen eingesetzt wird.

Bei der Anwendung eines Planungsverfahrens auf reale Problemstellungen führt das autonome Problemlösungsverhalten häufig zu Akzeptanzproblemen oder zu nicht adäquaten Lösungen. Komplexe Aufgabenstellungen können oftmals nur von interaktiven Planungssystemen sinnvoll gelöst werden, die einerseits in der Lage sind, Teilaufgaben automatisch zu bearbeiten und andererseits auf Benutzerinteraktionen opportunistisch zu reagieren, d.h. es wird kein autonomes Planungssystem benötigt, sondern ein Planungsassistent. Im einfachsten Fall einer Benutzerinteraktion handelt es sich um die Auswahl eines Ziels, mit dem der Planungsprozeß fortgesetzt werden soll oder um die Auswahl eines Schritts zur Erreichung eines Ziels. Beides kann mit Hilfe einer stackbasierten Implementierung eines Planungsverfahrens unterstützt und verwaltet werden. Soll darüberhinaus die interaktive Zurücknahme von Planungsentscheidungen möglich sein, zeigen stack-basierte Verfahren ein problematisches

*Die vorgestellte Arbeit wurde teilweise durch die Deutsche Forschungsgemeinschaft, SFB 314: „Künstliche Intelligenz – Wissensbasierte Systeme“, Projekt X 9 gefördert.

Verhalten. Aufgrund der vollständigen Anordnung der Entscheidungen müssen alle Entscheidungen, die zeitlich nach der zurückgenommenen Entscheidung getroffen wurden, ebenfalls zurückgenommen werden, obwohl sie sich u.U. auf unabhängige Teilprobleme beziehen.

Im folgenden wird die Verwaltung von Planungsentscheidung in CAPlan beschrieben, die es ermöglicht, Entscheidungen zurückzunehmen, wobei die aktuelle Teillösung nur so weit wie notwendig modifiziert wird. Da sich die Implementierung von CAPlan weitgehend an SNLP orientiert, werden zunächst in Abschnitt 2 die wesentlichen Ideen von SNLP im Überblick dargestellt. Abschnitt 3 beschreibt die wichtigen Eigenschaften der CAPlan-Architektur und vergleicht das Verhalten von SNLP und CAPlan anhand eines Beispiels. Die Mechanismen und zentralen Datenstrukturen zur Verwaltung von Planungsaufgaben und Planungsentscheidungen sind Gegenstand von Abschnitt 4. Das Papier endet mit einer abschließenden Diskussion der dargestellten Mechanismen (Abschnitt 5).

2 SNLP

Zunächst sollen einige grundlegenden Begriffe und Notationen erläutert werden, die charakteristisch für die Planung mit SNLP-basierten Planern wie CAPlan sind (vgl. [Barrett & Weld, 1993] und [Knoblock & Yang, 1994]). Weiter wird auf die Behandlung von Interaktionen (Threats) eingegangen und der SNLP-Algorithmus vorgestellt.

2.1 Notationen

Planungsprobleme bestehen aus einer initialen Situation und einer Menge $\{g_1, \cdots, g_n\}$ von Zielen. Ein *partiell geordneter Plan* ist ein Tupel (S, O, B), wobei S eine Menge von Planschritten[1], O eine Menge von Ordnungs-Constraints auf den Planschritten aus S und B eine Menge von Bindungs-Constraints für Variablen ist. In S gibt es zwei ausgezeichnete Planschritte, s_0 und s_∞; sie repräsentieren die aktuelle Problemstellung dadurch, daß s_0 als Effekte die initiale Situation und s_∞ als Vorbedingungen die Menge der zu erfüllenden Ziele hat. Die aktuellen Planungsziele G (offene Ziele) lassen sich als eine Menge von Tupeln (p, s) mit der Eigenschaft definieren, daß p eine Vorbedingung von Schritt s ist, die noch nicht erfüllt ist.[2] Der *initialer Plan* zu einem Planungsproblem ist gegeben durch $(S_0, O_0, B_0) = (\{s_0, s_\infty\}, \{s_0 \prec s_\infty\}, \emptyset)$, die initiale Menge der Planungsziele durch $G = \{(g_i, s_\infty) | i = 1, \cdots, n\}$.

Ein wesentlicher Punkt bei SNLP ist, daß die Gültigkeit jeder Vorbedingung eines Planschrittes durch die Erzeugung eines *Causal Links* garantiert wird. Wenn ein Schritt s_i einen Effekt hat, der verwendet wird, um die Vorbedingung p eines Schrittes s_j wahr zu machen, dann drückt der Causal Link $s_i \xrightarrow{p} s_j$ diesen Zusammenhang aus.
Ein Planschritt s_k interagiert mit einem Causal Link $s_i \xrightarrow{p} s_j$, wenn s_k einen Effekt q hat, der möglicherweise (d.h. eventuell erst mit zusätzlichen Bindungs-Constraints) identisch mit p oder der Negation von p ist. Dann heißt das Tupel $(s_k, s_i \xrightarrow{p} s_j)$ ein *Threat*. Man spricht von einem *positiven Threat*, wenn q möglicherweise identisch mit p ist; analog handelt es sich um einen *negativen Threat*, wenn q möglicherweise die Negation von p ist. Weiterhin soll zwischen *aktiven* und *potentiellen Threats* unterschieden werden. Ein Threat ist aktiv, wenn

[1] Ein Planschritt ist ein partiell instantiiertes Operator-Schema der Domäne, welches Vorbedingungen, Effekte, neu eingeführte Bindungs-Constraints für Variablen und neu hergeleitete Teilziele bestimmt.

[2] Im Text wird p alleine offenes Ziel genannt, wenn das zugehörige s aus dem Kontext klar hervorgeht.

der Schritt s_k möglicherweise zwischen s_i und s_j liegt und die u.U. notwendigen Bindungs-Constraints konsistent mit dem aktuellen Plan sind. Ansonsten handelt es sich um einem potentiellen Threat, d.h. entweder gilt im aktuellen Plan $s_k \prec s_i$ oder $s_j \prec s_k$ oder die notwendigen Constraints führen zu einem inkonsistenten Plan. Potentielle Threats spielen erst beim Rückzug von Planungsentscheidungen eine Rolle (vgl. Abschnitt 4.3). SNLP unterscheidet sich z.B. von NONLIN [Tate, 1977] dadurch, daß aus Gründen der Systematik neben negativen auch positive Threats aufgelöst werden. SNLP betrachtet nur aktive Threats.

2.2 Auflösen von Threats

Tritt im Plan (S, O, B) ein aktiver Threat $t = (s_k, s_i \xrightarrow{p} s_j)$ auf, so kann dieser grundsätzlich aufgelöst werden, indem der interagierende Schritt s_k durch zusätzliche Ordnungen aus dem Bereich des Causal Link gebracht wird oder indem durch zusätzliche Variablenbindungs-Constraints die Unifikation verhindert wird. Aus dem aktiven wird so ein potentieller Threat.

Es ergeben sich konkret die folgenden Auflösungsmöglichkeiten eines Threats t, auf die im folgenden Algorithmus mit *Resolve-Threat(t)* zurückgegriffen wird: (P bezeichne die Menge von Variablen-Paaren (x, y), die übereinstimmen müssen, damit der Threat entsteht)

- **Promotion:** $O' = O \cup \{s_j \prec s_k\}$, $B' = B$ • **Demotion:** $O' = O \cup \{s_k \prec s_i\}$, $B' = B$
- **Separation:** $O' = O \cup \{s_i \prec s_k, s_k \prec s_j\}$,
  $B' = B \cup \sigma$ mit $\sigma = \{\alpha | \alpha = \mathit{noncodesignate}(s) \cup \mathit{codesignate}(P - s), \emptyset \neq s \subseteq P\}$

  Hier bezeichnen die Ausdrücke $\mathit{codesignate}(R)$ bzw. $\mathit{noncodesignate}(R)$ die Menge der Gleichheits- bzw. Ungleichheits-Constraints zwischen den Variablenpaaren $(x, y) \in R$. Charakteristisch für SNLP ist, daß aus Gründen der Systematik bei jeder Möglichkeit von Separation alle Variablenpaare aus P betrachtet werden müssen.

2.3 Algorithmus

SNLP sucht im Raum der partiell geordneten Pläne. Der folgende Algorithmus beschreibt in Anlehnung an die Algorithmen aus [McAllester & Rosenblitt, 1991], [Barrett & Weld, 1993] und [Knoblock & Yang, 1994] die Vorgehensweise von SNLP und bildet das Grundgerüst des Algorithmus von CAPlan.

Die Parameter des Algorihmus sind: ein partieller Plan (S, O, B), eine Menge G von offenen Zielen, eine Menge L von Causal Links und eine Menge T von aktiven Threats. Der Algorithmus ist rekursiv und löst chronologisches Backtracking aus, wenn bei der Threat-Auflösung oder Operatorauswahl keine konsistente Möglichkeit (mehr) existiert. Ziel des Algorithmus ist es, für jede Vorbedingung eines Schrittes im Plan einen gegen alle Threats geschützten Causal Link zu erzeugen. Für ein gegebenes Planungsproblem wird der Algorithmus mit dem initialen Plan und der initialen Menge G von Planungszielen aufgerufen: $SNLP((S_0, O_0, B_0), G, \emptyset, \emptyset)$.

Algorithmus $SNLP((S, O, B), G, L, T)$

1. **Termination:** Falls $G = \emptyset$ und $T = \emptyset$, dann fertig (Erfolg).
2. **Threat-Behandlung:** (falls $T \neq \emptyset$)
   (a) **Threat-Auswahl:** Sei $t \in T$ mit $t = (s_k, s_i \xrightarrow{p} s_j)$

(b) **Threat-Auflösung:** (Backtracking-Punkt)
Resolve-Threat(t): Dies löst t auf und verändert O zu O' und B zu B'.

(c) **Threat-Aktualisierung:** $T' = T - \{t | t \in T$ kein aktiver Threat in $(S, O', B')\}$

(d) **Rekursiver Aufruf:** $SNLP((S, O', B'), G, L, T')$

3. **Zielabarbeitung:** (falls $G \neq \emptyset$)

(a) **Zielauswahl:** Sei $(p, s_{need}) \in G$ das nächste offene Ziel.

(b) **Operatorauswahl:** (Backtracking-Punkt)
Sei s_{add} ein Schritt (aus S oder neu) mit Effekt p, der nicht hinter s_{need} angeordnet ist. $S' = S \cup \{s_{add}\}$; $O' = O \cup \{s_{add} \prec s_{need}\}$; $B' = B \cup \{$Bindungs-Constraints, so daß s_{add} Effekt p hat$\}$; $G' = G - \{(p, s_{need})\} \cup Preconditions(s_{add})$, $L' = L \cup \{s_{add} \xrightarrow{p} s_{need}\}$.

(c) **Threat-Erkennung:** $T' = \{t | t = (s_k, s_i \xrightarrow{p} s_j)$ aktiver Threat in $(S', O', B')\}$

(d) **Rekursiver Aufruf:** $SNLP((S', O', B'), G', L', T')$

Bemerkung zum Algorithmus

Korrektheit, Vollständigkeit und Systematik[3] sind nachgewiesene Eigenschaften von SNLP. Der obige Algorithmus ist jedoch (ebenso wie der Algorithmus POCL aus [Barrett & Weld, 1993]) eine Vereinfachung des ursprünglichen Find-Completion-Algorithmus aus [McAllester & Rosenblitt, 1991] und die Vollständigkeit kann hier nicht garantiert werden, da der Algorithmus u.U. nicht terminiert, weil es zu endlosem Herleiten neuer Teilziele kommen kann. Aus diesem Grund löste der ursprüngliche Algorithmus aus [McAllester & Rosenblitt, 1991] beim Überschreiten einer Kostenobergrenze chronologisches Backtracking aus und verhindert so endloses Suchen. Vollständigkeit wird dann durch *Iterative Deepening* [Korf, 1987] erreicht.

In [Weberskirch, 1994] wird der obige Algorithmus so erweitert, daß das Terminationsproblem gelöst ist, ohne daß eine solche Kostenobergrenze und ein Iterative-Deepening-Verfahren verwendet wird. Dazu werden zunächst die für eine Nichtterminierung verantwortlichen Teilzielrekursionen erkannt. Eine Teilzielrekursion tritt auf, wenn durch Anwendung eines Operators *op* auf ein Ziel g wieder ein Teilziel $g' = g$ hergeleitet wird (eventuell erst nach mehreren anderen Operatoranwendungen). In diesem Fall muß verhindert werden, daß g' wieder durch den Operator *op* bearbeitet wird, weil dies unweigerlich zur Rekursion führt. Außerdem wird das einfache chronologische Backtracking durch Backjumping [Ginsberg, 1993] ersetzt.

3 CAPlan

Aufbauend auf das Planungsverfahren aus Abschnitt 2.3, soll hier auf die Modifikationen und Erweiterungen eingegangen werden, die aus einem SNLP-Planer einen Planungsassistenten machen. Zunächst wird eine Übersicht darüber gegeben, welche grundsätzlichen Möglichkeiten es gibt, den SNLP-basierten Planer CAPlan zu steuern. Dann geht es um die Schwierigkeiten, die durch Benutzerinteraktionen (insbes. durch die Rücknahme beliebiger Planungsentscheidungen durch den Benutzer) bei SNLP entstehen, und um deren Lösung in CAPlan.

[3] Systematik meint, daß jeder Plan (S, O, B) nur maximal einmal erzeugt wird.

3.1 Steuerung von CAPlan durch Kontrollkomponenten

Betrachtet man den Algorithmus aus Abschnitt 2.3, so kann man folgende Punkte identifizieren, an denen während der Planung eine Auswahlentscheidung getroffen werden muß:

- **Threat-Auswahl:** Welcher Threat aus der Menge T soll bearbeitet werden?
- **Threat-Bearbeitung:** Wie soll ein Threat aufgelöst werden? (vgl. Abschnitt 2.2)
- **Zielauswahl:** Welches offene Ziel aus der Menge G soll bearbeitet werden?
- **Operatorauswahl:** Mit welchem Operator soll ein Ziel bearbeitet werden?

Charakteristisch für SNLP und damit für CAPlan ist, daß die Threat- und die Zielauswahl keine Backtracking-Punkte im Algorithmus sind. Prinzipiell spielt es also keine Rolle, welcher Threat oder welches offene Ziel zuerst bearbeitet wird. Ein geschickte Threat- bzw. Zielauswahl kann jedoch den Suchraum drastisch reduzieren.

Ein anderer Aspekt wird im SNLP-Algorithmus zu starr betrachtet: Threats werden immer sofort aufgelöst, bevor die Erfüllung von Teilzielen fortgesetzt wird. In [Peot & Smith, 1993] und [Smith & Peot, 1993] wird gezeigt, daß das nicht zwingend ist und es nachweisbar bessere Strategien gibt, die die Systematik von SNLP nicht beeinflussen [Harvey *et al.*, 1993].

In CAPlan wird dieser Steuerungsproblematik dadurch Rechnung getragen, daß es im Planungssystem eine frei definierbare Menge von *Kontrollkomponenten* gibt, von denen eine an allen oben genannten Stellen entscheidet, welche Alternative gewählt werden soll bzw. ob Threats oder Ziele bearbeitet werden sollen (vgl. [Weberskirch, 1994]). Dazu ist es in CAPlan notwendig, den Algorithmus aus Abschnitt 2.3 dahingehend zu modifizieren, daß die Schritte 2 und 3 nicht mehr in einer festen Reihenfolge durchlaufen werden, sondern die Reihenfolge von der Entscheidung der momentan aktiven Kontrollkomponente abhängig zu machen ebenso wie alle anderen Auswahlentscheidungen. Diese momentan aktive Kontrollkomponente kann auch während der Bearbeitung eines Planungsproblems beliebig gewechselt werden. Damit wird eine flexible Architektur geschaffen, da verschiedene Kontrollstrategien leicht in Form verschiedener Kontrollkomponenten definierbar und kombinierbar sind.

Momentan gibt es für CAPlan folgende Kontrollkomponenten:

- **SNLP:** Hier wird die Vorgehensweise des SNLP-Algorithmus aus Abschnitt 2.3 als Default-Kontrollkomponente zur Verfügung gestellt.
- **UC (User Control):** Mit dieser Kontrollkomponente arbeitet CAPlan als Planungsassistent. Anfallende Entscheidungen werden vom Benutzer getroffen. Der Grad an Interaktion mit dem Benutzer ist variabel; er kann sich z.B. auf den Aspekt der Ziel- oder Operatorauswahl beschränken oder sich auf alle o.g. Entscheidungspunkte beziehen.
- **CbC (Case-based Control):** Mit CAPlan/CbC [Muñoz *et al.*, 1994] steht eine Kontrollkomponente zur Verfügung, die episodisches Wissen in Form von Fallbeispielen zur Steuerung der Planung verwendet (fallbasierte Planung).

3.2 Beispiel für die Planung mit CAPlan

Nachdem der Algorithmus und der Steuerungsmechanismus von CAPlan vorgestellt wurde, soll nun an einem einfachen Beispiel demonstriert werden, wie CAPlan bei der Planung

vorgeht und welche Defizite sich aus dem SNLP-Ansatz ergeben. Man gehe von einer Blocksworld-Domäne mit den Operatoren `PickUpBlock` und `PutDownBlock` aus (vgl. Abb. 1).[4]

PickUpBlock	
Vorbed.:	+Clear(x)
	+Handempty(z)
	+On(x,y)
Effekte:	+Holding(x,z)
	+Clear(y)
	-On(x,y)
	-Handempty(z)

PutDownBlock	
Vorbed.:	+Clear(y)
	+Holding(x,z)
Effekte:	+On(x,y)
	+Handempty(z)
	-Holding(x,z)
	-Clear(y)

Abbildung 1: Operatoren `PickUpBlock` und `PutDownBlock`

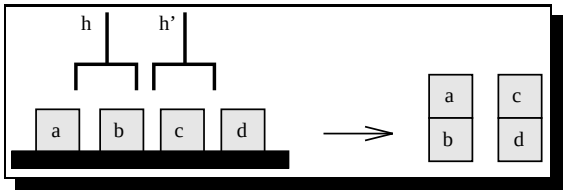

Abbildung 2: Beispiel für eine Problemstellung

Die Problemstellung sei wie in Abbildung 2 dargestellt: die Ziele `+On(a,b)` und `+On(c,d)` sind zu erreichen. Abbildung 3 zeigt einige Planzustände, die CAPlan durchläuft:[5]

Abbildung 3.a: Zunächst wird ein initialer Plan (Schritte START und FINISH) erzeugt und die Menge offener Ziele mit $G = \{(+On(a,b),FINISH),\ (+On(c,d),FINISH)\ \}$ initialisiert.

Abbildung 3.b: Hier wurde das Ziel `+On(c,d)` ausgewählt und mit dem Operator `PutDownBlock` (im Plan STEP-1) bearbeitet. Man sieht den Causal Link für die Vorbedingung `+On(c,d)` von FINISH. Außerdem hat der neue Schritt STEP-1 zwei Teilziele hergeleitet, die die Abbildung nicht zeigt.

Abbildung 3.c: Es wurde das offene Ziel `+Holding(c,x0)` ausgewählt und wieder unter Erzeugung eines neuen Schrittes (STEP-2) bearbeitet.

Abbildung 3.d: Hier wurden die Vorbedingungen von STEP-2 und die noch offene Vorbedingung von STEP-1 erfüllt, ohne daß ein neuer Schritt erzeugt werden mußte, da START bereits die geforderten Effekte hat. Die Planung für das erste Teilziel der Problembeschreibung ist damit fertig.

Abbildung 3.e: Schließlich sieht man den fertigen Plan. Er besteht aus zwei parallelen Teilplänen (Schritte STEP-2/STEP-1 bzw. STEP-4/STEP-3). Im Laufe der Planung sind Threats aufgetreten, deren Auflösung (durch Separation) nur indirekt dadurch zum Ausdruck kommt, daß bestimmte Variablenbindungen erfolgt sind.

Die Frage, die sich nun im Hinblick auf Benutzerinteraktionen stellt, ist die folgende: Was passiert, wenn der Benutzer beim fertigen Plan aus Abbildung 3.e entscheidet, z.B. die zweiten

[4] Zur Schreibweise folgende Bemerkung: `+Clear(y)` bzw. `-On(x,y)` als Effekt eines Operators heißt, daß der Operator `Clear(y)` bzw. `On(x,y)` gültig bzw. ungültig macht.

[5] In der Abbildung stehen Knoten mit abgerundeten Ecken für die Planschritte (START, FINISH, STEP-i), zwischen denen im Laufe der Planung Causal Links aufgebaut werden.

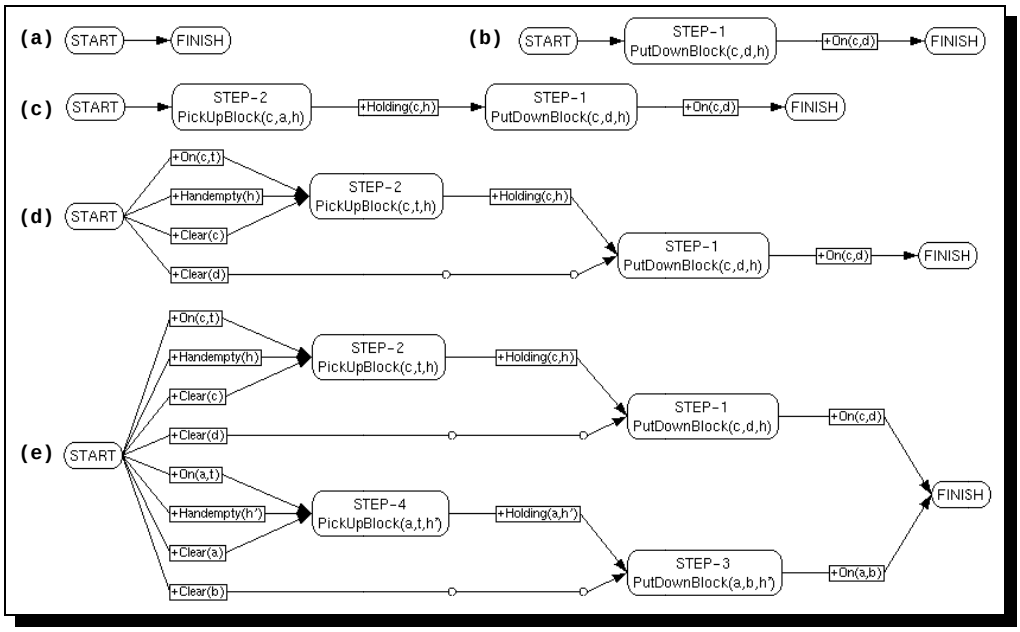

Abbildung 3: Einige erzeugte partielle Pläne zur Problemstellung aus Abbildung 2

Planungsentscheidung, die STEP-2 (`PickUpBlock`) einführte, zurückzunehmen und stattdessen eine andere Alternative (`PickUpBlock'`) zu wählen? Wir gehen hier davon aus, daß die Domänenmodellierung einen dritten Operator `PickUpBlock'` enthält. Um diese Frage zu beantworten, muß man den SNLP-Algorithmus unter dem Aspekt, wie er Backtracking durchführt, etwas genauer betrachten.

4 Repräsentation und Verwaltung von Entscheidungen

4.1 Stack von Causal Links und Threat-Auflösungen bei SNLP

Wie schon in Abschnitt 2.3 erwähnt, führt SNLP chronologisches Backtracking durch, um Inkonsistenzen im Plan aufzulösen. Die hierzu notwendige zeitliche Reihenfolge auf der Menge der Entscheidungen wird mit Hilfe eines Stacks verwaltet.

Bezogen auf das Beispiel aus Abbildung 2 sieht dies wie folgt aus: Nach dem letzten Planungsschritt aus Abbildung 3.e enthält der von SNLP benötigte Stack Einträge wie sie in Abbildung 4 aufgelistet sind. Ganz unten im Stack liegt der für das zuerst bearbeitete Ziel *(+On(c,d), FINISH)* erzeugte Causal Link. Darüber liegen alle anderen erzeugten Causal Links und Threat-Auflösungen.

Aus der Verwendung eines Stack ergibt sich, daß SNLP nur in der Lage ist, zur jeweils letzten Entscheidung zurückzuspringen; daher kann auf die Anforderung, eine beliebige, vorher getroffene Entscheidung d zurückzuziehen nur so reagiert werden, daß alle Entscheidungen d_i, die nach d getroffen wurden, ebenfalls zurückgezogen werden.

Ist im Beispiel etwa die Entscheidung d_2 (vgl. Abb. 4) zurückzuziehen, so werden von SNLP auch die Entscheidungen $d_3, \cdots, d_{15}$ zurückgezogen; es entsteht wieder der Plan aus Abbildung 3.b. Leider wird alles revidiert, was zur Erfüllung des Zieles *(+On(a,b), FINISH)* bereits getan wurde $(d_7, d_9, d_{12}, \cdots, d_{15})$, obwohl der durch diese Entscheidungen erzeugte Teilplan nach wie vor verwendet werden könnte, da er nicht von d_2 abhängt. Beim Rückzug

	:
d_{15} :	$START \stackrel{+Clear(b)}{\longrightarrow} STEP\text{-}3$
d_{14} :	$START \stackrel{+Clear[a]}{\longrightarrow} STEP\text{-}4$
d_{13} :	$START \stackrel{+Handempty[h']}{\longrightarrow} STEP\text{-}4$
d_{12} :	$START \stackrel{+On[a,t]}{\longrightarrow} STEP\text{-}4$
d_{11} :	$x18 \neq d$
d_{10} :	$x18 \neq c$
d_9 :	$STEP\text{-}4 \stackrel{+Holding[a,h']}{\longrightarrow} STEP\text{-}3$
d_8 :	$x0 \neq x16$
d_7 :	$STEP\text{-}3 \stackrel{+On[a,b]}{\longrightarrow} FINISH$
d_6 :	$START \stackrel{+Clear[d]}{\longrightarrow} STEP\text{-}1$
d_5 :	$START \stackrel{+Clear[c]}{\longrightarrow} STEP\text{-}2$
d_4 :	$START \stackrel{+Handempty[h]}{\longrightarrow} STEP\text{-}2$
d_3 :	$START \stackrel{+On[c,t]}{\longrightarrow} STEP\text{-}2$
d_2 :	$STEP\text{-}2 \stackrel{+Holding[c,h]}{\longrightarrow} STEP\text{-}1$
d_1 :	$STEP\text{-}1 \stackrel{+On[c,d]}{\longrightarrow} FINISH$

Abbildung 4: Stack von SNLP zu Abbildung 3.e

einer Entscheidung wird der Plan also mehr als notwendig modifiziert, da auch unabhängige Entscheidungen zurückgenommen.

Daraus ergibt sich für CAPlan die Forderung, die einfache Verwaltung der abgearbeiteten Ziele in Form eines Stacks durch einen leistungsfähigeren Mechanismus zu ersetzen.

4.2 Graphförmige Repräsentation von Entscheidungen

Es soll nun zunächst davon ausgegangen werden, daß während des Planungsprozesses keine Threats auftreten. Um eine minimale Modifikation des bestehenden Planes zu ermöglichen, werden in CAPlan die getroffenen Entscheidungen in Form eines Graphen verwaltet.

Knoten stehen entweder für noch offene Ziele oder für getroffene Entscheidungen, d.h. für erzeugte Causal Links und (falls notwendig) für neu eingeführte Schritte oder für Auflösungen von Threats. *Kanten* beschreiben Abhängigkeiten zwischen Entscheidungen bzw. offenen Zielen, derart daß eine Entscheidung d_j von d_i abhängig ist, wenn d_j Nachfolger von d_i ist.

Nachfolger eines Entscheidungsknotens entstehen, wenn eine Entscheidung neben einem Causal Link $s_i \stackrel{p}{\longrightarrow} s_j$ auch den benötigten Schritt s_i erzeugt. Für die Vorbedingungen dieses Schrittes s_i, die zunächst offene Planungsziele sind, werden Nachfolgerknoten erzeugt. Diese Knoten für offene Ziele werden durch einen Entscheidungsknoten ersetzt, sobald eine Entscheidung getroffen wird, wie das offene Ziel erfüllt wird. Wurde bei der Erzeugung eines Causal Links $s_i \stackrel{p}{\longrightarrow} s_j$ der Schritt s_i nicht neu erzeugt, so wird eine Abhängigkeitskante von dem Entscheidungsknoten d_k, der s_i erzeugt hat, zu dem Knoten d_l, der den Causal Link erzeugt, eingefügt, falls d_k nicht bereits Vorgänger von d_l ist. Hierdurch entsteht ein gerichteter azyklischer Graph, der die Abhängigkeiten der Entscheidungen widerspiegelt.

Betrachtet man zunächst etwa den Planzustand aus Abbildung 3.d, wo der Teilplan für das erste Teilziel *(+On(c,d),FINISH)* fertig ist, so zeigt Abbildung 5 den von CAPlan bis dahin

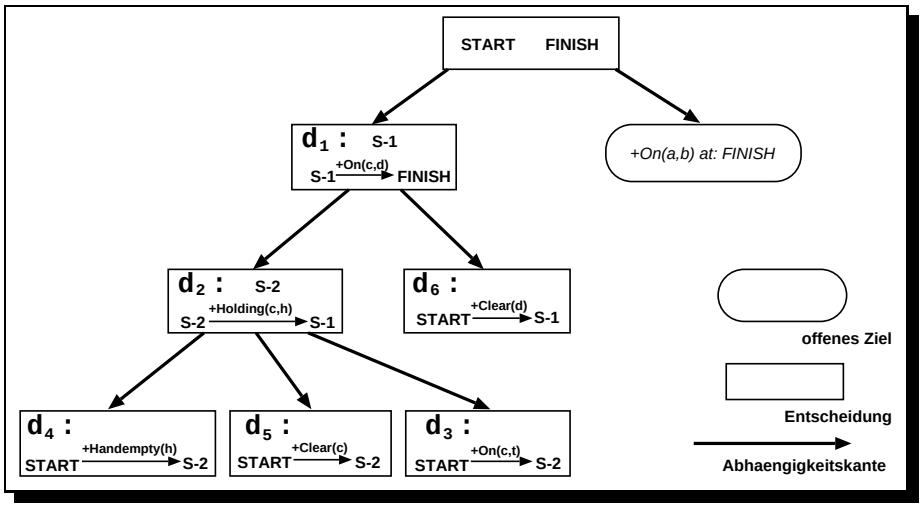

Abbildung 5: Graph von Entscheidungen zum Planzustand aus Abbildung 3.d. Die Namen der Schritte wurde hier abgekürzt, z.B. zu S-1 statt STEP-1.

aufgebauten Graph (hier nur ein Baum) von Entscheidungen. Es kommen folgende Aspekte zum Ausdruck: Die Wurzel steht für die Initialisierung des Systems mit dem Planungsproblem, wodurch die Schritte START und FINISH erzeugt werden. Es gibt zwei initiale Teilziele (Vorbedingungen von FINISH), die zu Nachfolgerknoten der Wurzel werden. Für das noch unbearbeitete Ziel *(+On(a,b), FINISH)* existiert noch keine Entscheidung, was hier durch einen Knoten mit abgerundeten Ecken angedeutet ist. Bei der Bearbeitung des anderen Teilziels wurde ein neuer Schritt STEP-1 und der Causal Link $\mathit{STEP\text{-}1} \stackrel{+On(c,d)}{\longrightarrow} \mathit{FINISH}$ erzeugt, was als Entscheidung d_1 repräsentiert ist. STEP-1 hat außerdem zwei neue Teilziele erzeugt: *(+Holding(c,x0), STEP-1), (+Clear(d), STEP-1).* Bei den Entscheidungen im Baum, die Blattknoten sind, wurde ein bereits existierender Schritt im Plan (hier START) verwendet. Es brauchte keine zusätzliche Abhängigkeitskante eingefügt werden, da der Wurzelknoten, der START enthält, bereits Vorgänger aller Blattknoten ist.

Wird im Planzustand aus Abbildung 3.d, zu dem Abbildung 5 gehört, wieder die Entscheidung d_2 zurückgenommen wird (es gibt bis dahin die Entscheidungen $d_1, \cdots, d_6$), so würde SNLP $d_2, \cdots, d_6$ zurücknehmen. CAPlan verwendet den Baum aus Abbildung 5 und revidiert d_2 und alle Nachfolger (d_3, d_4, d_5). CAPlan erkennt, daß d_6 von der Rücknahme von d_2 unabhängig ist und nicht zurückgenommen werden muß.

4.3 Berücksichtigung von Threats

Die Bearbeitung von Threats ist im Algorithmus aus Abschnitt 2.3 ein Backtracking-Punkt und kommt daher im Stack aus Abbildung 4 vor. Genauso muß die Threat-Bearbeitung auch im Graph von Entscheidungen berücksichtigt werden, da bei der Threat-Auflösung Entscheidungen getroffen werden, bestimmte Ordungen oder Bindungs-Constraints zu erzeugen.

Wird ein Threat erkannt, so wird im Abhängigkeitsgraph ähnlich wie für offene Vorbedingungen von Schritten ein neuer Knoten erzeugt, ein sog. *ProtectionGoal*, welches zunächst nur ausdrückt, daß der ihm zugeordnete Threat möglicherweise aktiv ist und aufgelöst werden muß. Aus der Definition eines Threats ergibt sich, daß ProtectionGoals immer von zwei anderen Entscheidungen abhängen müssen: einerseits von der Entscheidung, die den bedrohten Causal Link erzeugt hat, andererseits von der Entscheidung, die den Schritt erzeugt, der

diesen Causal Link bedroht.

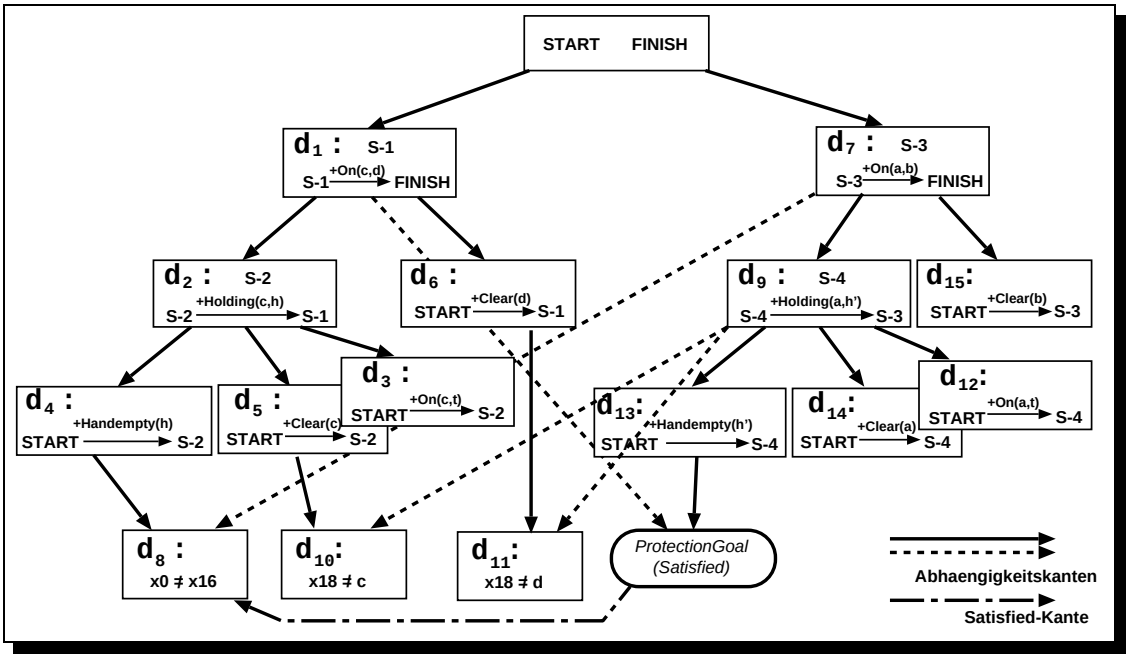

Abbildung 6: Abhängigkeitsgraph der Entscheidungen zum Planzustand aus Abbildung 3.e

Abbildung 6 zeigt diesen Sachverhalt am Beispiel aus Abschnitt 3.2. Der hier gezeigte Graph repräsentiert die Abhängigkeiten aller getroffenen Entscheidungen, wenn das Planungsproblem gelöst ist (vgl. Abb. 3.e). Im Laufe der Planung sind drei Threats aufgetaucht, die bearbeitet wurden. Abbildung 6 zeigt für jeden der Threats, welche Entscheidung getroffen wurde, um ihn aufzulösen (d_8, d_{10}, d_{11}). Betrachtet man z.B. den durch d_8 (Separation $x0 \neq x16$) aufgelösten Threat, so ist dieser Entscheidungsknoten abhängig von d_4 und d_7, weil durch diese Entscheidungen der bedrohte Causal Link bzw. der bedrohende Schritt (gestrichelte Linie) erzeugt wurde. Die so repräsentierte Abhängigkeit hat zur Folge, daß d_8 automatisch zurückgenommen wird, sobald mindestens eine der Entscheidungen d_4 oder d_7 zurückgenommen wird. Die Rücknahme des Knotens hat zur Folge, daß alle Constraints, die mit der Entscheidung in Plan aufgenommen wurden, ebenfalls zurückgenommen werden. Hierdurch wird sichergestellt, daß der Plan nach einer Benutzerinteraktion nur die minimal notwendige Menge von Constraints enthält (Vollständigkeit der Planung).

Am gleichen Beispiel betrachtet, ergibt sich: wenn die Entscheidung d_2 zurückgenommen wird, so nimmt CAPlan automatisch die abhängigen Entscheidungen d_3, d_4, d_5, d_8 und d_{10} zurück. Im Gegensatz zu SNLP bleiben $d_6, d_7, d_9, d_{11}, \cdots, d_{15}$ gültig, d.h. der Plan wird nur soweit modifiziert wie notwendig.

Ein weiterer Aspekt muß im Hinblick auf die Erzeugung von ProtectionGoals bedacht werden. Im Laufe des Planungsprozesses wird immer wieder nach Threats für Causal Links gesucht (vgl. Abschnitt 2.3) und dabei werden neben aktiven auch eine Reihe *potentieller Threats* entdeckt, potentiell deshalb, weil der aktuelle Plan (S, O, B) bereits Ordnungen oder Bindungs-Constraints enthält, die den Threat auflösen. Will man dem Benutzer gestatten, beliebige Entscheidungen wieder zurückzunehmen, so müssen auch diese potentiellen Threats repräsentiert werden, da sie infolge der Rücknahme von anderen Entscheidungen u.U. zu aktiven Threats werden können.

Potentielle Threats werden zunächst genauso wie normale Threats im Abhängigkeitsgraphen

durch ein ProtectionGoal eingetragen, jedoch wird zusätzlich vermerkt, aufgrund welcher anderen Entscheidungen sie nur potentielle Threats sind. Dies geschieht mittels eines speziellen Typs von Kanten (Satisfied-Kante) zwischen einer Entscheidung und einem ProtectionGoal. Ein ProtectionGoal muß bearbeitet werden, wenn es keine Satisfied-Kante mehr besitzt, die auf eine gültige Entscheidung zeigt, d.h. der zugehörige Threat ist dann aktiv.

Sieht man sich das wieder am Beispiel an (vgl. Abb. 6), so wird die Auswirkung klar. Im zu diesem Graph gehörigen Plan (vgl. Abb. 3.e) gibt es eine ganze Reihe von solchen potentielle Threats, von denen aus Gründen der Übersicht in Abbildung 6 nur einer dargestellt ist. Dieses ProtectionGoal ist als bereits erfüllt (*satisfied*) markiert (Satisfied-Kante zu d_8). Nimmt man aber wie in allen Beispielen zuvor auch hier die Entscheidung d_2 zurück, so wird dadurch zunächst u.a. auch d_8 zurückgenommen (ungültig). Das hat zur Folge, daß dieses ProtectionGoal nun bearbeitet werden muß. Die im Abhängigkeitsgraph gespeicherte Information über potentielle Threats verhindert also, daß nach dem Zurücknehmen einer Entscheidung alle Causal Links nochmals auf noch zu bearbeitende Threats hin überprüft werden müssen, was einen nicht unerheblichen Aufwand bedeuten würde.

Verwaltung des Entscheidungsgraphen

Die Verwaltung des Abhängigkeitsgraphen in CAPlan geschieht aufbauend auf ein TMS-basiertes System namens REDUX [Petrie, 1991], welches einige für die Bereiche Planung und Konfiguration grundlegenden Mechanismen zur Verfügung stellt. Insbesondere ist es in der Lage, Entscheidungen, Abhängigkeiten zwischen Entscheidungen, die durch Operatoranswendung entstehen, und Rückzugsbegründungen von Entscheidungen zu repräsentieren und unterstützt dependency-directed Backtracking.

Für CAPlan mußte REDUX jedoch noch um einige Aspekte erweitert werden. Threats werden durch Ziele repräsentiert und die von REDUX aufgebaute, baumförmige Abhängigkeitsstruktur muß zu einem gerichteten azyklischen Graphen erweitert werden.

5 Diskussion

Das zentrale Ziel, das wir mit der CAPlan-Architektur verfolgen, ist die Unterstützung der interaktiven Lösung eines Planungsproblems. Hierbei können einerseits einfache Teilprobleme automatisch durch das Planungssystem und andererseits komplexe Aufgabenstellungen interaktiv mit dem Benutzter bearbeitet werden. Die Architektur gibt den Benutzerentscheidungen die höchste Priorität, so daß Sytementscheidungen, die der Benutzer als falsch oder suboptimal einschätzt, jederzeit von ihm modifiziert werden können. Durch dieses opportunistische Sytemverhalten können Aufgaben interaktiv gelöst werden, obwohl das zugrundeliegende Domänenmodel unvollständig oder partiell falsch ist. Da Wissen über die Präferenz von Alternativen zu jedem Zeitpunkt interaktiv in den Lösungsprozeß eingebracht werden kann, berücksichtigt diese Architektur die Erfahrung, daß die Expertise der meisten Anwendungsdomänen mit den heutigen Techniken nicht vollständig oder zumindest nicht in einem Schritt modelliert werden kann.

Wir haben gezeigt, daß vor allem die Rücknahme von Entscheidungen bei stackbasierten Planungssystemen mit unnötigen Folgeänderungen verbunden sein kann. Zu ihrer Vermeidung werden Entscheidungen in CAPlan mit Hilfe gerichteter azyklischer Graphen verwaltet, die eine Erweiterung der Teilzielbäume oder der UND/ODER-Bäumen darstellen. Sie unterscheiden sich jedoch wesentlich von UND/ODER-Graphen, da in diesen Graphen eine Teilaufgabe durch unterschiedliche Problemlösungsschritte erreicht werden kann. Teilzielbäume zur Ver-

waltung von Entscheidungen werden auch in anderen Systemen verwendet, z.B. REDUX [Petrie, 1991] oder in einem System zur Integration von Aktionsplanung und Konfigurierung [Köhne, 1993]. Im Gegensatz zu CAPlan repräsentieren diese Systeme nur die eigentlichen Planungsziele, nicht jedoch die Ziele, eine Interaktion zwischen Planungsentscheidungen zu beheben. Dem REDUX-System liegt ein anderes Planungsmodell zugrunde, das nur negative Interaktionen berücksichtigt. Sie stellen Inkonsistenzen dar, die durch Backtracking behoben werden, so daß aus diesem Grund eine baumförmige Repräsentation ausreicht.

Einen interessanten Aspekt der REDUX-Architektur stellt die Verwaltung des Zielbaums mit Hilfe eines JTMS dar, wodurch die Aktualisierung der Entscheidungsstruktur auf die Label-Propagierung in einem TMS-Netzwerk zurückgeführt wird. Diese integrierte Abhängigkeitsverwaltung war der ausschlaggebende Grund, daß CAPlan aufbauend auf der modifizierten REDUX-Architektur realisiert wurde.

Literaturverzeichnis

BARRETT, A. UND WELD, D.S. 1993. *Partial-Order Planning: Evaluating Possible Efficiency Gains.* Technical Report 92-05-01 Expanded Version. Dep. of Computer Science and Engineering, University of Washington, Seattle, WA 98195.

GINSBERG, M.L. 1993. Dynamic Backtracking. *Journal of Artificial Intelligence Research*, **1**, 25–46.

HARVEY, W.D., GINSBERG, M.L. UND SMITH, D.E. 1993. Deferring Conflict Resolution Retains Systematicity. *Aus: Proceedings of AAAI-93.*

KNOBLOCK, C.A UND YANG, Q. 1994. Evaluating the Tradeoffs in Partial-Order Planning Algorithms. *Aus: Proceedings of the Canadian AI Conference.*

KÖHNE, A. 1993. *Integration von Aktionsplanung und Konfiguratuion.* Dissertationen zur Künstlichen Intelligenz (DISKI), Universität Karlsruhe.

KORF, R.E. 1987. Planning as Search: A Quantitative Approach. *Artificial Intelligence*, **33**, 65–88.

MCALLESTER, D. UND ROSENBLITT, D. 1991. Systematic Nonlinear Planning. *Seiten 634–639 aus: Proceedings of AAAI-91.*

MUÑOZ, H., PAULOKAT, J. UND WESS, S. 1994. Controlling non-linear hierarchical planning by case replay. *Aus:* KEANE, M., HATON, J. UND MANAGO, M. (Hrsg.), *Proceedings of the 2nd EWCBR.*

PEOT, M UND SMITH, D. 1993. Threat-removal strategies for partial-order planning. *Seiten 492–499 aus: Proceeding of AAAI-93.*

PETRIE, CH. 1991. *Planning and Replanning with Reason Maintenance.* Dissertation, University of Texas/Austin.

SMITH, D UND PEOT, M. 1993. Postponing threats in partial-order planning. *Seiten 500–506 aus: Proceedings of AAAI-93.*

TATE, A. 1977. Generating Project Networks. *Seiten 888–893 aus: Proceedings of IJCAI-77.*

WEBERSKIRCH, F. 1994. *Realisierung eines nichtlinearen Planungssystem zur Unterstützung der Arbeitsplanerstellung bei der computerintegrierten Fertigung (CIM).* Diplomarbeit, Universität Kaiserslautern.