Fachbereich Mathematik / Informatik
Studiengang Informatik

Vorschlag für eine Diplomarbeit

YACS: Ein hybrides Framework für Constraint-Solver
zur Unterstützung wissensbasierter Konfigurierung
in technischen Domänen

Wolfgang Runte
Matrikelnr.: 1140989

Stand: 13. Juni 2005

Inhaltsverzeichnis

1 Motivation und Problemstellung

Das wissensbasierte Konfigurierungswerkzeug EngCon verwendet u. a. Funktions- und Prädikat-Constraints[1] zur Beschreibung von Abhängigkeiten zwischen Konzepten der Wissensbasis (vgl. Hollmann et al. 2000; Ranze et al. 2002). Die Auflösung der Abhängigkeiten wird von einem Constraint-Solver geleistet (vgl. Syska und Cunis 1991). Dieser Constraint-Solver wird nicht von EngCon implementiert, sondern ist derzeit eine von einem Fremdhersteller eingebundene, externe Komponente.[2] Er ist ausschließlich für die Propagation und Auflösung von Constraints mit reellwertigen Intervalldomänen geeignet. Zudem ist die Anbindung an EngCon auf einen einzigen Constraint-Solver beschränkt und damit sehr unflexibel. Zur Erweiterung der Anwendungsmöglichkeiten des Konfigurierungswerkzeugs EngCon sowie zur Verbesserung der Effizienz ist ein Austausch oder eine Eigenimplementierung des Constraint-Solvers wünschenswert.

Der benötigte Constraint-Solver muss arithmetische Funktionen zur Berechnung der intensional in Form von algebraischen Ausdrücken formulierten Constraints innerhalb von EngCon bieten. Neben klassischen Constraint-Solvern zur Behandlung von finiten Domänen sind für die Constraint-Verarbeitung in EngCon Constraint-Lösungsmethoden für infinite, d. h. reellwertige Intervalldomänen erforderlich. Ein entsprechender Constraint-Solver muss einen hohen Präzisionsgrad durch Intervallarithmetik aufweisen (z. B. für Anwendungen im Maschinenbau) sowie unabhängig von der Constraint-Domäne ein inkrementell anwachsendes Constraint-Netz propagieren können.

Aufgrund der hohen Anforderungen von EngCon, insbesondere in Bezug auf unterschiedliche zu verarbeitende Wertedomänen, existiert kein Constraint-Solver, der sämtliche Anforderungen vollständig erfüllt. Darüber hinaus ergeben sich Anforderungen an den Constraint-Lösungsmechanismus häufig in Abhängigkeit von der Aufgabenstellung des jeweiligen Konfigurierungsproblems in EngCon. Spezielle benötigte Eigenschaften sind i. A. daher a priori nicht bekannt. 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 neben zusätzlichen Constraint-Lösungsmechanismen 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. Diese Constraint-Solver können eigenimplementierte aber auch Fremdsysteme sein. Eigene Implementierungen hätten den Vorteil der leichteren Erweiterbarkeit,[3] zudem entfällt

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

[2] Die Vorgänger von EngCon, die Konfigurierungswerkzeuge KonWerk und PlaKon, wurden in Common Lisp bzw. CLOS (*C*ommon Lisp Object System) implementiert. Hier fand hier ein eigens ins Lisp implementiertes Modul zur Durchführung von arithmetischen Berechnungen Verwendung, welches die von Davis (1987) und Hyvönen (1992) beschriebenen Verfahren zur Propagation von Intervall-Constraints anwendet (vgl. Gülden 1993).

[3] Denkbare Erweiterungen, die allerdings nicht in dieser Arbeit behandelt werden, wäre z. B. die Möglichkeit während der Laufzeit des Systems einzelne Constraints zurücknehmen zu können („Constraint-Relaxierung“) oder die Möglichkeit zur Beschreibung von „Constraint-Hierarchien“, in denen „harte“ und „weiche“ Constraints definiert werden können. Je nachdem auf welcher Stufe innerhalb der Hierarchie sich ein Constraint befindet, muss oder kann es erfüllt sein, um zu einer gültigen Lösung zu gelangen.

der bei geeigneten Constraint-Solvern von Fremdherstellern ggf. hohe Integrationsaufwand der Komponenten.

2 Anforderungen an die Problemlösung

In vielen Anwendungsbereichen lassen sich Problemstellungen als Constraint-Problem formulieren. Aufgrund einer Vielzahl unterschiedlicher Domänen und der Vielfältigkeit der Anwendungsszenarien ist es notwendig, an die jeweilige Domäne angepasste Constraint-Lösungsverfahren einzusetzen. Zur Behandlung unterschiedlicher Constraint-Domänen innerhalb des Constraint-Systems von EngCon ist eine Kooperation mehrerer Constraint-Solver 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 Entwurfs ist dabei entscheidend für die Austauschbarkeit einzelner Komponenten.

Im Detail stellen sich die folgenden Anforderungen an die in EngCon zu integrierende Constraint-Komponente:

Schnittstelle: Das Constraint-System muss unabhängig vom restlichen System funktionieren (*Black-Box*-Prinzip). Von außen darf lediglich beeinflusst werden, welche Constraint-Propagationsmethoden zum Einsatz kommen, nicht jedoch inhaltliche Details der Propagation. Die neue Komponente muss sich dazu in den vorhandenen Constraint-Mechanismus von EngCon einfügen (konzeptionelle Constraints, Pattern-Matcher). Die bereits existierenden Constraint-Propagationsmechanismen von EngCon bleiben unangetastet (Tupel-Constraints, Java-Constraints). Die Anbindung der Constraint-Solver muss über eine stringbasierte Schnittstelle erfolgen.

Dynamik: Die Constraint-Solver müssen in der Lage sein, ein inkrementell anwachsendes Constraint-Netz zu verarbeiten. Aufgrund der interaktiven Konfigurierung mit EngCon und der Möglichkeit von Konfigurierungskonflikten müssen sich zudem Zustände des Constraint-Netzes, die in der Vergangenheit liegen, wiederherstellen lassen.

Wertedomänen: Während einer Konfigurierung in EngCon müssen sowohl Elemente aus diskreten Werte*mengen* ausgewählt, als auch kontinuierliche Werte*bereiche* eingeschränkt werden können. Ein Constraint-System, welches ebensolche Konfigurierungsschritte unterstützt, muss infolgedessen Propagations- und Lösungsmechanismen sowohl für finite als auch infinite Domänen aufweisen.

Lösungsverfahren: Die Constraint-Lösungsverfahren müssen in der Lage sein, beliebige algebraische Constraint-Ausdrücke in Form von Gleichungen und Ungleichungen zu verarbeiten. Es sollte weiterhin möglich sein, n-stellige Constraints (bezogen auf die Anzahl der Variablen), nichtlineare Constraints und zyklische Strukturen des Constraint-Netzes zu behandeln.

Präzision und Effizienz: Der Erfordernis an die Präzision von Lösungen steht oftmals der Wunsch nach Effizienz bei den zum Einsatz kommenden Lösungsverfahren und

die Komplexität der Problemstellung entgegen. Das Constraint-System sollte demnach einen Kompromiss zwischen Effizienz und Qualität bzgl. der zu berechnenden Lösungen erlauben. Je nach Anwendungsdomäne, den vorhandenen Constraints, der bevorzugten Präzision und der benötigten Effizienz kann es sinnvoll sein, unterschiedliche Verfahren zur Auswertung der Constraints einzusetzen. Bei Einschränkungen bzgl. der Präzision von Lösungsverfahren ist zu beachten, dass keine möglichen Lösungen verloren gehen dürfen, da sonst die Vollständigkeit des Verfahrens nicht gewährleistet ist.

Hard- und Software-Umgebung: Die Anbindung bzw. Implementierung muss unter den von EngCon geforderten Bedingungen erfolgen, d. h. auf einer Microsoft-Windows-NT-Plattform auf x86-Hardware.[4] Vorzugsweise sollte eine Implementierung, ebenso wie die von EngCon, in der Programmiersprache Java erfolgen. Dies garantiert zudem weitestgehende Plattformunabhängigkeit der Constraint-Komponente und würde die Wiederverwendbarkeit auch für andere Projekte erhöhen.

Für eine Eigenentwicklung ist eine weitere Anforderung an den Entwurf die Anbindung eines ebenfalls austauschbaren Werkzeugs für intervallarithmetische Berechnungen. Derartige Operationen werden standardmäßig von den wenigsten Programmiersprachen unterstützt. Die Bibliothek IAMath[5] von *Timothy J. Hickey*[6] bietet sich für eine Anbindung an EngCon an. Die frei verfügbare Bibliothek ermöglicht grundlegende intervallarithmetische Operationen und Funktionen und ist ebenso wie EngCon vollständig in Java implementiert. Um eine eventuelle Weiterentwicklung nicht einzuschränken ist es auch hier erforderlich, sich nicht auf eine bestimmte Bibliothek festzulegen. Stattdessen ist im Entwurf eine Schnittstelle mit einer entsprechenden Kapselung vorzusehen.

Ferner sollte die eigenständige Wiederverwendbarkeit einer zu entwickelnden Constraint-Komponente gewährleistet sein. Dafür muss sie eine allgemeine Architektur und eine Schnittstelle aufweisen, die eine Nutzung auch in anderen Systemen losgelöst von EngCon ermöglicht.

3 Der Weg zur Lösung

Die in dieser Arbeit durchgeführte Evaluierung bestehender Werkzeuge zur Behandlung von Constraint-Problemen hat ergeben, dass diese nicht die konkreten Anforderungen bzgl. einer Integration in das Konfigurierungswerkzeug EngCon erfüllen. Es konnte kein System ermittelt werden, welches zusammen die relevantesten Forderungen erfüllt: Es muss „hybrid“ sein und damit sowohl finite Domänen als auch reellwertige Intervalle unterstützen, beliebige Relationen in Form von algebraischen Constraints sowie den inkrementellen Aufbau des Constraint-Netzes ermöglichen und von der Programmiersprache Java aus unter Microsoft Windows nutzbar sein. Eine einfache, stringbasierte Schnittstelle wird von keinem der betrachteten Systeme angeboten.

[4] Die Windows-NT-Produktfamilie von Microsoft für x86-Systeme besteht derzeit aus Windows XP, Windows 2000 und Windows NT 4.0.

[5] `http://interval.sourceforge.net/interval/java/ia_math/README.html`

[6] `http://www.cs.brandeis.edu/~tim`

Auch wenn im Bereich des *Constraint Logic Programming* (CLP) durchaus hybride Systeme existieren (z. B. BNR Prolog, CLP(BNR), ECL^{i}PS^e), besteht das Problem, dass sich diese Systeme i. d. R. nur schwer adaptieren lassen, da eine Java-Schnittstelle häufig nicht enthalten ist, bzw. weil diese Systeme für die (logische) Programmierung mit Constraints ausgelegt sind.[7] Damit wird zwar üblicherweise eine Schnittstelle zur Nutzung von in diesem Kontext benötigten, *global constraints* angeboten, aber keine einfach zu nutzende (stringbasierte) Schnittstelle für beliebige Relationen. Für die Integration in EngCon sind allerdings generische Constraint-Solver zur Auflösung derartiger Constraints erforderlich, anstatt generische, d. h. wiederverwendbare, Constraints aus dem Bereich *Constraint Programming* (CP).

Eine Schnittstelle zur Anbindung an EngCon muss in jedem Fall entwickelt und implementiert werden. Die durch diese Schnittstelle bereitgestellte Umgebung ist maßgeblich für die Integration von Constraint-Verfahren in EngCon, seien es Fremdsysteme oder Eigenentwicklungen. Der Nutzen einer möglicherweise komplizierten Anbindung eines Fremdsystems an diese Schnittstelle kann sich dagegen im Vergleich zur Eigenimplementierung von Lösungsalgorithmen als recht gering erweisen. Besonders wenn die im Fremdsystem vorhandenen Constraint-Lösungsverfahren nicht sehr anspruchsvoll sind, wie z. B. in JACK[8] (Abdennadher et al. 2001, 2002a, b) und GNU Prolog[9] (Diaz und Codognet 2001). Dabei würde von einem Fremdsystem ausschließlich der Zugriff auf dessen Constraint-Verarbeitung benötigt. Relevant für EngCon ist nicht die Deklarativität z. B. einer Prolog-Umgebung, sondern der stringbasierte Zugriff auf deren Constraint-Lösungsverfahren und die möglichst flexiblen Einbettung in EngCon.[10]

Um dies zu ermöglichen scheint eine Eigenentwicklung am Ehesten geeignet. Die Erfahrungen hinsichtlich der eingeschränkten Funktionalität und Skalierbarkeit, die bei einer prototypischen Integration von GNU Prolog in EngCon gemacht wurden, bestärken dies: Der Aufwand für die komplexe Integration eines Fremdsystems gestaltet sich u. U. größer, als die Implementierung eigener Algorithmen zur Constraint-Verarbeitung. Vorausgesetzt das entsprechende *Know-How* bzgl. der entsprechenden Constraint-Lösungsverfahren ist vorhanden. Ein Teil der Arbeit wird sich daher mit Verfahren zum effizienten Auflösen von Constraint-Problemen, sowohl über finite als auch über infinite Domänen, beschäftigen.

3.1 Klassische Constraint Satisfaction Probleme

Constraints über finite Domänen sind ein klassisches Gebiet der KI, auf dem bereits seit längerer Zeit geforscht wird. Sie kommen in vielen Anwendungen zum Einsatz, die Problemstellungen mit endlichen Wertebereichen beinhalten. Für klassische *Constraint Satis-*

[7] Zudem sind CLP-Systeme z. T. nur unter Unix-Systemen nutzbar, und nicht wie erforderlich unter Microsoft Windows. Ob diese Systeme in einer Umgebung wie Cygwin in der erforderlichen Stabilität nutzbar sind (vorausgesetzt der Quellcode für den notwendigen Kompilierungsvorgang ist verfügbar), kann nicht garantiert werden, wenn nicht entsprechende Pakete verfügbar sind.

[8] `http://www.pms.ifi.lmu.de/software/jack/`

[9] `http://gprolog.inria.fr`

[10] Eine deklarative Prolog-Schnittstelle zur Programmierung mit Constraints (CP) ist für den konkreten Einsatz in EngCon grundsätzlich eher ungeeignet und würde eine Nutzung des entsprechenden Constraint-Systems durch den entstehenden Overhead aufwendiger machen.

faction Probleme (CSP) mit finiten Domänen wurde daher eine Vielzahl an Constraint-Lösungsmethoden und Heuristiken entwickelt (vgl. Barták 1999a, b, 2001; Dechter 1992, 1999; Dechter und Rossi 2003; Güsgen 2000; Kumar 1992; Tsang 1993). Grundsätzlich wird zwischen Konsistenz- und (systematischen) Suchverfahren unterschieden. Durch Konsistenzverfahren allein ist es außer in Ausnahmefällen nicht möglich, Lösungen für CSPs zu finden. Stattdessen werden Konsistenzverfahren häufig als Preprozessing bzw. während eines Suchverfahrens, währenddessen die eigentlichen Lösungen generiert werden, als *look-ahead*-Mechanismus eingesetzt. Um eine weitere Optimierung des Laufzeitverhaltens zu erreichen, können Suchverfahren durch Heuristiken zur Variablen- und Werteauswahl unterstützt werden.

Eine besondere Stellung nehmen binäre CSPs ein, da viele der existierenden Lösungsalgorithmen auf binäre Constraints beschränkt sind. Verbunden wird dies häufig mit dem Hinweis, dass sich die Verfahren grundsätzlich auf n-äre CSPs verallgemeinern lassen. Aufgrund der hohen Komplexität n-ärer Constraints sind tatsächlich nur wenige Algorithmen verallgemeinert worden. Die Möglichkeit zur Binärisierung n-ärer Constraint-Netze verbessert die Situation nicht wirklich, da durch die Umwandlung in eine extensionale Repräsentation in Form von „umfassenden Variablen“ ein sehr hoher Platzbedarf verbunden mit relativ hohem Berechnungsaufwand entsteht. Die Binärisierung n-ärer CSPs, um anschließend binäre Lösungsalgorithmen verwenden zu können, ist daher nur für überschaubare Problemstellungen zu empfehlen.

Im Rahmen dieser Arbeit soll eine Grundausstattung an soliden Lösungsverfahren entstehen, die ein stabiles Laufzeitverhalten für möglichst viele Problemstellungen bieten. Verfahren zur Berechnung globaler Konsistenz sind i. A. zu aufwendig und zu ineffizient. Kantenkonsistenz dagegen ist ein weit verbreitetes Verfahren, welches einen angemessenen Kompromiss zwischen Berechnungsaufwand und der Menge der gefilterten Werte darstellt. Aufgrund der notwendigen extensionalen Repräsentation für den `AC-4`-Algorithmus und höher (durch *support*-Einträge), sowie aufgrund des durchschnittlich besseren Laufzeitverhaltens (vgl. Wallace 1993) sollte dem `AC-3`-Algorithmus bzw. einer der weiterentwickelten Varianten (`AC-8`, `AC-2000`, `AC-2001`, `AC-3.1`, `AC-`3_d) der Vorzug gegeben werden. Pfadkonsistenz ist für viele Problemstellungen zu aufwendig. Für die vorhandenen Algorithmen ist zudem eine Matrix-Repräsentation der Constraints erforderlich, was das Verfahren aufgrund des entstehenden Ressourcenbedarfs ungeeignet für umfangreiche Problemstellungen macht. Durch die Matrix-Repräsentation sind Algorithmen zur Herstellung von Pfadkonsistenz überdies auf binäre Constraints beschränkt. Ausgeschlossen werden soll ein derartiger Constraint-Solver allerdings nicht, denn u. U. ist ein Konsistenzgrad größer als Kantenkonsistenz für bestimmte, überschaubare Problemstellungen durchaus sinnvoll in Form einer Effizienzverbesserung. Weitere Konsistenzarten sind Verallgemeinerungen oder spezialisierte Varianten, die bspw. dazu dienen, möglichst frühzeitig Inkonsistenzen festzustellen oder geringfügig höhere bzw. niedrigere Konsistenzgrade anzubieten als die „Standardverfahren“. Diese Konsistenzen können bei speziellen Problemstellungen sinnvoll sein, sie werden allerdings nicht schwerpunktmäßig in dieser Arbeit behandelt.

Neben Filter- bzw. Konsistenzalgorithmen müssen durch die Constraint-Komponente verschiedene Suchverfahren angeboten werden, in deren Verlauf Lösungen ermittelt werden, sofern vorhanden. Für einfache Problemstellungen sollte, um unnötigen Overhead

zu vermeiden, einfaches, chronologisches Backtracking nutzbar sein. Außerdem sollte für die Verarbeitung komplexer Constraint-Netze eine (konfliktbasierte) Backjumping-Variante angeboten werden, denn Backjumping ist, im Gegensatz zu anderen Suchverfahren mit *look-back*-Strategie wie Backchecking und Backmarking, mit dynamischen und damit leistungsfähigen Heuristiken zur Variablenordnung einsetzbar. Diese Suchverfahren sollten außerdem mit Filteralgorithmen als *look-ahead*-Mechanismus kombiniert angeboten werden (*Forward Checking*, *Maintaining Arc Consistency*). Zur Variablenordnung sollte eine Kombination des *fail-first*-Prinzips und der *maximum-degree-ordering*-Heuristik ein stabiles Laufzeitverhalten bieten. Heuristiken zur Werteauswahl bringen, begründet durch den hohen Aufwand jeden einzelnen Wert zu betrachten, im Schnitt weniger Gewinn und müssen daher nicht vorrangig behandelt werden. Die Gefahr, durch derartige Ordnungsheuristiken unnötigen Overhead zu produzieren, ist ungleich größer.

Grundsätzlich sollten die oben genannten Verfahren auch für n-äre Constraint-Netze anwendbar sein, auch wenn der Einsatz aufgrund der hohen Komplexität derartig formulierter Problemstellungen nicht zu empfehlen ist.

3.2 Intervall Constraint Satisfaction Probleme

Die Domänen reellwertiger algebraischer Constraints innerhalb von *Intervall Constraint Satisfaction Problemen* (ICSP) werden in Form von Intervallen mit oberer und unterer Grenze definiert (vgl. Benhamou 2001; van Emden 2002). Kombinatorische Verfahren zur Aufzählung möglicher Lösungen versagen hier. Allerdings lassen sich basierend auf intervallarithmetischen Grundlagen Konsistenzalgorithmen in abgewandelter Form effizient anwenden. Die Wertebereiche der Constraint-Variablen werden hierbei möglichst weit eingeschränkt, ohne mögliche Lösungen des Problems zu verlieren. Durch ein Splitting der Wertebereiche können in einem weiterführenden Suchvorgang punktgenaue Lösungen ermittelt werden (vgl. Cleary 1987). Aufgrund einer Vielzahl von Splitting-Möglichkeiten führt dieses Verfahren bei umfangreicheren Problemen allerdings schnell zu einer kombinatorischen Explosion (vgl. Lhomme 1993, S. 236 f.).

Basierend auf dem Waltz-Filteralgorithmus zur Herstellung von Kantenkonsistenz (vgl. Waltz 1975) wurde von Davis (1987) das sog. Label Inference zur Einschränkung von Intervallgrenzen entwickelt. Fortführungen davon, die eine Zerlegung von Constraints in primitive Constraints vornehmen und dadurch eine generelle Anwendbarkeit ermöglichen, sind die Toleranzpropagation von Hyvönen (1989, 1991, 1992) und Hull-Konsistenz bzw. 2B-Konsistenz von Lhomme (1993).[11] Während im Rahmen der Toleranzpropagation von Hyvönen (1992) Möglichkeiten aufgezeigt werden, mit denen durch (*dynamic*) Splitting und der Eliminierung von Zyklen im Constraint-Netz ggf. globale Konsistenz erreicht werden kann, werden von Lhomme (1993) im Rahmen der Hull-Konsistenz höhere Konsistenzgrade durch 3B- bzw. kB-Konsistenz definiert.

Eine weiterführende Methode von Benhamou et al. (1994a, b) zur Herstellung von Box-Konsistenz setzt numerisch-mathematische Verfahren ein. Eingebettet in einen Waltz-ähnlichen Filteralgorithmus wird das Newton-Intervallverfahren, zur Einschränkung der

[11] Der durch lokale Toleranzpropagation erreichte Konsistenzgrad entspricht 2B-Konsistenz.

Intervallgrenzen genutzt. Auch hier lassen sich ggf. durch Splitting kanonische Lösungen generieren.

Ein weiterer Ansatz, die 2^k-Baum-Methode, ist in der Lage, für ununterbrochene Intervalle bzw. für Intervalle, die speziellen Konvexitätsbedingungen genügen, effizient globale Konsistenz sicherzustellen. Im Gegensatz zu Waltz-basierten Verfahren garantiert die 2^k-Baum-Methode durch ein binäres Suchverfahren Konvergenz. Sie hat allerdings im Gegensatz zu anderen numerischen Verfahren nicht die Berechnung eines einzigen, optimalen Fixpunktes zum Ziel, sondern ist eher dazu geeignet, Lösungsräume zu berechnen, die von Constraint-Systemen bestehend aus Ungleichungen gebildet werden. Für diese Fälle lässt sich eine kompakte Repräsentation des Lösungsraums berechnen (vgl. Haroud und Faltings 1994; Sam-Haroud und Faltings 1996a, b; Sam-Haroud 1995).

Für die zu entwickelnde Komponente zur Verarbeitung von Constraints mit infiniten Domänen in EngCon ist es vorrangig erforderlich, Algorithmen zum Einschränken der Intervallgrenzen zur Verfügung zu stellen. Berechnungsverfahren explizit für kanonische Lösungen stehen weniger im Fokus. Die Wertebereichseinschränkungen dagegen sollten so weit wie möglich vorgenommen, d. h. die Lösungen so dicht wie möglich umschlossen, werden. Dies lässt sich für eine Vielzahl an Problemstellungen effizient mit Verfahren ähnlich dem Waltz-Filteralgorithmus erreichen. Die (lokale) Toleranzpropagation und die Algorithmen zur Herstellung von Hull-Konsistenz (2B-Konsistenz, 3B-Konsistenz) implementieren dies für Intervalldomänen. Während für die Toleranzpropagation bzw. für Hull-Konsistenz lediglich eine Zerlegung der Constraints vorgenommen werden muss, ist für die Herstellung von Box-Konsistenz die Implementierung komplexer mathematischer Operationen und ein automatischer Gleichungsumformer erforderlich, durch den die Constraints in ein durch das Newton-Intervallverfahren verarbeitbares Format gebracht werden müssen. Für die 2^k-Baum-Methode sind eine komplexe Datenrepräsentation und aufwendige Projektionen zu implementieren. Im Gegensatz zu diesen Verfahren, die in ihrer Umsetzung zu Aufwendig sind, als dass sich dies im Rahmen dieser Arbeit bewerkstelligen ließe, ist eine Umsetzung der Toleranzpropagation bzw. von Algorithmen zur Herstellung von Hull-Konsistenz innerhalb eines vertretbaren Zeitaufwands möglich.

Aus Gründen der Komplexität sollten die in dieser Arbeit umgesetzten Verfahren ausschließlich konvexe Intervalldomänen verarbeiten. Zum Einen steigt bei der Berechnung mit diskontinuierlichen Intervallen, d. h. mit Vereinigungsmengen einzelner Teilintervalle, der Berechnungsaufwand stark an, und zum Anderen existiert für Berechnungen mit ununterbrochenen Intervallen mit der IAMath-Bibliothek bereits ein Werkzeug für intervallarithmetische Berechnungen für die Programmiersprache Java (vgl. Abschnitt 2, S. 3).

3.3 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. Im Besonderen sind dies Software-Frameworks, in denen ein Rahmenwerk für die Bewältigung eines bestimmten Aufgabenspektrums bereitgestellt wird. Sie bieten eine Architekturhilfe beim Aufteilen des Entwurfs in abstrakte Klassen und bei der Definition ihrer Zuständigkeiten

und Interaktionen. Ein objektorientiertes Framework wird für eine bestimmte Anwendung spezialisiert, indem der Entwickler anwendungsspezifische Unterklassen für abstrakte Framework-Klassen erstellt (vgl. Gamma et al. 1996, S. 37).

Objektorientierte Constraint-Frameworks im Speziellen dienen dazu, OOP-Sprachen und Techniken zur Constraint-Verarbeitung zu verbinden und in unterschiedlichen Szenarien nutzbar zu machen. Von Roy et al. (2000) wird ein Constraint-Framework als Möglichkeit beschrieben, eine flexible und erweiterbare Implementierung des Constraint-Mechanismus anzubieten. Anstatt wie bei CP- bzw. CLP-Systemen eine Domäne, nämlich die der (logischen) Programmierung mit Constraints zu fokussieren, bietet ein Constraint-Framework allgemeine Mechanismen, die eine Anpassung für spezielle Problemstellungen erlauben. In einem Framework werden demnach kollaborierende Komponenten integriert, die auf diese Weise eine wiederverwendbare Architektur für eine Vielzahl von Anwendungen zur Verfügung stellen. Weil ein Framework i. A. bereits funktionsfähige Mechanismen beinhaltet, kann es häufig auch *out of the box* wie eine Bibliothek eingesetzt werden.

Um je nach Problemstellung flexibel geeignete Lösungsverfahren einsetzen zu können, bietet sich für die zu entwickelnde Constraint-Komponente ein objektorientierter Framework-Ansatz an. Da für EngCon ein hybrides Constraint-System, sowohl für finite als auch infinite Constraint-Domänen benötigt wird, sind in jedem Fall unterschiedliche Constraint-Solver erforderlich. Neben eigenen implementierten Constraint-Solvern können in ein derartiges Framework über einen Wrapper-Mechanismus zudem bestehende Constraint-Systeme eingebunden werden, die, wenn sie die Anforderungen von EngCon auch nicht vollständig erfüllen, für bestimmte Problemstellungen ausreichend (oder gar notwendig) sind. Ein Framework bietet die notwendige Umgebung zur relativ einfachen Implementierung neuer Lösungsverfahren bzw. zur einfachen Integration von Fremdsystemen. Es bietet weiterhin den Vorteil der modularen Erweiterbarkeit und allgemeine Schnittstellen zum flexiblen Einsatz in unterschiedlichen Anwendungen auch außerhalb von EngCon.

3.4 Constraint-Lösungsstrategien

Flexibilität bzgl. 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: (1) Preprozessing, (2) Konsistenzherstellung und (3) Lösungssuche. Diese Phasen spiegeln sich innerhalb von Constraint-Lösungsstrategien wieder (vgl. Abbildung 1 auf der gegenüberliegenden Seite). In der ersten Phase wird ein Preprozessing des Constraint-Problems vorgenommen. Dies kann z. B. die Binärisierung eines Constraint-Netzes oder die Zerlegung von Constraints in primitive Constraints sein, 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.

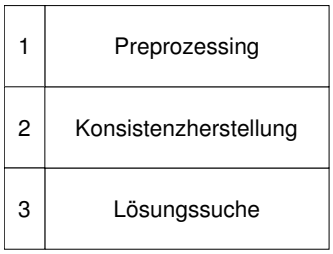

Abbildung 1: Aufbau einer Constraint-Lösungsstrategie

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.

Während es für Konsistenzverfahren durchaus sinnvoll erscheint bspw. Knotenkonsistenz herzustellen, bevor ein Algorithmus zum Herstellen von Kantenkonsistenz eingesetzt wird, erschließt sich der Sinn mehrerer Einträge im Fall von Suchverfahren nicht sofort. Für den Fall allerdings, dass für eine geeignete Anwendung aus Effizienzgründen anstatt systematischer Suchverfahren lokale bzw. stochastische Suchverfahren zum Einsatz kommen, kann hierdurch die Unvollständigkeit lokaler Verfahren abgemildert werden: Wenn ein (lokales) Suchverfahren keine Lösung für ein Constraint-Problem gefunden hat, sich aber mehrere Einträge in der Phase für die Lösungssuche befinden, kann entsprechend das nächste Suchverfahren angewendet werden.[12]

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 i. A. 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 auf der nächsten Seite zu sehen.

Zur Anwendung von derartigen Constraint-Lösungsstrategien ist eine Komponente bzw. eine Schnittstelle erforderlich, welche die Erstellung und Nutzung modularer Lösungsstrategien zur Verarbeitung von hybriden Constraint-Problemen, und basierend auf einer in eine Framework-Architektur integrierte Bibliothek von Constraint-Solvern erlaubt.

3.5 Das YACS-Framework

Aufgrund einer Vielzahl von Constraint-Lösungsverfahren und möglicher Kombinationen dieser, deren unterschiedlichen Eigenschaften und der problemabhängigen bzw. anwendungsspezifischen Effizienz unterschiedlicher Verfahren, ist eine Komponente notwendig,

[12] Lokale Suche wird aufgrund der Unvollständigkeit dieser Verfahren nicht in Zusammenhang mit ENGCON umgesetzt.

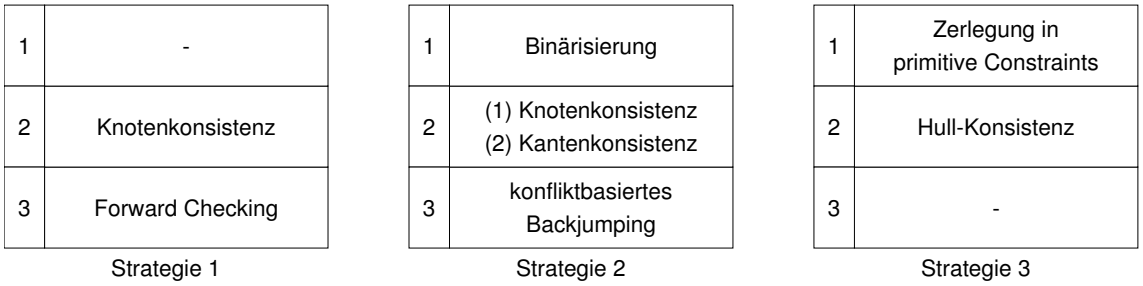

Abbildung 2: Beispiele für Constraint-Lösungsstrategien

mit der sich flexibel, je nach Problemstellung unterschiedliche Constraint-Lösungsmechanismen einsetzen lassen.

Das im Rahmen dieser Arbeit zu entwickelnde YACS-Framework stellt eine modulare und wiederverwendbare Constraint-Lösungskomponente dar. Der Name YACS steht für *Yet Another Constraint Solver*. YACS ist allerdings mehr als nur ein einzelner Constraint-Solver. Es bezeichnet vielmehr ein hybrides System zum flexiblen Einsatz von Constraint-Lösungsverfahren für finite und infinite Domänen. Sie sind eingebettet innerhalb einer modularen Framework-Architektur.

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.

Durch die Framework-Architektur wird sichergestellt, dass flexibel Lösungsverfahren 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.

3.5.1 Architektur

In Abbildung 3 auf der gegenüberliegenden Seite ist eine Übersicht über die Systemarchitektur von YACS, angebunden an das Konfigurierungswerkzeug EngCon, zu sehen. Aufsetzend auf einem *Domänen-Layer*, einer Umgebung zur arithmetischen Verarbeitung von finiten Domänen (FD) und reellwertigen Intervallen (IAMath), 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 (in diesem Fall das Constraint-System von EngCon) eine flexible Auswahl an Lösungsver-

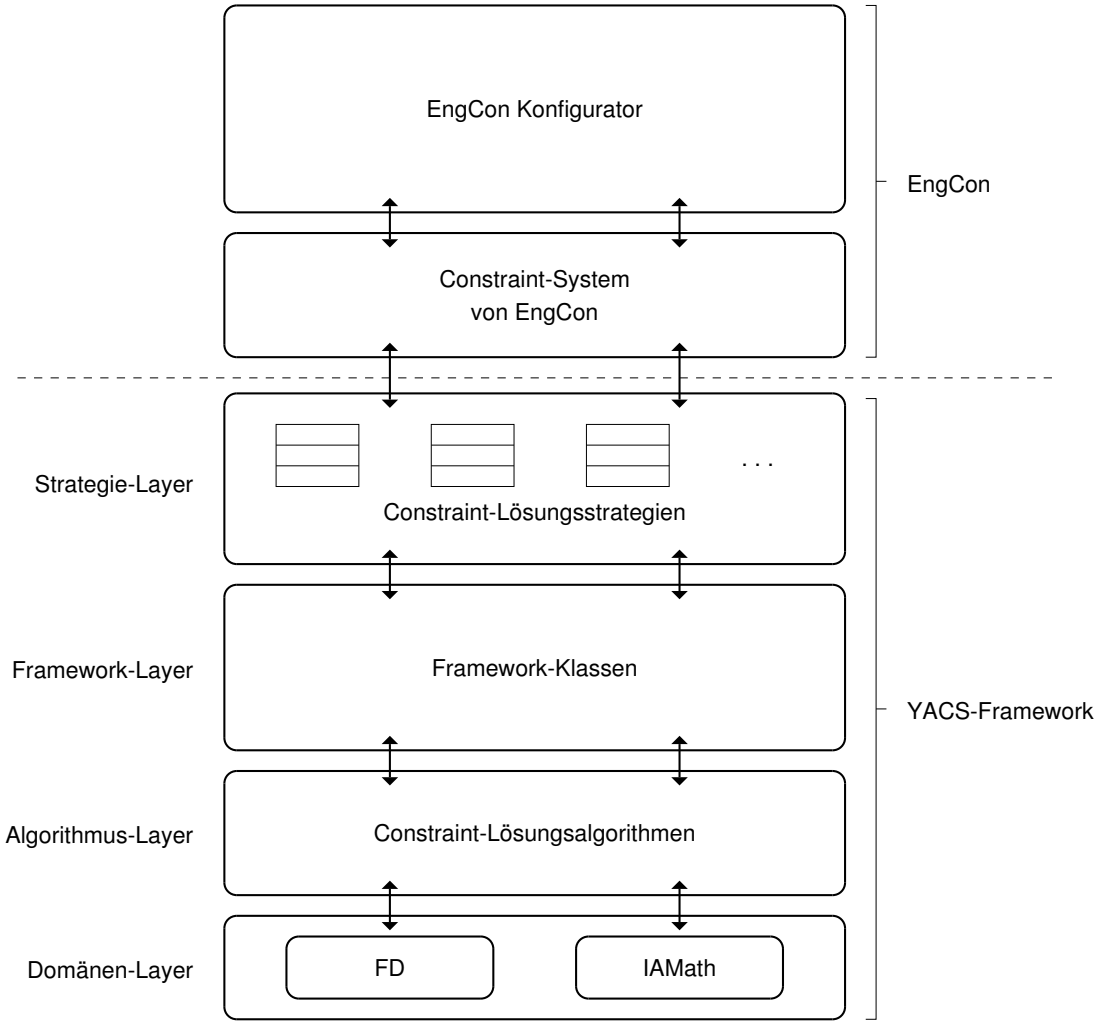

Abbildung 3: Systemarchitektur von YACS

fahren anbieten zu können.[13] Abstrahiert von den Lösungsverfahren können auf dieser Ebene aus einer Reihe vordefinierter Constraint-Lösungsstrategien problemabhängig die für die jeweilige Anwendung geeigneten Strategien ausgewählt werden.

4 Zusammenfassung

Für das Konfigurierungswerkzeug EngCon werden Constraint-Lösungsmethoden für finite und infinite Domänen benötigt. Finite Domänen werden durch diskrete Wertebereiche, infinite Domänen durch kontinuierliche, reellwertige Intervalle repräsentiert. Bestehende Constraint-Systeme erfüllen die Anforderungen für EngCon nur teilweise, zudem ist

[13] Auf eine detaillierte Darstellung der weiteren Systembestandteile von EngCon wurde in Abbildung 3 aus Gründen der Übersichtlichkeit verzichtet.

die Anbindung bestehender Systeme mit relativ hohem Integrationsaufwand verbunden. Die Eigenimplementierung geeigneter Constraint-Verfahren zum Auflösen von klassischen CSPs mit diskreten Domänen und ICSPs mit intervallwertigen Domänen scheint eine angemessene Lösung zu sein.

Aufsetzend auf einer mehrschichtigen Architektur, dem YACS-Framework, ist der Wissensingenieur für EngCon in der Lage, flexibel und abstrahiert von den eigentlichen Lösungsalgorithmen zwischen unterschiedlichen Verfahren zum Auflösen von Constraint-Problemen mit unterschiedlichen Wertedomänen auszuwählen. Das YACS-Framework umfasst eine hierfür geeignete, hybride Schnittstelle basierend auf Constraint-Lösungsstrategien. Eingebettet in eine Framework-Architektur und gesteuert über definierte Lösungsstrategien erhält der Benutzer Zugriff auf eine modulare Bibliothek von Constraint-Lösungsalgorithmen.

Das YACS-Framework bietet somit eine flexible und erweiterbare Plattform für den problemabhängigen und anwendungsspezifischen Einsatz unterschiedlicher Constraint-Lösungsmethoden für finite und infinite Domänen.

Literaturverzeichnis

Hinweis: Die Zahlen am Ende eines jeden Eintrags kennzeichnen die Seitenzahlen der vorliegenden Ausarbeitung, auf denen die jeweilige Quelle zitiert wird.

1. **Abdennadher et al. 2001** Abdennadher, Slim ; Krämer, Ekkerhard ; Saft, Matthias ; Schmauss, Matthias: JACK: A Java Constraint Kit. In: *Proceedings of the International Workshop on Functional and (Constraint) Logic Programming (WFLP'01)*, Universität Kiel (veröffentlicht als technischer Bericht Nr. 2017), 13.–15. September 2001. – URL `http://www.pms.ifi.lmu.de/publikationen/PMS-FB/PMS-FB-2001-10.pdf`. – Zugriffsdatum: 16. März 2005. – Zugl.: (Abdennadher et al. 2002a). – Gekürzte Fassung: (Abdennadher et al. 2002b) 4, 13

2. **Abdennadher et al. 2002a** Abdennadher, Slim ; Krämer, Ekkerhard ; Saft, Matthias ; Schmauss, Matthias: JACK: A Java Constraint Kit. In: *Electronic Notes in Theoretical Computer Science (ENTCS)* 64 (2002), September, S. 1–17. – URL `http://www.cs.guc.edu.eg/faculty/sabdennadher/Publikationen/paperENTC.ps.gz`. – Zugriffsdatum: 16. März 2005. – Zugl.: (Abdennadher et al. 2001). – Gekürzte Fassung: (Abdennadher et al. 2002b). – ISSN 1571-0661 4, 13

3. **Abdennadher et al. 2002b** Abdennadher, Slim ; Krämer, Ekkerhard ; Saft, Matthias ; Schmauss, Matthias: JACK: A Java Constraint Kit. In: *ALP Newsletter (Association for Logic Programming)* 15 (2002), Februar, Nr. 1. – URL `http://www.cs.kuleuven.ac.be/~dtai/projects/ALP/newsletter/feb02/nav/monfroy/monfroy_toprint.pdf`. – Zugriffsdatum: 2. Februar 2005. – Ausführliche Version: (Abdennadher et al. 2001, 2002a) 4, 13

4. **Barták 1999a** Barták, Roman: Constraint Programming: In Pursuit of the Holy Grail. In: *Proceedings of the Week of Doctoral Students (WDS'99), Part IV (Invited Lecture)*. Prague, Czechia : MatFyzPress, Juni 1999, S. 555–564. – URL `http://ktilinux.ms.mff.cuni.cz/~bartak/downloads/WDS99.pdf`. – Zugriffsdatum: 14. September 2004. – Vorhergehende Version: (Barták 1999b) 5, 13

5. **Barták 1999b** Barták, Roman: Constraint Programming: What is Behind? In: *Proceedings of the Workshop on Constraint Programming for Decision and Control (CPDC'99), Invited Talk*. Gliwice, Poland, Juni 1999, S. 7–15. – URL `http://ktilinux.ms.mff.cuni.cz/~bartak/downloads/CPDC99.pdf`. – Zugriffsdatum: 22. September 2004. – Erweiterte Fassung: (Barták 1999a) 5, 13

6. **Barták 2001** Barták, Roman: Theory and Practice of Constraint Propagation. In: *Proceedings of the 3rd Workshop on Constraint Programming for Decision and Control (CPDC'01), Invited Lecture*. Gliwice, Poland, Juni 2001, S. 7–14. – URL `http://ktilinux.ms.mff.cuni.cz/~bartak/downloads/CPDC2001.zip`. – Zugriffsdatum: 14. September 2004 5

7. **Benhamou 2001** 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`. – Zugriffsdatum: 14. September 2004. – ISBN 0-7923-6932-7 6

8. **Benhamou et al. 1994a** 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`. – Zugriffsdatum: 14. September 2004. – Vorhergehende Version: (Benhamou et al. 1994b). – ISBN 0-262-52191-1 6, 14

9. **Benhamou et al. 1994b** Benhamou, Frédéric ; McAllester, David ; Van Hentenryck, Pascal: CLP(Intervals) Revisited / University of Marseille. France, April 1994 (Technical Report CS-94-18). – Forschungsbericht. – 16 S. – URL `ftp://ftp.cs.brown.edu/pub/techreports/94/cs94-18.ps.Z`. – Zugriffsdatum: 22. September 2004. Überarb. Fassung: (Benhamou et al. 1994a) 6, 14

10. **Cleary 1987** Cleary, John G.: Logical Arithmetic. In: *Future Computing Systems* 2 (1987), Nr. 2, S. 125–149. – ISSN 0266-7207 6

11. **Cunis et al. 1991** 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 17

12. **Davis 1987** Davis, Ernest: Constraint Propagation with Interval Labels. In: *Artificial Intelligence* 32 (1987), Juli, Nr. 3, S. 281–331. – ISSN 0004-3702 1, 6

13. **Dechter 1992** Dechter, Rina: Constraint Networks. In: Shapiro, Stuart C. (Hrsg.): *Encyclopedia of Artificial Intelligence* Bd. 1. 2. Aufl. Chichester, London, New York : John Wiley & Sons, Februar 1992, S. 276–285. – URL `http://www.ics.uci.edu/~csp/r17-survey.pdf`. – Zugriffsdatum: 16. September 2004. – ISBN 0-4715-0307-X 5

14. **Dechter 1999** Dechter, Rina: Constraint Satisfaction. In: Wilson, Robert A. (Hrsg.) ; Keil, Frank C. (Hrsg.): *The MIT Encyclopedia of the Cognitive Sciences (MITECS)*. Cambridge, Massachusetts, USA : Bradford Books/The MIT Press, 1. Juni 1999, S. 195–197. – URL `http://www.ics.uci.edu/~csp/R68.pdf`. – Zugriffsdatum: 16. September 2004. – ISBN 0-262-73124-X 5

15. **Dechter und Rossi 2003** Dechter, Rina ; Rossi, Francesca: Constraint Satisfaction. In: Nadel, Lynn (Hrsg.): *Encyclopedia of Cognitive Science (ECS)* Bd. 1. London, New York, Tokyo : Nature Publishing Group/Macmillan Publishers, Juli 2003, S. 793–780. –

URL `http://www.ics.uci.edu/~csp/r85.pdf`. – Zugriffsdatum: 16. September 2004. – ISBN 0-333-79261-0 5

16. **Diaz und Codognet 2001** Diaz, Daniel ; Codognet, Philippe: Design and Implementation of the GNU Prolog System. In: *Journal of Functional and Logic Programming (JFLP)* 2001 (2001), Oktober, Nr. 6. – URL `http://danae.uni-muenster.de/lehre/kuchen/JFLP/articles/2001/S01-02/JFLP-A01-06.pdf`. – Zugriffsdatum: 13. Januar 2005. – ISSN 1080-5230 4

17. **van Emden 2002** Emden, Maarten H. van: Interval Constraints / University of Victoria, Computer Science Department. British Columbia, Canada, Stand: 26. Dezember 2002. – Tutorial. – URL `http://csr.uvic.ca/~vanemden/research/intConstrInd.html`. – Zugriffsdatum: 19. April 2005 6

18. **Freuder 1996** Freuder, Eugene C. (Hrsg.): *Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming (CP'96), Cambridge, Massachusetts, USA, 19.–22. August 1996.* Berlin, Heidelberg, New York : Springer Verlag, 1996. (LNCS 1118). – 572 S. – ISBN 3-540-61551-2 17

19. **Freuder und Mackworth 1994** Freuder, Eugene C. (Hrsg.) ; Mackworth, Alan K. (Hrsg.): *Constraint-Based Reasoning.* Cambridge, Massachusetts, USA : Bradford Books/The MIT Press, Februar 1994 (Special Issues of Artificial Intelligence). – 409 S. – ISBN 0-262-56075-5 16

20. **Gamma et al. 1996** Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software.* 1. Aufl. München, Boston, San Francisco : Addison-Wesley, 1996. – xx + 479 S. – ISBN 3-89319-950-0 8

21. **Gülden 1993** Gülden, Oliver: *Entwurf und Implementierung eines prototypischen Constraintsystems für das Konfigurierungswerkzeug KONWERK*, Universität Hamburg, Fachbereich Informatik, Studienarbeit, Dezember 1993. – iii + 76 S. – (PROKON-Memo Nr. 46) 1

22. **Güsgen 2000** Güsgen, Hans W.: Constraints. In: Görz, Günther (Hrsg.) ; Rollinger, Claus-Rainer (Hrsg.) ; Josef, Schneeberger (Hrsg.): *Handbuch der Künstlichen Intelligenz.* 3., vollst. überarb. Aufl. München : Oldenbourg Verlag, 2000, Kap. 8, S. 267–287. – ISBN 3-486-25049-3 5

23. **Haroud und Faltings 1994** Haroud, Djamila ; Faltings, Boi V.: Global Consistency for Continuous Constraints. In: Borning, Alan (Hrsg.): *Proceedings of the 2nd International Workshop on Principles and Practice of Constraint Programming (PPCP'94), Rosario, Orcas Island, Washington, USA, 2.–4. Mai 1994.* Berlin, Heidelberg, New York : Springer Verlag, 1994 (LNCS 874), S. 40–50. – URL `http://liawww.epfl.ch/Publications/Archive/Haroud1994.pdf`. – Zugriffsdatum: 14. September 2004. – Zugl.: Cohn, Anthony G. (Hrsg.): Proceedings of ECAI'94, John Wiley & Sons, 1994, S. 115–119. – ISBN 3-540-58601-6 7

24. **Hollmann et al. 2000** Hollmann, Oliver ; Wagner, Thomas ; Günter, Andreas: EngCon – A Flexible Domain-Independent Configuration Engine. In: *Proceedings of the Workshop on Configuration at the 14th European Conference on Artificial Intelligence (ECAI 2000).* Humboldt-Universität zu Berlin, 21.–22. August 2000, S. 94–96. – URL `http://www.hitec-hh.de/ueberuns/home/aguenter/literatur/ecai2000.pdf`. – Zugriffsdatum: 15. September 2004 1

25. **Hyvönen 1989** Hyvönen, Eero: Constraint Reasoning Based on Interval Arithmetic. In: (Sridharan 1989), S. 1193–1198. – ISBN 1-55860-094-9 6

26. **Hyvönen 1991** Hyvönen, Eero: Global Consistency in Interval Constraint Satisfaction. In: Mayoh, Brian H. (Hrsg.): *Proceedings of the 3rd Scandinavian Conference Conference on Artificial Intelligence (SCAI'91), Roskilde, Denmark, 21.–24. Mai 1991* Bd. 12. Amsterdam, The Netherlands : IOS Press, 1991, S. 241–251. – ISBN 90-5199-056-1 6

27. **Hyvönen 1992** 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. – Zugl.: (Freuder und Mackworth 1994), S. 71–112. – ISSN 0004-3702 1, 6

28. **Kumar 1992** Kumar, Vipin: Algorithms for Constraints Satisfaction Problems: A Survey. In: *AI Magazine* 13 (1992), Nr. 1, S. 32–44. – URL `http://www-users.cs.umn.edu/~kumar/papers/csp-aimagazine.ps`. – Zugriffsdatum: 21. September 2004. – ISSN 0738-4602 5

29. **Lhomme 1993** 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 6

30. **Ranze et al. 2002** 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`. – Zugriffsdatum: 15. September 2004. – ISBN 0-262-51129-0 1

31. **Roy et al. 2000** Roy, Pierre ; Liret, Anne ; Pachet, François: The Framework Approach for Constraint Satisfaction. In: *ACM Computing Surveys (CSUR)* 32 (2000),

März, Nr. 1es. – URL `http://www.csl.sony.fr/downloads/papers/1998/roy98a.pdf`. – Zugriffsdatum: 22. Februar 2005. – CSUR Electronic Symposium on Object-Oriented Application Frameworks – Artikel Nr. 63. – ISSN 0360-0300 8

32. **Sam-Haroud und Faltings 1996a** Sam-Haroud, Djamila ; Faltings, Boi V.: Consistency Techniques for Continuous Constraints. In: *Constraints, An International Journal* 1 (1996), September, Nr. 1–2, S. 85–118. – URL `http://liawww.epfl.ch/Publications/Archive/Sam-Haroud1996a.pdf`. – Zugriffsdatum: 14. September 2004. – ISSN 1383-7133 7

33. **Sam-Haroud und Faltings 1996b** Sam-Haroud, Djamila ; Faltings, Boi V.: Solving Non-Binary Convex CSPs in Continuous Domains. In: (Freuder 1996), S. 410–424. – URL `http://liawww.epfl.ch/Publications/Archive/Sam-Haroud1996.pdf`. – Zugriffsdatum: 14. September 2004. – ISBN 3-540-61551-2 7

34. **Sam-Haroud 1995** Sam-Haroud, Jamila: *Constraint Consistency Techniques for Continuous Constraints*. Lausanne, Switzerland, Swiss Federal Institute of Technology (EPFL), PhD Thesis No. 1423, 1995. – xviii + 178 S. – URL `http://liawww.epfl.ch/~haroud/Thesis-SamHaroud.ps.gz`. – Zugriffsdatum: 14. September 2004 7

35. **Sridharan 1989** Sridharan, Natesa S. (Hrsg.): *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI'89), Detroit, Michigan, USA, 20.–26. August 1989*. San Mateo, California, USA : Morgan Kaufmann Publishers, 1989. – ISBN 1-55860-094-9 16

36. **Syska und Cunis 1991** Syska, Ingo ; Cunis, Roman: Constraints in PLAKON. In: (Cunis et al. 1991), Kap. 6, S. 77–91. – ISBN 3-540-53683-3 1

37. **Tsang 1993** Tsang, Edward P. K.: *Foundations of Constraint Satisfaction*. London, San Diego, New York : Academic Press, 1993 (Computation in Cognitive Science). – 421 S. – URL `http://cswww.essex.ac.uk/CSP/papers/Tsang-Fcs1993.pdf/`. – Zugriffsdatum: 20. September 2004. – ISBN 0-12-701610-4 5

38. **Wallace 1993** Wallace, Richard J.: Why AC-3 is Almost Always Better than AC-4 for Establishing Arc Consistency in 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. 239–245. – ISBN 1-55860-300-X 5

39. **Waltz 1975** 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 6