

Cognitive Technologies

Managing Editors: D. M. Gabbay J. Siekmann

Editorial Board: A. Bundy J. G. Carbonell
M. Pinkal H. Uszkoreit M. Veloso W. Wahlster
M. J. Wooldridge

Advisory Board:

Luigia Carlucci Aiello
Franz Baader
Wolfgang Bibel
Leonard Bolc
Craig Boutilier
Ron Brachman
Bruce G. Buchanan
Anthony Cohn
Artur d'Avila Garcez
Luis Fariñas del Cerro
Koichi Furukawa
Georg Gottlob
Patrick J. Hayes
James A. Hendler
Anthony Jameson
Nick Jennings
Aravind K. Joshi
Hans Kamp
Martin Kay
Hiroaki Kitano
Robert Kowalski
Sarit Kraus
Maurizio Lenzerini
Hector Levesque
John Lloyd

Alan Mackworth
Mark Maybury
Tom Mitchell
Johanna D. Moore
Stephen H. Muggleton
Bernhard Nebel
Sharon Oviatt
Luis Pereira
Lu Ruqian
Stuart Russell
Erik Sandewall
Luc Steels
Oliviero Stock
Peter Stone
Gerhard Strube
Katia Sycara
Milind Tambe
Hidehiko Tanaka
Sebastian Thrun
Junichi Tsujii
Kurt VanLehn
Andrei Voronkov
Toby Walsh
Bonnie Webber

For further volumes:
<http://www.springer.com/series/5216>

Petra Hofstedt

Multiparadigm Constraint Programming Languages

Prof. Dr. Petra Hofstedt
Brandenburgische Technische
Universität (btu) Cottbus
Fakultät 1
Lehrstuhl Programmiersprachen
und Compilerbau
Postfach 101344
03013 Cottbus
Germany
petra.hofstedt@informatik.tu-cottbus.de

Managing Editors

Prof. Dov M. Gabbay
Augustus De Morgan Professor of Logic
Department of Computer Science
King's College London
Strand, London WC2R 2LS, UK

Prof. Dr. Jörg Siekmann
Forschungsbereich Deduktions- und
Multiagentensysteme, DFKI
Stuhlsatzenweg 3, Geb. 43
66123 Saarbrücken, Germany

Cognitive Technologies ISSN 1611-2482
ISBN 978-3-642-17329-5 e-ISBN 978-3-642-17330-1
DOI 10.1007/978-3-642-17330-1
Springer Heidelberg Dordrecht London New York

ACM Codes D.1, D.3, I.2

Library of Congress Control Number: 2011930425

© Springer-Verlag Berlin Heidelberg 2011

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Cover design: KunkelLopka GmbH, Heidelberg

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

For Sebastian and Pierre

Foreword

"Modeling" has become one of the primary concerns in modern Software Engineering. The reason is simple: starting development processes from clear and succinct models has proven to foster not only quality but also productivity. With the advance of modeling there also came a desire for automatic code generation from models. This way, the costly and error-prone implementation in terms of low-level languages should be saved. To this end, the models need to be "executable" in the widest sense.

In this general picture the concepts of constraint programming obtain a new and economically important role. Even though they are not in the current mainstream of UML-style graphical languages, they are extremely well suited for describing models. This is evident by considering the very nature of constraints: one formulates the properties that a software system shall fulfill; the implementation is done automatically by the constraint solver. So the time is ripe for constraint-based programming to come out of the more academic world, to which it still is constrained to a large extent, and show its potential for modeling real-world applications.

However, there is no silver bullet. Classical constraint systems in their pure forms are not expressive enough to be used in a large variety of application domains. Therefore they need to be augmented by other styles and concepts for programming and modeling. This leads into the realm of so-called "multiparadigm languages". There have been all kinds of approaches in Computer Science to address the "no-silver-bullet" issue. Examples range from voluminous languages such as PL/1 or ADA (which failed miserably), huge libraries (which are non-standardized and thus lead to severe problems in the long run), or Microsoft's .NET approach (which solves the problem at least on the low level of machine code). The keyword DSLs (domain-specific languages) can be viewed as the general circumscription for all kinds of attempts to address the multiparadigm idea. By contrast to .NET the emphasis here is on the integration at the level at which the users formulate their intentions.

In this dynamic evolution of ideas, concepts and efforts, the book by Petra Hofstedt provides a valuable snapshot and assessment of the state of the art and the future directions for potential evolutions. The first part of the book gives an overview of paradigms, languages and compilers that are currently available both for academic experiments and for practical applications. Based on a sound description of the mathematical foundations, the general concepts of multiparadigm constraint languages are explained and the technical means for their implementation in the form of efficient solvers are presented. Of particular interest is the classification of the interplay of the different styles of programming, such as constraint logic programming, constraint imperative programming, constraint object programming, et cetera.

The second part of the book complements this overview of languages and concepts by looking at the application domains. However, the word "case studies" does not refer here to a collection of selected examples. Rather it refers to the techniques that are needed to realize some of the multi-paradigmatic compositions. This is done on two orthogonal dimensions: First, one of the most complex combinations of paradigms is presented quite concretely, namely concurrent constraint-functional programming. Second, a generic framework is elaborated, which shall make it relatively easy to compose all kinds of paradigms within new language shells. This is best expressed by the catchphrase "from solver cooperation to language integration".

The book by Petra Hofstedt comes at a time of highly dynamic evolutions in Software and System Engineering, in particular with respect to modeling, specification and high-level problem description. It provides a valuable insight into one important aspect of this field, namely the activities centered around the declarative description of systems by way of their properties, that is, by way of constraints. It will help the reader to understand the foundations of the approach and to get a better judgement of the future perspectives.

Berlin, December 2010

Peter Pepper

Contents

1	Introduction	1
Part I Paradigms, Constraints, and Multiparadigm Programming		
2	Basic Notions	9
2.1	Signatures and Σ -structures	9
2.2	Terms, formulae, and validity	10
2.3	Substitutions and unifiers	14
3	Programming Languages and Paradigms	17
3.1	Programming languages and compilation	17
3.2	Programming paradigms	20
3.2.1	Imperative and declarative languages	20
3.2.2	Procedural and object-oriented languages	23
3.2.3	Functional programming languages	24
3.2.4	Logic programming languages	26
3.2.5	Constraint-based programming languages	31
3.2.6	Sequential, concurrent, and parallel languages	32
3.2.7	Languages of further paradigms	33
4	Constraints	35
4.1	Constraints and constraint systems	35
4.1.1	Linear arithmetic constraints	36
4.1.2	Finite domain constraints	37
4.2	Constraint solvers	39
4.3	Constraints as language constructs	43
4.3.1	Constraint abstractions	43
4.3.2	Tell-constraints for constraint propagation	45
4.3.3	Ask-constraints for process coordination	45
4.3.4	Computing solutions	47
4.4	Constraint-based applications	49

5	Multiparadigm Constraint Programming Languages	53
5.1	Language integration and programming libraries	54
5.2	Constraint logic programming (CLP)	55
5.3	Concurrent constraint logic programming (CCLP)	61
5.4	Functional logic programming (FLP)	65
5.5	Constraint functional logic programming (CFLP)	67
5.6	Constraint imperative programming (CIP)	69
5.6.1	The <i>Turtle</i> family for CIP	71
5.6.2	Constraints as objects	73
5.7	Further constraint-based paradigms	74

Part II Case Studies

6	Concurrent Constraint Functional Programming with CCFL	81
6.1	Programming with CCFL	81
6.1.1	Functional programming	82
6.1.2	Free variables	82
6.1.3	Constraint programming	83
6.2	Syntax	86
6.3	Semantics	90
6.4	Compiling CCFL into LMNTAL	101
6.4.1	LMNTAL	101
6.4.2	The structure of the compiler	104
6.4.3	The syntactic and semantic analyses	105
6.4.4	Encoding CCFL into LMNTAL	106
6.5	Parallel programming with π CCFL	114
6.6	Integration of external constraints into CCFL	117
6.6.1	External tell-constraints	117
6.6.2	External ask-constraints	119
6.6.3	Evaluation of external constraints	120
6.7	Conclusion and related work	121
7	A Generic Framework for Multiparadigm Constraint Programming	123
7.1	Constraint solver cooperation	124
7.1.1	The general model	124
7.1.2	Constraint propagation ($tell_\nu$)	126
7.1.3	Projection of constraint stores ($proj_{\nu \rightarrow \mu}$)	128
7.1.4	An example	130
7.1.5	Cooperation strategies	132
7.2	Language integration	133
7.2.1	A logic language as constraint solver	134
7.2.2	A functional logic language as constraint solver	138
7.3	Implementation	141

7.3.1	The architecture of META-S	142
7.3.2	Solvers	143
7.3.3	The strategy definition framework	148
7.3.4	Performance evaluation	153
7.4	Conclusion and related work	156
8	Outlook	159
	References	161
	Index	175

