LNCS 3199

Henk Schepers (Ed.)

Software and Compilers for Embedded Systems

8th International Workshop, SCOPES 2004
Amsterdam, The Netherlands, September 2004
Proceedings

Springer

Henk Schepers (Ed.)

Software and Compilers for Embedded Systems

8th International Workshop, SCOPES 2004
Amsterdam, The Netherlands, September 2-3, 2004
Proceedings

Springer

Volume Editor

Henk Schepers
Philips Research
Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
E-mail: Henk.Schepers@philips.com

Library of Congress Control Number: 3540230351

CR Subject Classification (1998): D.3, D.4, D.2, D.1, C.3, C.2

ISSN 0302-9743
ISBN 3-540-23035-1 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 11313403 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3199

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

Preface

This volume contains the proceedings of the 8th International Workshop on Software and Compilers for Embedded Systems (SCOPES 2004) held in Amsterdam, The Netherlands, on September 2 and 3, 2004. Initially, the workshop was referred to as the International Workshop on Code Generation for Embedded Systems. The first took place in 1994 in Schloß Dagstuhl, Germany. From its beginnings, the intention of the organizers has been to create an interactive atmosphere in which the participants can discuss and profit from the assembly of international experts in the field.

The name SCOPES has been used since the fourth edition in St. Goar, Germany, in 1999 when the scope of the workshop was extended to also cover general issues in embedded software design. Since then SCOPES has been held again in St. Goar in 2001; Berlin, Germany in 2002; Vienna, Austria in 2003; and now in Amsterdam, The Netherlands.

In response to the call for papers, almost 50 very strong papers were submitted from all over the world. All submitted papers were reviewed by at least three experts to ensure the quality of the workshop. In the end, the program committee selected 17 papers for presentation at the workshop. These papers are divided into the following categories: application-specific (co)design, system and application synthesis, data flow analysis, data partitioning, task scheduling and code generation.

In addition to the selected contributions, the keynote address was delivered by Mike Uhler from MIPS Technologies. An abstract of his talk is also included in this volume.

I want to thank all the authors for submitting their papers, and the program committee and the referees for carefully reviewing them. I thank Harry Hendrix and Jan van Nijnatten for supporting the review process and for compiling the proceedings. Finally, I thank Marianne Dalmolen for maintaining the web site and the local organization.

June 2004 Henk Schepers

Organization

SCOPES 2004 was organized by ACE Associated Compiler Experts and Philips Research in cooperation with EDAA.

Committee

General Chairs	Marco Roodzant, ACE (Associated Compiler Experts) Henk Schepers, Philips Research
Local Organization	Marianne Dalmolen, ACE (Associated Compiler Experts)
Program Committee	Uwe Assmann, Linköpings Universitet Lex Augusteijn, Silicon Hive Shuvra Bhattacharyya, University of Maryland Albert Cohen, INRIA Alex Dean, North Carolina State University Nikil Dutt, University of California at Irvine Antonio González, Universitat Politècnica de Catalunya & Intel David Gregg, Trinity College Dublin Rajiv Gupta, University of Arizona Seongsoo Hong, Seoul National University Nigel Horspool, University of Victoria Masaharu Imai, Osaka University Daniel Kästner, AbsInt Andreas Krall, Technische Universität Wien Rainer Leupers, RWTH Aachen Annie Liu, SUNY Stony Brook Peter Marwedel, Universität Dortmund Tatsuo Nakajima, Waseda University Alex Nicolau, University of California at Irvine Yunheung Paek, Seoul National University Santosh Pande, Georgia Institute of Technology Robert Pasko, IMEC Sreeranga Rajan, Fujitsu Miguel Santana, STMicroelectronics Hans van Someren, ACE (Associated Compiler Experts) Hiroyuki Tomiyama, Nagoya University Bernard Wess, Technische Universität Wien David Whalley, Florida State University

Referees

Christophe Alias
Cédric Bastoul
Marcel Beemster
Valerie Bertin
Doo-san Cho
Yulwon Cho
Junshik Choi
Jan van Dongen
Heiko Falk
Liam Fitzpatrick
Carlos Garcia
Laurent Gerard
Leszek Holenderski
Jan Hoogerbrugge
Martien de Jong
Saehwa Kim
Arvind Krishnaswamy
Fernando Latorre
Bengu Li
Klas Lindberg
Grigorios Magklis
Hyunok Oh
Bryan Olivier
Emre Ozer
Serge De Paoli
Jiyong Park
Sang-hyun Park
Greg Parsons
Zane Purvis
Robert Pyka
Frederic Riss
Ruben van Royen
Sergej Schwenk
Jaewon Seo
Aviral Shrivastava
Yoshinori Takeuchi
Sriraman Tallam
Hiroaki Tanaka
Thomas Thery
Osman Unsal
Xavier Vera
Manish Verma
Jens Wagner
Lars Wehmeyer
Sami Yehia
Thomas Zeitlhofer
Xiangyu Zhang
Xiaotong Zhuang

Lecture Notes in Computer Science

For information about Vols. 1–3099

please contact your bookseller or Springer

Vol. 3232: R. Heery, L. Lyon (Eds.), Research and Advanced Technology for Digital Libraries. XV, 528 pages. 2004.

Vol. 3220: J.C. Lester, R.M. Vicari, F. Paraguaçu (Eds.), Intelligent Tutoring Systems. XXI, 920 pages. 2004.

Vol. 3208: H.J. Ohlbach, S. Schaffert (Eds.), Principles and Practice of Semantic Web Reasoning. VII, 165 pages. 2004.

Vol. 3207: L.T. Jang, M. Guo, G.R. Gao, N.K. Jha, Embedded and Ubiquitous Computing. XX, 1116 pages. 2004.

Vol. 3206: P. Sojka, I. Kopecek, K. Pala (Eds.), Text, Speech and Dialogue. XIII, 667 pages. 2004. (Subseries LNAI).

Vol. 3205: N. Davies, E. Mynatt, I. Siio (Eds.), UbiComp 2004: Ubiquitous Computing. XVI, 452 pages. 2004.

Vol. 3203: J. Becker, M. Platzner, S. Vernalde (Eds.), Field Programmable Logic and Application. XXX, 1198 pages. 2004.

Vol. 3199: H. Schepers (Ed.), Software and Compilers for Embedded Systems. X, 259 pages. 2004.

Vol. 3198: G.-J. de Vreede, L.A. Guerrero, G. Marín Raventós (Eds.), Groupware: Design, Implementation and Use. XI, 378 pages. 2004.

Vol. 3194: R. Camacho, R. King, A. Srinivasan (Eds.), Inductive Logic Programming. XI, 361 pages. 2004. (Subseries LNAI).

Vol. 3193: P. Samarati, P. Ryan, D. Gollmann, R. Molva (Eds.), Computer Security – ESORICS 2004. X, 457 pages. 2004.

Vol. 3192: C. Bussler, D. Fensel (Eds.), Artificial Intelligence: Methodology, Systems, and Applications. XIII, 522 pages. 2004. (Subseries LNAI).

Vol. 3189: P.-C. Yew, J. Xue (Eds.), Advances in Computer Systems Architecture. XVII, 598 pages. 2004.

Vol. 3186: Z. Bellahsène, T. Milo, M. Rys, D. Suciu, R. Unland (Eds.), Database and XML Technologies. X, 235 pages. 2004.

Vol. 3184: S. Katsikas, J. Lopez, G. Pernul (Eds.), Trust and Privacy in Digital Business. XI, 299 pages. 2004.

Vol. 3183: R. Traunmüller (Ed.), Electronic Government. XIX, 583 pages. 2004.

Vol. 3182: K. Bauknecht, M. Bichler, B. Pröll (Eds.), E-Commerce and Web Technologies. XI, 370 pages. 2004.

Vol. 3181: Y. Kambayashi, M. Mohania, W. Wöß (Eds.), Data Warehousing and Knowledge Discovery. XIV, 412 pages. 2004.

Vol. 3180: F. Galindo, M. Takizawa, R. Traunmüller (Eds.), Database and Expert Systems Applications. XXI, 972 pages. 2004.

Vol. 3179: F.J. Perales, B.A. Draper (Eds.), Articulated Motion and Deformable Objects. XI, 270 pages. 2004.

Vol. 3178: W. Jonker, M. Petkovic (Eds.), Secure Data Management. VIII, 219 pages. 2004.

Vol. 3177: Z.R. Yang, H. Yin, R. Everson (Eds.), Intelligent Data Engineering and Automated Learning – IDEAL 2004. XVIII, 852 pages. 2004.

Vol. 3175: C.E. Rasmussen, H.H. Bülthoff, B. Schölkopf, M.A. Giese (Eds.), Pattern Recognition. XVIII, 581 pages. 2004.

Vol. 3174: F. Yin, J. Wang, C. Guo (Eds.), Advances in Neural Networks - ISNN 2004. XXXV, 1021 pages. 2004.

Vol. 3172: M. Dorigo, M. Birattari, C. Blum, L. M.Gambardella, F. Mondada, T. Stützle (Eds.), Ant Colony, Optimization and Swarm Intelligence. XII, 434 pages. 2004.

Vol. 3170: P. Gardner, N. Yoshida (Eds.), CONCUR 2004 - Concurrency Theory. XIII, 529 pages. 2004.

Vol. 3166: M. Rauterberg (Ed.), Entertainment Computing – ICEC 2004. XXIII, 617 pages. 2004.

Vol. 3163: S. Marinai, A. Dengel (Eds.), Document Analysis Systems VI. XII, 564 pages. 2004.

Vol. 3162: R. Downey, M. Fellows, F. Dehne (Eds.), Parameterized and Exact Computation. X, 293 pages. 2004.

Vol. 3160: S. Brewster, M. Dunlop (Eds.), Mobile Human-Computer Interaction – MobileHCI 2004. XVIII, 541 pages. 2004.

Vol. 3159: U. Visser, Intelligent Information Integration for the Semantic Web. XIV, 150 pages. 2004. (Subseries LNAI).

Vol. 3158: I. Nikolaidis, M. Barbeau, E. Kranakis (Eds.), Ad-Hoc, Mobile, and Wireless Networks. IX, 344 pages. 2004.

Vol. 3157: C. Zhang, H. W. Guesgen, W.K. Yeap (Eds.), PRICAI 2004: Trends in Artificial Intelligence. XX, 1023 pages. 2004. (Subseries LNAI).

Vol. 3156: M. Joye, J.-J. Quisquater (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2004. XIII, 455 pages. 2004.

Vol. 3155: P. Funk, P.A. González Calero (Eds.), Advances in Case-Based Reasoning. XIII, 822 pages. 2004. (Subseries LNAI).

Vol. 3154: R.L. Nord (Ed.), Software Product Lines. XIV, 334 pages. 2004.

Vol. 3153: J. Fiala, V. Koubek, J. Kratochvíl (Eds.), Mathematical Foundations of Computer Science 2004. XIV, 902 pages. 2004.

Vol. 3152: M. Franklin (Ed.), Advances in Cryptology – CRYPTO 2004. XI, 579 pages. 2004.

Vol. 3150: G.-Z. Yang, T. Jiang (Eds.), Medical Imaging and Augmented Reality. XII, 378 pages. 2004.

Vol. 3149: M. Danelutto, M. Vanneschi, D. Laforenza (Eds.), Euro-Par 2004 Parallel Processing. XXXIV, 1081 pages. 2004.

Vol. 3148: R. Giacobazzi (Ed.), Static Analysis. XI, 393 pages. 2004.

Vol. 3146: P. Érdi, A. Esposito, M. Marinaro, S. Scarpetta (Eds.), Computational Neuroscience: Cortical Dynamics. XI, 161 pages. 2004.

Vol. 3144: M. Papatriantafilou, P. Hunel (Eds.), Principles of Distributed Systems. XI, 246 pages. 2004.

Vol. 3143: W. Liu, Y. Shi, Q. Li (Eds.), Advances in Web-Based Learning – ICWL 2004. XIV, 459 pages. 2004.

Vol. 3142: J. Diaz, J. Karhumäki, A. Lepistö, D. Sannella (Eds.), Automata, Languages and Programming. XIX, 1253 pages. 2004.

Vol. 3140: N. Koch, P. Fraternali, M. Wirsing (Eds.), Web Engineering. XXI, 623 pages. 2004.

Vol. 3139: F. Iida, R. Pfeifer, L. Steels, Y. Kuniyoshi (Eds.), Embodied Artificial Intelligence. IX, 331 pages. 2004. (Subseries LNAI).

Vol. 3138: A. Fred, T. Caelli, R.P.W. Duin, A. Campilho, D.d. Ridder (Eds.), Structural, Syntactic, and Statistical Pattern Recognition. XXII, 1168 pages. 2004.

Vol. 3137: P. De Bra, W. Nejdl (Eds.), Adaptive Hypermedia and Adaptive Web-Based Systems. XIV, 442 pages. 2004.

Vol. 3136: F. Meziane, E. Métais (Eds.), Natural Language Processing and Information Systems. XII, 436 pages. 2004.

Vol. 3134: C. Zannier, H. Erdogmus, L. Lindstrom (Eds.), Extreme Programming and Agile Methods - XP/Agile Universe 2004. XIV, 233 pages. 2004.

Vol. 3133: A.D. Pimentel, S. Vassiliadis (Eds.), Computer Systems: Architectures, Modeling, and Simulation. XIII, 562 pages. 2004.

Vol. 3132: B. Demoen, V. Lifschitz (Eds.), Logic Programming. XII, 480 pages. 2004.

Vol. 3131: V. Torra, Y. Narukawa (Eds.), Modeling Decisions for Artificial Intelligence. XI, 327 pages. 2004. (Subseries LNAI).

Vol. 3130: A. Syropoulos, K. Berry, Y. Haralambous, B. Hughes, S. Peter, J. Plaice (Eds.), TeX, XML, and Digital Typography. VIII, 265 pages. 2004.

Vol. 3129: Q. Li, G. Wang, L. Feng (Eds.), Advances in Web-Age Information Management. XVII, 753 pages. 2004.

Vol. 3128: D. Asonov (Ed.), Querying Databases Privately. IX, 115 pages. 2004.

Vol. 3127: K.E. Wolff, H.D. Pfeiffer, H.S. Delugach (Eds.), Conceptual Structures at Work. XI, 403 pages. 2004. (Subseries LNAI).

Vol. 3126: P. Dini, P. Lorenz, J.N.d. Souza (Eds.), Service Assurance with Partial and Intermittent Resources. XI, 312 pages. 2004.

Vol. 3125: D. Kozen (Ed.), Mathematics of Program Construction. X, 401 pages. 2004.

Vol. 3124: J.N. de Souza, P. Dini, P. Lorenz (Eds.), Telecommunications and Networking - ICT 2004. XXVI, 1390 pages. 2004.

Vol. 3123: A. Belz, R. Evans, P. Piwek (Eds.), Natural Language Generation. X, 219 pages. 2004. (Subseries LNAI).

Vol. 3122: K. Jansen, S. Khanna, J.D.P. Rolim, D. Ron (Eds.), Approximation, Randomization, and Combinatorial Optimization. IX, 428 pages. 2004.

Vol. 3121: S. Nikoletseas, J.D.P. Rolim (Eds.), Algorithmic Aspects of Wireless Sensor Networks. X, 201 pages. 2004.

Vol. 3120: J. Shawe-Taylor, Y. Singer (Eds.), Learning Theory. X, 648 pages. 2004. (Subseries LNAI).

Vol. 3118: K. Miesenberger, J. Klaus, W. Zagler, D. Burger (Eds.), Computer Helping People with Special Needs. XXIII, 1191 pages. 2004.

Vol. 3116: C. Rattray, S. Maharaj, C. Shankland (Eds.), Algebraic Methodology and Software Technology. XI, 569 pages. 2004.

Vol. 3115: P. Enser, Y. Kompatsiaris, N.E. O'Connor, A.F. Smeaton, A.W.M. Smeulders (Eds.), Image and Video Retrieval. XVII, 679 pages. 2004.

Vol. 3114: R. Alur, D.A. Peled (Eds.), Computer Aided Verification. XII, 536 pages. 2004.

Vol. 3113: J. Karhumäki, H. Maurer, G. Paun, G. Rozenberg (Eds.), Theory Is Forever. X, 283 pages. 2004.

Vol. 3112: H. Williams, L. MacKinnon (Eds.), Key Technologies for Data Management. XII, 265 pages. 2004.

Vol. 3111: T. Hagerup, J. Katajainen (Eds.), Algorithm Theory - SWAT 2004. XI, 506 pages. 2004.

Vol. 3110: A. Juels (Ed.), Financial Cryptography. XI, 281 pages. 2004.

Vol. 3109: S.C. Sahinalp, S. Muthukrishnan, U. Dogrusoz (Eds.), Combinatorial Pattern Matching. XII, 486 pages. 2004.

Vol. 3108: H. Wang, J. Pieprzyk, V. Varadharajan (Eds.), Information Security and Privacy. XII, 494 pages. 2004.

Vol. 3107: J. Bosch, C. Krueger (Eds.), Software Reuse: Methods, Techniques and Tools. XI, 339 pages. 2004.

Vol. 3106: K.-Y. Chwa, J.I. Munro (Eds.), Computing and Combinatorics. XIII, 474 pages. 2004.

Vol. 3105: S. Göbel, U. Spierling, A. Hoffmann, I. Iurgel, O. Schneider, J. Dechau, A. Feix (Eds.), Technologies for Interactive Digital Storytelling and Entertainment. XVI, 304 pages. 2004.

Vol. 3104: R. Kralovic, O. Sykora (Eds.), Structural Information and Communication Complexity. X, 303 pages. 2004.

Vol. 3103: K. Deb, e. al. (Eds.), Genetic and Evolutionary Computation – GECCO 2004. XLIX, 1439 pages. 2004.

Vol. 3102: K. Deb, e. al. (Eds.), Genetic and Evolutionary Computation – GECCO 2004. L, 1445 pages. 2004.

Vol. 3101: M. Masoodian, S. Jones, B. Rogers (Eds.), Computer Human Interaction. XIV, 694 pages. 2004.

Vol. 3100: J.F. Peters, A. Skowron, J.W. Grzymała-Busse, B. Kostek, R.W. Świniarski, M.S. Szczuka (Eds.), Transactions on Rough Sets I. X, 405 pages. 2004.

Table of Contents

Invited Talk

Application Specific (Co)Design

System and Application Synthesis

Data Flow Analysis

Data Partitioning

Task Scheduling

Code Generation

The New Economics of Embedded Systems

Michael Uhler

Chief Technology Officer
MIPS Technologies
uhler@mips.com

Abstract. There is a fundamental shift coming in the economic model used to build successful embedded systems. Below 100nm fabrication geometries, the cost of designing and creating masks for new semiconductor devices is projected to rocket up, making it uneconomical to create a new device for every system, or even to do another manufacturing pass to fix bugs.

So why is this an interesting topic at a workshop on software and compilers? Because the increased cost of hardware, and the increasing demand for new capability in embedded systems mean that programmable products will explode as the solution of choice to cost-effective embedded systems. However, the need to keep power dissipation under control means that both programmability and configurability will be critical in new system design. This puts an increased burden on compilers to generate code for configurable processors, and requires additional capability in software to manage the complexity.

This talk will briefly review the reasons for the economic change, but focus primarily on a vision for a new type of embedded system in which software and compilers play a critical role. In some sense, the original RISC concepts have returned in that software will assume an increasing role in the design of new embedded systems.

H. Schepers (Ed.): SCOPES 2004, LNCS 3199, p. 1, 2004.

A Framework for Architectural Description of Embedded System

Daniela Cristina Cascini Peixoto[1] and Diógenes Cecílio da Silva Júnior[2]

[1] Computer Science Department
Universidade Federal de Minas Gerais, Brasil
cascini@dcc.ufmg.br
[2] Electrical Engineering Department
Universidade Federal de Minas Gerais, Brasil
diogenes@ufmg.br

Abstract. In this paper a new approach for describing embedded systems is presented. The approach is based on the composition of hardware and software components with the addition of an interface between them. Non-functional constraints for components and their interfaces can also be modeled and verified. As such, the component-based view presented here differs from traditional component-based views, where focus is laid on the functional part. The ideas discussed in this paper have been implemented in a tool. This tool enables the description of an embedded system through a specific language. It can also allow the behavioral simulation and the non-functional verification of the hardware and software components.

1 Introduction

As with all computer systems, the embedded computer is composed of hardware and software. In the early days of microprocessors much of the design time was spent on the hardware, in defining address decoding, memory map and so on. A comparatively simple program was developed, limited in size and complexity by memory size.

Current embedded system designs are complex and demand high development efforts. The design attention has shifted to software development and much of the hardware system is now contained on a single chip, in the form of a microcontroller [18].

The design of mixed hardware/software systems presents several challenges to the designer. Not the least of these is the fact that even though the hardware and the software are interdependent, they are typically described and designed using different formalisms, languages, and tools. These notations include a wide variety of hardware description languages, continuous modeling languages, protocol specification languages and dynamic synchronization languages.

Combining hardware and software designs tasks into a common methodology has several advantages. One is that accelerates the design process. Another is that addressing the design of hardware and software components simultaneously may enable hardware/software trade-offs to be made dynamically, as design progresses.

In this paper we propose a notation to describe embedded system architecture, exposing its gross organization as a collection of interacting hardware and software

H. Schepers (Ed.): SCOPES 2004, LNCS 3199, pp. 2–16, 2004.

components. A well-defined architecture allows an engineer to reason about system properties at a high level of abstraction. Typical properties of concern include protocols of interaction, bandwidths, memory size, and so on.

Our primary considerations here to support architectural abstractions, identify, classify and code the components and the ways they interact.

Another point is to verify the correctness of the design. It is needed that the final design has the same behavior as specified. It is also clear that some way is needed for the specification and verification of non-functional constraints. In our approach, domain modeling is used to ensure this.

Our framework provides a simulation environment to validate the functional specification of the components, as well as the interaction between them. This can be used to gain information about an embedded system before a prototype is actually built. Furthermore, this simulation allows designers to perform experiments that could be impractical in a prototyping environment. For example, designers can perform comparative studies of a single system using a variety of processors or other hardware components. This helps ensure completeness of the specification and avoids inconsistency and errors.

2 Related Works

There are four primary areas of related work that address similar problems. The first and most influential of these areas is software architecture description languages, tools and environments. The second area is environments for hardware-software co-design. This area includes environments for modeling, simulation and prototyping of heterogeneous systems. The third area is notations for specification of system-level design problems. Finally, the fourth area presents models for the design of embedded software.

2.1 Software Architecture Description Languages, Toolkits and Environments

Software architecture has been a very active research field in Software Engineering [9,16]. Its goal is to provide a formal way of describing and analyzing large software systems. Abstractly, software architectures involve the description of elements, patterns that guide the composition, and constraints on these patterns.

Numerous Architecture Description Languages (ADLs) have been created for describing the structure and behavior of software systems at the architectural level of abstraction. Most of these ADLs offer a set of tools that support the design and analysis of software system architectures specified with the ADL.

Examples of ADLs include Aesop [7], Rapide [10], Wright [3], UniCon [15], ACME [8] and Meta-H [5]. Although all of these languages are concerned with architectural design, each language provides certain distinctive capacibilities: Aesop supports design using architectural styles; Rapide allows architectural designs to be simulated, and has tools for analyzing the results of those simulations; Wright supports the specification and analysis of interactions between architectural components; UniCon has a high-level compiler for architectural designs that supports a heterogeneous mixture of component and connector abstractions; ACME supports the interchange of architectural descriptions between a wide variety of different architecture design and analysis tools; Meta-H provides specific guidance for designers of real-time avionics control software.

However, none of these ADLs offers sufficient support to capture design properties of hardware components, such as organization or capacity.

Generically, the hardware architecture is described at the RTL (*Register Transfer Level*), what confuses its design with its final implementation. Widely used languages to support the description at this level are VHDL (*VHSIC Hardware Description Language*) and Verilog HDL. The hardware architecture can also specify an ISA (*Instruction Set Architecture*), which describes the computer architecture or a CPU (*Central Processor Unit*).

2.2 Hardware-Software Co-design Environments

Over the past several years there has been a great deal of interest in the design of mixed hardware-software systems, sometimes referred to as hardware-software co-design.

Approaches to hardware-software co-design can be characterized by the design activities for which hardware and software are integrated, which include: hardware-software co-simulation, hardware-software co-synthesis and hardware-software partitioning [1].

Numerous methodologies and languages have been created for each of these design activities. Examples include Ptolemy II [4], LYCOS [11] and POLIS [6].

Ptolemy II is an environment for simulation and prototyping of heterogeneous systems. Its focus is on embedded systems. Ptolemy II takes a component view of design, in that models are constructed as a set of interacting components. A model of computation governs the semantics of the interaction, and thus imposes a discipline on the interaction of the components.

LYCOS is a co-synthesis environment that can be used for hardware-software partitioning of an application onto a target architecture consisting of a single CPU and a single hardware component. LYCOS provides the choice between different partition models and partitioning algorithms, one of which is a novel algorithm, called PACE.

POLIS is a co-design system in which hardware-software partitioning is obtained by user interaction. In POLIS analysis and transformation are done on a uniform and formal internal hardware/software representation called Co-design Finite State Machines, CFSMs. Partitioning is done manually by assigning each CFSM to either hardware or software. POLIS will assist the designer providing estimation tool.

2.3 System Level Design Languages

A well-known solution in computer science for dealing with complexity is to move to a higher level of abstraction, in this case to system level. From this level, the design methodology works its way through several refinement steps down to the implementation.

Several languages were developed to deal with this question. Examples include SystemC [17] and Rosetta [2].

SystemC permits the specification and design of a hardware-software system at various levels of abstraction. It also creates executable specification of the design.

This language is a superset of C++. It provides the necessary constructs to model system architecture including hardware timing, concurrency, and reactive behavior that are missing in standard C++. However, SystemC does not provide any support for representing design aspects like timing, structural and physical constraints. For example, cycle time memory capacity or power consumption.

Rosetta is a non-executable language that addresses the problem of defining the constraints of the desired design and not its behavior. This language is mainly used to mathematically analyze the performance of a system. Rosetta provides modeling support for different design domains using multiple-domain semantics and syntaxes appropriate for each of them.

2.4 Embedded System Design Approaches

Architecture systems introduce the notion of components, ports and connector as first class representations. However, most of the approaches proposed in the literature do not take into account the specific properties of software systems for embedded devices.

Koala [12] introduces a component model that is used for embedded software in consumer electronic devices. Koala components may have several "provides" and "requires" interfaces. In order to generate efficient code from Koala specifications, partial evaluation techniques are employed. However, Koala does not take into account non-functional requirements such as timing and memory consumption. Koala lacks a formal execution model and automated scheduler generation is not supported.

PECOS (*Pervasive Component System*) [13] model provides a component-based technology for the development of embedded software systems. PECOS is divided into two sub-models: structural and execution. The structural model defines the entities (port, connector and components), its characteristics and properties. The execution model deals with execution semantics. It specifies the synchronization between components that live in different threads of control, cycle time and required deadlines that a component has to be executed. An automated scheduler is provided by the composite component for their children components.

These environments do not provide any mechanism for representation of the hardware components and the interaction between the hardware/software components.

3 LACCES Design Language

LACCES (*Language of Components and Connectors for Embedded Systems*) design language is used for capturing both embedded system architecture design expertise and the architectural constraints. The language provides constructs for capturing all requirements for design an embedded system with hardware and software components.

LACCES meet three fundamental requirements: it is capable of describing architectural structure, properties, and topology of individuals embedded system designs; it allows the behavioral and non-functional evaluation and analysis of the design; and the language supports incremental capture of architectural design expertise and incremental modifications of architectural descriptions.

This notation addresses several issues in novel ways:

- It permits the specification of hardware and software components and their interaction.
- It supports abstraction idioms commonly used by designers. These idioms are captured through a set of primitives. Different types of elements can be modeled.
- It provides a clear separation of concerns regarding important properties of a system, for example, computation and communication; behavior and structure; and functionality and timing.
- It has the ability to capture a broad range of system requirements, namely, functionality, timing, logical structure, and physical structure constraints.
- It defines a function to map the description to an executable specification that can be used to validate system functionality before implementation begins. It also helps to avoid inconsistency and errors of the functionality specification.

The first step towards the definition of the language was to survey several designs and compile a list of system primitives, which are constructs necessary to represent embedded systems. These primitives are broad enough to represent a large range of components and connectors and covers diverse aspects such as behavior, logical structure, timing and physical structure. These primitives form the basis of LACCES.

The set of primitives is also complete and consistent, so new structures can be represented without need to extend the language. It is also possible to combine these primitives to obtain more complex blocks, and to decompose a complex block or design into a collection of these primitives.

To accommodate the wide variety of auxiliary information LACCES supports annotation of architectural properties. Components and connectors can be annotated. These properties are described in terms of four different domains.

The motivating factor to use four orthogonal domains is the separation of concerns [14]. The first concern is the separation of time from functionality. This led to the use of two domains, *Behavioral* and the *Timing* domains.

A topological or logical organization is usually employed to facilitate the design capture. To represent this logical organization the *Structural* domain is used. It is a mapping of behavioral and timing representations onto a set of constraints such as capacity and organization.

The *Physical* domain is used to represent clients' and designers' physical requirements and constraints and actual implementation.

There is no direct relationship among objects in different domains. The domains can be partially or completely orthogonal to each other. Orthogonal domains mean that the domains are independent of each other and changes in one domain do not affect the other domain. For example, a single behavior, represented in the behavioral domain, can be mapped to several different implementations in the physical domain.

Each element may be represented in all four domains. Some elements may not have representation in some domains, especially in the physical domains if there are not specified physical constraints.

3.1 Components

Components are the core entity in this model. They are used to organize the computation and data into parts that have well-defined semantic and behavior.