Jean-Michel Bruel
Zohra Bellahsène (Eds.)

LNCS 2426

Advances in Object-Oriented Information Systems

OOIS 2002 Workshops
Montpellier, France, September 2002
Proceedings

Springer

Jean-Michel Bruel Zohra Bellahsène (Eds.)

Advances in Object-Oriented Information Systems

OOIS 2002 Workshops
Montpellier, France, September 2, 2002
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Jean-Michel Bruel
LIUPPA, Computer Science Research Department
University of Pau
B.P. 1155, 64013 Pau Cedex, France
E-mail: Jean-Michel.Bruel@univ-pau.fr

Zohra Bellahsène
LIRMM
161 rue Ada, 34392 Montpellier Cedex 5, France
E-mail: bella@lirmm.fr

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Advances in object oriented information systems : OOIS 2002 workshops, Montpellier, France, September 2, 2002 ; proceedings / Jean-Michel Bruel ; Johra Bellahsene (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Lecture notes in computer science ; Vol. 2426)
ISBN 3-540-44088-7

CR Subject Classification (1998): H.2, H.4, H.5, H.3, I.2, D.2, D.4, K.4.4, J.1

ISSN 0302-9743
ISBN 3-540-44088-7 Springer-Verlag Berlin Heidelberg New York

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, re-use of illustrations, recitation, broadcasting, reproduction on microfilms 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-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

© Springer-Verlag Berlin Heidelberg 2002
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Da-TeX Gerd Blumenstein
Printed on acid-free paper SPIN: 10873861 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 2426
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Preface

For the first time four workshops have been held in conjunction with the 8th Object-Oriented Information Systems conference, OOIS 2002, to encourage interaction between researchers and practitioners. Workshop topics are, of course, inline with the conference's scientific scope and provide a forum for groups of researchers and practitioners to meet together more closely and to exchange opinions and advanced ideas, and to share preliminary results on focused issues in an atmosphere that fosters interaction and problem solving.

The conference hosted four one-day workshops. The four selected workshops were fully in the spirit of a workshop session hosted by a main conference. Indeed, OOIS deals with all the topics related to the use of object-oriented techniques for the development of information systems. The four workshops are very specific and contribute to enlarging the spectrum of the more general topics treated in the main conference. The first workshop focused on a very specific and key concept of object-oriented development, the specialization/generalization hierarchy. The second one explored the use of "non-traditional" approaches (at the edge of object-oriented techniques, such as aspects, AI, etc.) to improve reuse. The third workshop dealt with optimization in Web-based information systems. And finally the fourth workshop investigated issues related to model-driven software development.

Each workshop was organized by a group of international organizers, leading a program committee in the process of reviewing submissions. Together the workshops selected 30 papers, involving about 80 authors, and gathered a good number of participants to the campus of the University of Montpellier on September 2, 2002.

The editors would like to thank Springer-Verlag for publishing this year both the main conference and workshops proceedings in the *Lecture Notes in Computer Science* series. They would also like to thank all the workshop organizers and program committee members for their support and collaboration in the success of this first series of workshops and in the preparation of this volume. Finally, they are also grateful to the local organizers for their support.

September 2002

Jean-Michel Bruel
Zohra Bellahsene

此为试读,需要完整PDF请访问: www.ertongbook.com

Organization

This volume is a compilation of the four OOIS workshops organized at the University of Montpellier. It is organized in four chapters. Each chapter contains an introduction, written by the workshop organizers, which provides an overview of the workshop contribution, along with the Program Committee and details of other related information, followed by the accepted papers of the workshop.

The proceedings of the main conference were published as LNCS Vol. 2425.

OOIS 2002 Executive Committee

General Chair:	Colette Roland, Paris I University, France
Program Co-Chairs:	Zohra Bellahsène, LIRMM, France Dilip Patel, South Bank University, UK
Workshops:	Jean-Michel Bruel, LIUPPA, France Computer Science Research Department, B.P. 1155, F-64013, Pau Université, Cedex, France E-mail: Jean-Michel.Bruel@univ-pau.fr

Lecture Notes in Computer Science

For information about Vols. 1–2346
please contact your bookseller or Springer-Verlag

Vol. 2347: P. De Bra, P. Brusilovsky, R. Conejo (Eds.), Adaptive Hypermedia and Adaptive Web-Based Systems. Proceedings, 2002. XV, 615 pages. 2002.

Vol. 2348: A. Banks Pidduck, J. Mylopoulos, C.C. Woo, M. Tamer Ozsu (Eds.), Advanced Information Systems Engineering. Proceedings, 2002. XIV, 799 pages. 2002.

Vol. 2349: J. Kontio, R. Conradi (Eds.), Software Quality – ECSQ 2002. Proceedings, 2002. XIV, 363 pages. 2002.

Vol. 2350: A. Heyden, G. Sparr, M. Nielsen, P. Johansen (Eds.), Computer Vision – ECCV 2002. Proceedings, Part I. XXVIII, 817 pages. 2002.

Vol. 2351: A. Heyden, G. Sparr, M. Nielsen, P. Johansen (Eds.), Computer Vision – ECCV 2002. Proceedings, Part II. XXVIII, 903 pages. 2002.

Vol. 2352: A. Heyden, G. Sparr, M. Nielsen, P. Johansen (Eds.), Computer Vision – ECCV 2002. Proceedings, Part III. XXVIII, 919 pages. 2002.

Vol. 2353: A. Heyden, G. Sparr, M. Nielsen, P. Johansen (Eds.), Computer Vision – ECCV 2002. Proceedings, Part IV. XXVIII, 841 pages. 2002.

Vol. 2355: M. Matsui (Ed.), Fast Software Encryption. Proceedings, 2001. VIII, 169 pages. 2001.

Vol. 2356: R. Kohavi, B.M. Masand, M. Spiliopoulou, J. Srivastava (Eds.), WEBKDD 2002 – Mining Log Data Across All Customers Touch Points. Proceedings, 2001. XI, 167 pages. 2002. (Subseries LNAI).

Vol. 2358: T. Hendtlass, M. Ali (Eds.), Developments in Applied Artificial Intelligence. Proceedings, 2002 XIII, 833 pages. 2002. (Subseries LNAI).

Vol. 2359: M. Tistarelli, J. Bigun, A.K. Jain (Eds.), Biometric Authentication. Proceedings, 2002. X, 197 pages. 2002.

Vol. 2360: J. Esparza, C. Lakos (Eds.), Application and Theory of Petri Nets 2002. Proceedings, 2002. X, 445 pages. 2002.

Vol. 2361: J. Blieberger, A. Strohmeier (Eds.), Reliable Software Technologies – Ada-Europe 2002. Proceedings, 2002 XIII, 367 pages. 2002.

Vol. 2362: M. Tanabe, P. van den Besselaar, T. Ishida (Eds.), Digital Cities II. Proceedings, 2001. XI, 399 pages. 2002.

Vol. 2363: S.A. Cerri, G. Gouardères, F. Paraguaçu (Eds.), Intelligent Tutoring Systems. Proceedings, 2002. XXVIII, 1016 pages. 2002.

Vol. 2364: F. Roli, J. Kittler (Eds.), Multiple Classifier Systems. Proceedings, 2002. XI, 337 pages. 2002.

Vol. 2365: J. Daemen, V. Rijmen (Eds.), Fast Software Encryption. Proceedings, 2002. XI, 277 pages. 2002.

Vol. 2366: M.-S. Hacid, Z.W. Raś, D.A. Zighed, Y. Kodratoff (Eds.), Foundations of Intelligent Systems. Proceedings, 2002. XII, 614 pages. 2002. (Subseries LNAI).

Vol. 2367: J. Fagerholm, J. Haataja, J. Järvinen, M. Lyly, P. Råback, V. Savolainen (Eds.), Applied Parallel Computing. Proceedings, 2002. XIV, 612 pages. 2002.

Vol. 2368: M. Penttonen, E. Meineche Schmidt (Eds.), Algorithm Theory – SWAT 2002. Proceedings, 2002. XIV, 450 pages. 2002.

Vol. 2369: C. Fieker, D.R. Kohel (Eds.), Algorithmic Number Theory. Proceedings, 2002. IX, 517 pages. 2002.

Vol. 2370: J. Bishop (Ed.), Component Deployment. Proceedings, 2002. XII, 269 pages. 2002.

Vol. 2371: S. Koenig, R.C. Holte (Eds.), Abstraction, Reformulation, and Approximation. Proceedings, 2002. XI, 349 pages. 2002. (Subseries LNAI).

Vol. 2372: A. Pettorossi (Ed.), Logic Based Program Synthesis and Transformation. Proceedings, 2001. VIII, 267 pages. 2002.

Vol. 2373: A. Apostolico, M. Takeda (Eds.), Combinatorial Pattern Matching. Proceedings, 2002. VIII, 289 pages. 2002.

Vol. 2374: B. Magnusson (Ed.), ECOOP 2002 – Object-Oriented Programming. XI, 637 pages. 2002.

Vol. 2375: J. Kivinen, R.H. Sloan (Eds.), Computational Learning Theory. Proceedings, 2002. XI, 397 pages. 2002. (Subseries LNAI).

Vol. 2377: A. Birk, S. Coradeschi, T. Satoshi (Eds.), RoboCup 2001: Robot Soccer World Cup V. XIX, 763 pages. 2002. (Subseries LNAI).

Vol. 2378: S. Tison (Ed.), Rewriting Techniques and Applications. Proceedings, 2002. XI, 387 pages. 2002.

Vol. 2379: G.J. Chastek (Ed.), Software Product Lines. Proceedings, 2002. X, 399 pages. 2002.

Vol. 2380: P. Widmayer, F. Triguero, R. Morales, M. Hennessy, S. Eidenbenz, R. Conejo (Eds.), Automata, Languages and Programming. Proceedings, 2002. XXI, 1069 pages. 2002.

Vol. 2381: U. Egly, C.G. Fermüller (Eds.), Automated Reasoning with Analytic Tableaux and Related Methods. Proceedings, 2002. X, 341 pages. 2002 .(Subseries LNAI).

Vol. 2382: A. Halevy, A. Gal (Eds.), Next Generation Information Technologies and Systems. Proceedings, 2002. VIII, 169 pages. 2002.

Vol. 2383: M.S. Lew, N. Sebe, J.P. Eakins (Eds.), Image and Video Retrieval. Proceedings, 2002. XII, 388 pages. 2002.

Vol. 2384: L. Batten, J. Seberry (Eds.), Information Security and Privacy. Proceedings, 2002. XII, 514 pages. 2002.

Vol. 2385: J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, V. Sorge (Eds.), Artificial Intelligence, Automated Reasoning, and Symbolic Computation. Proceedings, 2002. XI, 343 pages. 2002. (Subseries LNAI).

Lecture Notes in Computer Science

For information about Vols. 1–2346
please contact your bookseller or Springer-Verlag

Vol. 2347: P. De Bra, P. Brusilovsky, R. Conejo (Eds.), Adaptive Hypermedia and Adaptive Web-Based Systems. Proceedings, 2002. XV, 615 pages. 2002.

Vol. 2348: A. Banks Pidduck, J. Mylopoulos, C.C. Woo, M. Tamer Ozsu (Eds.), Advanced Information Systems Engineering. Proceedings, 2002. XIV, 799 pages. 2002.

Vol. 2349: J. Kontio, R. Conradi (Eds.), Software Quality – ECSQ 2002. Proceedings, 2002. XIV, 363 pages. 2002.

Vol. 2350: A. Heyden, G. Sparr, M. Nielsen, P. Johansen (Eds.), Computer Vision – ECCV 2002. Proceedings, Part I. XXVIII, 817 pages. 2002.

Vol. 2351: A. Heyden, G. Sparr, M. Nielsen, P. Johansen (Eds.), Computer Vision – ECCV 2002. Proceedings, Part II. XXVIII, 903 pages. 2002.

Vol. 2352: A. Heyden, G. Sparr, M. Nielsen, P. Johansen (Eds.), Computer Vision – ECCV 2002. Proceedings, Part III. XXVIII, 919 pages. 2002.

Vol. 2353: A. Heyden, G. Sparr, M. Nielsen, P. Johansen (Eds.), Computer Vision – ECCV 2002. Proceedings, Part IV. XXVIII, 841 pages. 2002.

Vol. 2355: M. Matsui (Ed.), Fast Software Encryption. Proceedings, 2001. VIII, 169 pages. 2001.

Vol. 2356: R. Kohavi, B.M. Masand, M. Spiliopoulou, J. Srivastava (Eds.), WEBKDD 2002 – Mining Log Data Across All Customers Touch Points. Proceedings, 2001. XI, 167 pages. 2002. (Subseries LNAI).

Vol. 2358: T. Hendtlass, M. Ali (Eds.), Developments in Applied Artificial Intelligence. Proceedings, 2002 XIII, 833 pages. 2002. (Subseries LNAI).

Vol. 2359: M. Tistarelli, J. Bigun, A.K. Jain (Eds.), Biometric Authentication. Proceedings, 2002. X, 197 pages. 2002.

Vol. 2360: J. Esparza, C. Lakos (Eds.), Application and Theory of Petri Nets 2002. Proceedings, 2002. X, 445 pages. 2002.

Vol. 2361: J. Blieberger, A. Strohmeier (Eds.), Reliable Software Technologies – Ada-Europe 2002. Proceedings, 2002 XIII, 367 pages. 2002.

Vol. 2362: M. Tanabe, P. van den Besselaar, T. Ishida (Eds.), Digital Cities II. Proceedings, 2001. XI, 399 pages. 2002.

Vol. 2363: S.A. Cerri, G. Gouardères, F. Paraguaçu (Eds.), Intelligent Tutoring Systems. Proceedings, 2002. XXVIII, 1016 pages. 2002.

Vol. 2364: F. Roli, J. Kittler (Eds.), Multiple Classifier Systems. Proceedings, 2002. XI, 337 pages. 2002.

Vol. 2365: J. Daemen, V. Rijmen (Eds.), Fast Software Encryption. Proceedings, 2002. XI, 277 pages. 2002.

Vol. 2366: M.-S. Hacid, Z.W. Raś, D.A. Zighed, Y. Kodratoff (Eds.), Foundations of Intelligent Systems. Proceedings, 2002. XII, 614 pages. 2002. (Subseries LNAI).

Vol. 2367: J. Fagerholm, J. Haataja, J. Järvinen, M. Lyly, P. Råback, V. Savolainen (Eds.), Applied Parallel Computing. Proceedings, 2002. XIV, 612 pages. 2002.

Vol. 2368: M. Penttonen, E. Meineche Schmidt (Eds.), Algorithm Theory – SWAT 2002. Proceedings, 2002. XIV, 450 pages. 2002.

Vol. 2369: C. Fieker, D.R. Kohel (Eds.), Algorithmic Number Theory. Proceedings, 2002. IX, 517 pages. 2002.

Vol. 2370: J. Bishop (Ed.), Component Deployment. Proceedings, 2002. XII, 269 pages. 2002.

Vol. 2371: S. Koenig, R.C. Holte (Eds.), Abstraction, Reformulation, and Approximation. Proceedings, 2002. XI, 349 pages. 2002. (Subseries LNAI).

Vol. 2372: A. Pettorossi (Ed.), Logic Based Program Synthesis and Transformation. Proceedings, 2001. VIII, 267 pages. 2002.

Vol. 2373: A. Apostolico, M. Takeda (Eds.), Combinatorial Pattern Matching. Proceedings, 2002. VIII, 289 pages. 2002.

Vol. 2374: B. Magnusson (Ed.), ECOOP 2002 – Object-Oriented Programming. XI, 637 pages. 2002.

Vol. 2375: J. Kivinen, R.H. Sloan (Eds.), Computational Learning Theory. Proceedings, 2002. XI, 397 pages. 2002. (Subseries LNAI).

Vol. 2377: A. Birk, S. Coradeschi, T. Satoshi (Eds.), RoboCup 2001: Robot Soccer World Cup V. XIX, 763 pages. 2002. (Subseries LNAI).

Vol. 2378: S. Tison (Ed.), Rewriting Techniques and Applications. Proceedings, 2002. XI, 387 pages. 2002.

Vol. 2379: G.J. Chastek (Ed.), Software Product Lines. Proceedings, 2002. X, 399 pages. 2002.

Vol. 2380: P. Widmayer, F. Triguero, R. Morales, M. Hennessy, S. Eidenbenz, R. Conejo (Eds.), Automata, Languages and Programming. Proceedings, 2002. XXI, 1069 pages. 2002.

Vol. 2381: U. Egly, C.G. Fermüller (Eds.), Automated Reasoning with Analytic Tableaux and Related Methods. Proceedings, 2002. X, 341 pages. 2002 .(Subseries LNAI).

Vol. 2382: A. Halevy, A. Gal (Eds.), Next Generation Information Technologies and Systems. Proceedings, 2002. VIII, 169 pages. 2002.

Vol. 2383: M.S. Lew, N. Sebe, J.P. Eakins (Eds.), Image and Video Retrieval. Proceedings, 2002. XII, 388 pages. 2002.

Vol. 2384: L. Batten, J. Seberry (Eds.), Information Security and Privacy. Proceedings, 2002. XII, 514 pages. 2002.

Vol. 2385: J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, V. Sorge (Eds.), Artificial Intelligence, Automated Reasoning, and Symbolic Computation. Proceedings, 2002. XI, 343 pages. 2002. (Subseries LNAI).

Table of Contents

Efficient Web-Based Information Systems

Model-Driven Approaches to Software Development

此为试读,需要完整PDF请访问: www.ertongbook.com

MAnaging SPEcialization/Generalization HIerarchies

Marianne Huchard[1], Hernan Astudillo[2], and Petko Valtchev[3]

[1] L.I.R.M.M., France
[2] Financial Systems Architects, New York, USA
[3] Université de Montréal, Canada

Preface

In object-oriented approaches (modeling, programming, databases, knowledge representation), the core of systems is, most of the time, a specialization hierarchy, that organizes concepts of the application domain or software artifacts useful in the development. These concepts are usually known as classes, interfaces and types. Software Engineering methods for design and analysis are concerned by application domain modeling as well as transferring the model into the target programming language chosen for implementation. For programming languages and database systems, the specialization hierarchy is implemented by inheritance, that also supports feature (specification or code) sharing and reuse. In Knowledge Representation and data-mining approaches, the modeling aspect of a class hierarchy prevails, whereas its main purpose is to guide the process of reasoning and rule discovery.

Despite their wide and long use in these domains, specialization hierarchies still give rise to controversial interpretations and implementations. The design, implementation and maintenance of such hierarchies are complicated by their size, the numerous and conflicting generalization criteria, and the natural evolution of the domains themselves and of the knowledge about them, which of course must be reflected by the hierarchies.

The fact that two workshops ("The Inheritance Workshop" at ECOOP 2002 [2] and *MASPEGHI* at OOIS 2002) hold the same year on close topics indicates that it is time to bring to the fore specialization/generalization as a specific research field.

Among the 15 early submissions, we selected 9 papers that cover five main areas:

- general discussion about common sense specialization and its implementation,
- lattice/order theory (aspects useful for hierarchy manipulation),
- modeling (points of view and new paradigms),
- programming (analysis of practices, meta-programming),
- linguistic issues (taxonomy construction).

The web site of *MASPEGHI* remained open until the workshop for refereed late submissions that are gathered in [1].

J.-M. Bruel and Z. Bellahsène (Eds.): OOIS 2002 Workshops, LNCS 2426, pp. 1–2, 2002.
© Springer-Verlag Berlin Heidelberg 2002

Organization

The workshop was organized by Marianne Huchard (LIRMM, France), Hernan Astudillo (Financial Software Architects, USA) and Petko Valtchev (Université de Montréal, Canada).

Program Committee

Michel Dao (France Télécom R&D, France)
Robert Godin (UQAM, Canada)
Haim Kilov (Financial Systems Architects, USA)
Thérèse Libourel (LIRMM, France)
Juan Lloréns (Universidad Carlos III, Spain)
Joaquin Miller (Financial Systems Architects, USA)
Amedeo Napoli (INRIA Lorraine, France)
Ruben Prieto-Diaz (James Madison University, USA)
Derek Rayside (University of Toronto, Canada)
Houari Sahraoui (Université de Montréal, Canada)
Markku Sakkinen (Jyväskylän yliopisto, Finland)
Gregor Snelting (Universität Passau, Germany)

Additional Referees

Y. Ahronovitz, G. Ardourel, R. Ducournau, M. Lafourcade, C. Roume (LIRMM, France); Gillian Gass, Sara Scharf (University of Toronto, Canada).

Primary Contact

For more details about the workshop please contact:
Marianne Huchard
LIRMM, CNRS et Université Montpellier 2
161, rue Ada – 34392 Montpellier cedex 5, France
email: `huchard@lirmm.fr`
tel: +33 (0)4 67 41 86 58
fax: +33 (0)4 67 41 85 00
url: `http://www.lirmm.fr/~huchard/MASPEGHI/`

References

1. M. Huchard, H. Astudillo and P. Valtchev (editors) *Late Submissions of the workshop Managing SPEcialization/Generalization HIerarchies (MASPEGHI), OOIS'2002* Research Report LIRMM, CNRS et Université Montpellier 2, n.02087, August 2002.
2. A. Black, E. Ernst, P. Grogono and M. Sakkinen (editors) *Proceedings of the Inheritance workshop at ECOOP 2002* Publications of Information Technology Research Institute, University of Jyväskylä, 12/2002, ISBN: 951-39-1252-3.

"Real World" as an Argument for Covariant Specialization in Programming and Modeling

Roland Ducournau

L.I.R.M.M., Université Montpellier 2
161, rue Ada – 34392 Montpellier cedex 5, France
ducournau@lirmm.fr
http://www.lirmm.fr/~ducour/

Abstract. Class specialization is undoubtedly one of the most original and powerful features of object orientation as it structures object models at all stages of software development. Unfortunately, the semantics of specialization is not defined with the same accuracy in the various fields. In programming languages, specialization is constrained by type theory and by a type safe policy, whereas its common sense semantics dates back to the Aristotelian tradition. The well known covariant vs. contravariant controversy originates here. In this paper, we investigate how modeling and programming languages deal with this mismatch. We claim that type errors are part of the real world, so they should be taken into account at all stages of software development. Modeling as well as programming languages should adopt a covariant policy.

1 Introduction

Originated in SIMULA more than 30 years ago [3], object orientation has become, by now, quite hegemonic in the field of programming languages and software engineering, not to speak of databases or knowledge representation. This hegemony has often been explained by the closeness of various object-oriented concepts to corresponding common sense notions as they have been elaborated in classic philosophy [21,22]. Noticing that, one could hope for a *seamless* development process from so-called real world to program implementation, through analysis and design steps. However, this apparently uniform model presents some discontinuities, particularly when specialization is concerned.

Class specialization is undoubtedly one of the most original and powerful features of object orientation, yielding most of its qualities and breaking with previous programming paradigms. A large part of the literature is devoted to it, and it is the central point of many active topics of research such as inheritance (programming languages), classification or subsumption (knowledge representation), polymorphism or subtyping (type theory). Unfortunately, the semantics of specialization is not defined with the same accuracy in those various fields. Moreover, specialization may be constrained, in some field, by some external considerations. For instance, the well known *covariant vs. contravariant controversy* (e.g. [8], [18, chapter 17] or [25]) can be explained as a conflict between the

J.-M. Bruel and Z. Bellahsène (Eds.): OOIS 2002 Workshops, LNCS 2426, pp. 3–12, 2002.
© Springer-Verlag Berlin Heidelberg 2002

demands of a type safe policy and the needs for expressivity. In this paper, we look at this well known controversy from the point of view of our common sense understanding of the "real world" and investigate whether modeling languages answer adequately to this requirement. Type errors are part of the real world. A dramatic example has been given by the "mad cow" disease: `cows`, as a specialization of `herbivorous`, should only eat `grass`, not `meat`, but it happened that they were feeded with remains of cows. So, we claim that type errors should be taken into account at all stages of software development: analysis and design methods, as well as programming languages should adopt a covariant policy.

The rest of this paper is organized as follows: section 2 briefly recalls the *de facto* standard object model, then states how specialization can be related to common sense reasoning and Aristotelian tradition and gives some hints regarding how knowledge representation formalizes it. Next section takes the viewpoint of programming languages and type theory and states the covariance vs. contravariance controversy. The case of most widely used languages is examined and some alternatives such as multiple dispatch are investigated. Section 4 looks at analysis and design methods, mainly UML, and concludes to their current abdication to impose a semantics in front of JAVA's one. In conclusion, we sketch out the specifications of a language adapted to the semantics of specialization.

2 Semantics of Specialization

The *de facto* standard object model is the class-based model, consisting of *classes*, organized in a *specialization* hierarchy, and *objects* created as instances of those classes by an instantiation process. Each class is described by a set of properties, *attributes* for the state of its instances and *methods* for their behavior. Applying a method to an object follows the metaphor of *message sending* (also called *late binding*): the invoked method is selected according to the class of the object (called the *receiver*). This is the core of the model and it suffices to state the point of the specialization semantics. It is a *de facto* standard since it covers all of the widely used languages as the core of analysis and design models.

Though novel in computer science, specialization has quite ancient roots in the Aristotelian tradition, in the well known syllogism: *Socrates is a human, humans are mortals, thus Socrates is a mortal.* Here *Socrates* is an instance, *human* and *mortal* are classes. The interested reader will find in [21,22] a deep analysis of the relationships between object orientation and Aristotle syllogistic.

2.1 Inclusion of Extensions, Intensions and Domains

According to the Aristotelian tradition, as revised with the computer science vocabulary, one can generalize this example by saying that *instances of a class are also instances of its superclasses.* More formally, $\prec$ is the specialization relationship ($B \prec A$ means that B is a subclass of A) and Ext is a function which maps classes to the sets of their instances, their *extensions.* Then:

$$B \prec A \Longrightarrow Ext(B) \subseteq Ext(A) \tag{1}$$

This is the essence of specialization and it has two logical consequences: inclusion of intensions (i.e. inheritance) and inclusion of properties' domains (i.e. covariant refinement). When considering the properties of a class, one must remember that they are properties of instances of the class, factorized in the class. Let B be a subclass of A: instances of B being instances of A, have all the properties of instances of A. One says that subclasses inherit properties from superclasses. More formally, Int is a function which maps classes to the sets of their properties, their *intensions*:

$$B \prec A \Longrightarrow Int(A) \subseteq Int(B) \tag{2}$$

Properties have a value in each object and can be described in the class by a *domain*, that is the set of values taken by the property in all the class's instances. For instance, the class `Person` has a property `age` whose domain is $[0, 120]$. When specializing a class, one refines the domains of inherited properties: for instance, a subclass `Child` of `Person` will have domain $[0, 12]$ for its property `age`. The function Dom maps classes and properties to sets of values. Then:

$$B \prec A \ \ \& \ \ P \in Int(A) \Longrightarrow Dom(B, p) \subseteq Dom(A, p) \tag{3}$$

The `age` example concerns attributes. Methods may have several domains, for parameters and returned value. As an example, consider classes of `Animals`, in a hierarchy à la Linnaeus, with a method `eat` defined with different domains in classes such as `herbivorous`, `carnivorous`, and so on. [18, chapitre 17] develops a longer example, more oriented towards programming languages.

The inclusions of extensions and intensions have opposite directions, while those of extensions and domains have the same: intensions can be said *contravariant* whereas domains are *covariant*, both w.r.t. extensions, i.e. specialization.

2.2 Specialization in Knowledge Representation

Though quite intuitive, inclusion (3) cannot be proved to be entailed by (1) without a careful definition of class extensions which needs a model-theoretic approach. Such a semantics of specialization has been formalized in knowledge representation systems called *description logics* or languages of the KL-ONE family [27,10]. In previous works, we showed that such a formalization could be exported to a more standard object model but this is not a common approach [12]. A main feature of this semantics is that the equations corresponding to (1-3) can be equivalences, not mere implications: in other words, classes can be defined as necessary and sufficient conditions and specialization between classes (then called *subsumption*) can be deduced from class properties, which leads to *classification*. Previous examples obviously need such semantics since `adult` and `child` are defined by their `age`, as well as `herbivorous` and `carnivorous` by what they `eat`. However, such a semantics is not necessarily adapted to programming languages nor to analysis and design modeling, as it has a major drawback, being essentially monotonous: one can add values, not modify them. Nevertheless, it could give some hints to precise the semantics of object models, as well as semantical bases to automatic computation of class hierarchies [13].

3 Programming Languages, Subtyping and Polymorphism

Object-oriented programming languages can be considered as a mixture of object-oriented notions and programming languages notions. We will just consider the notion of type, central in programming languages, and focus on *statically typed* languages. Arguments in favor of static typing are numerous. The main one concerns reliability. Static, i.e. compile-time, analysis is needed to avoid dynamic, i.e. run-time, errors. Static typing allows a simple and efficient static analysis, whereas dynamic typing requires more expensive and less effective analyses. Anyway, static typing is another *de facto* standard.

3.1 Contravariance of Subtyping

In a statically typed language, every entity in the program text which can be bound to a value at run-time is annotated by a type, its *static type*. At run-time, every value has a type, its *dynamic type*, i.e. the class which creates the value as its instance. In such a context, an entity is said to be *polymorphic* when it can be bound to values of distinct types, and the dynamic types of the values must *conform to* the static type of the entity. Otherwise, there is a run-time type error, which may lead to an `unknown message` error when a method, called upon this entity, is known by the static type, not by the dynamic one.

Types and classes are quite similar—a type can be seen as a set of values (extension) and a set of operators (intension)—and the conformance relationship between types, denoted by ¡:, is analogous to specialization between classes. Statically typed languages allow a static (compile-time) type error checking, i.e. a *type safe* compilation. A simple way to allow this is to define conformance through the notion of *substitutability*: a type t_2 conforms to a type t_1 iff any expression of type t_1 can be substituted by (bound to) any value of type t_2 without any run-time type error. Types can be identified with classes or, preferably, types can be associated to classes but the very point is to liken class specialization and subtyping. Class specialization can support polymorphism—an instance of a subclass can be substituted to an instance of a superclass—as long as the type of the subclass conforms to the type of the superclass. Of course, with a type safe policy. Class specialization is thus constrained by type safety.

This constraint revolves around the way types of properties can be redefined (overridden) in a subclass. Let A be a class and m a method defined in A, noted m_A. Method types are noted in a functional way, with arrow types: m_A has, for instance, type $t \to t'$. Let B a subclass of A, where m is redefined in m_B, with type $u \to u'$. The type of B conforms to the type of A, only if $u \to u'$ is a subtype of $t \to t'$. Subtyping on arrow types is defined as follows [7]:

$$u \to u' \text{ ¡: } t \to t' \iff t \text{ ¡: } u \ \& \ u' \text{ ¡: } t' \tag{4}$$

A function of type $t \to t'$ can be replaced by a function of type $u \to u'$ if the latter accepts more values as parameter (t ¡: u) and returns less values (u' ¡: t'). Following Cardelli, the return type is said *covariant*, while the parameter type is *contravariant*: this is known as the *contravariance rule*. Attribute redefinition is

此为试读,需要完整PDF请访问: www.ertongbook.com