Tiziana Margaria
Bernhard Steffen (Eds.)

LNCS 4313

Leveraging Applications of Formal Methods

First International Symposium, ISoLA 2004
Paphos, Cyprus, October/November 2004
Revised Selected Papers

Springer

Tiziana Margaria Bernhard Steffen (Eds.)

Leveraging Applications of Formal Methods

First International Symposium, ISoLA 2004
Paphos, Cyprus, October 30 - November 2, 2004
Revised Selected Papers

Springer

Volume Editors

Tiziana Margaria
Lehrstuhl für Service and Software Engineering
Universität Potsdam , Germany
E-mail: margaria@cs.uni-potsdam.de

Bernhard Steffen
Universität Dortmund
FB Informatik, Lehrstuhl 5
Dortmund, Germany
E-mail: steffen@cs.uni-dortmund.de

Library of Congress Control Number: 2006935874

CR Subject Classification (1998): F.3, D.2.4, D.3, C.3, D.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-48928-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-48928-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11925040 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 4313

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Preface

This volume contains the main proceedings for ISoLA 2004, the 1st International Symposium on Leveraging Applications of Formal Methods held in Paphos (Cyprus) October–November 2004. Besides the 12 papers in this volume, other ISoLA 2004 contributions were selected for thematical special issues of the international journals of *Theoretical Computer Science* (TCS-B), *Software Tools for Technolgoy Transfer* (STTT), as well as *Integrated Design and Process Science* (SDPS transactions).

ISoLA 2004 served the need of providing a forum for developers, users, and researchers to discuss issues related to the adoption and use of rigorous tools and methods for the specification, analysis, verification, certification, construction, test, and maintenance of systems from the point of view of their different application domains. Thus, the ISoLA series of events serves the purpose of bridging the gap between designers and developers of rigorous tools, and users in engineering and in other disciplines, and of fostering and exploiting synergetic relationships among scientists, engineers, software developers, decision makers, and other critical thinkers in companies and organizations. In particular, by providing a venue for the discussion of common problems, requirements, algorithms, methodologies, and practices, ISoLA aims at supporting researchers in their quest to improve the utility, reliability, flexibility, and efficiency of tools for building systems, and users in their search for adequate solutions to their problems.

September 2006

Tiziana Margaria
Bernhard Steffen

Organization

Committees

Symposium Chair	Tiziana Margaria, University of Potsdam, Germany
Program Chair	Bernhard Steffen, University of Dortmund, Germany
Organization Chair	Anna Philippou, University of Cyprus, Cyprus
Industrial Chair	Manfred Reitenspiess, Fujitsu Siemens, Germany

Program Committee

Ed Brinksma	University of Twente, Netherlands
Peter Buchholz	University of Dortmund, Germany
Muffy Calder	University of Glasgow, UK
Radhia Cousot	Ecole Polytechnique, France
Jin Song Dong	National University of Singapore
Ibrahim Esat	Brunel University, UK
John Fitzgerald	University of Newcastle, UK
Robert Giegerich	University of Bielefeld, Germany
Joshua Guttman	MITRE, USA
John Hatcliff	Kansas State University, USA
Mats Heimdahl	University of Minnesota, USA
Joost-Pieter Katoen	University of Twente, Netherlands
Jens Knoop	TU Vienna, Austria
Joost Kok	University of Leiden, Netherlands
Bernd Krämer	FU Hagen, Germany
Liviu Miclea	University of Cluj-Napoca, Romania
Alice Miller	University of Glasgow, UK
Zebo Peng	University of Linkping, Sweden
Alexander Petrenko	ISPRAS, Russia
Mauro Pezzè	University of Milano-Bicocca, Italy
Paolo Prinetto	Politecnico Torino, Italy
Franz Rammig	University of Paderborn, Germany
Konstantinos Sagonas	University of Uppsala, Sweden
Bernhard Steffen	University of Dortmund, Germany
Murat Tanik	University of Alabama, USA
Marcel Verhoef	Chess, Netherlands

Gottfried Vossen	University of Mnster, Germany
Hai Wang	University of Manchester, UK
Alan Wassyng	McMaster University, Canada
Michel Wermelinger	Universidade of Nova de Lisboa, Portugal
Martin Wirsing	LMU München, Germany
Keijiro Yamaguchi	NEC, Japan
Lenore Zuck	NYU, USA

Organization Committee

Andreas Andreou	University of Cyprus - Cyprus
Yannis Demopoulos	University of Cyprus - Cyprus
Chryssis Georgiou	University of Cyprus - Cyprus
Marios Mavronicolas	University of Cyprus - Cyprus

Industrial Board

Mirko Conrad	DaimlerChrysler AG
Limor Fix	Intel, Haifa, Israel (Hardware)
Mike Hinchey	NASA, USA (Space and Robotics)
Manfred Reitenspiess	Fujitsu-Siemens, Munich Germany (Telecommunication Platforms)
Yaron Wolfsthal	IBM, Haifa Israel (HW/SW Systems)
Jianli Xu	Nokia
Yervant Zorian	Virage Logic USA (HW Systems)

Reviewers

Robby
P. Abdulla
E. Abraham
J. Andersson
R. Banach
D. Clarke
J. Deneux
J. Hatcliff
J. Ivers
J. Jacob
D. Kroening
B. Lisper
J. Rehof
M. Schordan
A. Stam
B. Steffen
L. v.d. Torre
A. Wall
Q. Yi
G. Zavattaro
W. Zimmermann

Lecture Notes in Computer Science

For information about Vols. 1–4204

please contact your bookseller or Springer

Vol. 4313: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods. IX, 197 pages. 2006.

Vol. 4300: Y.Q. Shi (Ed.), Transactions on Data Hiding and Multimedia Security I. IX, 139 pages. 2006.

Vol. 4292: G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, A. Nefian, G. Meenakshisundaram, V. Pascucci, J. Zara, J. Molineros, H. Theisel, T. Malzbender (Eds.), Advances in Visual Computing, Part II. XXXII, 906 pages. 2006.

Vol. 4291: G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, A. Nefian, G. Meenakshisundaram, V. Pascucci, J. Zara, J. Molineros, H. Theisel, T. Malzbender (Eds.), Advances in Visual Computing, Part I. XXXI, 916 pages. 2006.

Vol. 4283: Y.Q. Shi, B. Jeon (Eds.), Digital Watermarking. XII, 474 pages. 2006.

Vol. 4281: K. Barkaoui, A. Cavalcanti, A. Cerone (Eds.), Theoretical Aspects of Computing - ICTAC. XV, 371 pages. 2006.

Vol. 4279: N. Kobayashi (Ed.), Programming Languages and Systems. XI, 423 pages. 2006.

Vol. 4278: R. Meersman, Z. Tari, P. Herrero (Eds.), On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, Part II. XLV, 1004 pages. 2006.

Vol. 4277: R. Meersman, Z. Tari, P. Herrero (Eds.), On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, Part I. XLV, 1009 pages. 2006.

Vol. 4276: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, Part II. XXXII, 752 pages. 2006.

Vol. 4275: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, Part I. XXXI, 1115 pages. 2006.

Vol. 4273: I.F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, L. Aroyo (Eds.), The Semantic Web - ISWC 2006. XXIV, 1001 pages. 2006.

Vol. 4272: P. Havinga, M. Lijding, N. Meratnia, M. Wegdam (Eds.), Smart Sensing and Context. XI, 267 pages. 2006.

Vol. 4271: F.V. Fomin (Ed.), Graph-Theoretic Concepts in Computer Science. XIII, 358 pages. 2006.

Vol. 4270: H. Zha, Z. Pan, H. Thwaites, A.C. Addison, M. Forte (Eds.), Interactive Technologies and Sociotechnical Systems. XVI, 547 pages. 2006.

Vol. 4269: R. State, S. van der Meer, D. O'Sullivan, T. Pfeifer (Eds.), Large Scale Management of Distributed Systems. XIII, 282 pages. 2006.

Vol. 4268: G. Parr, D. Malone, M. Ó Foghlú (Eds.), Autonomic Principles of IP Operations and Management. XIII, 237 pages. 2006.

Vol. 4267: A. Helmy, B. Jennings, L. Murphy, T. Pfeifer (Eds.), Autonomic Management of Mobile Multimedia Services. XIII, 257 pages. 2006.

Vol. 4266: H. Yoshiura, K. Sakurai, K. Rannenberg, Y. Murayama, S. Kawamura (Eds.), Advances in Information and Computer Security. XIII, 438 pages. 2006.

Vol. 4265: N. Lavrač, L. Todorovski, K.P. Jantke (Eds.), Discovery Science. XIV, 384 pages. 2006. (Sublibrary LNAI).

Vol. 4264: J.L. Balcázar, P.M. Long, F. Stephan (Eds.), Algorithmic Learning Theory. XIII, 393 pages. 2006. (Sublibrary LNAI).

Vol. 4263: A. Levi, E. Savas, H. Yenigün, S. Balcisoy, Y. Saygin (Eds.), Computer and Information Sciences – ISCIS 2006. XXIII, 1084 pages. 2006.

Vol. 4261: Y. Zhuang, S. Yang, Y. Rui, Q. He (Eds.), Advances in Multimedia Information Processing - PCM 2006. XXII, 1040 pages. 2006.

Vol. 4260: Z. Liu, J. He (Eds.), Formal Methods and Software Engineering. XII, 778 pages. 2006.

Vol. 4259: S. Greco, Y. Hata, S. Hirano, M. Inuiguchi, S. Miyamoto, H.S. Nguyen, R. Słowiński (Eds.), Rough Sets and Current Trends in Computing. XXII, 951 pages. 2006. (Sublibrary LNAI).

Vol. 4257: I. Richardson, P. Runeson, R. Messnarz (Eds.), Software Process Improvement. XI, 219 pages. 2006.

Vol. 4256: L. Feng, G. Wang, C. Zeng, R. Huang (Eds.), Web Information Systems – WISE 2006 Workshops. XIV, 320 pages. 2006.

Vol. 4255: K. Aberer, Z. Peng, E.A. Rundensteiner, Y. Zhang, X. Li (Eds.), Web Information Systems – WISE 2006. XIV, 563 pages. 2006.

Vol. 4254: T. Grust, H. Höpfner, A. Illarramendi, S. Jablonski, M. Mesiti, S. Müller, P.-L. Patranjan, K.-U. Sattler, M. Spiliopoulou (Eds.), Current Trends in Database Technology – EDBT 2006. XXXI, 932 pages. 2006.

Vol. 4253: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part III. XXXII, 1301 pages. 2006. (Sublibrary LNAI).

Vol. 4252: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part II. XXXIII, 1335 pages. 2006. (Sublibrary LNAI).

Vol. 4251: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part I. LXVI, 1297 pages. 2006. (Sublibrary LNAI).

Vol. 4249: L. Goubin, M. Matsui (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2006. XII, 462 pages. 2006.

Vol. 4248: S. Staab, V. Svátek (Eds.), Managing Knowledge in a World of Networks. XIV, 400 pages. 2006. (Sublibrary LNAI).

Vol. 4247: T.-D. Wang, X. Li, S.-H. Chen, X. Wang, H. Abbass, H. Iba, G. Chen, X. Yao (Eds.), Simulated Evolution and Learning. XXI, 940 pages. 2006.

Vol. 4246: M. Hermann, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning. XIII, 588 pages. 2006. (Sublibrary LNAI).

Vol. 4245: A. Kuba, L.G. Nyúl, K. Palágyi (Eds.), Discrete Geometry for Computer Imagery. XIII, 688 pages. 2006.

Vol. 4244: S. Spaccapietra (Ed.), Journal on Data Semantics VII. XI, 267 pages. 2006.

Vol. 4243: T. Yakhno, E.J. Neuhold (Eds.), Advances in Information Systems. XIII, 420 pages. 2006.

Vol. 4242: A. Rashid, M. Aksit (Eds.), Transactions on Aspect-Oriented Software Development II. IX, 289 pages. 2006.

Vol. 4241: R.R. Beichel, M. Sonka (Eds.), Computer Vision Approaches to Medical Image Analysis. XI, 262 pages. 2006.

Vol. 4239: H.Y. Youn, M. Kim, H. Morikawa (Eds.), Ubiquitous Computing Systems. XVI, 548 pages. 2006.

Vol. 4238: Y.-T. Kim, M. Takano (Eds.), Management of Convergence Networks and Services. XVIII, 605 pages. 2006.

Vol. 4237: H. Leitold, E. Markatos (Eds.), Communications and Multimedia Security. XII, 253 pages. 2006.

Vol. 4236: L. Breveglieri, I. Koren, D. Naccache, J.-P. Seifert (Eds.), Fault Diagnosis and Tolerance in Cryptography. XIII, 253 pages. 2006.

Vol. 4234: I. King, J. Wang, L. Chan, D. Wang (Eds.), Neural Information Processing, Part III. XXII, 1227 pages. 2006.

Vol. 4233: I. King, J. Wang, L. Chan, D. Wang (Eds.), Neural Information Processing, Part II. XXII, 1203 pages. 2006.

Vol. 4232: I. King, J. Wang, L. Chan, D. Wang (Eds.), Neural Information Processing, Part I. XLVI, 1153 pages. 2006.

Vol. 4231: J. F. Roddick, R. Benjamins, S. Si-Saïd Cherfi, R. Chiang, C. Claramunt, R. Elmasri, F. Grandi, H. Han, M. Hepp, M. Hepp, M. Lytras, V.B. Mišić, G. Poels, I.-Y. Song, J. Trujillo, C. Vangenot (Eds.), Advances in Conceptual Modeling - Theory and Practice. XXII, 456 pages. 2006.

Vol. 4230: C. Priami, A. Ingólfsdóttir, B. Mishra, H.R. Nielson (Eds.), Transactions on Computational Systems Biology VII. VII, 185 pages. 2006. (Sublibrary LNBI).

Vol. 4229: E. Najm, J.F. Pradat-Peyre, V.V. Donzeau-Gouge (Eds.), Formal Techniques for Networked and Distributed Systems - FORTE 2006. X, 486 pages. 2006.

Vol. 4228: D.E. Lightfoot, C.A. Szyperski (Eds.), Modular Programming Languages. X, 415 pages. 2006.

Vol. 4227: W. Nejdl, K. Tochtermann (Eds.), Innovative Approaches for Learning and Knowledge Sharing. XVII, 721 pages. 2006.

Vol. 4226: R.T. Mittermeir (Ed.), Informatics Education – The Bridge between Using and Understanding Computers. XVII, 319 pages. 2006.

Vol. 4225: J.F. Martínez-Trinidad, J.A. Carrasco Ochoa, J. Kittler (Eds.), Progress in Pattern Recognition, Image Analysis and Applications. XIX, 995 pages. 2006.

Vol. 4224: E. Corchado, H. Yin, V. Botti, C. Fyfe (Eds.), Intelligent Data Engineering and Automated Learning – IDEAL 2006. XXVII, 1447 pages. 2006.

Vol. 4223: L. Wang, L. Jiao, G. Shi, X. Li, J. Liu (Eds.), Fuzzy Systems and Knowledge Discovery. XXVIII, 1335 pages. 2006. (Sublibrary LNAI).

Vol. 4222: L. Jiao, L. Wang, X. Gao, J. Liu, F. Wu (Eds.), Advances in Natural Computation, Part II. XLII, 998 pages. 2006.

Vol. 4221: L. Jiao, L. Wang, X. Gao, J. Liu, F. Wu (Eds.), Advances in Natural Computation, Part I. XLI, 992 pages. 2006.

Vol. 4219: D. Zamboni, C. Kruegel (Eds.), Recent Advances in Intrusion Detection. XII, 331 pages. 2006.

Vol. 4218: S. Graf, W. Zhang (Eds.), Automated Technology for Verification and Analysis. XIV, 540 pages. 2006.

Vol. 4217: P. Cuenca, L. Orozco-Barbosa (Eds.), Personal Wireless Communications. XV, 532 pages. 2006.

Vol. 4216: M.R. Berthold, R. Glen, I. Fischer (Eds.), Computational Life Sciences II. XIII, 269 pages. 2006. (Sublibrary LNBI).

Vol. 4215: D.W. Embley, A. Olivé, S. Ram (Eds.), Conceptual Modeling - ER 2006. XVI, 590 pages. 2006.

Vol. 4213: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Knowledge Discovery in Databases: PKDD 2006. XXII, 660 pages. 2006. (Sublibrary LNAI).

Vol. 4212: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Machine Learning: ECML 2006. XXIII, 851 pages. 2006. (Sublibrary LNAI).

Vol. 4211: P. Vogt, Y. Sugita, E. Tuci, C. Nehaniv (Eds.), Symbol Grounding and Beyond. VIII, 237 pages. 2006. (Sublibrary LNAI).

Vol. 4210: C. Priami (Ed.), Computational Methods in Systems Biology. X, 323 pages. 2006. (Sublibrary LNBI).

Vol. 4209: F. Crestani, P. Ferragina, M. Sanderson (Eds.), String Processing and Information Retrieval. XIV, 367 pages. 2006.

Vol. 4208: M. Gerndt, D. Kranzlmüller (Eds.), High Performance Computing and Communications. XXII, 938 pages. 2006.

Vol. 4207: Z. Ésik (Ed.), Computer Science Logic. XII, 627 pages. 2006.

Vol. 4206: P. Dourish, A. Friday (Eds.), UbiComp 2006: Ubiquitous Computing. XIX, 526 pages. 2006.

Vol. 4205: G. Bourque, N. El-Mabrouk (Eds.), Comparative Genomics. X, 231 pages. 2006. (Sublibrary LNBI).

Table of Contents

Interaction and Coordination of Tools for Structured Data

Farhad Arbab[1,2] and Joost N. Kok[2]

[1] Center for Mathematics and Computer Science (CWI)
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
farhad@cwi.nl
[2] Leiden Institute for Advanced Computer Science (LIACS), Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
{farhad, joost}@liacs.nl

Abstract. This paper has an introductory nature. It sets the scene for the three papers in the thematic session "Structured Data Tools". We discuss interaction and coordination in general and look more specifically at the XML data format and the dataflow driven coordination language Reo.

1 Introduction

The size, speed, capacity, and price of computers have all dramatically changed in the last half-century. Still more dramatic are the subtle changes of the society's perception of what computers are and what they can, should, and are expected to do. Clearly, this change of perception would not have been possible without the technological advances that reduced the size and price of computers, while increasing their speed and capacity. Nevertheless, the social impact of this change of perception and its feedback influence on the advancement of computer science and technology, are too significant to be regarded as mere by-products of those technological advances.[1]

The social perception of what computers are (to be used for) has evolved through three phases:

1. Computers as fast number crunchers
2. Computers as symbol manipulators
3. Computers as mediators and facilitators of interaction

The general faster-cheaper-smaller trend of computer technology aside, two specific transformations marked the above phase-transitions. The advent of fast, large main memory and mass-storage devices suitable to store and access the significantly more voluminous amounts of data required for non-numerical symbol manipulation made symbolic computation possible. The watershed that set forth the second transition was the availability of affordable computers and digital telecommunication that together fueled the explosion of the Internet.

[1] See [3] for an expanded version of this discussion.

T. Margaria and B. Steffen (Eds.): ISoLA 2004, LNCS 4313, pp. 1–8, 2006.

We are still at the tail-end of the second transition (from symbolic computation to interaction) and trying to come to terms with its full implications on computer science and technology. This involves revisiting some established areas, such as concurrency, from a new perspective, and leads to a specific field of study concerned with theories and models for coordination of interactive concurrent computations. Moreover, it presents new challenges in software engineering to address coarse-grain reuse in component based software and to tackle architectures of complex systems whose organization and composition must dynamically change, e.g., to accommodate mobility, or evolve and be reconfigured to adapt to short- as well as long-term changes in their environment.

Two key concepts emerge as core concerns: *interaction* and *coordination.* While researchers have worked on both individually in the past, we propose that their combination deserves still more serious systematic study because it offers insight into new approaches to coordination of cooperation of interacting components that comprise such complex systems. In our thematic session we concentrate on interaction of components through dataflow. This has two aspects: the nature of data such that they are suited for interaction *and* the coordination of components. First we put interaction and coordination in a context and later we focus on the XML format and the Reo coordination language, being the topics of the papers that belong to this thematic session.

The overview of the rest of the paper is as follows. We discuss interaction and coordination in two separate sections. In these two sections we first treat general issues and then focus on the data aspects. In the final section, we briefly discuss the three papers of the thematic session.

2 Interaction

In the real world, computer systems and databases contain data in incompatible formats. However, interaction implies data exchange between such systems and this exchange has been one of the most time-consuming challenges for developers. Converting the data to a common format can greatly reduce this complexity and create data that can be read by many different types of applications.

Traditional database systems rely on an old model: the relational data model. When it was proposed in the early 1970's by Codd, the relational model generated a revolution in data management. In this model data are represented as relations in first-order structures and queries as first-order logic formulas. It enabled researchers and implementors to separate the logical aspect of the data from its physical implementation. Thirty years of research and development followed, and they led to today's relational database systems.

The advent of the Internet led to a variety of new generation data management applications for which the relational model is no longer suited. For instance, data are now frequently accessed through the Web, is distributed over several sources, and resides there in various formats (e.g., HTML, XML, text files, relational). To accommodate all forms of data and access to them, the database research community has introduced the "semi-structured data model", where data are

self-describing, irregular, and graph-like. The new model captures naturally Web data, such as HTML, XML, or other application specific formats like trees and molecules. (See Foundations of Semistructured Data, Dagstuhl Seminar [10].) Data are typically subject to frequent changes in both structure, contents, and interfacing.

As a general trend, data have become more structured: examples of structured data include graphs, trees, molecules and XML documents. In many application areas very large quantities of structured data are generated. Handling these large quantities of structured data requires rigorous data tools. Data tools are tools for handling and expanding the use of such data, including those to acquire, store, organize, archive and analyze data. It is important that these tools are efficient, correct, flexible and reliable. Examples include tools for data conversion, data integration, data security and data mining. Typical areas of application are in health-care, bioscience, financial sector and e-commerce, interoperability and integration.

In our thematic session we focus on XML which is representative of issues in structured data tools. The features of XML that make it particularly appropriate for data transfer are (see [16]):

1. XML uses plain text files.
2. XML can represent records, lists and trees.
3. XML is platform-independent.
4. the XML format is self-documenting in that it describes the structure and field names as well as the syntax for specific values.
5. XML is used as the format for document storage and processing, its hierarchical structure being suitable for most types of documents.
6. XML has already been in use (as SGML) for longer than a decade, and is very popular by itself, with extensive available experience and software.

The XML format is not only a common format, but it also provides a basis for the coordination of systems.

3 Coordination

Coordination languages, models, and systems constitute a recent field of study in programming and software systems, with the goal of finding solutions to the problem of managing the interaction among concurrent programs. Coordination can be defined as the study of the dynamic topologies of interactions, and the construction of protocols to realize such topologies that ensure well-behavedness. Analogous to the way in which topology abstracts away the metric details of geometry and focuses on the invariant properties of (seemingly very different) shapes, coordination abstracts away the details of computation, and focuses on the invariant properties of (seemingly very different) programs. As such, coordination focuses on patterns that specifically deal with interaction.

The inability to deal with the cooperation model of a concurrent application in an explicit form contributes to the difficulty of developing working concurrent applications that contain large numbers of active entities with non-trivial

cooperation protocols. In spite of the fact that the implementation of a complex protocol is often the most difficult and error prone part of an application development effort, the end result is typically not recognized as a "commodity" in its own right, because the protocol is only implicit in the behavior of the rest of the concurrent software. This makes maintenance and modification of the cooperation protocols of concurrent applications much more difficult than necessary, and their reuse next to impossible. Coordination languages can be thought of as the linguistic counterpart of the ad hoc platforms that offer middle-ware support for software composition.

Coordination languages are most relevant specifically in the context of open systems, where their participants are not fixed at the outset. Coordination is also relevant in design, development, debugging, maintenance, and reuse of all concurrent systems, where it addresses a number of important software engineering issues. The current interest in constructing applications out of independent software components necessitates specific attention to the so-called *glue-code*. The purpose of the glue-code is to compose a set of components by filling the significant interface gaps that naturally arise among them, simply because they are not (supposed to be) tailor-made to work with one another. Using components, thus, means understanding how they individually interact with their environment, and specifying how they should engage in mutual, cooperative interactions in order for their composition to behave as a coordinated whole. Many of the core issues involved in component composition have already been identified and studied as key concerns in work on coordination. Coordination models and languages address such key issues in Component Based Software Engineering as specification, interaction, and dynamic composition of components. Specifically, *exogenous* coordination models (discussed later in this section) and languages provide a very promising basis for the development of effective glue-code languages because they enable third-party entities to wield coordination control over the interaction behavior of mutually anonymous entities involved in a collaboration activity from outside of those participating entities.

One of the best known coordination languages is Linda [9,13], which is based on the notion of a shared tuple space. The tuple space of Linda is a centrally managed resource and contains all pieces of information that processes wish to communicate with each other. Linda processes can be written in any language augmented with Linda primitives. There are only four primitives provided by Linda, each of which associatively operates on a single tuple in the tuple space. The primitive **in** searches the tuple space for a matching tuple and deletes it; **out** adds a tuple to the tuple space; **read** searches for a matching tuple in the tuple space; and **eval** starts an active tuple (i.e., a process). Numerous other coordination models and language extensions, e.g., JavaSpace of Jini [12,14], are based on Linda-like models.

Besides the "generative tuple space" of Linda, a number of other interesting models have been proposed and used to support coordination languages and systems. Examples include various forms of "parallel multiset rewriting" or "chemical reactions" as in Gamma [6], models with explicit support for coordinators

as in Manifold [4,8], "software bus" as in ToolBus [7], and a calculus of generalized-channel composition as in Reo [2]. A significant number of these models and languages are based on a few common notions, such as pattern-based, associative communication [1], to complement the name-oriented, data-based communication of traditional languages for parallel programming. See [15] for a comprehensive survey of coordination models and languages.

Some of the important properties of different coordination languages, models, and systems become clear when we classify them along the following three dimensions: focus of coordination, locus of coordination, and modus of coordination. Although a detailed description of most individual coordination models and languages is beyond the scope of our interest in this paper, an overview of the dimensions of this classification helps to clarify at a more abstract level the issues they address, and thus the concerns of coordination as a field.

Focus of coordination refers to the aspect of the applications that a coordination model, language, or system emphasizes as its primary concern. Significant aspects used by various models as their focus of coordination include data, control, and dataflow, yielding *data-oriented*, *control-oriented*, and *dataflow-oriented* families of coordination models, languages, and systems.

For instance, Linda uses a data-oriented coordination model, whereas Manifold is a control-oriented coordination language. The activity in a data-oriented application tends to center around a substantial shared body of data; the application is essentially concerned with what happens to the data. Examples include database and transaction systems such as banking and airline reservation applications. On the other hand, the activity in a control-oriented application tends to center around processing or flow of control and, often, the very notion of the data, as such, simply does not exist; such an application is essentially described as a collection of activities that genuinely consume their input data, and subsequently produce, remember, and transform "new data" that they generate by themselves. Examples include applications that involve work-flow in organizations, and multi-phase applications where the content, format, and/or modality of information substantially changes from one phase to the next.

Dataflow-oriented models, such as Reo, use the flow of data as the only (or at least the primary) control mechanism. Unlike data-oriented models, dataflow models are oblivious to the actual content, type, or structure of data and are instead concerned with the flow of data from their sources to their destinations. Unlike control-oriented models, events that trigger state transitions are limited to only those that arise out of the flow of data.

Locus of coordination refers to where coordination activity takes place, classifying coordination models and languages as *endogenous* or *exogenous*. Endogenous models and languages, such as Linda, provide primitives that must be incorporated *within* a computation for its coordination. In applications that use such models, primitives that affect the coordination of each module are inside the module itself. In contrast, exogenous models and languages, such as Manifold and Reo, provide primitives that support coordination of entities from *without*.

In applications that use exogenous models primitives that affect the coordination of each module are outside the module itself.

Modus of coordination refers to how coordination is carried out in a model or language: how the coordination rules of an application are defined and enforced. The repertoire of coordination rules supported by a coordination model or language can be very different in its nature than that of another. Some, e.g., Linda, Manifold, and Reo, provide primitives for building coordination rules. Others propose rule-based languages where rules act as trigger conditions for action or as constraints on the behavior of active agents to coordinate them in a system. One way or the other, coordination rules provide a level of abstraction which hides much of the complexity of coordination activity from programmers. Models that use more declarative coordination rules can support increased reasoning power.

In our thematic session we consider the Reo language. Reo is a channel-based exogenous coordination model wherein complex coordinators, called *connectors*, are compositionally built out of simpler ones. The simplest connectors in Reo are a set of *channels* with well-defined behavior supplied by users [2]. The emphasis in Reo is on connectors, their behavior, and their composition, not on the entities that connect, communicate, and cooperate through them. The behavior of every connector in Reo imposes a specific coordination pattern on the entities that perform normal I/O operations through that connector, without the knowledge of those entities. This makes Reo a powerful "glue language" for compositional construction of connectors to combine component instances into a software system and exogenously orchestrate their mutual interactions.

Reo's notion of components and connectors is depicted in Figure 1, where component instances are represented as boxes, channels as straight lines, and connectors are delineated by dashed lines. Each connector in Reo is, in turn, constructed compositionally out of simpler connectors, which are ultimately composed out of primitive channels.

For instance, the connector in Figure 1.a may in fact be a flow-regulator (if its three constituent channels are of the right type as described under write-cue regulator, below, and shown in Figure 2.a). Figure 1.a would then represent a system composed out of two *writer* component instances (`C1` and `C3`), plus a *reader* component instance (`C2`), glued together by our flow-regulator connector. Every component instance performs its I/O operations following its own timing

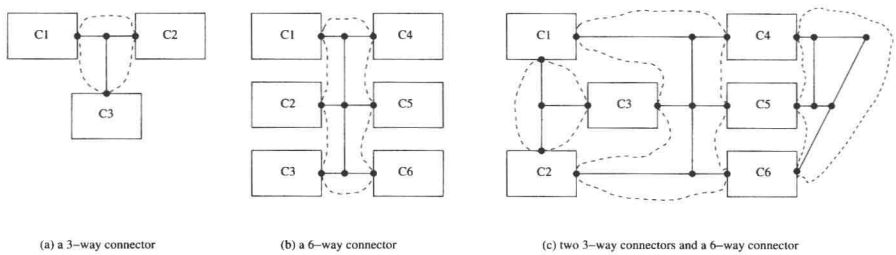

Fig. 1. Connectors and component composition

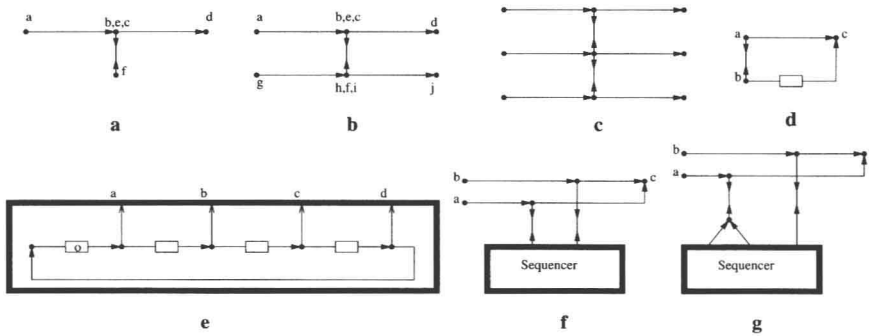

Fig. 2. Examples of connector circuits in Reo

and logic, independently of the others. None of these component instances is aware of the existence of the others, the specific connector used to glue it with the rest, or even of its own role in the composite system. Nevertheless, the protocol imposed by our flow-regulator glue code (see [2] and [5]) ensures that a data item passes from `C1` to `C2` only whenever `C3` writes a data item (whose actual value is ignored): the "tokens" written by `C3`, thus, serve as cues to regulate the flow of data items from `C1` to `C2`. The behavior of the connector, in turn, is independent of the components it connects: without their knowledge, it imposes a coordination pattern among `C1`, `C2`, and `C3` that regulates the precise timing and/or the volume of the data items that pass from `C1` to `C2`, according to the timing and/or the volume of tokens produced by `C3`. The other connectors in Figure 1 implement more complex coordination patterns.

Figure 2.a shows a write-cue regulator connector circuit built out of two synchronous channels and a synchronous-drain, whose behavior is analogous to that of a transistor. Figures 2.b and c show this transistor used to construct more complex Reo connector circuits, specifically, two barrier-synchronizer circuits for, respectively, two- and three-pairs of readers and writers. Figure 2.d shows a Reo connector circuit for an alternator. Figure 2.e shows a sequencer circuit, and Figures 2.f and g show more complex alternator circuits that use this sequencer as a component. See the first paper in this thematic session for a brief overview of Reo and [2] for more details on Reo channels and these and other examples.

There are three papers in the thematic session: the first presents an application of the Reo coordination language; the second paper describes a structured data transformation tool; and the third paper discusses the application of this tool in enterprise architectures. We discuss them in turn.

- *Modeling Coordination in Biological Systems.* The Reo coordination language is used for modeling in systems biology. Various forms of coordination in living cells are discussed and metabolization of galactose is modeled.
- *A Rule Markup Language and its application to UML.* A data transformation tool for XML, called RML, is introduced. RML can be use for rule-based transformations of XML documents. RML rules are stated in XML and are