INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67, D–80538 München

Ludwig
Maximilians–
Universität
München

Der Mietspiegel im Internet

Ein Fall für Constraint-Logikprogrammierung

Thom Frühwirth, Slim Abdennadher

erscheint in KI-Themenheft Constraints
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1996-16, Dezember 1996

Der Mietspiegel im Internet
Ein Fall für Constraint-Logikprogrammierung

Thom Frühwirth*und Slim Abdennadher
Institut für Informatik, Ludwig-Maximilians-Universität
Oettingenstraße 67, 80538 München
{Thom.Fruehwirth,Slim.Abdennadher}@informatik.uni-muenchen.de
http://www.pst.informatik.uni-muenchen.de/personen/fruehwir/miet-demo.html

Zusammenfassung

Der elektronische Mietspiegel im World-Wide-Web erlaubt es, in wenigen Minuten die ortsübliche Vergleichmiete einer Wohnung zu berechnen. Die Berechnungsregeln und die Datenbank dieses Expertensystems konnten direkt in einer Constraint-Logikprogrammiersprache als Regeln und Fakten implementiert werden. Durch die Anwendung von Constraint-Technologie kann die Vergleichsmiete auch berechnet werden, wenn ungenaue oder unvollständige Angaben vorliegen. Damit kann erstmals ein Mietspiegel auch zur Ermittlung des Preisniveaus bei der Wohnungssuche verwendet werden.

Constraint-Logikprogrammierung

Der Begriff "Constraint-Logikprogrammierung" (CLP) [vH91, FHK+92, JM94] bezeichnet eine Familie von Programmiersprachen, die in der zweiten Hälfte der achtziger Jahre als eine natürliche Fusion zweier deklarativer Paradigmen entstand, von Constraintlösen und Logikprogrammierung. CLP(R) (J. Jaffar, Monash University Melbourne, 1987), Prolog III (A. Colmerauer, Uni Marseille, 1987) und CHIP (P. v. Hentenryck, ECRC München, 1988) waren die ersten Sprachen der CLP.

Mit der Einbettung von Constraints in Logikprogrammiersprachen wurde es möglich, schnell und elegant komplexe kombinatorische Probleme durch eine Verbindung aus Constraintlösen und Suche zu lösen. Unter "Constraintlösen" versteht man das Lösen von Problemen, indem man "Constraints" (Bedingungen, Einschränkungen) angibt, die von einer Lösung erfüllt werden müssen. Zum Beispiel: Wir haben unser Fahrrad mit einem Zahlenschloß abgesperrt. Wir können uns nicht mehr an die erste Zahl erinnern. Wir wissen nur mehr: Sie ist ungerade, natürlich einstellig und außerdem keine Primzahl. Indem wir

*Bei Entwicklung dieser Anwendung am ECRC, München, beschäftigt.

die unvollständigen Informationen über die Zahl kombinieren, können wir die gesuchte Zahl, nämlich 9, ermitteln. Dabei sind *ungerade, einstellig* und *keine Primzahl* die Constraints, die die Zahl beschreiben. Man beachte, daß das Constraint *ungerade* für sich allein eine unendliche Menge von Lösungen besitzt. Im allgemeinen reichen Constraints allein nicht aus, um ein Problem vollständig zu lösen. Man muß zwischendurch immer wieder suchen. Hätte uns bei diesem Beispiel die letzte Information gefehlt, so wären wir darauf angewiesen gewesen, die Zahlen 1, 3, 5, 7 und 9 auszuprobieren.

Seit Anfang der neunziger Jahre wird constraintbasierte Programmierung mit großem Erfolg von mehreren Firmen (Cosytec, ILOG) weltweit kommerziell eingesetzt. Die Zahl der jährlichen kommerziellen Anwendungen wird momentan (1996) auf 300 geschätzt, der Umsatz mit Constraint-Technologie auf etwa 100 Millionen Dollar, mit stark steigender Tendenz [Wal96]. Anwendungsgebiete sind Produktions- und Ressourcenplanung (insbesondere Zeit- und Kapazitätswirtschaft), Personalplanung, Transportoptimierung, Layoutgenerierung und CAD-Systeme. Das System DAYSY von Cosytec zum Beispiel adaptiert für die Lufthansa den Einsatz von Personal nach Störungen im Flugbetrieb (Verspätungen, Erkrankung,...), sodaß die änderungen im Personalplan und die Kosten minimiert werden.

Constraint Handling Rules

Die Erfahrungen mit kommerziellen Anwendungen zeigen, daß oftmals kein homogenes Constraintproblem vorliegt, sondern eine subtile Kombination verschiedenster Constraintsysteme. Oft treten auch neuartige Constraints auf, die nur mit großem Aufwand in existierende Constraints übersetzt werden können. Um Constraints so verwenden zu können, wie sie in einer Anwendung auftreten, haben wir eine spezielle Sprache, Constraint Handling Rules (CHR), zum Schreiben von Constraintlösern entwickelt [Frü95]. Damit lassen sich schnell und ausreichend effizient constraintbasierte Algorithmen anwendungs-orientiert spezifizieren und implementieren. Unter anderem haben wir mit CHR eine Applikationsstudie implementiert, die die Berechnung einer Vergleichsmiete für eine beliebige Mietwohnung in München ermöglichen kann. Derzeit gibt es Implementierungen von CHR in Eclipse (eine Prolog-Erweiterung) [Ecl94, BFL+94], Common Lisp und OZ (beides am DFKI) sowie Sicstus Prolog [FH96].

CHR-Regeln werden in einem deklarativen, nebenläufigen Regelformalismus geschrieben, der auf sogenannten "guarded rules" [Smo93, Sha89] aufbaut. Sie ähneln jenen von Produktionssystemen wie sie in Expertensystemen Anwendung finden, sind aber speziell zum Lösen von Constraintsystemen entworfen worden.

Beispiel: Wir schreiben ein Constraint für die zweistellige Relation `=<`, für den Fall, daß die Argumente ungebundene Variablen sind, indem wir einfach die Axiome der Relation geeignet angeben:

```
Reflexivitaet @ X =< Y <=> X = Y | true.
Antisymmetrie @ X =< Y, Y =< X <=> X = Y.
Transitivitaet @ X =< Y, Y =< Z ==> X =< Z.
```

Die Reflexivitäts-Regel besagt, daß `X =< Y` wahr ist, aber nur falls `X = Y` gilt. Diese Vorbedingung für die Anwendung einer Regel wird "Wächter" genannt. Aufgrund der Reflexivitäts-Regel kann jedes Vorkommen von `A =< A` durch `true` (wahr) ersetzt werden. Diese Regel erkennt die Erfüllbarkeit eines Constraints. Die Antisymmetrie-Regel drückt aus, daß man die beiden Ungleichheitsconstraints `X =< Y` und `Y =< X` zum Gleichheitsconstraint `X = Y` vereinfachen kann. Die Antisymmetrie-Regel hat keinen Wächter, ist also immer anwendbar. Diese beiden Regeln sind Simplifikationsregeln, die Constraints vereinfachen bzw. lösen. Die Transitivitäts-Regel besagt, daß die Konjunktion `X =< Y, Y =< Z` das Constraint `X =< Z` impliziert. Mittels einer Propagationsregel werden logische Konsequenzen als redundante Constraints eingefügt. Dies ist durchaus sinnvoll, wie das folgende Beispiel zeigt:

```
:- A =< B, C =< A, B =< C.
% C =< A, A =< B fuegt C =< B mittels Transitivitaet ein.
% C =< B, B =< C wird zu B=C mittels Antisymmetrie vereinfacht.
% B =< A, A =< B wird zu A=B mittels Antisymmetrie vereinfacht.
A=B, B=C.
```

Mit Hilfe der drei CHR-Regeln kann die Anfrage also mit "`A, B` und `C` haben den gleichen Wert" beantwortet werden, obwohl für keine der drei variablen ein Wert bekannt ist.

Der Münchner Mietspiegel

Der "Münchner Mietspiegel" wird regelmäßig vom Stadtrat der Stadt München zusammen mit den Münchner Mietervereinen, dem Münchner Haus- und Grundbesitzer Verein, Sachverständigen zu Mietfragen und dem Mieterbeirat der Stadt herausgegeben. Mit einem Mietspiegel läßt sich eine Vergleichsmiete zu einer bestehenden Mietwohnung errechnen. Diese Vergleichsmiete ist auch in gerichtlich zu klärenden Rechtsstreitigkeiten zu Mietfragen als gültiges Beweismittel zugelassen. Die Berechnung basiert auf Größe, Alter, Lage der Wohnung und einer Reihe von detaillierten Fragen über die Wohnung und das Haus, die aus einer statistischen Erhebung als relevant hervorgingen. Der Mietspiegel wird auf Grundlage einer repräsentativen Stichprobe nicht preisgebundener Wohnungen entwickelt. Wegen dieses statistischen Ansatzes tritt eine inhärente Ungenauigkeit auf, die im Mietspiegel nicht ausreichend behandelt wird.

Mit Papier und Bleistift braucht man ohne die Hilfe eines Experten (z.B. von einem Mieterverein) leicht ein Wochenende, um die Vergleichsmiete zu berechnen. Einige Fragen des Münchner Mietspiegels sind schwer zu beantworten. Man benötigt Angaben, über die man in der Regel nur ungefähr Bescheid weiß, z.B. das genaue Baujahr des Hauses oder die Höhe der Kachelung im Bad einer Wohnung. Unser Prototyp "The Munich Rent Advisor" (MS) kann die Berechnungszeit auf wenige Minuten reduzieren [AF96]. Mittels Constraints ist der MS sogar in der Lage, mit ungenauen und unvollständigen Angaben und der statistischen Unschärfe umzugehen (Bild 1). Damit läßt sich erstmals ein

Mietspiegel auch dazu verwenden, bei der Wohnungsuche das Mietpreisniveau zu bestimmen, ohne sich auf eine bestimmte Wohnung festlegen zu müssen.

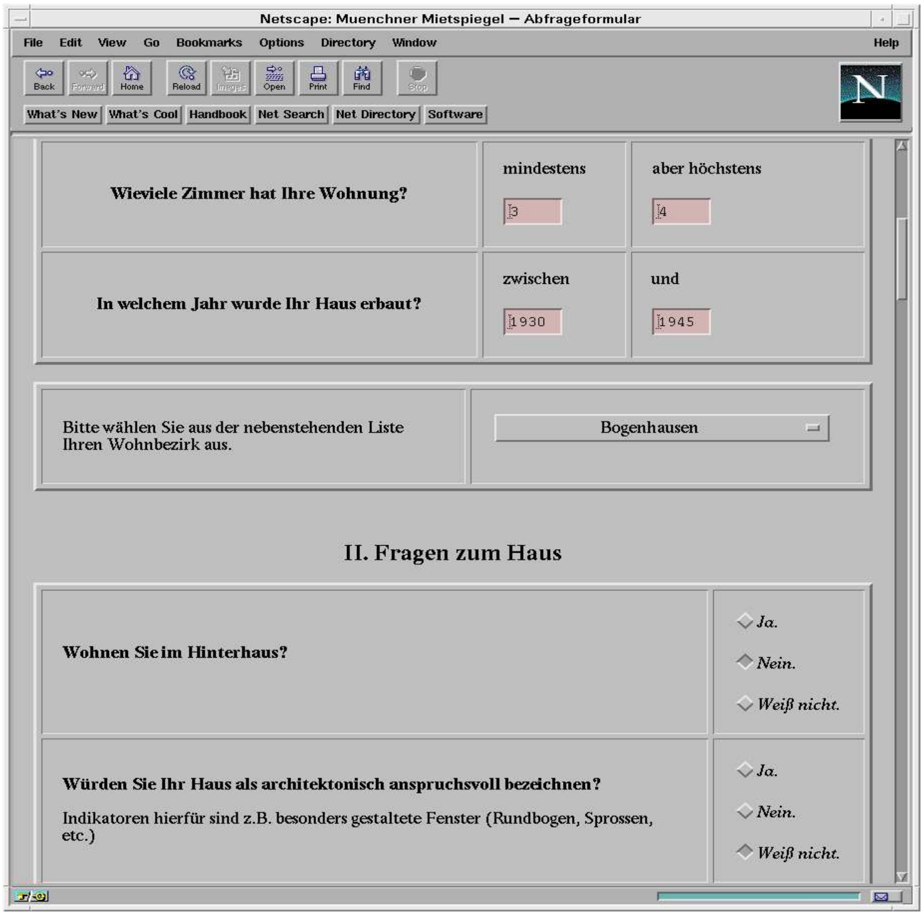

Abbildung 1: Mietspiegelformular (Ausschnitt)

Unser Ansatz war, die Tabellen, Regeln und arithmetischen Formeln mit genauen Daten in der CLP-Sprache Eclipse zu implementieren. Wegen der Deklarativität der verwendeten Programmiersprache war die Implementierung sehr einfach.

Dann haben wir Constraints eingeführt, um die Ungenauigkeit (wegen des statistischen Modells) und Unvollständigkeit (wegen ungenauer oder fehlender Benutzerantworten) zu erfassen. Dabei blieben die Tabellen, Regeln und Formeln praktisch unverändert. Jedoch werden die Formeln zur Vergleichsmietenberechnung nun als Constraints betrachtet. Das Verhalten dieser Constraints wurde in einem Constraintlöser spezifiert. Es genügte, einen Löser aus der

CHR-Bibliothek von Eclipse zu erweitern.

Weil sich Mietspiegel mit jeder neuen Ausgabe, oft deutlich, ändern und damit viele Bürger den MS jederzeit benutzen können, wurde der MS im Internet als World Wide Web (WWW) Anwendung realisiert, in Deutsch und Englisch. Das Anfrageformular wird vom Benutzer ausgefüllt und direkt von Eclipse ausgewertet. Eclipse stellt eine Reihe von vordefinierten Relationen für die Kommunikation mit dem Internet zur Verfügung. Aus diesem Grund war es einfach den Web-Server, der das Anfrageformular bearbeitet, direkt in Eclipse zu schreiben. Für die Bekanntgabe der berechneten Vergleichsmiete wird HTML-Code generiert, der ohne Umweg über ein File auf den I/O-Stream geschrieben wird (Bild 2). Zusätzlich werden auf weiteren Webseiten Erläuterungen zum Mietspiegel angeboten – mit Querverweisen (Links) zu den relevanten Institutionen wie Wohnungsamt oder Mietervereinen.

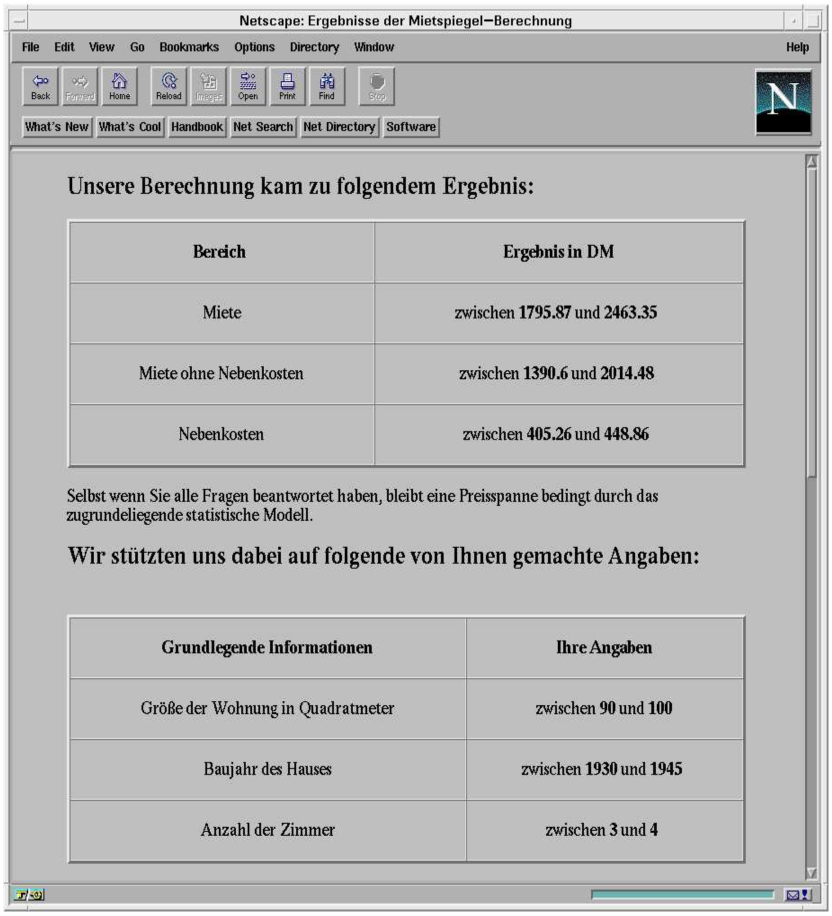

Abbildung 2: Ergebnis (Ausschnitt)

Der Mietspiegel

Die monatliche Vergleichsmiete einer Wohnung berechnet sich im Prinzip nach folgender Formel:

$$
\begin{aligned}
Vergleichsmiete &= [m^2 * Basismiete\_pro\_m^2 \\
&* (Summe\_der\_Zu\_und\_Abschläge + 100) * 0,01 \\
&* (Bandbreite + 100) * 0,01 \\
&* (Zusätzliche\_Bandbreite + 100) * 0,01] \\
&+ Nebenkosten
\end{aligned}
$$

Die $Basismiete\_pro\_m^2$ entnimmt man einer Tabelle mit etwa 200 Einträgen. Die Summe der Zu- und Abschläge wird mittels der Antworten aus dem Formular berechnet. Neben Fragen nach der Größe der Wohnung, ihrer Lage und dem Alter des Hauses, gibt es 6 Ja/Nein-Fragen über das Haus, wie z.B. die Anzahl der Stockwerke, über optische Gestaltung und Aufzug, und 13 Ja/Nein-Fragen über die Wohnung, wie z.B. über Heizungssystem, Bad- und Küchenausstattung. Die größte Abweichung im Münchner Mietspiegel hängt von der Zimmerzahl, einem eventuell vorhandenen Balkon und dem Alter des Hauses ab.

Die sich so ergebende Netto-Vergleichsmiete unterliegt einer möglichen statistischen Schwankung, einer sogenannten Bandbreite. Diese Bandbreite stellt mögliche und erlaubte Abweichungen einer Vergleichsmiete von der tatsächlichen Miete dar, die man wegen der begrenzten Zahl der Stichproben nicht weiter einengen kann. Diese Bandbreite erhöht sich daher noch zusätzlich, wenn die Wohnung in einem Neubau liegt oder die Wohnung aufgrund ihrer Quadratmeterzahl erheblich von einer durchschnittlichen Wohnung abweicht.

Gesondert müssen die Nebenkosten behandelt werden. Nebenkosten sind Anteile an Fixkosten des Hauses, wie z.B. Kosten für Müllabfuhr und Kabel-TV sowie andere Abgaben. Im MS werden 16 Nebenkostenarten behandelt und dann auf die Nettomiete addiert. Normalerweise wird der Benutzer die Fragen nach den Nebenkosten übergehen, da ihre genaue Berechnung nur im Falle einer gerichtlichen Auseinandersetzung notwendig ist.

Mietspiegel Constraints

Ein Vorteil des MS ist, daß man mit ungenauen Werten rechnen kann. Dazu braucht man arithmetische Operationen über Intervallen. Wir haben einen existierenden Constraintlöser über sogenannten endlichen Bereichen (engl. finite domains) aus der CHR-Bibliothek von Eclipse so erweitert, daß er auch Intervallconstraints über nichtlinearen Gleichungen behandeln kann.

Das Constraint der Form `X::Min:Max` schränkt die Variable `X` auf ganzzahlige Zahlenwerte zwischen `Min` und `Max` ein. Alle Variablen werden eingangs auf ihren erlaubten Wertebereich eingeschränkt, z.B. `Groesse::22:160`, d.h. die Größe einer Wohnung liegt zwischen 22 und 160 m^2.

Mit dem gegebenen Constraintlöser über endlichen Bereichen kann man nur Aufgaben behandeln, bei denen Gleichheiten oder Ungleichheiten zwischen zwei Variablen vorhanden sind. Der Löser enthält u.a. die CHR-Regeln:

```
X::Min:Min <=> X=Min.
X::Min:Max <=> gebunden(X) | Min =< X,X =< Max.
X::Min:Max ==> Min =< Max.

X =< Y, Y::Min:Max ==> gebunden(X) | Y::max(X,Min):Max.
Y =< X, Y::Min:Max ==> gebunden(X) | Y::Min:min(X,Max).
X =< Y, Y::Min:Max ==> ungebunden(X) | X =< Max.
Y =< X, Y::Min:Max ==> ungebunden(X) | X >= Min.
```

So vereinfacht sich zum Beispiel das Constraint `A::1:2, B::2:3, A>=B` unter Anwendung der letzten vier Regeln zu `A::2:2, B::2:2, A>=B` und schließlich unter zweifacher Anwendung der ersten Regel zu `A=2, B=2`.

Dieser Löser wurde für den MS um die Behandlung linearer und nichtlinearer Gleichungen der Form

`C1*X1+C2*X2+...+Cn*Xn+C0 = Y` und `C*X1*X2*...*Xn = Y` mit `n >= 0`,

erweitert, wobei die `Ci`'s und `C` Zahlen sind und die `Xi`'s und `Y` verschiedene Variablen. Diese Gleichungen und Ungleichungen zwischen zwei Variablen oder Werten sind notwendig, um die im Mietspiegel vorhandenen Formeln zur Berechnung der Nebenkosten und Vergleichsmiete auszudrücken. In der Implementierung wird z. B. die Gleichung `C1*X1+C2*X2+...+Cn*Xn+C0 = Y` durch das Constraint

`summe(0:0, C1*X1+C2*X2+...+Cn*Xn+C0, Y)`

dargestellt.

```
summe(Min:Max, C0, Resultat) <=> Resultat::Min+C0:Max+C0.

summe(Min:Max, C*X+Rest, Resultat) <=> gebunden(X) |
    NewMin = Min + C*X,
    NewMax = Max + C*X,
    summe(NewMin:NewMax, Rest, Resultat).

summe(Min:Max, C*X+Rest, Resultat), X::XMin:XMax <=>
    NewMin = Min + min(C*XMin,C*XMax),
    NewMax = Max + max(C*XMin,C*XMax),
    summe(NewMin:NewMax, Rest, Resultat),
    X::XMin:XMax.
```

Das Constraint `summe` durchläuft rekursiv das lineare Polynom. In jedem Rekursionsschritt wird ein Summand der Form `C*X` bearbeitet und sein Wert zum vorläufigen Resultat im ersten Argument addiert, das in Form eines Intervalls gegeben ist. Falls `X` schon bekannt ist, addiert man `C*X` zu den Intervallenden `Min` und `Max` (zweite Regel), falls `X` noch unbekannt ist, aber ein Constraint der Form `X::XMin:XMax` existiert, dann vergrößert sich das Resultatsintervall

`Min:Max` um das mit `C` multiplizierte Intervall von `X`. Analog wird das nichtlineare Constraint der Form `C*X1*X2*...*Xn = Y` implementiert.

Zusammenfassung

Noch vor wenigen Jahren wurden constraintbasierte Systeme der Forschung zugezählt, heute sind sie Stand der Technik und haben sich im kommerziellen Praxiseinsatz bewährt. Erfolgreiche Anwendungen existieren in der Produktions- und Ressourcenplanung, Personalplanung, Transportoptimierung, Layoutgenerierung und in CAD-Systemen.

In unserem Artikel haben wir eine Sprache zum Schreiben von Constraintlösern, Constraint Handling Rules (CHR), vorgestellt, die sich gut zur Implementierung von heterogenen Constraints eignet, wie sie in realistischen Anwendungen vorzufinden sind. Die wichtigsten Eigenschaften dieser Sprache sind die schnelle Erstellung von Prototypen, von Erweiterungen, von Spezialisierungen und von Kombinationen von Constraintlösern sowie die Möglichkeit, in eine beliebige Sprache eingebettet zu werden.

Wir haben eine innovative Internetanwendung von CHR vorgestellt, die es den Münchner Bürgern ermöglicht, jederzeit die ortsübliche Vergleichsmiete ihrer Wohnung in wenigen Minuten berechnen zu lassen, auch wenn die Angaben ungenau oder unvollständig sind.

Mit dem gleichen Ansatz der Constraint-Technologie lassen sich unserer Einschätzung nach allgemein berechnungsorientierte Anwendungen, sei es in der Finanzberatung, Stundenplanung oder in der Wettervorhersage, um die Behandlung von ungenauen Angaben erweitern, ohne daß die Anwendung selbst von Grund auf neu geschrieben werden muß.

Biografien. Dr. Dipl.-Ing. Thom Frühwirth, wissenschaftlicher Assistent am Institut für Informatik der LMU München, war zuletzt fünf Jahre am European Computer-Industry Research Centre in München im Bereich Constraint-Logikprogrammierung tätig und vorher mehrere Jahre an Forschungsinstituten in den USA, Israel und Österreich.

Dipl.-Inform. Slim Abdennadher ist seit Januar 1993 wissenschaftlicher Mitarbeiter am Institut für Informatik der LMU München. Seine Arbeitsschwerpunkte sind Constraint-Programmierung im besonderen und Künstliche Intelligenz im allgemeinen.

Literatur

[AF96] S. Abdennadher and T. Frühwirth. The Munich Rent Advisor. 1st Workshop on Logic Programming Tools for Internet Applications, JICSLP96. Bonn, September 1996.

[BFL+94] P. Brisset, T. Frühwirth, P. Lim, M. Meier, T. Le Provost, J. Schimpf, and M. Wallace. *Eclipse 3.4 Extensions User Manual.* ECRC Munich Germany, July 1994.

[Ecl94] *Eclipse 3.4 User Manual*, ECRC GmbH, München, July 1994.

[FH96] T. Frühwirth and Christian Holzbaur. Realizing Constraint Handling Rules via Attributed Variables. Zur Veröffentlichung eingereicht, November 1996.

[FHK+92] Thom Frühwirth, Alexander Herold, Volker Küchenhoff, Thierry Le Provost, Pierre Lim, Eric Monfroy, and Mark Wallace. Constraint logic programming: An informal introduction. In G. Comyn, N. E. Fuchs, and M. J. Ratcliffe, editors, *Logic Programming in Action*, LNCS 636, pages 3–35. Springer-Verlag, 1992.

[Frü95] T. Frühwirth. Constraint handling rules. In A. Podelski, editor, *Constraint Programming: Basics and Trends*, LNCS 910. Springer-Verlag, March 1995.

[JM94] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 20:503–581, 1994.

[Sha89] E. Shapiro. The family of concurrent logic programming languages. In *ACM Computing Surveys*, volume 21:3, pages 413–510, September 1989.

[Smo93] G. Smolka. Residuation and guarded rules for constraint logic programming. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming: Selected Research*. MIT Press, 1993.

[vH91] P. van Hentenryck. Constraint logic programming. In *The Knowledge Engineering Review*, volume 6, pages 151–194, 1991.

[Wal96] Mark Wallace. Practical applications of constraint programming. *Constraints Journal*, 1(1,2):139–168, September 1996. Kluwer Academic Publishers.