Lecture Notes in Computer Science 615

O. Lehrmann Madsen (Ed.)

ECOOP '92 European Conference on Object-Oriented Programming

Utrecht, The Netherlands, June/July 1992
Proceedings

Springer-Verlag

O. Lehrmann Madsen (Ed.)

ECOOP '92
European Conference on Object-Oriented Programming

Utrecht, The Netherlands, June 29-July 3, 1992
Proceedings

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
W-7500 Karlsruhe, FRG

Juris Hartmanis
Department of Computer Science
Cornell University
5149 Upson Hall
Ithaca, NY 14853, USA

Volume Editor

Ole Lehrmann Madsen
Computer Science Department, Aarhus University
Ny Munkegade, DK-8000 Aarhus C, Denmark

CR Subject Classification (1991): D.2.2, D.3.2, H.2.4

ISBN 3-540-55668-0 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-55668-0 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera ready by author/editor
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Lecture Notes in Computer Science 615
Edited by G. Goos and J. Hartmanis

Advisory Board: W. Brauer D. Gries J. Stoer

Preface

This volume constitutes the proceedings of the sixth European Conference on Object-Oriented Programming (ECOOP), held in Utrecht (the Netherlands), June 29 - July 3, 1992. Previous ECOOP conferences were held in Paris (France), Oslo (Norway), Nottingham (England), Ottawa (Canada, jointly with OOPSLA), and Geneva (Switzerland). Next year's ECOOP is planned to take place in Kaiserslautern (Germany).

Since the "French initiative" to organize the first conference in Paris, ECOOP has been a very successful forum for discussing the state of art of object-orientation. ECOOP has been able to attract papers of a high scientific quality as well as high quality experience papers describing the pros and cons of using object-orientation in practice. This duality between theory and practice within object-orientation makes a good example of experimental computer science. The enormous commercial success of object-orientation may also be a problem for object-orientation. You cannot open a journal these days without reading about the great advantages of object-orientation. It is quite clear to me that many people are overselling object-orientation. The result of this may be that industry reacts negatively to object-orientation when it does not live up to all the given promises. We are sure that this year's ECOOP will continue the tradition of being a high quality conference and that it will contribute to a more balanced view on object-orientation.

This volume consists of 23 papers, including one invited paper and 22 papers selected by the programme committee from 124 submissions. The selection process was very difficult due to the large amount of good papers and many good papers could not be accepted due to space limitations. Each submitted paper has been reviewed by 3-4 people. The selection of papers was based only on the quality of the papers themselves. The quality of a conference is determined by the quality of the submitted papers and the quality of the reviewing. The members of the programme committee and the other referees have invested a large amount of work in the reviewing. The authors of submitted papers, the programme committee and the other referees deserve my sincere thanks.

The conference includes presentation of invited and selected papers, a panel, tutorials, workshops, demonstrations and an exhibition. In addition, the 25th anniversary of Simula 67, the first object-oriented language, will be celebrated. I would like to thank Mehmet Akşit, Pierre America, Gert Florijn, Paul Hendriks and Chris Laffra for the great job they have done in organizing this conference. I would also like to thank the participants for their contributions. Further thanks are due to the sponsoring organizations; and to Susanne Brøndberg, who was a great help in organizing the programme committee work. The success of ECOOP'92 is due to all these people and organizations.

May 1992

Ole Lehrmann Madsen
ECOOP'92 Programme Chair

List of Referees

Jay Almarode
Birger Andersen
Patrick Arnold
Cecily Bailes
Brian Barry
Barbara Beaudoing
Jean Bell
Dag Belsnes
Mats Bengtsson
Stephanie Bodoff
Isabelle Borne
Malik Bouabsa
Marc-Michael Brandis
Alfred Brown
Böhnlein
Craig Chambers
Bernadette Charron-Bost
Robert Cole
Derek Coleman
Richard Connor
Betsy Cordingley
Andreas Creppert
Michael Crowther
Quintin Cutts
Ole-Johan Dahl
Lars-Ove Dahlin
Laurent Dami
Christophe Dony
Denise Draper
Roger Duke
Nigel Edwards
Oren Etzioni
Ed Felten
Bjorn Freeman-Benson
David Freestone
Philippe Gautron
Stein Gjessing
Vera Goebel
Joseph Goguen
Patrick Goldsack
Thorsten Gorchs
Matthieu Goutet
Nicolas Graube
Peter Grogono
Hele-Mai Haav
Michel Habib
Fiona Hayes
Görel Hedin
Andrew Herbert
Alex Heung
Toshio Hirotsu
Ian Holland
Kohei Honda
Yutaka Ishikawa
Kent Johnson
Niels Christian Juul
Paul King
Graham Kirby
Jørgen Lindskov Knudsen
Karl-Heinz Koester
Dimitri Konstantas
Kai Koskimies
A. Kotz
Stein Krogdahl
Susanna Lambertini
Douglas Lea
Sylvie Lemarié
Loïc Lescaudron
Hank Levy
Peter Linington
Brendan Mahony
Jacques Malenfant
Roberto Maiocchi
Mihail Matskin
Satoshi Matsuoka
Arne Maus
Mike Milinkovich
Tatsuo Minohara
Gerhard Mueller
David Munro
Birger Møller-Pedersen
Jean-Marc Nerson
Thai Nguyen
Peter Nickolas
Else Nordhagen
David Notkin
John O'Connell
Tim O'Connor
Maria Orlowska
Olaf Owe
François Pachet
Jens Palsberg
Michael Papathomas
Jaan Penjam
Seppo Puuronen
Timothy Regan
Didier Rémy
Barbara Rieche
Renaud Rioboo
Gordon Rose
Steve Rudkin
Heikki Saikkonen
Hayssam Saleh
Anne Salvesen
Michael Schwartzbach
Graeme Smith
Akikazu Takeuchi
Magnus Taube
François-Xavier Testard-Vaillant
Dave Thomson
Claudio Trotta
Mark Utting
Tarmo Uustalu
Juha Vihavainen
Benjamin Volosh
Shigeru Watari
Andrew Watson
Phillip Williams
Shirley Williams
Alan Wills
Jeremy Wilson
Michael Wilson
Francis Wolinski
Phillip M. Yelland
Yauhiko Yokote

Organization

Conference Chair:	Pierre America (The Netherlands)
Programme Chair:	Ole Lehrmann Madsen (Denmark)
Organizing Chair:	Gert Florijn (The Netherlands)
Tutorials:	Mehmet Akşit (The Netherlands)
Exhibition:	Paul Hendriks (The Netherlands)
Demonstrations:	Chris Laffra (The Netherlands)

Sponsor

Software Engineering Research Centre - SERC, the Netherlands

Co-sponsoring Organizations

BCS (British Computer Society)	ICG
Datex	Koning en Hartman
DND (Norwegian Computer Society)	NGI (Dutch Computer Society)
Electric Engineering	Object Management Group
Georg Heeg Smalltalk-80-Systems	OCG (Austrian Computer Society)
Getronics Network Services	Philips
Getronics Service	SI (Swiss Computer Society)
IBM	Sun Microsystems

Programme Committee

Ole Lehrmann Madsen	Aarhus University, Denmark
Alan Borning	University of Washington, USA
Jean-Pierre Briot	Université Pierre et Marie Curie, France
Pierre Cointe	Université Pierre et Marie Curie, France
Elspeth Cusack	British Telecom, U.K.
Nigel Derrett	Hewlett-Packard Laboratories, U.K.
Klaus Dittrich	Universität Zurich, Switzerland
Roger Duke	University of Queensland, Australia
Eric Juul	Københavns Universitet, Denmark
Boris Magnusson	Lund University, Sweden
Bertrand Meyer	Interactive Software Engineering, USA
Ron Morrison	University of St. Andrews, U.K.
Oscar Nierstrasz	Université de Genève, Switzerland
Walter Olthoff	German Research Center for AI, Germany
Barbara Pernici	Universitá di Udine, Italy
Jean-François Perrot	Université Pierre et Marie Curie, France
Anna-Kristin Pröfrock	Siemens Nixdorf, Germany
Trygve Reenskaug	Taskon A/S, Norway
Markku Sakkinen	University of Jyväskylä, Finland
Dave Thomas	Object Technology International, Canada
Mario Tokoro	Sony CSL, Japan
Enn Tyugu	Institute of Cybernetics, Estonia
Rebecca Wirfs-Brock	Instantiations, USA
Akinori Yonezawa	University of Tokyo, Japan
Ed Yourdon	American Programmer, USA

Lecture Notes in Computer Science

For information about Vols. 1–529
please contact your bookseller or Springer-Verlag

Vol. 530: D. H. Pitt, P.-L. Curien, S. Abramsky, A. M. Pitts, A. Poigné, D. E. Rydeheard (Eds.), Category Theory and Computer Science. Proceedings, 1991. VII, 301 pages. 1991.

Vol. 531: E. M. Clarke, R. P. Kurshan (Eds.), Computer-Aided Verification. Proceedings, 1990. XIII, 372 pages. 1991.

Vol. 532: H. Ehrig, H.-J. Kreowski, G. Rozenberg (Eds.), Graph Grammars and Their Application to Computer Science. Proceedings, 1990. X, 703 pages. 1991.

Vol. 533: E. Börger, H. Kleine Büning, M. M. Richter, W. Schönfeld (Eds.), Computer Science Logic. Proceedings, 1990. VIII, 399 pages. 1991.

Vol. 534: H. Ehrig, K. P. Jantke, F. Orejas, H. Reichel (Eds.), Recent Trends in Data Type Specification. Proceedings, 1990. VIII, 379 pages. 1991.

Vol. 535: P. Jorrand, J. Kelemen (Eds.), Fundamentals of Artificial Intelligence Research. Proceedings, 1991. VIII, 255 pages. 1991. (Subseries LNAI).

Vol. 536: J. E. Tomayko, Software Engineering Education. Proceedings, 1991. VIII, 296 pages. 1991.

Vol. 537: A. J. Menezes, S. A. Vanstone (Eds.), Advances in Cryptology – CRYPTO '90. Proceedings. XIII, 644 pages. 1991.

Vol. 538: M. Kojima, N. Megiddo, T. Noma, A. Yoshise, A Unified Approach to Interior Point Algorithms for Linear Complementarity Problems. VIII, 108 pages. 1991.

Vol. 539: H. F. Mattson, T. Mora, T. R. N. Rao (Eds.), Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. Proceedings, 1991. XI, 489 pages. 1991.

Vol. 540: A. Prieto (Ed.), Artificial Neural Networks. Proceedings, 1991. XIII, 476 pages. 1991.

Vol. 541: P. Barahona, L. Moniz Pereira, A. Porto (Eds.), EPIA '91. Proceedings, 1991. VIII, 292 pages. 1991. (Subseries LNAI).

Vol. 542: Z. W. Ras, M. Zemankova (Eds.), Methodologies for Intelligent Systems. Proceedings, 1991. X, 644 pages. 1991. (Subseries LNAI).

Vol. 543: J. Dix, K. P. Jantke, P. H. Schmitt (Eds.), Nonmonotonic and Inductive Logic. Proceedings, 1990. X, 243 pages. 1991. (Subseries LNAI).

Vol. 544: M. Broy, M. Wirsing (Eds.), Methods of Programming. XII, 268 pages. 1991.

Vol. 545: H. Alblas, B. Melichar (Eds.), Attribute Grammars , Applications and Systems. Proceedings, 1991. IX, 513 pages. 1991.

Vol. 546: O. Herzog, C.-R. Rollinger (Eds.), Text Understanding in LILOG. XI, 738 pages. 1991. (Subseries LNAI).

Vol. 547: D. W. Davies (Ed.), Advances in Cryptology – EUROCRYPT '91, Proceedings, 1991. XII, 556 pages. 1991.

Vol. 548: R. Kruse, P. Siegel (Eds.), Symbolic and Quantitative Approaches to Uncertainty. Proceedings, 1991. XI, 362 pages. 1991.

Vol. 549: E. Ardizzone, S. Gaglio, F. Sorbello (Eds.), Trends in Artificial Intelligence. Proceedings, 1991. XIV, 479 pages. 1991. (Subseries LNAI).

Vol. 550: A. van Lamsweerde, A. Fugetta (Eds.), ESEC '91. Proceedings, 1991. XII, 515 pages. 1991.

Vol. 551:S. Prehn, W. J. Toetenel (Eds.), VDM '91. Formal Software Development Methods. Volume 1. Proceedings, 1991. XIII, 699 pages. 1991.

Vol. 552: S. Prehn, W. J. Toetenel (Eds.), VDM '91. Formal Software Development Methods. Volume 2. Proceedings, 1991. XIV, 430 pages. 1991.

Vol. 553: H. Bieri, H. Noltemeier (Eds.), Computational Geometry - Methods, Algorithms and Applications '91. Proceedings, 1991. VIII, 320 pages. 1991.

Vol. 554: G. Grahne, The Problem of Incomplete Information in Relational Databases. VIII, 156 pages. 1991.

Vol. 555: H. Maurer (Ed.), New Results and New Trends in Computer Science. Proceedings, 1991. VIII, 403 pages. 1991.

Vol. 556: J.-M. Jacquet, Conclog: A Methodological Approach to Concurrent Logic Programming. XII, 781 pages. 1991.

Vol. 557: W. L. Hsu, R. C. T. Lee (Eds.), ISA '91 Algorithms. Proceedings, 1991. X, 396 pages. 1991.

Vol. 558: J. Hooman, Specification and Compositional Verification of Real-Time Systems. VIII, 235 pages. 1991.

Vol. 559: G. Butler, Fundamental Algorithms for Permutation Groups. XII, 238 pages. 1991.

Vol. 560: S. Biswas, K. V. Nori (Eds.), Foundations of Software Technology and Theoretical Computer Science. Proceedings, 1991. X, 420 pages. 1991.

Vol. 561: C. Ding, G. Xiao, W. Shan, The Stability Theory of Stream Ciphers. IX, 187 pages. 1991.

Vol. 562: R. Breu, Algebraic Specification Techniques in Object Oriented Programming Environments. XI, 228 pages. 1991.

Vol. 563: A. Karshmer, J. Nehmer (Eds.), Operating Systems of the 90s and Beyond. Proceedings, 1991. X, 285 pages. 1991.

Vol. 564: I. Herman, The Use of Projective Geometry in Computer Graphics. VIII, 146 pages. 1992.

Vol. 565: J. D. Becker, I. Eisele, F. W. Mündemann (Eds.), Parallelism, Learning, Evolution. Proceedings, 1989. VIII, 525 pages. 1991. (Subseries LNAI).

Vol. 566: C. Delobel, M. Kifer, Y. Masunaga (Eds.), Deductive and Object-Oriented Databases. Proceedings, 1991. XV, 581 pages. 1991.

Vol. 567: H. Boley, M. M. Richter (Eds.), Processing Declarative Kowledge. Proceedings, 1991. XII, 427 pages. 1991. (Subseries LNAI).

Vol. 568: H.-J. Bürckert, A Resolution Principle for a Logic with Restricted Quantifiers. X, 116 pages. 1991. (Subseries LNAI).

Vol. 569: A. Beaumont, G. Gupta (Eds.), Parallel Execution of Logic Programs. Proceedings, 1991. VII, 195 pages. 1991.

Vol. 570: R. Berghammer, G. Schmidt (Eds.), Graph-Theoretic Concepts in Computer Science. Proceedings, 1991. VIII, 253 pages. 1992.

Vol. 571: J. Vytopil (Ed.), Formal Techniques in Real-Time and Fault-Tolerant Systems. Proceedings, 1992. IX, 620 pages. 1991.

Vol. 572: K. U. Schulz (Ed.), Word Equations and Related Topics. Proceedings, 1990. VII, 256 pages. 1992.

Vol. 573: G. Cohen, S. N. Litsyn, A. Lobstein, G. Zémor (Eds.), Algebraic Coding. Proceedings, 1991. X, 158 pages. 1992.

Vol. 574: J. P. Banâtre, D. Le Métayer (Eds.), Research Directions in High-Level Parallel Programming Languages. Proceedings, 1991. VIII, 387 pages. 1992.

Vol. 575: K. G. Larsen, A. Skou (Eds.), Computer Aided Verification. Proceedings, 1991. X, 487 pages. 1992.

Vol. 576: J. Feigenbaum (Ed.), Advances in Cryptology - CRYPTO '91. Proceedings. X, 485 pages. 1992.

Vol. 577: A. Finkel, M. Jantzen (Eds.), STACS 92. Proceedings, 1992. XIV, 621 pages. 1992.

Vol. 578: Th. Beth, M. Frisch, G. J. Simmons (Eds.), Public-Key Cryptography: State of the Art and Future Directions. XI, 97 pages. 1992.

Vol. 579: S. Toueg, P. G. Spirakis, L. Kirousis (Eds.), Distributed Algorithms. Proceedings, 1991. X, 319 pages. 1992.

Vol. 580: A. Pirotte, C. Delobel, G. Gottlob (Eds.), Advances in Database Technology – EDBT '92. Proceedings. XII, 551 pages. 1992.

Vol. 581: J.-C. Raoult (Ed.), CAAP '92. Proceedings. VIII, 361 pages. 1992.

Vol. 582: B. Krieg-Brückner (Ed.), ESOP '92. Proceedings. VIII, 491 pages. 1992.

Vol. 583: I. Simon (Ed.), LATIN '92. Proceedings. IX, 545 pages. 1992.

Vol. 584: R. E. Zippel (Ed.), Computer Algebra and Parallelism. Proceedings, 1990. IX, 114 pages. 1992.

Vol. 585: F. Pichler, R. Moreno Díaz (Eds.), Computer Aided System Theory – EUROCAST '91. Proceedings. X, 761 pages. 1992.

Vol. 586: A. Cheese, Parallel Execution of Parlog. IX, 184 pages. 1992.

Vol. 587: R. Dale, E. Hovy, D. Rösner, O. Stock (Eds.), Aspects of Automated Natural Language Generation. Proceedings, 1992. VIII, 311 pages. 1992. (Subseries LNAI).

Vol. 588: G. Sandini (Ed.), Computer Vision – ECCV '92. Proceedings. XV, 909 pages. 1992.

Vol. 589: U. Banerjee, D. Gelernter, A. Nicolau, D. Padua (Eds.), Languages and Compilers for Parallel Computing. Proceedings, 1991. IX, 419 pages. 1992.

Vol. 590: B. Fronhöfer, G. Wrightson (Eds.), Parallelization in Inference Systems. Proceedings, 1990. VIII, 372 pages. 1992. (Subseries LNAI).

Vol. 591: H. P. Zima (Ed.), Parallel Computation. Proceedings, 1991. IX, 451 pages. 1992.

Vol. 592: A. Voronkov (Ed.), Logic Programming. Proceedings, 1991. IX, 514 pages. 1992. (Subseries LNAI).

Vol. 593: P. Loucopoulos (Ed.), Advanced Information Systems Engineering. Proceedings. XI, 650 pages. 1992.

Vol. 594: B. Monien, Th. Ottmann (Eds.), Data Structures and Efficient Algorithms. VIII, 389 pages. 1992.

Vol. 595: M. Levene, The Nested Universal Relation Database Model. X, 177 pages. 1992.

Vol. 596: L.-H. Eriksson, L. Hallnäs, P. Schroeder-Heister (Eds.), Extensions of Logic Programming. Proceedings, 1991. VII, 369 pages. 1992. (Subseries LNAI).

Vol. 597: H. W. Guesgen, J. Hertzberg, A Perspective of Constraint-Based Reasoning. VIII, 123 pages. 1992. (Subseries LNAI).

Vol. 598: S. Brookes, M. Main, A. Melton, M. Mislove, D. Schmidt (Eds.), Mathematical Foundations of Programming Semantics. Proceedings, 1991. VIII, 506 pages. 1992.

Vol. 599: Th. Wetter, K.-D. Althoff, J. Boose, B. R. Gaines, M. Linster, F. Schmalhofer (Eds.), Current Developments in Knowledge Acquisition - EKAW '92. Proceedings. XIII, 444 pages. 1992. (Subseries LNAI).

Vol. 600: J. W. de Bakker, K. Huizing, W. P. de Roever, G. Rozenberg (Eds.), Real-Time: Theory in Practice. Proceedings, 1991. VIII, 723 pages. 1992.

Vol. 601: D. Dolev, Z. Galil, M. Rodeh (Eds.), Theory of Computing and Systems. Proceedings, 1992. VIII, 220 pages. 1992.

Vol. 602: I. Tomek (Ed.), Computer Assisted Learning. Proceedigs, 1992. X, 615 pages. 1992.

Vol. 603: J. van Katwijk (Ed.), Ada: Moving Towards 2000. Proceedings, 1992. VIII, 324 pages. 1992.

Vol. 604: F. Belli, F.-J. Radermacher (Eds.), Industrial and Engineering Applications of Artificial Intelligence and Expert Systems. Proceedings, 1992. XV, 702 pages. 1992. (Subseries LNAI).

Vol. 605: D. Etiemble, J.-C. Syre (Eds.), PARLE '92. Parallel Architectures and Languages Europe. Proceedings, 1992. XVII, 984 pages. 1992.

Vol. 606: D. E. Knuth, Axioms and Hulls. IX, 109 pages. 1992.

Vol. 607: D. Kapur (Ed.), Automated Deduction – CADE-11. Proceedings, 1992. XV, 793 pages. 1992. (Subseries LNAI).

Vol. 608: C. Frasson, G. Gauthier, G. I. McCalla (Eds.), Intelligent Tutoring Systems. Proceedings, 1992. XIV, 686 pages. 1992.

Vol. 609: G. Rozenberg (Ed.), Advances in Petri Nets 1992. VIII, 472 pages. 1992.

Vol. 610: F. von Martial, Coordinating Plans of Autonomous Agents. XII, 246 pages. 1992. (Subseries LNAI).

Vol. 612: M. Tokoro, O. Nierstrasz, P. Wegner (Eds.), Object-Based Concurrent Computing. Proceedings, 1991. X, 265 pages. 1992.

Vol. 613: J. P. Myers, Jr., M. J. O'Donnell (Eds.), Constructivity in Computer Science. Proceedings, 1991. X, 247 pages. 1992.

Vol. 614: R. G. Herrtwich (Ed.), Network and Operating System Support for Digital Audio and Video. Proceedings, 1991. XII, 403 pages. 1992.

Vol. 615: O. Lehrmann Madsen (Ed.), ECOOP '92. European Conference on Object Oriented Programming. Proceedings. X, 426 pages. 1992.

Contents

This article was processed using the LaTeX macro package with LLNCS style

On Unifying Relational and Object–Oriented Database Systems

Won Kim

UniSQL, Inc.
9390 Research Blvd.
Austin, Texas 78759

Abstract. The past several years have been the gestation period for a new generation of database technology. There has been a flurry of activities to develop and experiment with database systems that support an object–oriented data model or that extend the relational data model with some object–oriented facilities. These activities have been fueled by the emergence of a broad spectrum of database applications which relational database systems cannot support and the increasing need to achieve another productivity leap in application development. As a result of these efforts, there is now a sufficient body of knowledge for the development of a commercially viable next–generation database system. Such a system should support a unified relational and object–oriented data model; that is, a full object–oriented data model in a way that is completely compatible with the relational model. This article examines motivations for unifying the relational and object–oriented data models in a single database system, and outlines design and implementation issues that must be addressed in building such a system.

1 Introduction

During the 1980s, relational database systems [11, 12, 13, 33] dominated the database market for business data processing applications. The relational database language SQL [3] became an industry standard. The theory, implementation, and use of database systems became a major discipline of computer science. The simplicity of the relational model of data and the dynamic management of a database have been accepted as vehicles for significant productivity enhancement in application development.

However, even as the acceptance of relational database systems spread, their limitations were exposed by the emergence of various classes of new applications. These applications include multimedia systems, statistical and scientific modeling and analysis systems, geographic information systems, engineering and design systems, knowledge–based systems, and so on. The limitations of relational database systems that these applications exposed fall into two categories: data model and computational model. The data–modeling facilities that relational systems lack include those for specifying, querying and updating

complex nested entities (such as design and engineering objects, and compound documents); arbitrary user–defined data types; frequently useful relationships, such as generalization and aggregation relationships; temporal evolution of data (i.e., temporal dimension of data, and versioning of data); and so on. The computation–modeling facilities that relational systems lack include the management of memory–resident objects for extensive pointer–chasing applications (e.g., simulation of a computer–aided design); long–duration, cooperative transactions; and so on.

The need to reduce the cost of developing and operating these applications pointed to the need for a fundamental advancement in database technology, that is, a paradigm shift, rather than incremental extensions of the capabilities of existing data management systems. The basis of this fundamental advancement in database technology is the object–oriented paradigm developed in object–oriented programming languages. There are two major reasons the object–oriented paradigm is a sound basis for a new generation of database technology.

One is that the object–oriented paradigm [16, 29, 32] can be the basis for a data model which subsumes the relational (and pre–relational) data models. Solutions to most of the data–modeling–related difficulties of relational database systems are inherent in an object–oriented data model. Relational systems are designed to manage only limited types of data, such as integer, floating–point number, string, Boolean, date, time, and money. In other words, they are not designed to manage arbitrary user–defined data types. On the other hand, a central tenet of an object–oriented data model is the uniform treatment of arbitrary data types and the facility to add new data types. Further, an object–oriented data model allows the representation of not only data, and relationships and constraints on the data, as the relational data model does; but also the encapsulation of data and programs that operate on the data, and provides a uniform framework for the treatment of arbitrary user–defined data types.

Another reason is that the object–oriented paradigm, through the notions of encapsulation and inheritance (reuse), is fundamentally designed to reduce the difficulty of developing and evolving complex software systems or designs. This is precisely the goal that has driven the data management technology from file systems to relational database systems during the past three decades. An object–oriented data model thus inherently satisfies the objective of reducing the difficulty of design and evolving very large and complex databases. The notions of encapsulation and inheritance are a key to the search for further productivity enhancement in database application development.

The promises of a database technology based on an object–oriented data model are clear. These promises fueled, during the past several years, a rush to develop a first wave of object–oriented database systems [30, 2, 8, 15, 14, 20]. About a half dozen systems are currently commercially available; some are persistent storage systems for C++ applications, some provide low–level support for engineering and design applications; and some offer a relatively richer set of database features [25].

If the object–oriented technology can truly deliver the fundamental advance in database technology to the overall database market, that is, transition the database technology past the

current relational technology, it needs to be unified with the relational technology. The relational paradigm should be extended with a set of fundamental object–oriented concepts found in most object–oriented programming languages and application systems. The database language that embodies the united modeling paradigm should be a minimal extension to the SQL relational database language. The database language should then be embedded in a wide variety of host programming lanugages to bring object–oriented modeling facilities to programmers of these languages.

SQL–compatibility is a key to the adoption of new database systems that incorporate object–oriented modeling facilities. It will minimize the time necessary for the current relational users to learn the new technology, and facilitate the migration of relational applications to the new systems. However, SQL–compatibility will not solve the mismatch that has existed until now between a database language and the host programming language in which the database language is embedded. Ideally, database application programmers should use a single database programming language for both general–purpose programming and database management. Once object–oriented programming languages become solidly established as database application programming languages, database systems that "seamlessly" support the languages will certainly be useful.

Some vendors currently offer unified database programming languages by augmenting object–oriented programming languages, notably, C++ and Smalltalk, with persistent storage, transaction management, and limited query facilities. However, important challenges remain. Persistent storage and database management facilities (especially the query facilities) must be provided without introducing obtrusive syntax or semantics that is different from that of the programming language being augmented. Database management facilities and persistent storage should be provided to objects of all data types allowed in the programming language. Further, database management facilities should be provided to not only persistent objects, but also nonpersistent objects. Finally, the extensions to one programming language should be applicable with minimal changes to a variety of other programming languages. It is important to note that, regardless of whether the database language is an extension of SQL or some object–oriented programming language, most of the considerations outlined in this paper remain valid. Ultimately, a database system must address the impact of object–oriented modeling concepts on the database architecture.

A formidable array of technical challenges must be overcome before a next–generation database system can be built that incorporates object–oriented modeling concepts into relational database systems. The purpose of this paper is to outline these challenges and how they may be met. First, the relational and object–oriented data models must be combined into a single coherent unified model. Second, a database language must be designed to allow the specification of data and relationships among them, and the querying and updating of the data. Third, the database system must be built to fully support all the facilities the database language allows. Fourth, the system must deliver high performance; in particular, it must deliver performance comparable to relational database systems for all operations that can be performed using relational database systems. The strength of an object–oriented data model is also its weakness; the richness of an object–oriented data model that makes it possible to

model complex objects and their relationships in nonbusiness data–processing applications necessarily introduces added complexities with which the users must cope. The richness of an object–oriented data model also implies added difficulties in implementing an object–oriented database system. The designers of the unified database language and database system must carefully revisit the philosophy and design rationale behind relational database systems, and address a host of technical problems in database architecture.

2 Design Issues

A database system is in essence a software that implements all the functions supported in a database language. A database language in turn is an embodiment of a data model. For example, the relational database language SQL stipulates how the users of a database system should create tables; specify data types and integrity constraints on the tables and columns within tables; populate the tables with rows; query, update, and delete the contents of the database; and so forth. A data model is thus the foundation of any database system.

Further, a database language consists of three sublanguages: data–definition, query and data manipulation, and data control language. A data–definition language allows the definition of a database. A query and data manipulation language allows the database to be populated, and the database contents to be queried, updated, and deleted. A data control language allows the specification of database sharing and administration.

In this section, I will show first that a unified relational and object–oriented data model is in essence an object–oriented data model obtained by extending the relational model with key concepts found in object–oriented programming. Next, I will outline in terms of the three database languages architectural issues in building a unified database system.

2.1 Data Model

The primary advantage of the relational model of data is its simplicity; however, simplicity is also a major disadvantage of the relational model. A relational database consists of a set of relations (tables), and a relation in turn consists of rows and columns. An entry in a table may have a single value, and the value may belong to a set of system–defined data types (e.g., integer, string, float, date, time, money). The user may impose further restrictions, called integrity constraints, on these values (e.g., the integer value of an employee age may be restricted to between 18 and 65).

This simple data model is really a subset of an object–oriented data model; that is, the relational model can be generalized to arrive at an object–oriented data model. First, let us regard a table as a data type, and allow each entry of a table to be a single value belonging to any arbitrary user–defined table, rather than just the system–defined data types. The first CREATE TABLE statement in Figure 1 shows the specification of an Employee table under the relational model. The values of the Hobby and WorksFor columns are restricted to character strings. The second CREATE TABLE in Figure 1 reflects data–type generalization for the columns of a table. The value for the Hobby column no longer needs to be restricted to a character string; it may now be a row of a user–defined table Activity.

Allowing a column of a table to hold a row of another table (i.e., data of arbitrary type) directly leads to nested tables; that is, the value of an entry of a table can now be a row of another table, and the value of an entry of that table can in turn be a row of another table, and so forth, recursively. This gives a database system the potential to support such applications as multimedia systems (which manage image, audio, graphic, text data, and compound documents that comprise of such data), scientific data processing systems (which manipulate vectors, matrices,, etc.), engineering and design systems (which deal with complex nested objects). This is the basis for bridging the large gulf in data types supported in today's programming languages and database systems.

Next, let us allow the entry of a table to have any number of values, rather than just a single value; and further allow the set of values to be of more than one data type. The restriction in the relational model that the entry of a table may hold only a single value forces

1. *CREATE TABLE Employee*
   (Name CHAR(20), Job CHAR(20), Salary FLOAT,
   Hobby CHAR(20), WorksFor CHAR(20));

2. *CREATE TABLE Employee*
   (Name CHAR(20), Job CHAR(20), Salary FLOAT,
   Hobby ACTIVITY, WorksFor COMPANY);

   CREATE TABLE Company
   (Name CHAR(20), Location STATECITY, Budget FLOAT);

   CREATE TABLE StateCity
   (State CHAR(20), City CHAR(20));

   CREATE TABLE Activity
   (Name CHAR(20), NumPlayers INTEGER);

3. *CREATE TABLE Employee*
   (Name CHAR(20), Job CHAR(20), Salary FLOAT,
   Hobby set–of ACTIVITY, WorksFor COMPANY);

4. *CREATE TABLE Employee*
   (Name CHAR(20), Job CHAR(20), Salary FLOAT,
   Hobby set–of ACTIVITY, WorksFor COMPANY)
   PROCEDURE RetirementBenefits FLOAT ;

5. *CREATE TABLE Employee*
   (Name CHAR(20), Job CHAR(20), Salary FLOAT,
   Hobby set–of ACTIVITY, WorksFor COMPANY)
   PROCEDURE RetirementBenefits FLOAT
   AS CHILD OF Person ;

   CREATE TABLE Person
   (SSN CHAR(9), Age INTEGER, Address CHAR(20));

Figure 1. Successive Extensions to the Relational Model

the creation of an extra table if the column of a table should hold more than one value. For example, suppose that an employee may have more than one hobby. It is not possible to define the Hobby column in the Employee table to store employee hobbies, and thus an extra table, say EmployeeHobby, needs to be created. The Employee and EmployeeHobby tables must be joined to retrieve information about employee's hobbies. The third CREATE TABLE statement in Figure 1 shows that the data type of the Hobby column is a *set–of* Activity, that is, the value of the Hobby column may be a set of rows of a single user–defined table Activity.

Next, let us allow a table to have procedures that operate on the column values in each row [31]. The fourth CREATE TABLE statement in Figure 1 shows the PROCEDURE clause for specifying a procedure, RetirementBenefits, which computes the retirement benefit for any given employee and returns a floating–point numeric value. A column may be regarded as a restricted procedure.

A table now encapsulates a state and a behavior of its rows; the state is the set of column values, and the behavior is the set of procedures that operate on the column values. The user may write any procedure and attach it to a table to operate on the values of any row or rows of the table. There is virtually unlimited application of procedures. It is often convenient to think of columns as special procedures; the system provides "procedures" that read or update the values of columns.

Next, let us organize all tables in the database into a hierarchy, such that between a pair of tables P and C, P is made the parent of C, if C is to take (*inherit*) all columns and procedures defined in P, besides those defined in C. Further, let us allow C to have more than one parent table from which it may take columns and procedures (this is called *multiple inheritance*). The hierarchy of tables is a directed acylic graph, with a single system–defined root. The child table is said to inherit columns and procedures from the parent table. Further, an IS–A (generalization and specialization) relationship holds between a child table and its parent table [28]. In the fifth CREATE TABLE in Figure 1, the Employee table is defined as a CHILD OF another user–defined table Person. The Employee table automatically inherits the three columns of the Person table; that is, the Employee table will now have the SSN, Age, and Address columns.

The hierarchical organization of tables also presents two types of *name conflict* problem [29, 6]. First, the names of the columns and procedures defined for a child table may be the same as those defined in a parent table. In this case, the names in the child table will override those of the parent table, that is, the child table does not take columns and procedures of the parent table by the same names. Second, if a child table is to have more than one parent table, more than one parent table may have columns and procedures with the same names. In this case, there needs to be a rule by which the child table may take appropriate columns and procedures.

This table hierarchy, with the semantics defined above, offers two advantages over the conventional relational model of a simple collection of largely independent (unrelated) tables. First, it makes it possible for a user to create a new table as a child table of an existing table; the new table inherits (reuses) all columns and procedures specified in the existing table