Ein hybrides Framework für Constraint-Solver zur Unterstützung wissensbasierter Konfigurierung

Wolfgang Runte

Technologie-Zentrum Informatik, Universität Bremen
Postfach 33 04 40, D-28334 Bremen
Am Fallturm 1, D-28359 Bremen
woru@tzi.de
http://www.tzi.de

Zusammenfassung In wissensbasierten Konfigurierungssystemen werden Komponenten zur Constraint-Verarbeitung für die Verwaltung von Abhängigkeiten während des Konfigurierungsprozesses eingesetzt. Die Effizienz von Constraint-Lösungsverfahren ist allerdings stark problemabhängig. Zudem kann es aufgrund einer Vielzahl unterschiedlicher Domänen und der Vielfältigkeit der Anwendungsszenarien notwendig werden, an die jeweilige Domäne angepasste Constraint-Lösungsverfahren einzusetzen. Zur Behandlung von unterschiedlichen Constraint-Domänen innerhalb des strukturbasierten Konfigurierungswerkzeugs ENGCON sind Constraint-Solver sowohl für finite als auch infinite Domänen erforderlich. Die Steuerung muss durch eine Komponente geleistet werden, mit der sich unterschiedliche Constraint-Solver je nach Bedarf und domänenspezifisch einsetzen lassen. Die Modularität ist dabei entscheidend für die Austauschbarkeit einzelner Komponenten. Es stellt daher einen sinnvollen Ansatz dar, ein modulares Constraint-Framework einzusetzen, in das je nach Bedarf, und dem in der jeweiligen Wissensbasis definiertem Problem, unterschiedliche Constraint-Solver mit jeweils für das spezifische Problem effizienten Lösungsverfahren eingesetzt werden können. Das Framework wird durch einen strategiebasierten Ansatz unterstützt, der eine flexible Kooperation unterschiedlicher Constraint-Lösungsalgorithmen erlaubt.

Key words: Wissensbasierte Konfigurierung, Konfiguration, Constraint Satisfaction Problem, Constraint-Solver, Java-Framework, finite, infinite, diskrete, kontinuierliche Domänen, reellwertige Intervalle

1 Motivation

Für die Konfigurierung variantenreiche Produkte lassen sich von Konfigurierungswerkzeugen mit unterschiedlichen wissensbasierten Methoden aus einzelnen Komponenten komplexe Aggregate erstellen. Constraints[1] sind ein Mittel

[1] *constraint* (engl.): Einschränkung, Beschränkung, Restriktion

zur Repräsentation von Abhängigkeiten zwischen den Komponenten einer Konfiguration (vgl. [11], [23], [25]).

Wissensbasierte Konfigurierungswerkzeuge verwenden Constraints zur Beschreibung von Abhängigkeiten zwischen Konzepten der Wissensbasis [19]. Die Auflösung der Abhängigkeiten wird von integrierten Constraint-Systemen verwaltet [26]. Die eingesetzten Constraint-Solver hingegen werden üblicherweise nicht innerhalb des eigenen Systems implementiert, sondern sondern sind von Fremdherstellern eingebundene externe Komponenten. Diese weisen häufig Eigenschaften auf, die den Einsatz innerhalb eines wissensbasierten Konfigurierungssystems einschränken. Zudem ist die Anbindung an das Konfigurierungssystem in der Regel auf einen einzigen Constraint-Solver beschränkt und damit sehr unflexibel.

Benötigte Constraint-Solver müssen arithmetische Funktionen zur Berechnung von intensional in Form von algebraischen Ausdrücken formulierten Constraints bieten. Neben klassischen Constraint-Solvern zur Behandlung von finiten Domänen sind für die Constraint-Verarbeitung in wissensbasierten Konfigurierungswerkzeugen Constraint-Lösungsmethoden für infinite, d. h. reellwertige Intervalldomänen erforderlich. Entsprechende Constraint-Solver müssen eine hohe Präzision durch Intervallarithmetik aufweisen (z. B. für Anwendungen im Maschinenbau) sowie unabhängig von der Constraint-Domäne ein inkrementell anwachsendes Constraint-Netz propagieren können.

Darüber hinaus ergeben sich Anforderungen an den Constraint-Lösungsmechanismus häufig in Abhängigkeit von der Aufgabenstellung des jeweiligen Konfigurierungsproblems. Neben stabilen Constraint-Lösungsverfahren, die eine hohe Effizienz für möglichst viele Problemstellungen bieten, ist es deshalb erforderlich, problemabhängig unterschiedliche Verfahren nutzen zu können. Benötigt wird eine Komponente, an der sich flexibel unterschiedliche Constraint-Solver mit verschiedenen Eigenschaften, sowohl bezogen auf die Lösungsverfahren als auch auf die zu verarbeitenden Wertedomänen, einbinden lassen.

Die vorliegende Ausarbeitung gliedert sich in die folgenden Anschnitte: Nach einer kurzen Einführung in die wissensbasierte Konfigurierung mit EngCon in Abschnitt 2 werden in Abschnitt 3 die benötigen Methoden zur Constraint-Verarbeitung vorgestellt. In Abschnitt 4 wird das Konzept für ein hybrides Constraint-Framework entwickelt. Anschließend wird in Abschnitt 5 eine Übersicht über verwandte Arbeiten gegeben. Die Ausarbeitung endet in Abschnitt 6 mit einer Zusammenfassung und einem Ausblick.

2 Wissensbasierte Konfigurierung mit EngCon

Wissensbasierte Konfigurierungssysteme nutzen deklarativ und explizit repräsentiertes Wissen, um komplexe Konfigurierungsaufgaben zu lösen oder Experten bei der Lösung einer Konfigurierungsaufgabe zu unterstützen. Sowohl EngCon [19] als auch dessen Vorläufer PlaKon [5] und KonWerk [10] sind in erster Linie strukturbasierte Konfigurierungswerkzeuge. Der Schwerpunkt dieser Systeme liegt auf einem begriffshierarchie-orientierten Kontrollmechanismus. Zusätzlich

kommen weitere Inferenzmechanismen zum Einsatz, wie z. B. ein ausgereiftes Constraint-System.

Der Schwerpunkt von EngCon liegt im Gegensatz zu klassischen Systemen nicht auf Inferenzen, die aufgrund von Expertenregeln gebildet werden („regelhafte Expertise“), sondern auf Inferenzmechanismen, die aufgrund der wissensbasierten Architektur der Domäne angewendet werden. Zur Repräsentation des Objektwissens der Domäne wird in EngCon eine framebasierte Repräsentation verwendet, die die zusammengefasste Spezifikation der Objektstruktur, der Eigenschaften und möglichen Belegungen in einer sog. *Begriffshierarchie* innerhalb der Wissensbasis ermöglicht. In einer Ontologie stehen hier die Konzepte über *is-a*- und *has-parts*-Relationen in taxonomischen und partonomischen Hierarchien zueinander in Beziehung.

Die Art des Vorgehens bei der Lösung des Konfigurierungsproblems wird durch den strukturbasierten Konfigurierungsansatz anhand der Struktur der Konfigurierungsobjekte innerhalb dieses Domänenmodells definiert. Zur Erstellung einer Konfiguration werden die Konfigurierungsschritte *Zerlegung* (Instantiierung der Bestandteile eines Aggregates), *Spezialisierung* („Verfeinerung“ einer Instanz zu einer spezielleren Instanz) und *Parametrierung* (Setzen bzw. Einschränken der Wertebereiche von Objekteigenschaften) eingesetzt [19].

Die Wissensbasis von EngCon ist eine flexibel austauschbare, deklarative Beschreibung, d. h. für jede Anwendung kann spezielles Wissen definiert und eingebunden werden. EngCon ist dadurch domänenunabhängig und kann durch einen Austausch der Wissensbasis beliebige variantenreiche Komponenten konfigurieren.

Eine Besonderheit von EngCon ist der inkrementell und interaktiv verlaufende Konfigurierungsprozess. Am Ende einer strukturbasierten Konfigurierung steht immer genau eine Lösung. Im Gegensatz zu anderen Konfigurierungswerkzeugen, die häufig eine Breitensuche vornehmen und abschließend alle gefundenen Lösungen präsentieren, findet während der Konfigurierung mit EngCon eine benutzergesteuerte Tiefensuche statt, mit dem Ziel, interaktiv eine für den Benutzer geeignete Lösung zu finden.

3 Constraints

Die Verwendung von Constraints ist eine vielfach eingesetzte Methode zur Repräsentation und Auswertung von Abhängigkeiten. Durch Constraints werden Relationen zwischen (Constraint-)Variablen definiert. Neben der Definition solcher „Beschränkungen“ werden Constraints eingesetzt, um die Werte von Variablen dynamisch den Anforderungen der Constraints entsprechend anzupassen. Constraints sind in diesem Sinne Randbedingungen, welche die Konsistenz der Variablenwerte in einem System sicherstellen [26].

3.1 Finite-Domain-Constraints

Das allgemeine *Constraint Satisfaction Problem* (CSP) bezeichnet eine Klasse von kombinatorischen Problemen, die mittels einer Menge von Randbedingungen

bzw. Constraints über eine Menge von Variablen formuliert werden. Die Aufgabe besteht darin, eine wohlgeformte Belegung für eine endliche Menge von Variablen zu finden. Wohlgeformt bedeutet in diesem Fall eine konsistente Belegung der Variablen mit Werten, so dass alle Randbedingungen erfüllt werden [7].

Definition 1. (Constraint Satisfaction Problem, CSP)
Ein Constraint Satisfaction Problem wird durch ein Tripel (V, D, C) *beschrieben, wobei* $V = \{v_1, \ldots, v_n\}$ *eine endliche Menge von Variablen mit assoziierten Wertebereichen* $D = \{D_1, \ldots, D_n\}$ *mit* $\{v_1 : D_1, \ldots, v_n : D_n\}$ *ist.* C *ist eine endliche Menge von Constraints* $C_j(V_j)$, $j \in \{1, \ldots, m\}$, *wobei jedes Constraint* $C_j(V_j)$ *eine Teilmenge* $V_j = \{v_{j_1}, \ldots, v_{j_k}\} \subseteq V$ *der Variablen zueinander in Relation setzt und deren gültige Wertekombinationen auf eine Teilmenge von* $D_{j_1} \times \cdots \times D_{j_k}$ *beschränkt.*

Klassischerweise bezieht sich die Literatur bei der Definition von CSPs häufig auf eine strengere Form, in der die Domänen der Variablen aus diskreten, endlichen Mengen (engl. *finite domains*, FD) bestehen (vgl. [28], [17]).

Definition 2. (Finite Constraint Satisfaction Problem, FCSP)
Sei $P = (V, D, C)$ *ein CSP. Wenn die Domäne* D_i *jeder Variablen* $v_i \in V$ *diskret und endlich ist, wird* P *ein Finite Constraint Satisfaction Problem (FCSP) genannt.*

Weil bei einem FCSP die Wertebereiche der Variablen endlich sind, ist auch der Lösungsraum endlich. Die Anzahl der möglichen Lösungen ergibt sich wiederum aus dem kartesischen Produkt aller Wertebereiche $D_1 \times \cdots \times D_n$. Die Kardinalität dieser Menge, und entsprechend auch der Aufwand zur Berechnung dieser möglichen Lösungen, wächst exponentiell mit der Anzahl der Variablen.

3.2 Intervall-Constraints

Neben klassischen CSP über finite Domänen stellt sich für die Constraint-Verarbeitung in wissensbasierten Konfigurierungssystemen das Problem der Behandlung von reellwertigen algebraischen Constraints. Die Wertebereiche der Constraint-Variablen werden hier als Ober- und Untergrenzen von reellwertigen Intervallen definiert, zwischen denen sich unendlich viele, nicht abzählbare Elemente befinden [2]. Diese Wertebereiche werden deshalb auch *kontinuierlich* genannt.

Intervall-Constraints werden z. B. eingesetzt, wenn unscharfe Informationen behandelt werden müssen, d. h. wenn das Wissen über Parameter nur in Form eines Werteintervalls bekannt ist, oder wenn die Aufzählung aller Lösungen nicht möglich ist, da unendlich viele existieren.

Im Gegensatz zu diskreten, endlichen Wertebereichen, für die Konsistenz- und Lösungsalgorithmen die einzelnen Werte in den Domänen der Constraint-Variablen mittels kombinatorischer Methoden aufzählen und auf Zugehörigkeit zu den Constraint-Relationen überprüfen können, lässt sich im Fall von reellwertigen Intervallen nicht für jeden einzelnen Wert bestimmen, ob er als konsistente

Belegung geeignet ist. Während CSPs über endliche Domänen zur Klasse der NP-vollständigen Probleme zählen, sind CSPs über unendliche Domänen unentscheidbar. Daher werden hier durch Konsistenz- und Suchverfahren lediglich die Ober- und Untergrenzen der Intervalle überprüft bzw. angepasst, so dass inkonsistente Werte ausgeschlossen werden (vgl. [4], [6], [14], [16], [3]).

Definition 3. (Intervall Constraint Satisfaction Problem, ICSP)
Ein Intervall Constraint Satisfaction Problem wird durch ein Tripel (V, I, C) *beschrieben. Neben den die Menge der Constraints* C *beschränkenden Variablen* $V = \{v_1, \ldots, v_n\}$ *wird eine Menge von Intervallen* $I = \{I_1, \ldots, I_n\}$ *als Wertebereiche der Variablen mit* $\{v_1 : I_1, \ldots, v_n : I_n\}$ *definiert.* C *ist die endliche Menge von Constraints* $C_j(V_j)$*,* $j \in \{1, \ldots, m\}$*. Jedes Constraint* $C_j(V_j)$ *setzt eine Teilmenge* $V_j = \{v_{j_1}, \ldots, v_{j_k}\} \subseteq V$ *zueinander in Relation, und beschränkt den Lösungsraum der involvierten Variablen auf eine Teilmenge des kartesischen Produkts* $I_{j_1} \times \cdots \times I_{j_k}$ *von* k *Subintervallen.*

Das kartesische Produkt mehrerer Intervalle wird aus geometrischen Gründen auch kurz *Box* genannt. Ziel ist es, eine Menge n-stelliger, möglichst *kanonischer* Boxen zu isolieren, die den Lösungsraum des CSP approximieren, ohne gültige Lösungen zu verlieren. Jede n-stellige Box approximiert jeweils eine mögliche Lösung des CSP. Eine Box wird „kanonisch“ genannt, wenn sie aus Intervallen besteht, deren Grenzen entweder jeweils dieselben oder direkt aufeinander folgende Zahlen sind, d. h. wenn sie möglichst punktgenaue Lösungen darstellen. Um dies zu erreichen wird mittels Such- und Konsistenztechniken sowie mathematischer Verfahren durch den Raum navigiert, der durch das kartesische Produkt $I_1 \times \cdots \times I_n$ aufgespannt wird [2].

4 Das YACS-Framework

Zur Behandlung unterschiedlicher Constraint-Domänen innerhalb eines wissensbasierten Konfigurierungswerkzeugs ist eine Kooperation von mehreren Constraint-Solvern erforderlich. Diese Kooperation muss durch eine Komponente geleistet werden, mit der sich unterschiedliche Constraint-Solver je nach Bedarf und domänenspezifisch einsetzen lassen. Die Modularität des Systems ist dabei entscheidend für die Austauschbarkeit einzelner Komponenten. Die Constraint-Komponente muss im Detail die folgenden Anforderungen erfüllen:

- Das System muss eine modulare Architektur und einheitliche Schnittstellen für unterschiedliche Lösungsverfahren aufweisen.

- Constraint-Lösungsverfahren müssen sich flexibel einbinden und problemabhängig austauschen lassen.

- Das System muss *hybrid* sein, d. h. neben finiten Domänen müssen infinite Domänen in Form von reellwertigen Intervallen unterstützt werden.

– Der inkrementelle Aufbau des Constraint-Netzes innerhalb einer interaktiv durchgeführten, wissensbasierten Konfigurierung muss unterstützt werden.

Der im folgenden Vorgestellte Ansatz ermöglicht die flexible Kooperation und Kombination von Lösungsalgorithmen zu Constraint-Solvern mit neuen oder erweiterten Eigenschaften. Der Name YACS steht dabei für *Yet Another Constraint Solver*. YACS ist allerdings mehr als nur ein einzelner Constraint-Solver. Es bezeichnet vielmehr ein hybrides System für den flexiblen Einsatz von Constraint-Lösungsverfahren für finite und infinite Domänen. Sie sind eingebettet innerhalb einer modularen Framework-Architektur.

4.1 Der Framework-Ansatz

Durch das Aufkommen von objektorientierten Sprachen und objektorientierter Programmierung (OOP) entstanden Ansätze, welche verstärkt die Steigerung der Wiederverwendbarkeit von einmal entwickelten Software-Komponenten zum Ziel hatten [15]. Im Besonderen sind dies objektorientierte Software-Frameworks, in denen ein Rahmenwerk und eine Architekturhilfe für die Bewältigung eines bestimmten Aufgabenspektrums bereitgestellt wird [8]. Objektorientierte Constraint-Frameworks im Speziellen dienen dazu, OOP-Sprachen und Techniken zur Constraint-Verarbeitung zu verbinden und in unterschiedlichen Szenarien nutzbar zu machen. Ein Constraint-Framework ist eine Möglichkeit, Constraints als Inferenz-Mechanismus unabhängig von einer konkreten Domäne nutzbar für unterschiedliche Anwendungen zu machen. Ein Framework bietet hierfür allgemeine Mechanismen, die zur Nutzung durch eine bestimmte Anwendung an die jeweils spezielle Problemstellung angepasst werden können [21].

Durch ein Constraint-Framework wird ein allgemeiner Kontrollzyklus vorgegeben, in den unterschiedliche Lösungsverfahren je nach Bedarf eingebunden werden können. Allgemeine Verfahren zur Constraint-Verarbeitung sind innerhalb eines Frameworks in einer erweiterbaren Bibliothek bereits enthalten. Die Architektur eines Frameworks sollte dabei eine einfache Nutzung garantieren, und in diesem Fall die komplexen Mechanismen des CSP-Formalismus vor dem Benutzer weitestgehend verbergen [21].

4.2 Constraint-Lösungsstrategien

Flexibilität hinsichtlich der einzusetzenden Lösungsverfahren kann durch ein Konzept von modularen und austauschbaren Constraint-Lösungsstrategien erreicht werden. Zur Strukturierung des Constraint-Lösungsvorgangs wird dieser Prozess in drei Phasen eingeteilt: *Preprozessing*, *Konsistenzherstellung* und *Lösungssuche*. Diese Phasen spiegeln sich innerhalb von Constraint-Lösungsstrategien wieder (vgl. Abbildung 1). In der ersten Phase wird ein Preprozessing des Constraint-Problems vorgenommen. Dies kann sich z. B. auf die Binärisierung eines Constraint-Netzes oder die Zerlegung von Constraints in primitive Constraints beziehen, um anschließend darauf aufbauende Lösungsalgorithmen

anwenden zu können. In der zweiten Phase werden Filter- bzw. Konsistenzalgorithmen zur Einschränkung der Domänen der Constraint-Variablen angewendet. Da dies allein i. A. nicht zu einer Lösung des Constraint-Problems führt, können in einer dritten Phase Suchverfahren zum Auffinden von Lösungen in den reduzierten Wertebereichen eingesetzt werden.

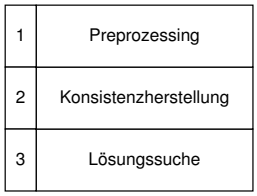

Abbildung 1. Aufbau einer Constraint-Lösungsstrategie

Zu beachten ist, dass in jeder Phase mehrere Einträge innerhalb einer Lösungsstrategie existieren können. So ist es z. B. möglich, mehrere Preprozessing-Schritte auf ein Problem anzuwenden, bevor Verfahren aus der nächsten Phase zum Einsatz kommen. Dies gilt ebenso für Konsistenz- und Suchverfahren.

Ebenso ist es möglich, dass für eine Phase innerhalb einer Strategie keine Einträge existieren. Nicht für alle Konsistenz- und Suchverfahren ist ein Preprozessing erforderlich. Gleichfalls kann ein Suchverfahren auch ohne vorherigen Filteralgorithmus angewendet werden, insbesondere wenn das Suchverfahren bereits Filtermechanismen enthält. Sind für eine Anwendung keine exakten Lösungen sondern nur eingeschränkte Wertebereiche erforderlich, kann auf ein Suchverfahren in der dritten Phase verzichtet werden. Mehrere Beispiele für mögliche Constraint-Lösungsstrategien sind in Abbildung 2 zu sehen.

Abbildung 2. Beispiele für Constraint-Lösungsstrategien

Die Verwaltung derartiger Constraint-Lösungsstrategien muss von einer Komponente vorgenommen werden, die in der Lage ist, aufgrund einer klaren Spezifikation die Zuordnung der jeweiligen Strategien zu einzelnen Teilproblemen des gesamten Constraint-Problems vornehmen zu können.

4.3 Ein hybrides Constraint-System

Ein hybrides Constraint-System zeichnet sich dadurch aus, dass es in der Lage ist, ein hybrides Constraint Satisfaction Problem zu verarbeiten:

Definition 4. (Hybrides Constraint Satisfaction Problem)
Ein System zur Verarbeitung eines hybriden Constraint Satisfaction Problems H wird durch die Angabe von sieben Komponenten

$$H = (C, S, \delta, V_{fd}, D_{fd}, V_{int}, D_{int})$$

beschrieben. Dabei ist $C = \{C_1, \ldots, C_m\}$ *eine endliche Menge von Constraints und* $S = \{S_1, \ldots, S_n\}$ *eine endliche Menge von Constraint-Lösungsstrategien. Die Funktion* δ *ordnet jedem Constraint* C_i, $i \in \{1, \ldots, m\}$, *eine eindeutige Strategie* S_j, $j \in \{1, \ldots, n\}$, *zu:*

$$\delta : C_i \rightarrow S_j.$$

Der Bezeichner V_{fd} *steht für eine endliche Menge von FD-Variablen* $\{v_1, \ldots, v_k\}$, *mit denen die Wertebereiche* $D_{fd} = \{D_1, \ldots, D_k\}$ *mit* $\{v_1 : D_1, \ldots, v_k : D_k\}$ *assoziiert sind. Ebenso sind die Intervallvariablen* $V_{int} = \{v_1, \ldots, v_l\}$ *mit den Wertebereichen* $D_{int} = \{D_1, \ldots, D_l\}$ *mit* $\{v_1 : D_1, \ldots, v_l : D_l\}$ *assoziiert. Jedes Constraint* C_i *setzt eine Teilmenge der Variablen aus* V_{fd} *und* V_{int} *zueinander in Relation und beschränkt deren gültige Wertekombinationen auf eine Teilmenge des kartesischen Produkts ihrer Wertebereiche.*

Ein hybrides CSP vereinigt somit Constraints über Variablen mit finiten und infiniten Domänen. Jedem Constraint ist eine Lösungsstrategie zu dessen Verarbeitung zugeordnet. Dies führt zu einer Aufteilung des ursprünglichen Constraint-Problems in unterschiedliche Teilprobleme, welche durch die jeweils zuständige Constraint-Lösungsstrategie definiert werden (vgl. Abbildung 3).[2]

Ausführungsmodell Der „YACS Constraint-Manager“ (YCM) ist die Komponente an der Schnittstelle zwischen einer vorhandenen Anwendung und dem YACS-Framework. Dem YCM obligt die Verwaltung der neu hinzukommenden unterschiedlichen Constraint-Netze, der Constraint-Lösungsstrategien, der Constraint-Solver und letztendlich der Steuerung des Lösungsprozesses.

Von dem aufrufenden System erhält der Constraint-Manager YCM die Informationen, welche Constraints mit welcher Strategie aufzulösen sind. Der Constraint-Manager aktiviert die entsprechenden Constraint-Lösungskomponenten und übergibt in der jeweiligen Bearbeitungsphase das entsprechende Constraint-Netz zur Verarbeitung an die dafür vorgesehene Komponente.

Der Prozess des Constraint-Lösens ist analog zum Aufbau der Constraint-Lösungsstrategien in drei Phasen unterteilt. In jeder Phase werden sequentiell jeweils die Constraint-Netze aller Strategien der Reihe nach bearbeitet. Das

[2] Teilprobleme entstehen, indem mehrere Constraints derselben Strategie zugeordnet werden.

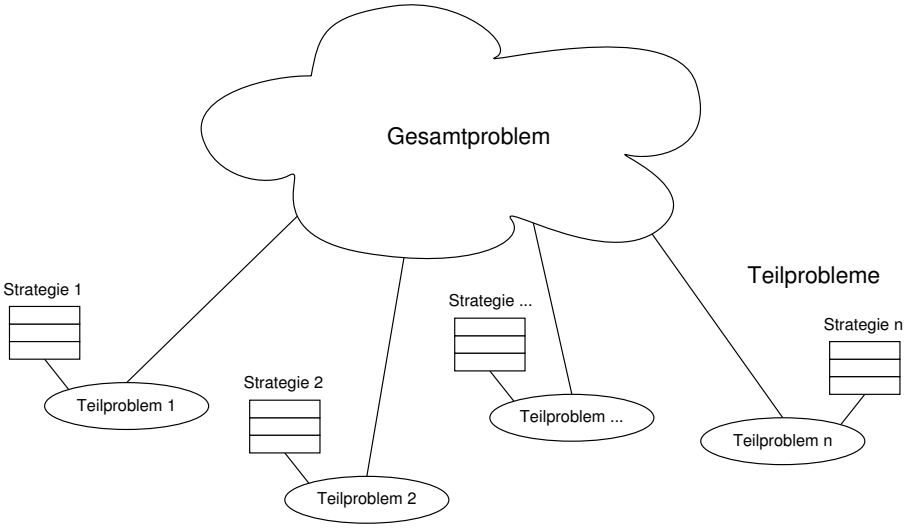

Abbildung 3. Zuständigkeiten unterschiedlicher Strategien für Teilbereiche des Constraint-Problems

heißt die in den Strategien angegebenen Constraint-Verfahren werden in den entsprechenden Phasen auf die zugehörigen Constraint-Netze angewendet.

- *Phase 1 (Preprozessing):* Das jeweilige Constraint-Netz wird wenn möglich vollständig konvertiert oder es wird festgestellt, dass es sich mit den gegebenen Algorithmen nicht umformen lässt.

- *Phase 2 (Konsistenzherstellung):* Es wird solange propagiert, bis keine Änderungen der Wertebereiche mehr eintreten (d. h. Konsistenz hergestellt ist) oder eine Inkonsistenz auftritt.

- *Phase 3 (Lösungssuche):* Die Suchalgorithmen finden die geforderten Lösungen oder stellen fest, dass keine Lösung existiert.

Maximale Flexibilität wird erreicht, wenn in der Wissensbasis für jedes Constraint der Name einer entsprechenden Strategie zu dessen Lösung angegeben werden kann. Der Name der jeweiligen Strategie entspricht dabei einem eigenen Teilbereich des ursprünglichen Constraint-Problems. Jede Strategie ist für die Verarbeitung eines (Sub-)Constraint-Netzes vorgesehen. Bei der Initialisierung wird durch den Constraint-Manager von YACS sichergestellt, dass alle geforderten Strategien existieren und angewendet werden können.

Die Strategien werden separat von der Wissensbasis des Konfigurators definiert. Die Anwendung übergibt die Constraints jeweils mit dem Namen der zugehörigen Strategie an den YACS Constraint-Manager. Dieser generiert daraus die unterschiedlichen Constraint-Netze und wendet die entsprechenden Constraint-Lösungsstrategien an.

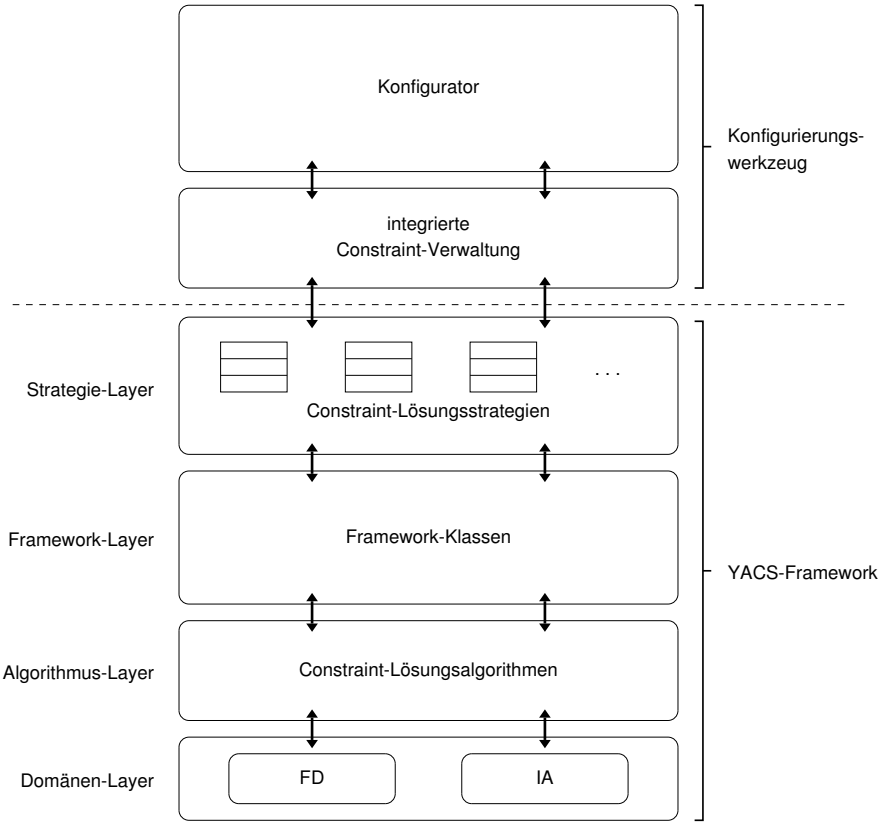

Abbildung 4. Systemarchitektur von YACS

4.4 Architektur des YACS-Frameworks

Das YACS-Framework stellt eine modulare und wiederverwendbare Constraint-Lösungskomponente dar. YACS ist ein hybrides System für den flexiblen Einsatz von Constraint-Lösungsverfahren für finite und infinite Domänen. Die Lösungsverfahren sind eingebettet innerhalb einer strategiebasierten, modularen Framework-Architektur:

- *Constraint-Lösungsstrategien:* Der flexible Einsatz von Constraint-Lösungsverfahren wird über ein Strategiekonzept realisiert. Abstrahiert von den eigentlichen Lösungsalgorithmen können von dem Wissensingenieur problemabhängig bzw. anwendungsspezifisch unterschiedliche Constraint-Lösungsstrategien eingesetzt werden. Diese Lösungsstrategien müssen vorab in Abhängigkeit von den vorhandenen Lösungsverfahren definiert werden.

- *Framework-Architektur:* Durch die Framework-Architektur wird sichergestellt, dass Lösungsverfahren flexibel ausgetauscht und zudem auf einfache

Weise neue Lösungsalgorithmen implementiert bzw. Fremdsysteme integriert werden können. Das YACS-Framework stellt hierfür einen geeigneten Rahmen mit einheitlichen Schnittstellen bereit.

In Abbildung 4 ist eine Übersicht über die Systemarchitektur von YACS, angebunden an ein wissensbasiertes Konfigurierungswerkzeug, zu sehen. Aufsetzend auf einem *Domänen-Layer*, einer Umgebung zur arithmetischen Verarbeitung von finiten Domänen (FD) und reellwertigen Intervallen (eine Intervallarithmetik, kurz IA), werden die eigentlichen Algorithmen zum Auflösen von Constraint-Problemen implementiert (*Algorithmus-Layer*). Constraint-Verfahren aus Fremdsystemen können an dieser Stelle über Wrapper-Klassen eingebunden werden. Die Algorithmen bzw. die sie umschließenden (Wrapper)-Klassen müssen wiederum den Schnittstellen des *Framework-Layers* von YACS genügen.

Der durch das Framework vereinheitlichte Zugriff auf Constraint-Lösungsverfahren ermöglicht es dem *Strategie-Layer*, dem Anwender bzw. dem übergeordnetem System eine flexible Auswahl an Lösungsverfahren anbieten zu können. Abstrahiert von den Lösungsverfahren können auf dieser Ebene aus einer Reihe vordefinierter Constraint-Lösungsstrategien problemabhängig die für die jeweilige Anwendung geeigneten Strategien ausgewählt werden.

YACS wurde in der Programmiersprache Java implementiert und prototypisch in das Konfigurierungswerkzeug EngCon integriert. Die benötigten Constraint-Lösungsstrategien werden separat mittels XML definiert, eingelesen und angewendet. Die Intervallarithmetik wurde mit Hilfe der Bibliothek *IAMath*[3] realisiert. Der Prototyp von YACS ist im Internet verfügbar.[4]

5 Verwandte Arbeiten

Bekannte Constraint-Bibliotheken mit integrierten Constraint-Solvern sind z. B. die *C-Lib*[5], die *Java Constraint Library* (JCL)[6] [27] und der *IASolver*[7] [12]. Diese Systeme sind allerdings nicht hybrid, d. h. es lassen sich ausschließlich entweder finite oder infinite Domänen verarbeiten. Zudem ist eine inkrementelle Constraint-Verarbeitung nicht vorgesehen und die Erweiterbarkeit ist aufgrund der fehlenden Framework-Architektur nur eingeschränkt möglich.

Constraint-Frameworks im eigentlichen Sinn sind z. B. die Systeme *BackTalk* [20] und *POOC* [24]. Beide sind allerdings ebenfalls auf finite Domänen beschränkt. *POOC* fokussiert zudem den Bereich *Constraint Programming* (CP), d. h. es werden in erster Linie die für die Programmierung mit Constraints benötigten *global constraints* angeboten.

In [13] wird ein allgemeines und formales Schema für die Kombination von Constraint-Systemen und die Kooperation von Constraint-Solvern zur Behand-

[3] `http://interval.sourceforge.net/interval/java/ia_math/`
[4] `http://sourceforge.net/projects/constraints`
[5] `http://ai.uwaterloo.ca/~vanbeek/software/software.html`
[6] `http://liawww.epfl.ch/JCL/`
[7] `http://www.cs.brandeis.edu/~tim/Applets/IAsolver.html`

lung von Constraints mit vermischten Wertedomänen vorgestellt. Strategien werden hier in Form von frei modellierbaren *Kooperationsstrategien* eingesetzt. Mit diesen kann beschrieben werden, welche Constraint-Verfahren wie und in welcher Reihenfolge (sequentiell/parallel) kombiniert werden.

In [18] wird eine Sprache namens *BALI* zur Steuerung von Kooperationen entwickelt. *BALI* erlaubt es, die Kooperation unterschiedlicher Constraint-Solver auf hoher Ebene durch eine eigene Sprache zu beschreiben, und auf diese Weise effizient neue Prototypen von kooperierenden Solvern zu erstellen. Neben unterschiedlichen Kooperationsprimitiven, welche die sequentielle, die unabhängig parallele und die nebenläufige Ausführung von Constraint-Lösungsmechanismen erlauben, bietet *BALI* mehrere Lösungsstrategien an. Darunter befindet auch sich eine inkrementelle Variante [18].

6 Zusammenfassung und Ausblick

Aufgrund einer Vielzahl von Constraint-Lösungsverfahren und möglicher Kombinationen derselben, deren unterschiedlichen Eigenschaften und der problemabhängigen bzw. anwendungsspezifischen Effizienz unterschiedlicher Verfahren, ist zur Unterstützung der wissensbasierten Konfigurierung eine Komponente notwendig, mit der sich flexibel, je nach Problemstellung unterschiedliche Constraint-Lösungsmechanismen einsetzen lassen.

Das Framework-Konzept von YACS bietet eine flexible und nutzerfreundliche Architektur zur Implementierung von Constraint-Lösungsverfahren. Die Framework-Architektur wiederum wird in Bezug auf die Abstraktion von den tatsächlich eingesetzten Verfahren durch ein Strategiekonzept ergänzt.

In den Constraint-Lösungsstrategien werden die tatsächlich einzusetzenden Lösungsverfahren definiert. Dies geschieht problemabhängig und flexibel je nach Anwendung und Einsatzzweck. Der Wissensingenieur kann sich somit bei der Erstellung der Wissensbasis auf vordefinierte und dokumentierte Constraint-Lösungsstrategien stützen, und diese zur Behandlung der im Rahmen der Konfigurierung entstehenden Constraint-Probleme gezielt einsetzen. Sollten Anpassungen notwendig werden, so ist eine einfache Wartung, Pflege und auch Neuimplementierung oder -anbindung von Constraint-Lösungsverfahren durch eine modulare Struktur und die Framework-Architektur gewährleistet.

Die Möglichkeit, auf unkomplizierte und durchschaubare Art und Weise Anpassungen an YACS bzw. an den Constraint-Lösungsstrategien von YACS vornehmen zu können, kann sich positiv auf die Akzeptanz der Lösung bei den zuständigen Wissensingenieuren bei gleichzeitig höchstmöglicher Flexibilität auswirken. Das YACS-Framework wurde zudem für den Anwendungsfall der strukturbasierten Konfigurierung entwickelt und getestet. YACS unterstützt daher die domänenspezifischen Eigenheiten wie ein inkrementell anwachsendes Constraint-Netz und sowohl finite als auch infinite Wertebereiche.

Zur Erhöhung der Flexibilität ist die Anwendung strategiebasierter Ansätze ein gängiges Mittel zur Steuerung der Propagation und Lösungssuche von kooperierenden Constraint-Solvern. Das Konzept für YACS wurde aus der Idee heraus

geboren, größtmögliche Flexibilität hinsichtlich der einzusetzenden Constraint-Lösungsmechanismen zu bieten. Weniger im Vordergrund steht der Aspekt der flexiblen *Steuerung* von Kooperationen unterschiedlicher Lösungsverfahren. Weitere Arbeiten werden sich daher damit befassen, das Ausführungsmodell flexibler zu gestalten, so dass z. B. neben einer sequentiellen Abfolge von Constraint-Lösungsalgorithmen eine parallele Verarbeitung gewährleistet werden kann.

Ein weiteres Betätigungsfeld bietet die Untersuchung der „Überlappung" von Teilproblemen unterschiedlicher Constraint-Lösungsstrategien. Eine Überlappung entsteht, wenn eine Variable in strukturell unterschiedlichen Teilproblemen existiert, d. h. in Constraints, die unterschiedlichen Lösungsstrategien zugeordnet sind. Werden bei Überlappungen infinite Variablen mit intervallwertigen Domänen von diskreten Werten finiter Variablen beschränkt und umgekehrt, besteht die Aufgabe darin, ein *heterogenes Constraint-Problem* zu verarbeiten (vgl. [1], [9]). In jedem Fall ist bei Überlappungen von Teilproblemen ein *Meta-Constraint-Solver* erforderlich, wenn globale Lösungen generiert werden sollen.

Eine ausführliche Fassung dieser Ausarbeitung ist in [22] zu finden.

Literatur

1. Benhamou, Frédéric: Heterogeneous Constraint Solving. In: Hanus, Michael (Hrsg.) ; Rodríguez-Artalejo, Mario (Hrsg.): *Proceedings of the 5th International Conference on Algebraic and Logic Programming (ALP'96), Aachen, 25.–27. September 1996.* Berlin, Heidelberg, New York : Springer Verlag, 1996 (LNCS 1139), S. 62–76. – URL `http://www.sciences.univ-nantes.fr/info/perso/permanents/benhamou/PAPERS/Ben_ALP96.pdf`. – ISBN 3-540-61735-3
2. Benhamou, Frédéric: Interval Constraints. In: Floudas, Christodoulos A. (Hrsg.) ; Pardalos, Panos M. (Hrsg.): *Encyclopedia of Optimization* Bd. 3. Dordrecht, Netherlands : Kluwer Academic Publishers, August 2001, S. 45–48. – URL `http://www.sciences.univ-nantes.fr/info/perso/permanents/benhamou/PAPERS/Ben99_EoO.pdf`. – ISBN 0-7923-6932-7
3. Benhamou, Frédéric ; McAllester, David ; Van Hentenryck, Pascal: CLP(Intervals) Revisited. In: Bruynooghe, Maurice (Hrsg.): *Logic Programming, Proceedings of the 1994 International Symposium (ILPS'94), Ithaca, New York, USA, 13.–17. November 1994.* Cambridge, Massachusetts, USA : The MIT Press, 1994, S. 124–138. – URL `http://www.sciences.univ-nantes.fr/info/perso/permanents/benhamou/PAPERS/BenMcAlVHen94.pdf`. – ISBN 0-262-52191-1
4. Cleary, John G.: Logical Arithmetic. In: *Future Computing Systems* 2 (1987), Nr. 2, S. 125–149. – ISSN 0266-7207
5. Cunis, Roman (Hrsg.) ; Günter, Andreas (Hrsg.) ; Strecker, Helmut (Hrsg.): *Das PLAKON-Buch, Ein Expertensystemkern für Planungs- und Konfigurierungsaufgaben in technischen Domänen.* Berlin, Heidelberg, New York : Springer Verlag, 1991 (Informatik-Fachberichte, Subreihe Künstliche Intelligenz 266). – 279 S. – ISBN 3-540-53683-3
6. Davis, Ernest: Constraint Propagation with Interval Labels. In: *Artificial Intelligence* 32 (1987), Juli, Nr. 3, S. 281–331. – ISSN 0004-3702
7. Dechter, Rina: *Constraint Processing.* San Francisco, California, USA : Morgan Kaufmann Publishers, 2003 (The Morgan Kaufmann Series in Artificial Intelligence) – xx + 481 S. – ISBN 1-558-60890-7

8. Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software.* 1. Aufl. München : Addison-Wesley, 1996. – xx + 479 S. – ISBN 3-89319-950-0
9. Gelle, Esther ; Faltings, Boi V.: Solving Mixed and Conditional Constraint Satisfaction Problems. In: *Constraints, An International Journal* 8 (2003), April, Nr. 2, S. 107–141. – URL `http://liawww.epfl.ch/Publications/Archive/Gelle2003.pdf`. – ISSN 1383-7133
10. Günter, Andreas: KONWERK – ein modulares Konfigurierungswerkzeug. In: Richter, Michael M. (Hrsg.) ; Maurer, Frank (Hrsg.): *Expertensysteme 95, Beiträge zur 3. Deutschen Expertensystemtagung (XPS'95), Kaiserslautern, 1.–3. März 1995.* Sankt Augustin : Infix Verlag, 1995, S. 1–18. – URL `http://www.hitec-hh.de/ueberuns/home/aguenter/literatur/konwerk.pdf`. – ISBN 3-932-79295-5
11. Günter, Andreas ; Kühn, Christian: Knowledge-Based Configuration – Survey and Future Directions. In: Puppe, Frank (Hrsg.): *Knowledge-Based Systems – Survey and Future Directions, Proceedings of the 5th Biannual German Conference on Knowledge-Based Systems (XPS 1999), Würzburg, 3.–5. März 1999.* Berlin, Heidelberg, New York : Springer Verlag, 1999 (LNCS 1570), S. 47–66. – URL `http://www.hitec-hh.de/ueberuns/home/aguenter/literatur/xps-99.pdf`. – ISBN 3-540-65658-8S
12. Hickey, Timothy J. ; Qiu, Zhe ; Emden, Maarten H. van: Interval Constraint Plotting for Interactive Visual Exploration of Implicitly Defined Relations. In: *Reliable Computing* 6 (2000), Februar, Nr. 1, S. 81–92. – URL `http://csr.uvic.ca/~vanemden/Publications/graphics.pdf`. – ISSN 1385-3139
13. Hofstedt, Petra: Cooperating Constraint Solvers. In: Dechter, Rina (Hrsg.): *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP 2000), Singapore, 18.–21. September 2000.* Berlin, Heidelberg, New York : Springer Verlag, 2000. (LNCS 1894), S. 520–524. – URL `http://uebb.cs.tu-berlin.de/~ph/ph.papers/cp2000.pdf` – ISBN 3-540-41053-8
14. Hyvönen, Eero: Constraint Reasoning Based on Interval Arithmetic: The Tolerance Propagation Approach. In: *Artificial Intelligence* 58 (1992), Dezember, Nr. 1–3, S. 71–112. – Special Volume on Constraint Based Reasoning. – ISSN 0004-3702
15. Johnson, Ralph E.: Components, Frameworks, Patterns. In: Harandi, Medhi (Hrsg.): *Proceedings of the 1997 Symposium on Software Reusability (SSR'97), Boston, Massachusetts, USA, 17.–20. Mai 1997.* New York, NY, USA : ACM Press, 1997, S. 10–17. – URL `ftp://st.cs.uiuc.edu/pub/papers/frameworks/framework97.ps`. – ISBN 0-89791-945-9 – Zugl.: ACM SIGSOFT Software Engineering Notes, 22 (1997), Nr. 3, S. 10–17. – ISSN 0163-5948
16. Lhomme, Olivier: Consistency Techniques for Numeric CSPs. In: Bajcsy, Ruzena (Hrsg.): *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93), Chambéry, France, 28. August – 3. September 1993.* San Mateo, California, USA : Morgan Kaufmann Publishers, 1993, S. 232–238. – ISBN 1-55860-300-X
17. Mackworth, Alan K.: Consistency in Networks of Relations. In: *Artificial Intelligence* 8 (1977), Februar, Nr. 1, S. 99–118. – ISSN 0004-3702
18. Monfroy, Eric: The Constraint Solver Collaboration Language of BALI. In: Gabbay, Dov (Hrsg.) ; Rijke, Maarten de (Hrsg.): *Proceedings of the 2nd International Workshop Frontiers of Combining Systems (FroCoS'98), Amsterdam, The Netherlands, 2.–4. Oktober 1998.* Baldock, Hertfordshire, UK : Research Studies Press, 2000 (Studies in Logic and Computation Vol. 7), S. 211–230. – URL `http://www.sciences.univ-nantes.fr/info/perso/permanents/monfroy/Papers/frocos98.ps.gz`

19. Ranze, Christoph ; Scholz, Thorsten ; Wagner, Thomas ; Günter, Andreas ; Herzog, Otthein ; Hollmann, Oliver ; Schlieder, Christoph ; Arlt, Volker: A Structure-Based Configuration Tool: Drive Solution Designer – DSD. In: Dechter, Rina (Hrsg.) ; Sutton, Rich (Hrsg.) ; Kearns, Michael (Hrsg.) ; Chien, Steve (Hrsg.) ; Riedl, John (Hrsg.): *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'02) and 14th Conference on Innovative Applications of Artificial Intelligence (IAAI'02), Edmonton, Alberta, Canada, 28. Juli – 1. August 2002.* Menlo Park, California, USA/Cambridge, Massachusetts, USA : AAAI Press/The MIT Press, September 2002, S. 845–852. – URL `http://www.hitec-hh.de/ueberuns/home/aguenter/literatur/IAAI2002.pdf`
20. Roy, Pierre ; Liret, Anne ; Pachet, François: Constraint Satisfaction Problems Framework. In: Fayad, Mohamed E. (Hrsg.) ; Schmidt, Douglas C. (Hrsg.) ; Johnson, Ralph E. (Hrsg.): *Implementing Application Frameworks: Object-Oriented Frameworks at Work.* Chichester, London, New York : John Wiley & Sons, September 1999 (Wiley Computer Publishing), Kap. 17, S. 369–401. – ISBN 0-471-25201-8
21. Roy, Pierre ; Liret, Anne ; Pachet, François: The Framework Approach for Constraint Satisfaction. In: *ACM Computing Surveys (CSUR)* 32 (2000), März, Nr. 1es. – URL `http://doi.acm.org/10.1145/351936.351949`. – CSUR Electronic Symposium on Object-Oriented Application Frameworks – Artikel Nr. 13. – ISSN 0360-0300
22. Runte, Wolfgang: *YACS: Ein hybrides Framework für Constraint-Solver zur Unterstützung wissensbasierter Konfigurierung*, Universität Bremen, Diplomarbeit, 27. Januar 2006. – xx + 405 S. – URL `http://www.informatik.uni-bremen.de/~woru/pub/diplom/runte06diplom.pdf`
23. Sabin, Daniel ; Weigel, Rainer: Product Configuration Frameworks – A Survey. In: *IEEE Intelligent Systems* 13 (1998), Juli/August, Nr. 4, S. 42–49. – ISSN 1094-7167l
24. Schlenker, Hans ; Ringwelski, Georg: POOC: A Platform for Object-Oriented Constraint Programming. In: O'Sullivan, Barry (Hrsg.): *Recent Advances in Constraints, Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming (CSCLP'02), Cork, Ireland, 19.–21. Juni 2002, Selected Papers.* Berlin, Heidelberg, New York : Springer Verlag, 2003 (LNCS 2627), S. 159–170. – URL `http://4c.ucc.ie/web/upload/publications/inProc/ercim02pooc.pdf`. – ISBN 3-540-00986-8
25. Stumptner, Markus: An Overview of Knowledge-Based Configuration. In: *AI Communications (AICOM)* 10 (1997), Juli, Nr. 2, S. 111–125. – ISSN 0921-7126
26. Syska, Ingo ; Cunis, Roman: Constraints in PLAKON. In: Cunis, Roman (Hrsg.) ; Günter, Andreas (Hrsg.) ; Strecker, Helmut (Hrsg.): *Das PLAKON-Buch, Ein Expertensystemkern für Planungs- und Konfigurierungsaufgaben in technischen Domänen.* Berlin, Heidelberg, New York : Springer Verlag, 1991 (Informatik-Fachberichte, Subreihe Künstliche Intelligenz 266), Kap. 6, S. 77–91. – ISBN 3-540-53683-3
27. Torrens, Marc ; Weigel, Rainer ; Faltings, Boi V.: Distributing Problem Solving on the Web Using Constraint Technology. In: *Proceedings of the 10th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'98), Taipei, Taiwan, 10.–12. November 1998.* Los Alamitos, California, USA, : IEEE Computer Society Press, 1998, S. 42–49. – URL `http://liawww.epfl.ch/Publications/Archive/Torrens1998.pdf`. – ISBN 0-7803-5214-9
28. Waltz, David L.: Understanding Line Drawings of Scenes with Shadows. In: Winston, Patric Henry (Hrsg.): *The Psychology of Computer Vision.* New York, NY, USA : McGraw-Hill, 1975, S. 19–91. – ISBN 0-07-071048-1