Realisierung und Anwendung von Constraint-Hierarchien für die Konfiguration technischer Systeme in ConBaCon*

A. Schiemann, U. John, U. Geske, D. Boulanger
GMD — Forschungszentrum Informationstechnik GmbH
GMD FIRST, Rudower Chaussee 5, D-12489 Berlin

e-mail: {schieman, john, geske, dmitri}@first.gmd.de

Zusammenfassung

Constraint-Hierarchien ermöglichen die Spezifikation und Verarbeitung von weichen (möglichst einzuhaltenden) Constraints. In dieser Arbeit wird die Implementation eines neuen Verfahrens zur Verarbeitung von Constraint-Hierarchien vorgestellt. Es ermöglicht, spezifizierte Constraint-Hierarchien, die mit trivialen oder nicht-trivialen Fehlerfunktionen behaftet sind, zu verarbeiten. Dies geschieht durch Transformation in Finite-Domain-Constraints, die von gängigen FD-Constraint-Lösern (z.B. ECLIPSE und CHIP) verarbeitet werden können. Die Realisierung wurde in das System ConBaCon (**Con**straint **Ba**sed **Con**figuration) integriert, mit dem die Konfiguration und Rekonfiguration von technischen Systemen durchgeführt werden kann.

1 Einführung

Die Konfiguration technischer Systeme, z.B. Prozeßkontrollanlagen ([9]), ist hochkomplex und daher zeit- und personalintensiv. Dadurch bedingt, sind die erarbeiteten Konfigurationen nicht selten fehlerhaft, was sich meist erst in späteren praktischen Testphasen zeigt. Dies führt zu zusätzlichen, potentiell vermeidbaren Kosten (z.B. Vertragsstrafen, Umbaukosten) und birgt u.U. Gefahren für den praktischen Betrieb (z.B. Prozeßkontrollanlagen für Kraftwerke). Darüber hinaus läßt sich unterstellen, daß gefundene Lösungen, soweit sie fehlerfrei sind, im allgemeinen nicht einer optimumnahen Lösung entsprechen, beispielsweise der verbrauchsärmsten oder billigsten. Die Modellierung und Realisierung von expertenunterstützenden Systemen für die Konfiguration technischer Systeme ist daher ein permanentes Thema praxis-orientierter Forschung. Verschiedene Ansätze wurden entwickelt und geprüft (z.B. [1], [3], [7]). Neben erreichten Verbesserungen weisen diese aber immer noch essentielle Mängel auf. Starke Abweichung vom Optimum, nicht-terminierende Konfigurationsläufe und das Unvermögen, Präferenzen oder weiche Constraints zu spezifizieren bzw. zu verarbeiten sind einige von ihnen. Unser Ansatz basiert auf der Constraint Logischen Programmierung und ermöglicht das Erzeugen von "guten" Problemlösungen in kurzer Zeit, wobei steuernde und simulierende Interaktionen seitens des Projektanten unterstützt werden. Die erforderlichen weichen Constraints werden dabei mit Hilfe von Constraint-Hierarchien[1] modelliert und durch Transformation in Finite-Domain-Constraints verarbeitet.

*Das dieser Arbeit zugrundeliegende Forschungsvorhaben VERMEIL wird mit Mitteln des Bundesministeriums für Bildung, Wissenschaft, Forschung und Technologie (Förderkennzeichen 01 IN 507 B) gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

[1]im folgenden auch mit 'CH' abgekürzt

2 Constraint-Hierarchien

2.1 Theoretische Grundlagen

Constraints beschreiben Bedingungen, die einzuhalten sind, und stellen somit ein universelles Beschreibungsmittel von Problemen dar. Bei praktischen Anwendungen der Constraint Logischen Programmierung (CLP) zeigt sich jedoch, daß nicht immer alle Bedingungen exakt einzuhalten sind. Lösungen sind nur durch Abschwächung bzw. Aufweichung (Constraint-Relaxation) der Bedingungen möglich. Die Bedingungen sollen – wenn sie schon nicht einzuhalten sind – bestmöglich berücksichtigt werden.

Eine Möglichkeit, dies zu modellieren, stellen Constraint-Hierarchien dar. Die Constraints werden dazu in mehrere Hierarchiestufen (siehe Abbildung 1) eingeteilt, wobei die Stufen den "Härtegrad" der Constraints repräsentieren. Constraints auf der Hierarchiestufe 0 sind notwenige, unbedingt zu erfüllende. Mit wachsendem Level werden die Constraints der jeweiligen Hierarchiestufe weniger bedeutend – also schwächer bzw. weicher – und gehen somit entsprechend dieser Priorität in die Lösungsberechnung ein. Eine Lösung einer solchen Constraint-Hierarchie ist eine Wertebele-

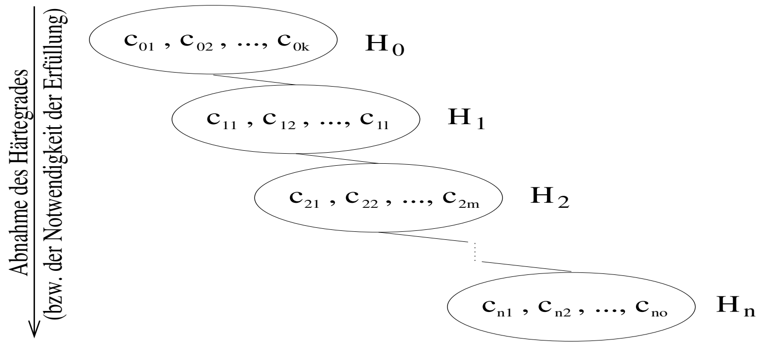

Abbildung 1: Struktur einer Constraint-Hierarchie

gung[2], die alle notwendigen Constraints auf der Stufe H_0 erfüllt und die restlichen Constraints bestmöglich berücksichtigt.

Um Wertebelegungen (σ, θ) einer Constraint-Hierarchie H miteinander vergleichen zu können, wird ein Komparator $better_{e,g}$ bezüglich einer Fehlerfunktion e und einer Kombinationsfunktion g definiert. Die Fehlerfunktion e ist ein Maß für die Erfüllung *eines* Constraints. Eine Erweiterung dieser Fehlerfunktion ist die Funktion E, die wertebelegte Constraint-Vektoren auf reellwertige Vektoren abbildet. Für einen Constraint-Vektor $C = (c_1, \ldots, c_k)$ und eine Wertebelegung σ ist sie definiert durch $E(C\sigma) := [e(c_1\sigma), \ldots, e(c_k\sigma)]$. Die Kombinationsfunktion g kombiniert die "Einzelfehler" $e(c_1\sigma), \ldots, e(c_k\sigma)$ auf jeder Hierarchiestufe H_i zu einem "Gesamtfehler" $g(E(H_i))$. Als Beispiele für solche Funktionen seien hier die *triviale Fehlerfunktion* für e, die bei Erfüllung des Constraints den Wert 0 und bei Verletzung dessen den Wert 1 liefert, und die Summe der Einzelfehler als Kombinationsfunktion g genannt.

Die Verallgemeinerung einer Kombinationsfunktion g ist eine Funktion G. Sie ist definiert durch $G([E(H_1\sigma), \ldots, E(H_n\sigma)]) := [g(E(H_1\sigma)), \ldots, g(E(H_n\sigma))]$.

$[g(E(H_1\sigma)), \ldots, g(E(H_n\sigma))]$ heißt *kombinierte Fehlerfolge* und stellt den "Gesamtfehler" dar, den eine Wertebelegung σ in der Hierarchie H verursacht. Die so ermittelten Gesamtfehler zweier Wertebelegungen werden lexikographisch miteinander verglichen. Eine Wertebelegung θ ist besser

[2]Eine Wertebelegung bildet die freien Variablen der Constraints auf Werte aus ihrem Domänbereich ab.

als eine Wertebelegung σ, wenn der Gesamtfehler, den die Wertebelegung θ erzeugt, ab einer Hierarchiestufe k kleiner ist, als der Gesamtfehler, den die Wertebelegung σ erzeugt. Das heißt, für eine kombinierte Fehlerfolge $(x_1, \ldots, x_n)$ und eine kombinierte Fehlerfolge $(y_1, \ldots, y_n)$ gilt: $(x_1, \ldots, x_n) <_G (y_1, \ldots, y_n) \leftrightarrow \exists k : k \leq n \wedge x_k <_g y_k \wedge (\forall i < k : x_i <>_g y_i)$, wobei $<_g$ irreflexiv, antisymetrisch und transitiv ist und $<>_g$ eine reflexive und symmetrische Relation ist.

Eine Lösung $S(H)$ einer Hierarchie H bezüglich einer Fehlerfunktion e und einer Kombinationsfunktion g ist wie folgt definiert:

$$\begin{aligned} S_0(H) &:= \{\theta \mid \forall c \in H_0 : e(c\theta) = 0\} \\ S(H) &:= \{\theta \mid \theta \in S_0(H) \wedge \forall \sigma \in S_0(H) : \neg better_{e,g}(\sigma, \theta, H)\}. \end{aligned}$$

Eine Wertebelegung θ erfüllt die notwendigen Constraints auf der Hierarchiestufe H_0 und es gibt keine Wertebelegung σ, die ebenfalls alle notwendigen Constraints erfüllt und besser ist als θ. Dies schließt jedoch nicht aus, daß mehrere "gleich gute" Lösungen existieren.

Für eine ausführlichere Darstellung der theoretischen Grundlagen sei auf die Dissertation von M. A. Wilson [11] verwiesen.

2.2 Realisierung eines Constraint-Hierarchie-Lösers

Aufgebaut wird auf der in [12, 13] vorgestellten Transformation von Constraint-Hierarchien in "Geordnete Constraint-Hierarchien"[3]. Dieses Verfahren wurde prototypisch implementiert. Gegenüber dem in [12] zugrunde liegenden Prototypen, können nun beliebig viele Constraints pro Level *fair* berücksichtigt werden. Unter *Fairness* verstehen wir hierbei, daß das Ergebnis der Gesamtfehlerermittlung unabhängig von der Reihenfolge der Constraints auf den einzelnen Hierarchiestufen ist.

Neben der Realisierung der trivialen Fehlerfunktion galt unser spezielles Augenmerk der Implementierung eines Constraint-Hierarchie-Lösers mit folgender naheliegenden nichttrivialen Fehlerfunktion. Der Fehler e für ein Constraint $a \leq b$ auf den Hierarchiestufen $H_1, \ldots, H_n$ ist definiert durch:

$$e(a \leq b) \quad := \quad \begin{cases} 0 & \text{falls } a \leq b \\ a - b & \text{sonst} \end{cases}$$

Dabei sind a und b lineare Terme[4].

Die zu lösenden Constraint-Hierarchien dürfen auf der Hierarchiestufe H_0 Constraints enthalten, die durch den zugrunde liegenden Constraint-Löser lösbar sind. Auf den Hierarchiestufen $H_1, \ldots, H_n$ sind lineare[5] sowie alldistinct/1[6]- und allequal/1[7]-Constraints zulässig. Entsprechend der Auswahl eines Komparators durch den Benutzer (siehe dazu Tabelle 1) wird die Constraint-Hierarchie in ein herkömmliches Constraintsystem transformiert und dieses durch den Constraint-Löser des CLP-Systems gelöst.

[3] Die Constraints sind auf den jeweiligen Stufen nach ihrer Priorität total geordnet, so daß jedes Constraint somit als eigene Hierarchiestufe betrachtet werden kann.

[4] Unter einem linearen Term wird in CHIP eine natürliche Zahl, eine Domain-Variable, das Produkt aus einer natürlichen Zahl und einer Domain-Variablen sowie die Summe zweier linearer Terme verstanden.

[5] Darunter sind die mathematischen Constraints $\# <, \# \leq, \# =, \# \geq, \# >$ und $\# \neq$ zu verstehen.

[6] Bei `alldistinct(Liste)` sollen die Domänwerte der Variablen der Liste paarweise von einander verschieden sein.

[7] Analog zu `alldistinct(Liste)`, nur sollen hier die Domänwerte der Variablen der Liste gleich sein.

Das Lösen einer Constraint-Hierarchie läßt sich in drei Stufen darstellen:

1. Vorverarbeitung der Constraints auf den Hierarchiestufen $H_1, \ldots, H_n$

Alle durch den Constraint-Hierarchie-Löser verarbeitbaren weichen Constraints (auf den Hierarchiestufen $H_1, \ldots, H_n$) werden in Constraints der Form $t \leq b$ überführt, wobei t und b lineare Terme sind:

$$
\begin{array}{rclcl}
t & < & b & \equiv & t+1 \leq b \\
t & = & b & \equiv & t \leq b \wedge b \leq t \\
t & \geq & b & \equiv & b \leq t \\
t & > & b & \equiv & b+1 \leq t \\
t & \neq & b & \equiv & t+1 \leq b \vee b+1 \leq t
\end{array}
$$

Die Disjunktion wird durch die Benutzung jeweils einer "Boolschen Variablen" in eine Konjunktion überführt (siehe [10]), `alldistinct/1` und `allequal/1`-Constraints werden zuvor in eine Menge von "$\neq$"- bzw. "="-Constraints umgewandelt.

2. Transformation der Constraints

Nach der Vorverarbeitung liegen nur noch Constraints der Form $t \leq b$ vor. Sei $t_i^l \leq b_i^l$ das i-te Constraint auf der Hierarchiestufe l ($l \geq 1$), so wird dieses in ein neues Constraint $t_i^l - z_l \leq b_i^l$ transformiert[8]. Dabei ist z_l eine "neue" Variable, die den Fehler des Constraints repräsentiert. Erweitert auf die gesamte Hierarchie H läßt sich die Transformation der vorliegenden Constraintmenge in das Constraintsystem H' wie folgt definieren:

$$
\begin{array}{rcl}
H'_0 & = & H_0 \cup \bigcup_{l=1}^{m} [\{z_l \geq 0\} \cup \bigcup_{i=1}^{k_l} \{t_i^l - z_l \leq b_i^l\}] \\
H'_l & = & \{z_l \leq 0\} \quad \text{für } l = 1, \ldots, m
\end{array}
$$

Die notwendigen Constraints H'_0 der transformierten Hierarchie H' werden aus den notwendigen Constraints H_0 der ursprünglichen Hierarchie H vereinigt mit den transformierten, ursprünglich "weichen" Constraints und der Bedingung, daß der Fehler eine nichtnegative Zahl ist, gebildet. Der Fehler auf jedem Level soll möglichst klein sein (H'_l). Dabei sind $z_1, \ldots, z_m$ "neue" und paarweise verschiedene Variablen ($var(H) \cap \{z_1, \ldots, z_m\} = \emptyset$ und $z_i \not\equiv z_j$ für $i \neq j$), die die zusammengefaßten Fehler der Constraints auf den jeweiligen Hierarchiestufen repräsentieren und sich je nach gewähltem Komparator unterschiedlich berechnen.

3. Labeling der Fehlervariablen

Nach Anwendung der Transformation erhält man ein mehrstufiges Minimierungsproblem, dessen Nebenbedingungen die Constraints in H'_0 und dessen Zielfunktionen die Minimierung der Fehler z_l auf den jeweiligen Hierarchiestufen H'_l, beginnend mit z_1, sind. Dieses wird duch das Labeling der Fehlervariablen und der Variablen der ursprünglichen Constraint-Hierarchie gelöst.

Die Transformation besitzt linearen Aufwand. Bei der Abarbeitung der transformierten Form H' verursacht die Suche nach der minimalen Wertebelegung für die Fehler z_l, wegen der Benutzung eines unvollständigen Constraint-Lösers, eine höhere Komplexität. Daher wurden mehrere Labelingstrategien - u.a. ein branch-und-bound-Algorithmus und das "incremental search - Verfahren", vorgestellt in [4] - realisiert (siehe dazu [10]).

Im Prototypen, der sowohl in CHIP als auch in ECLIPSE implementiert ist, beschränken wir uns auf globale Komparatoren, welche im Gegensatz zu lokalen oder regionalen Komparatoren,

[8] Bei Benutzung der nichttrivialen Fehlerfunktion.

die Einzelfehler zu einem Gesamtfehler zusammenfassen. Insbesondere sind dies *weighted-sum-metric-better*, *worst-case-metric-better* und *least-squares-metric-better* (siehe Tabelle 1) sowie ein kombinierter Komparator aus den beiden ersteren. Dieser besitzt die Vorteile des *least-squares-metric-better*-Komparators und findet darüber hinaus entsprechende, intuitiv erwartete Lösungen auch für Hierarchien, wie beispielsweise diejenigen, die auf einem Level größer Null das Constraint *alldistinct(Liste)* mit n Listenelementen beinhalten, für die aber jeweils nur $n-1$ Domänwerte auf dem Level 0 definiert sind:

$$\begin{aligned} H_0 &:= \{1 \leq X_i \leq n-1 \mid 1 \leq i \leq n\}, \\ H_1 &:= \{X_i \neq X_j \mid 1 \leq i < j \leq n\}. \end{aligned}$$

Bei der Implementierung des Prototypen wurde darauf Wert gelegt, daß die Syntax der zu lösen-

Komparator	weighted-sum-metric-better	worst-case-metric-better	least-squares-metric-better
Beschreibung	Geamtfehler der jeweiligen Hierarchiestufe entspricht der Summe der Einzelfehler der Constraints	Gesamtfehler des Levels bestimmt sich nach dem größten Einzelfehler	Gesamtfehler wird aus der Summe der Quadrate der Einzelfehler berechnet
Vorteile	gerechtere Lösung bei Ungleich-Constraints	gleichmäßige Fehlerminimierung	gerechtere Lösung bei Ungleich-Constraints *und* gleichmäßige Fehlerminimierung

Tabelle 1: Einteilung der Komparatoren

den Constraint-Hierarchie der Syntax für Constraints bekannter Constraint-Löser, wie CHIP und ECLIPSE ähnlich ist. So unterscheidet sie sich von einem “normalen” Constraintsystem nur dadurch, daß jedes Constraint mit einer Zahl – welche die Hierarchiestufe angibt – zu einem zweistelligen Term mit dem Funktor *c/2* zusammengesetzt wird. Diese Terme bilden dann die Elemente einer Liste – der Constraint-Hierarchie, welche wiederum das Argument des Aufrufs der Prozedur *hierarchy/1* ist. Dabei impliziert die Verwendung des gleichen Levels die gleiche Priorität der entsprechenden Constraints. Die Contraints in der Liste müssen keiner Ordnung unterliegen, da sie vor der Transformation entsprechend ihrer “Härte” in eine interne Repräsentation umgewandelt werden. Zur Verdeutlichung der Syntax ein Beispiel:

Ein Kabelbündel mit 17 Kabeln soll in 3 Kabelbäume aufgeteilt werden. Dabei steht für den ersten Kabelbaum etwa die Hälfte des Kabelschachtdurchmessers zur Verfügung, für den zweiten etwa der dritte Teil und für den letzten noch ca. ein Neuntel:

als “normales” Constraintsystem nicht lösbar:

```
kabel(X, Y, Z) :-
   [X,Y,Z] :: 0..20,
   S = 17,
   X + Y + Z #= S,
   2 * X #= S,
   3 * Y #= S,
   9 * Z #= S.
```

als Constraint-Hierarchie mit Fehlerwert[9] 1 auf Level 1 lösbar (X=9, Y=6, Z=2):

```
kabel(X, Y, Z) :-
   [X,Y,Z] :: 0..20,
   S = 17,
   hierarchy([ c(0, X + Y + Z #= S),
               c(1, 2 * X #= S),
               c(1, 3 * Y #= S),
               c(1, 9 * Z #= S) ]).
```

[9]bei Benutzung der nichttrivialen Fehlerfunktion und des worst-case-metric-better-Komparators

Wie man sieht, sind "normale" Constraintsysteme ohne großen Aufwand in hierarchische umzuformen.

3 Konfiguration mit weichen Constraints

3.1 Die Konfigurationsproblematik

Bei der Aufgabe, ein Gerät aus Baugruppen zusammenzusetzen, ist zunächst eine vollständige korrekte Spezifikation der Zusammensetzung des Gerätes gesucht, so daß alle technischen und problemspezifischen Anforderungen erfüllt sind. Darüber hinaus sollen bestimmte Zielstellungen (z.B. Unterstützung späterer Erweiterbarkeit, Kostenminimalität oder Verbrauchsarmut) befriedigt werden. Für die Problemlösung stehen den Projektanten Informationen bzgl. der verfügbaren technischen Module, deren Struktur und Eigenschaften sowie Wissen über Kompatibilität und Constraints zwischen den Modulen zur Verfügung. Daneben verfügen sie über Vorgehenswissen in Form von Heuristiken[10]. Die für die Akzeptanz eines rechnergestützten Konfigurationssystems notwendige Zielstellung, einerseits alle Aspekte eines beliebigen Konfigurationsproblems in einfacher Weise formal spezifizieren zu können und andererseits diese effzient in systeminterne Repräsentationen überführen zu können, verdeutlicht die Relevanz einer entsprechenden Spezifikationssprache. Als Grundlage einer solchen Sprache entwickelten wir die Semi-Level-Spezifikationssprache ConBaCon-L (vgl. [7]).

3.2 Problemspezifikation

Das Konfigurationswerkzeug ConBaCon bietet Konstrukte zur Spezifikation von *Objektstrukturen*, *Objekteigenschaften*, *High-Level-Constraints zwischen Objekten und/oder Eigenschaftswertemengen* sowie von *weichen Constraints*. Die High-Level-Constraints sind unterteilt in *Simple Constraints*, *Compositionale Constraints*, *Conditionale Constraints* und *Tabellarische Constraints*. *Simple Constraints* sind:

- Anforderungen bezüglich der Existenz oder Nicht-Existenz von Objekten in der gewünschten Lösung,
- Anforderungen bezüglich technischer Bezeichner von Objekten[11],
- Anforderungen bezüglich möglicher und nicht-möglicher Eigenschaftswertemengen,
- Anforderungen (Relationen) zwischen verschiedenen Eigenschaftswertemengen.

Compositionale Constraints sind neben simplen Constraints konjunktive und disjunktive Verknüpfungen von simplen Constraints. *Conditionale Constraints* sind folgende Konstrukte:

- [if(Comp_Cons$_I$), then(Comp_Cons$_T$)]

  Wenn das compositionale constraint Comp_Cons$_I$ erfüllt ist, muß das compositionale Constraint Comp_Cons$_T$ auch erfüllt sein.

[10] Da die Vorgehensheuristiken *eine* (hoffentlich korrekte) Lösung liefern, die selten globale Zielstellungen erfüllt, und i. allg. lediglich der manuellen Handhabbarkeit des Problems dienen, spielen sie im formalen Problemmodell keine Rolle.

[11] Objekte können u.U. abhängig von ihren konkreten Eigenschaftswerten verschiedene technische Bezeichner führen.

- [iff(Comp_Cons_1), then(Comp_Cons_2)]

  Genau dann, wenn das compositionale Constraint Comp_Cons_1 erfüllt ist, muß das compositionale Constraint Comp_Cons_2 erfüllt sein.

Die Verarbeitung *Tabellarischer Constraints* kann einerseits über eine Transformation in sog. explizite Alternativen (vgl. [8]) oder über eine Transformation der Tupel-Mengen zu conditionalen Constraints realisiert werden.

Zusätzlich zu den genannten Spezifikationskonstrukten, ist es notwendig, die Spezifikation von *weichen Constraints*, zu unterstützen, da solche in praxisrelevanten Konfigurationsproblemen oftmals in Form von Präferenzen vorkommen. So können in ConBaCon-L *weiche simple Constraints* in der Form $weak(Simp\_Cons, Level)$ spezifiziert werden, wobei Level den Weichheitsgrad des Constraints $Simp\_Cons$ benennt (vgl. Kapitel 2). Perspektivisch ist auch die Spezifizierbarkeit und Verarbeitung *weicher compositionaler Constraints* ($weak(Comp\_Cons, Level)$) und *weicher conditionaler Constraints* ($[if(Comp\_Cons_I), then(weak(Comp\_Cons_T, Level))]$) notwendig.

Der Gerätespezifikation mit Hilfe der vorher genannten Constraints ist die Spezifikation problem- bzw. kundenspezifischer Constraints für das gewünschte Gerät hinzuzufügen.

3.3 Anwendung von Constraint-Hierarchien

Der Konfigurationsraum, der durch ConBaCon nach Absetzen der spezifizierten Constraints berechnet wird, hat eine Form wie in Abbildung 2 (durch die Menge aller Bestandteile des Baumes dargestellt; gestrichelte Kanten stellen Alternativen im Sinne der *is_a*-Relation dar, vollständige Linien repräsentieren Komponentenbeziehungen im Sinne der *consist_of*-Relation). Durch Inter-

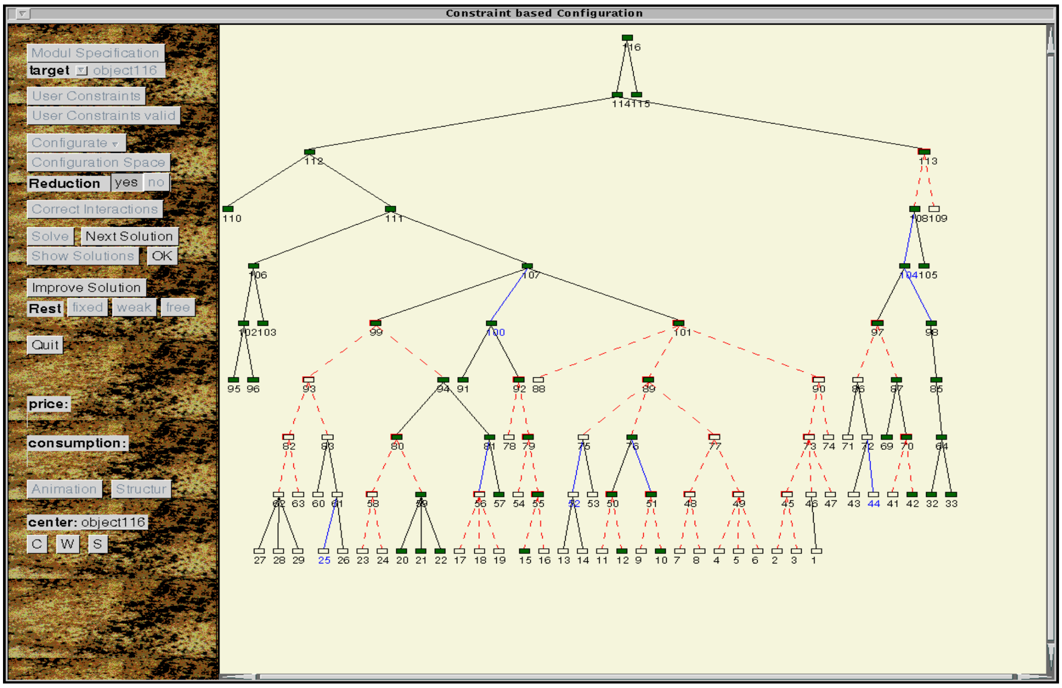

Abbildung 2: *Konfigurationsraum mit einer Lösung*

aktionen auf der graphischen Benutzeroberfläche kann der Projektant weitere Constraints zur Spezifikation hinzufügen (oder auch wieder entfernen). Wird im Laufe des Interaktionszyklus eine

Lösung generiert, die an bestimmten Stellen verbesserungsbedürftig ist, kann mit dem herkömmlichen Ansatz, der lediglich "harte" Constraints unterstützte, eine lokale Verbesserung nur unter *strikter Beibehaltung* aller anderen gewünschten Bestandteile spezifiziert werden. Im Normalfall wird dieser Ansatz scheitern und keine Lösung generieren, da einzelne harte Constraints verletzt werden. Die intuitiv verständliche Forderung, die lokalen Verbesserungen unter *bestmöglicher Beibehaltung* der restlichen Bestandteile durchzuführen, ließe sich lediglich durch sehr zeitintensive und komplexe Simulationszyklen realisieren. Eine Behebung dieser Schwierigkeit ist die Erweiterung des bestehenden Ansatzes um Spezifikations- und Verarbeitungsmöglichkeiten von weichen Constraints, die mit Hilfe des Modells von Constraint-Hierarchien modelliert und verarbeitet werden. Zur Illustration dieser gewünschten Systemeigenschaft dient das in Abbildung 2 verwendete Beispiel, welches mit dem Konfigurations-Problem-Generator GCP[12] generiert wurde. Die Problemspezifikation besteht aus 116 Objekten mit zufällig generierten Eigenschaften (Objekt 116 soll konfiguriert werden). Eine mögliche Lösung ist in Abbildung 2 dargestellt (die gefüllten Rechtecke entsprechen Lösungsbestandteilen). Beispielhaft sei angenommen, die gefundene Lösung wird vom Projektanten mit Ausnahme des Vorkommens von Objekt 108 (oben rechts) als gut bewertet. Darum gibt er dem System die Verbesserungsforderung, daß Objekt 109 anstelle des Objektes 108 zu verwenden sei. Auf Grund des integrierten CH-Lösers ist der Nutzer in der Lage zu fordern, daß alle anderen Bestandteile der bestehenden Lösung in der verbesserten Lösung *so weit wie möglich* enthalten sein sollen. Das Ergebnis des Verbesserungsprozesses ist in Abbildung 3 dargestellt. Objekt 108 wurde mit allen Komponenten durch Objekt 109 ersetzt. Der

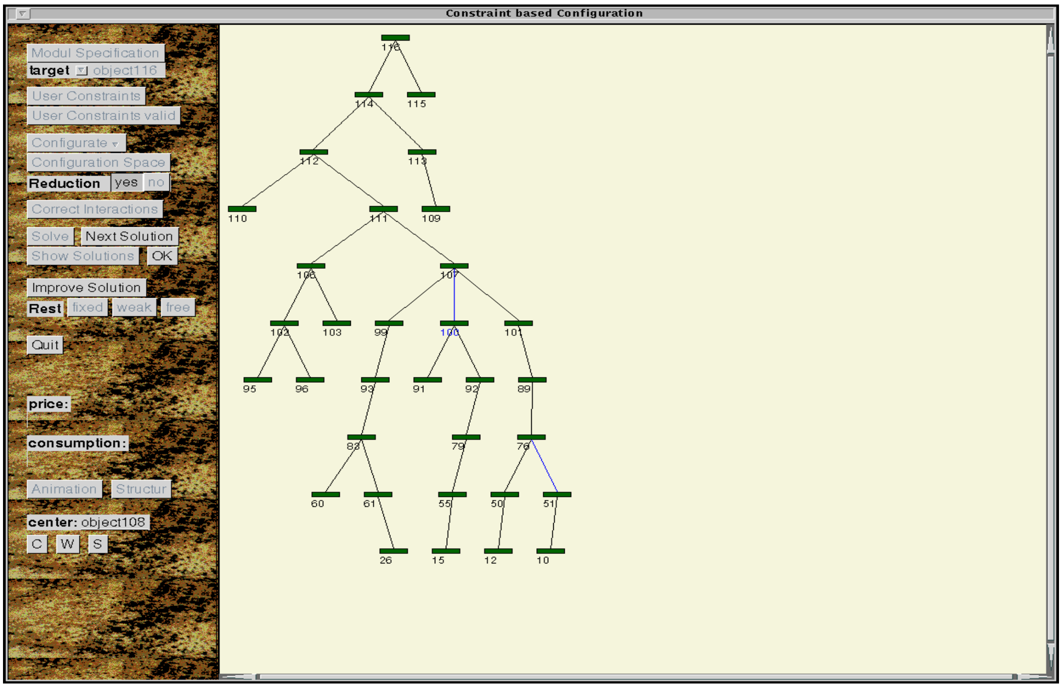

Abbildung 3: *Eine verbesserte Lösung*

Wunsch, alle anderen Lösungsbestandteile zu erhalten, konnte aufgrund spezifizierter Constraints nicht vollständig erfüllt werden. Zum Beispiel war in der initialen Problembeschreibung mittels conditionaler Constraints spezifiziert worden, daß Objekt 93 (links) in der Lösung vorkommen muß, wenn Objekt 109 Bestandteil der Lösung ist. Infolge desssen kann das zu 93 alternative Objekt 94 nicht in der verbesserten Lösung vorkommen. Die anderen Bestandteile der ursprünglichen

[12]GCP - Generation of Configuration Problems; vgl. [7]

Lösung sind, wie gewünscht, ebenfalls Teil der neuen Lösung. So sind z.B. die Objekte 15, 12 und 10 enthalten. Die Vorgehensweise bei der Ableitung dieser Lösung ist die folgende: Der Nutzer spezifiziert die möglichst beizubehaltenden Lösungsbestandteile (oder auch Lösungseigenschaften) als weich[13]. Zusätzlich zu den in der initialen Problemspezifikation gegebenen Constraints, die ständig aktiv sind, werden die interaktiv spezifizierten harten Constraints abgesetzt. Die interaktiv spezifizierten weichen Constraints werden dem CH-Löser übergeben. Dieser bestimmt den kleinsten Fehler der zugehörigen Constraint-Hierarchie und reduziert die Domänen der beteiligten Domänvariablen. Der neue Konfigurationsraum (samt verzögerter Constraints) steht danach fest. Innerhalb dessen kann nun die Lösung generiert werden.

Abbildung 4 macht deutlich, welche Schwierigkeiten das gleiche Verbesserungsproblem für ein constraint-basiertes Konfigurationswerkzeug ohne die Fähigkeit, weiche Constraints zu verarbeiten, bereiten würde. Zwei Vorgehensweise kämen in Betracht. Einerseits wäre es, wie bereits

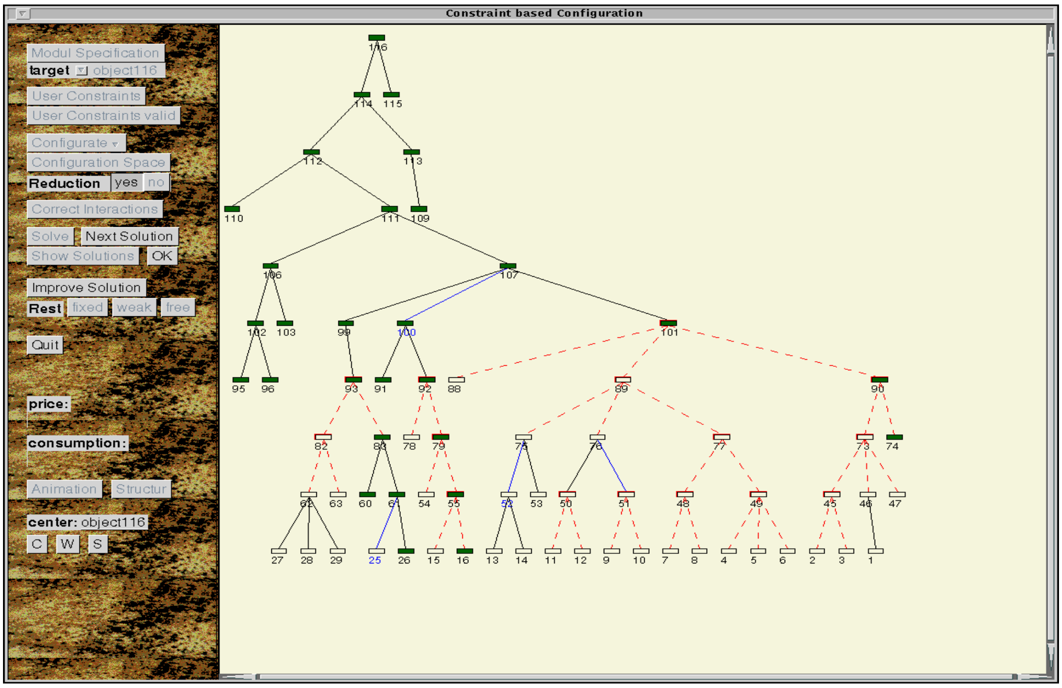

Abbildung 4: *Eine nur teilweise verbesserte Lösung*

angedeutet, möglich, alle weichen Constraints als harte Constraints zu spezifizieren. Andererseits könnten alle weichen Constraints als nicht existent betrachtet werden. Im ersten Fall kann das System nur eine Lösung ermitteln, wenn *alle* Constraints erfüllt werden können. Im zweiten Fall (Abbildung 4) berechnet das System einen neuen Konfigurationsraum, der die (harten) Verbesserungs-Constraints berücksichtigt. Alle erwünschten Bestandteile der ursprünglichen Lösung werden dagegen vergessen. Der Nutzer könnte nun versuchen, diesen Mangel durch Angabe entsprechender interaktiver Constraints, die den Konfigurationsraum entsprechend einschränken, zu beheben. Dies wäre sehr zeitintensiv und nutzerseitig in hohem Maße unübersichtlich.

[13]Darüber hinaus kann der Nutzer auch Bestandteile oder Eigenschaften als hart oder als "beliebig" spezifizieren.

4 Ausblick

Die Realisierung der Verarbeitung weicher Constraints auf Grundlage des Modells von Constraint-Hierarchien bringt für den Fall der Konfiguration technischer Systeme erhebliche Vorteile gegenüber herkömmlichen Ansätzen. Diesbezüglich vorgesehene weiterführende Arbeiten auf diesem Gebiet sind zum einen die bereits in Abschnitt 3.2 benannten Erweiterungen bzgl. der Spezifikation und Verarbeitung von *weichen compositionalen Constraints* und *weichen conditionalen Constraints* und zum anderen seitens des Constraint-Hierarchie-Lösers Auswertungsrealisierungen weiterer Constrainttypen bis hin zu globalen Constraints, wie *cumulative* und *cardinality.* Darüber hinaus ist alternativ zu der beschriebenen Realisierung des Constraint-Hierarchie-Lösers eine Realisierung mittels Conditional Propagation Rules, welche Bestandteil des CHIP-Systems sind, geplant. Die gesammelten Erfahrungen sollen die Integration von Constraint-Hierarchie-Verarbeitungsmechanismen in andere constraint-basierte Planungswerkzeuge erleichtern.

Literatur

[1] T. Axling; S. Haridi: A Tool for Developing Interactive Configuration Applications. Journal of Logic Programming, 1996.

[2] Th. Christaller; D. Schuett (Eds.): Grundlagen und Anwendungen der KI. Springer Verlag, 1993.

[3] A. Günter: Modelle beim Konfigurieren. In [2].

[4] Pascal van Hentenryck and Thierry le Provost. Incremental Search in Constraint Logic Programming. New Generation Computing, 1991.

[5] Over Constraint Systems, Springer Verlag, 1996.

[6] U. John: Constraint-Based Simulation of Configuration Processes. Proceedings of IMACS'97, August 1997.

[7] U. John: Constraint-basierte Konfiguration. To appear as GMD - Arbeitspapier, 1997.

[8] A. Nareyek; U. Geske: Behandlung expliziter Alternativen in CLP(FD). GMD-Studien, Nr. 297, GMD, Sankt Augustin, Schloß Birlinghoven, 1996.

[9] M. Polke: Prozeßleittechnik. Oldenburg Verlag, 1994.

[10] André Schiemann. Erweiterung und Implementierung von Verfahren zum Lösen von Constraint-Hierarchien unter Berücksichtigung mehrerer Fehlerfunktionen. Diplomarbeit (to appear), TU-Berlin, 1997.

[11] M. A. Wilson: Hierarchical Constraint Logic Programming. PhD thesis, Dept. of Computer Science and Engineering, University of Washington, 1993.

[12] A. Wolf: Transforming Ordered Constraint Hierarchies into Ordinary Constraint Systems. In [5].

[13] A. Wolf: Solving Hierarchies of Finite-Domain Constraints. Journal on Experimental and Theoretical AI, to appear in a special issue on non-standard constraint processing, 1997.