From Model-Driven Design to Resource Management for Distributed Embedded Systems

Edited by
Bernd Kleinjohann
Lisa Kleinjohann
Ricardo J. Machado
Carlos Pereira
P.S. Thiagarajan

Springer

ifip

FROM MODEL-DRIVEN DESIGN TO RESOURCE MANAGEMENT FOR DISTRIBUTED EMBEDDED SYSTEMS

IFIP – The International Federation for Information Processing

IFIP was founded in 1960 under the auspices of UNESCO, following the First World Computer Congress held in Paris the previous year. An umbrella organization for societies working in information processing, IFIP's aim is two-fold: to support information processing within its member countries and to encourage technology transfer to developing nations. As its mission statement clearly states,

> *IFIP's mission is to be the leading, truly international, apolitical organization which encourages and assists in the development, exploitation and application of information technology for the benefit of all people.*

IFIP is a non-profitmaking organization, run almost solely by 2500 volunteers. It operates through a number of technical committees, which organize events and publications. IFIP's events range from an international congress to local seminars, but the most important are:

• The IFIP World Computer Congress, held every second year;
• Open conferences;
• Working conferences.

The flagship event is the IFIP World Computer Congress, at which both invited and contributed papers are presented. Contributed papers are rigorously refereed and the rejection rate is high.

As with the Congress, participation in the open conferences is open to all and papers may be invited or submitted. Again, submitted papers are stringently refereed.

The working conferences are structured differently. They are usually run by a working group and attendance is small and by invitation only. Their purpose is to create an atmosphere conducive to innovation and development. Refereeing is less rigorous and papers are subjected to extensive group discussion.

Publications arising from IFIP events vary. The papers presented at the IFIP World Computer Congress and at open conferences are published as conference proceedings, while the results of the working conferences are often published as collections of selected and edited papers.

Any national society whose primary activity is in information may apply to become a full member of IFIP, although full membership is restricted to one society per country. Full members are entitled to vote at the annual General Assembly, National societies preferring a less committed involvement may apply for associate or corresponding membership. Associate members enjoy the same benefits as full members, but without voting rights. Corresponding members are not represented in IFIP bodies. Affiliated membership is open to non-national societies, and individual and honorary membership schemes are also offered.

FROM MODEL-DRIVEN DESIGN TO RESOURCE MANAGEMENT FOR DISTRIBUTED EMBEDDED SYSTEMS

IFIP TC 10 Working Conference on Distributed and Parallel Embedded Systems (DIPES 2006), October 11-13, 2006, Braga, Portugal

Edited by

Bernd Kleinjohann
University of Paderborn
Germany

Lisa Kleinjohann
University of Paderborn/ C-Lab
Germany

Ricardo J. Machado
Universidade do Minho
Portugal

Carlos E. Pereira
Universidade Federal do Rio Grande do Sul
Brazil

P.S. Thiagarajan
National University of Singapore
Republic of Singapore

Springer

Library of Congress Control Number: 2006931703

From Model-Driven Design to Resource Management for Distributed Embedded Systems

Edited by B. Kleinjohann, L. Kleinjohann, R. Machado, C. Pereira, and P.S. Thiagarajan

p. cm. (IFIP International Federation for Information Processing, a Springer Series in Computer Science)

ISSN: 1571-5736 / 1861-2288 (Internet)
ISBN: 10: 0-387-39361-7
ISBN: 13: 9780-387-39361-7
eISBN: 10: 0-387-39362-5
Printed on acid-free paper

Copyright © 2006 by International Federation for Information Processing.
All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.
The use in this publication of trade names, trademarks, service marks and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

9 8 7 6 5 4 3 2 1
springer.com

Contents

Preface ix
Conference Committee xi

1 **Challenges in the Design of Distributed Embedded Systems**

Design Challenges in Multiprocessor System-on-Chips
Wayne Wolf *1*

Some Issues in Model-Based Development for Embedded Control Systems
Paul Caspi *9*

2 **Model-Driven Development**

MDE Benefits for Distributed, Real-Time and Embedded Systems
Francois Terrier, Sébastien Gérard *15*

Reifying the Semantic Domains of Component Contracts
Jean-Marc Jézéquel *25*

3 **Model-Based Conformance Testing and Requirement Adaptation**

Model Based Test Selection for Infinite State Reactive Systems
Thierry Jéron *35*

Continuous Engineering of Embedded Systems
Bernhard Steffen, Tiziana Margaria *45*

4 Applications and Code Realization

Prototyping an Ambient Light System – A Case Study
Henning Zabel, Achim Rettberg 55

The Paderkicker Team: Autonomy in Realtime Environments
Willi Richert, Bernd Kleinjohann, Markus Koch, Alexander Bruder, Stefan Rose, Phillip Adelt 65

Modular Compilation of Synchronous Programs
Klaus Schneider, Jens Brandt, Eric Vecchié 75

5 Timing Analysis

Trends in Timing Analysis
Björn Lisper 85

Traffic Scheduling Anomalies in Temporal Partitions
Luís Almeida, Paulo Pedreiras, Ricardo Marau 95

6 Towards Timing Predictability

Pulsed Data Streams
Hermann Kopetz 105

From Time-Triggered to Time-Deterministic Real-Time Systems
Peter Puschner, Raimund Kirner 115

7 Scheduling and Timed Multitasking

Lazy Scheduling for Energy Harvesting Sensor Nodes
Clemens Moser, Davide Brunelli, Lothar Thiele, Luca Benini 125

Transient Processor/Bus Fault Tolerance for Embedded Systems
Alain Girault, Hamoudi Kalla, Yves Sorel 135

Distributed Timed Multitasking - A Model Of Computation for Hard Real-Time Distributed Systems
Christo Angelov, Jesper Berthing 145

8 Service Configuration and Task Allocation

Iterative Refinement Approach for QoS-Aware Service Configuration

Luís Noguera, Luís Miguel Pinho 155

A Fast and Efficient Isomorphic Task Allocation Scheme for K-ary N-cube Systems
D. Doreen Hephzibah Miriam, T. Srinivasan 165

Communication Aware Component Allocation Algorithm for a Hybrid Architecture
Marcelo Götz, Achim Rettberg, Carlos Eduardo Pereira 175

9 Design Space Exploration and Memory Management

Multi-Objective Design Space Exploration of Embedded System Platfoms
Jan Madsen, Thomas K. Stidsen, Peter Kjœrulf, Shankar Mahadevan 185

Dynamic Memory Management for Embedded Real-Time Systems
A. Crespo, I. Ripoll, M. Masamano 195

10 Power Management

Reliability-Aware Power Management of Multi-Core Processors
Jan Haase, Markus Damm, Dennis Hauser, Klaus Waldschmidt 205

Evaluating Energy-Aware Task Allocation Strategies for MPSoCs
Fabio Wronski, Eduardo W. Brião, Flávio R. Wagner 215

Integration of Energy Reduction into High-Level Synthesis by Partitioning
Achim Rettberg, Franz J. Rammig 225

11 Work in Progress

A Demonstration Case on the Transformation of Software Architectures for Service Specification
João M. Fernandes, Ricardo J. Machado, Paula Monteiro, Helena Rodrigues 235

A Model-Based Analysis of a Wndmill Communication System
Simon Tjell 245

Pre-Runtime Scheduling Considering Timing and Energy Constraints in

Embedded Systems with Multiple Processors
Eduardo Tavares, Meuse Olivera Jr., Paulo Maciel, Bruno Souza, Silvino Neto, Raimundo Barreto, Romulo Freitas, Marcelo Custodio 255

A Hierarchical Approach for Power Management on Mobile Embedded Systems
Arliones Stevert Hoeller Jr., Lucas Francisco Wanner, Antônio Augusto Fröhlich 265

Preface

This volume consists of the proceedings of the 5th IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES 2006). This bi-annual series of conferences are intended as a forum where significant research on models, methods, tools and applications in the area of distributed embedded systems are presented.

Embedded computing systems have started to carry out the key control functions in diverse domains such as telecommunications, automotive electronics, avionics and even complete industrial manufacturing lines. Traditionally, such embedded control systems have been implemented in a monolithic, centralized manner. However, distributed and parallel solutions have been steadily gaining popularity. In a distributed setup, the control task is carried out by a number of controllers distributed over the entire system and interconnected as a network by communication components such as field buses. More demanding local control applications require controllers based on parallel architectures or processors with dedicated co-processors. Distribution and parallelism in embedded system design increase the engineering challenges and demand new development methods and tools. The main goal of the DIPES series of conferences is to bring together the research community dealing with problems in this important area.

The previous conferences were located in Toulouse, France (2004), Montreal, Canada (2002) and Schloß Eringerfeld, Germany (1998 and 2000). This year's conference was organized by the Escola de Engenharia, Universidade do Minho and took place in Braga, Portugal. We cordially thank the organizers, especially Joao M. Fernandes, for the great deal of work they have put in to enable the smooth running of the conference.

The DIPES 2006 conference comprises a combination of invited presentations by the leading researchers in the field and papers selected from a pool of submitted papers. We are grateful to our invited speakers for agreeing to present their research at the conference and for the papers they have provided for inclusion in this volume. In this connection, we also wish to thank Franz Rammig and Lisa Kleinjohann for helping to put together a first rate set of invited speakers and conference program.

This year we received 21 submissions from which 12 papers were selected for regular presentations and 4 for short presentations. We thank the authors for their submissions and the members of the Program Committee for their timely reviews.

Bernd Kleinjohann, Ricardo J. Machado, Carlos E. Pereira, P. S. Thiagarajan

IFIP TC10 Working Conference on Distributed and Parallel Embedded Systems (DIPES 2006) October 11-13, 2006, Braga, Portugal

General Chair
Bernd Kleinjohann, University of Paderborn (Germany)

Organizing Chair
Ricardo J. Machado, Universidade do Minho (Portugal)

Program Committee Co-Chairs
Carlos E. Pereira, UFRGS (Brazil)
P. S. Thiagarajan, NUS (Republic of Singapore)

Program Committee
Arndt Bode (Germany)
Joao Cardoso (Portugal)
Nikil Dutt (USA)
Bernhard Eschermann (Switzerland)
Joao Fernandes (Portugal)
Guang R. Gao (USA)
Uwe Glässer (Canada)
Luis Gomes (Portugal)
Uwe Honekamp (Germany)
Pao-Ann Hsiung (Taiwan)
Ahmed Jerraya (France)
Jens B. Jorgensen (Denmark)
Kane Kim (USA)
Moon Hae Kim (Korea)
Bernd Kleinjohann (General Chair, Germany)
Lisa Kleinjohann (Germany)
Herman Kopetz (Austria)
Johan Lilius (Finland)
Ricardo J. Machado (Portugal)

Erik Maehle (Germany)
Carlos E. Pereira (PC Co-Chair, Brazil)
Luis Pinho (Portugal)
Byron Purves (USA)
Peter Puschner Austria)
Franz J. Rammig (Germany)
Achim Rettberg (Germany)
Matthias Riebisch (Germany)
Bernd-Heinrich (Germany)
Edwin Sha (USA)
Zili Shao (Hong Kong)
Joachim Stroop (Germany)
P. S. Thiagarajan (PC Co-Chair, Republic of Singapore)
Flavio R. Wagner (Brazil)
Dieter Wuttke (Germany)
Alex Yakovlev (UK)
Laurence T. Yang (Canada)

Organizing Committee *(Univ. do Minho and Univ. of Paderborn)*
Ricardo J. Machado, Chair and Editorial Liaison
Joao M. Fernandes, Co-Chair and Finance
Lisa Kleinjohann, Web Services
M. Joao Nicolau, Internet Services
Achim Rettberg, Web Services
Oscar Ribeiro, Local Arrangements
Paula Monteiro, Local Arrangements

Sponsoring and Co-Organizing Institution
IFIP TC 10, WG 10.5, SIG-ES in co-operation with WG 10.1, 10.3, 10.4

Acknowledgement
Special thanks to the authors for their contribution, and for the Program Committee members for their time reviewing the contributions.

DESIGN CHALLENGES IN MULTIPROCESSOR SYSTEMS-ON-CHIP

Wayne Wolf
Department of Electrical Engineering, Princeton University

Abstract: A multiprocessor system-on-chip is an integrated system that performs real-time tasks at low power and for low cost. The stringent requirements on multiprocessor systems-on-chips force us to use advanced design methods to create these systems. Both hardware and software design must be taken into account. In this paper, we will survey some important challenges in the design of these systems.

1. INTRODUCTION

A multiprocessor system-on-chip (MPSoC) is an integrated system designed with multiple processing elements. MPSoCs are already in common use; over the next several years very large MPSoCs will come onto the market.

MPSoCs are interesting examples of complex embedded systems. A variety of techniques must be used to successfully create MPSoCs and their embedded applications. These techniques must take into account real-time performance, power/energy consumption, and cost.

In this paper, we will survey some design challenges in MPSoCs. We will start by reviewing the requirements on these systems. We will then look at several aspects of these systems: processors, multiprocessor architecture, programs, and task-level scheduling.

Please use the following format when citing this chapter:

Wolf, W., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 1–8.

2. REQUIREMENTS AND APPROACHES

MPSoCs must provide high levels of performance for applications like video compression and high-speed data communication. But that performance must meet real-time, not just average performance requirements. Real-time computing constraints are deadlines that must be met. Average high performance is not satisfactory if one of the system's tasks does not meet its deadlines.

Many multiprocessor SoCs also operate under tight power and energy requirements. Most embedded applications are somewhat power and heat sensitive. Battery-powered systems must make the best use of available battery energy.

Embedded systems are generally cost-sensitive. The cost of the hardware platform, including processors and memory, must be kept down while meeting the real-time performance requirements.

One traditional approach to meeting these requirements is to make use of knowledge of the application. General-purpose computers are designed around benchmarks but not highly tuned to a particular application. If we know details of the application to be run, we can customize the MPSoC to provide features where needed and eliminate them where they are not necessary.

An implication of this approach is that many MPSoCs are heterogeneous, with multiple types of CPUs, irregular memory hierarchies, and irregular communication. Heterogeneity allows architects to support necessary operations while eliminating the costs of unnecessary features. Heterogeneity is also important to real-time performance---limiting access to part of the system can make that part more predictable.

3. PROCESSOR SELECTION

The choice of processor is one of the basic decisions in the design of an embedded system. The decision may be made on several grounds, both technical and non-technical: software performance, I/O system configuration, support tools, setup costs, etc. The designer may also choose between an existing processor and a customized CPU.

Customized processor architectures can be implemented on systems-on-chips or using FPGAs. When choosing a CPU based on software performance, two major alternatives exist: hardware/software partitioning and custom instruction sets. While these techniques have not traditionally been seen as alternatives, both have a similar goal, namely the cost-effective speedup of a program. Hardware/software partitioning works at coarser granularity while custom instruction sets find speedups at finer levels of granularity.

Hardware/software partitioning builds a custom heterogeneous system with a CPU and a hardwired accelerator, based on program characteristics and performance requirements. A variety of hardware/software partitioning systems have been developed, including COSYMA [7], Vulcan [10], CoWare [25], the system of Eles et al. [Ele97], Lycos [17], and COSYN [5]. These co-synthesis algorithms work on fairly large blocks of program behavior, generally either loop nests or task graphs. They choose units to implement in hardware based in part on the cost of communication between the processor and the accelerator.

The goal of instruction set design is to find operations that can be profitably packaged as instructions. These are generally combinations of more basic instructions or perhaps work on specialized registers, and so show coarser granularity than standard instructions, but finer granularity than accelerators. Instruction sets can be designed by hand, and CPU microarchitectures that are designed to be customized are known as configurable processors. Several companies offer configurable processors. Several university configurable processors also exist, including LISA [11] and PEAS-III [13, 23].

The largest cost in using configurable processors, when compared to accelerators, is the overhead of instruction interpretation. Instruction fetch, decode, and execution on a CPU are more expensive than dedicated logic on an accelerator. However, a configurable processor has two advantages over an accelerator. First, it can be used for many tasks and so may be more heavily utilized. Second, we can eliminate the cost of communication between the accelerator and the processor (so long as we have the registers required to perform the operations). As MPSoCs become larger and the area cost of CPUs becomes less critical, we may see more configurable processors.

4. MULTIPROCESSOR CONFIGURATION

Many embedded systems are multiprocessors, so we need to configure a multiprocessor for use as a platform for the application. Embedded multiprocessors are often heterogeneous, in order to meet real-time performance requirements as well as power and cost requirements.

Hardware/software co-design algorithms have been developed to synthesize multiprocessor architectures, for example systems by Dave and Jha [4] and Wolf [26], but system-on-chip multiprocessors have often been designed by hand. Hand-designed architectures may be satisfactory for multiprocessors with a few processors, but as we move to systems-on-chips with a dozen or more large processors, we may see MPSoC designers rely more heavily on design space exploration tools.

As in general-purpose computing systems, busses don't scale to large multiprocessors. As a result, MPSoCs are starting to use on-chip networks that route packets between processors and memory. A number of networks-on-chips have been developed, including Nostrum [15], SPIN [9], Slim-spider [16], OCCN [3], QNoC [1], xpipes/NetChip [20,14], and the network of Xu et al. [27].

5. PROGRAM DESIGN

When optimizing embedded programs for the target platform, memory system behavior is a prime target for optimization. Many embedded systems are memory intensive and the program's interaction with memory helps to determine both the system performance and energy consumption.

Code placement was originally developed for general-purpose machines but is also useful in embedded systems. Code placement determines the addresses for sections of code to minimize the cost of cache interactions between instructions. Hwu and Chang [12] extracted information from traces and placed code using a greedy algorithm. McFarling [18] used a combination of trace data and program behavior to determine how to place code.

A variety of methods for optimizing the cache behavior of data have been developed, both by the scientific computing and embedded communities. Panda et al. [21] used a cluster interference graph to optimize the placement of data in memory. Panda et al. [22] developed algorithms for placing data in scratch pads, which are software-controlled memories at the same level of memory hierarchy as level-one caches.

Overall methodologies for memory-intensive systems have also been developed, most notably by Catthoor et al. [2]

6. PROCESS-LEVEL DESIGN

Traditional scheduling algorithms treat jobs as atomic; in some cases, jobs are assumed to arrive dynamically so their characteristics are not known to the scheduler. While embedded systems tasks may use some dynamic tasks, the critical code is often known in advance. Knowledge of the task allows us to combine scheduling algorithms with memory hierarchy analysis, power management, and other aspects of the system

When we build embedded systems on multiprocessor platforms, we often rely on middleware to manage the multiprocessor. Single-processor management is handled by an operating system, while middleware negotiates resource requests across the multiprocessor platform. One approach to building embedded system middleware is to rely on existing standards, such as CORBA [19, 24]. An alternative is to design custom middleware services. Thanks to the tight performance/energy/cost constraints of embedded systems, we should expect to see at least customized versions of middleware standards.

7. SUMMARY

Embedded systems must provide very high levels of performance, but under much more serious power and cost constraints than general-purpose systems. MPSoC designers need to take advantage of the knowledge of computer system design gained over the past several decades, but embedded computing is developing additional techniques to solve its unique problems. Optimizations of both hardware and software are often necessary to achieve the strict requirements of multiprocessor systems-on-chips.

REFERENCES

[1] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *The Journal of Systems Architecture*, 50(2-3), February 2004, pp. 105-128.

[2] Francky Catthoor, Sven Wuytack, Eddy De Greef, Florin Balasa, Lode Nachtergaele, and Arnout Vandecappelle, *Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design*, Norwell MA: Kluwer Academic Publishers, 1998.

[3] Marcello Coppola, Stephane Curaba, Miltos D. Grammatikakis, Giuseppe Maruccia, and Francesco Papariello, "OCCN" a network-on-chip modeling and simulation framework," in *Proceedings of the Conference on Design Automation and Test in Europe*, vol. 3, IEEE Computer Society Press, 2004, p. 30174.

[4] Bharat P. Dave and Niraj K. Jha, "COHRA: hardware-software cosynthesis of hierarchical heterogeneous distributed embedded systems," *IEEE Transactions on CAD/ICAS*, 17(10), October 1998, pp. 900-919.

[5] Bharat P. Dave, Ganesh Lakshminarayana, and Niraj K. Jha, "COSYN: hardware-software co-synthesis of heterogeneous distributed embedded systems," *IEEE Transactions on VLSI Systems*, 7(1), March 1999, pp. 92-104.

[6] Petru Eles, Zebo Peng, Krzysztof Kuchcinski, and Alexa Doboli, "System level hardware/software partitioning based on simulated annealing and tabu search," *Design Automation for Embedded Systems*, 2, 1996, pp. 5-32.

[7] Rolf Ernst, Joerg Henkel, and Thomas Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Design and Test of Computers,* 10(4), December 1993, pp. 64–75.

[8] G. Goossens, "Application-specific networks-on-chips," in A. Jerraya and W. Wolf, eds., *Multiprocessor Systems-on-Chips*, Morgan Kaufman, 2004.

[9] Pierre Guerrier and Alain Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proceedings of the Conference on Design Automation and Test in Europe*, ACM Press, 2000, pp. 250-256.

[10] Rajesh K. Gupta and Giovanni De Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Design and Test of Computers*, 10(3), September 1993, pp. 29–40.

[11] Andreas Hoffman, Tim Kogel, Achim Nohl, Gunnar Braun, Oliver Schliebusch, Oliver Wahlen, Andreas Wieferink, and Heinrich Meyr, "A novel methodology for the design of application-specific instruction-set processors (ASIPs) using a machine description language," *IEEE Transactions on CAD/ICAS*, 20(11), November 2001, pp. 1138-1354.

[12] Wen-Mei W. Hwu and Pohua P. Chang, "Achieving high instruction cache performance with an optimizing compiler," in *Proceedings of the 16th Annual International Symposium on Computer Architecture*, ACM, 1989, pp. 242-251.

[13] Makiko Itoh, Shigeaki Higaki, Jun Sato, Akichika Shiomi, Yoshinori Takuchi, Akira Kitajima, and Masaharu Imai, "PEAS-III: an ASIP design environment," in *International Conference on Computer Design: VLSI in Computers and Processors*, IEEE Computer Society Press, 2000, pp. 430-436.

[14] A. Jalabert, S. Murali, L. Benini, and G. De Micheli, "xpipesCompiler: a tool for instantiating application specific networks-on-chips," in *Proceedings of Design Automation and Testing in Europe Conference*, IEEE, 2004, pp. 884-889.

[15] Sashi Kumar, Axel Jantsch, Juha-Pekka Soininen, Martti Forsell, Mikael Millberg, Jonny Oberg, Kari Tiensyrja, and Ahmed Hemani, "A network on chip architecture and design methodology," in Proceedings

of the IEEE Computer Society Annual Symposium on VLSI, IEEE Computer Society Press, 2002.

[16] Se-Joong Lee, Kangmin Lee, and Hoi-Jun Yoo, "Analysis and implementation of practical, cost-effective networks-on-chips," *IEEE Design & Test of Computers*, 22(5), September/October 2005, pp. 422-433.

[17] J. Madsen, J. Grode, P. V. Knudsen, M. E. Petersen, and A. Haxthausen, "LYCOS: the Lyngby Co-Synthesis System," *Design Automation for Embedded Systems*, 2, 1997, pp. 195-235.

[18] Scott McFarling, "Program optimization for instruction caches," *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, 1989, pp. 183-191.

[19] Object Management Group, CORBA Basics, 2006, http://www.omg.org/gettingstarted/corbafaq.htm.

[20] M. D. Osso, G. Biccari, L. Giovanni, D. Bertozzi, and L. Benini, "xpipes: a latency insensitive parameterized network-on-chip architecture for multi-processor SoCs," in *Proceedings of the 21st International Conference on Computer Design*, IEEE Computer Society Press, 2003, pp. 536-539.

[21] Preeti Ranjan Panda, Nikil D. Dutt, and Alexandru Nicolau, "Memory data organization for improved cache performance in embedded processor applications," ACM Transactions on Design Automation of Electronic Systems, 2(4), October 1997, pp. 384-409.

[22] Preeti Ranjan Panda, Nikil D. Dutt, and Alexandru Nicolau, "On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems," *ACM Transactions on Design Automation of Embedded Systems*, 5(3), July 2000, pp. 682-704.

[23] Toshiyuki Sasaki, Shinsuke Kobayashi, Tomohide Maeda, Makkiko Itoh, Yoshinori Takeushi, and Masahiru Imai, "Rapid prototyping of complex instructions for embedded processors using PEAS-III," in *Proceedings, SASIMI 2001*, October, 2001, pp. 61-66.

[24] Douglas C. Schmidt and Fred Kuhns, "An overview of the real-time CORBA specification," *IEEE Computer*, 33(6), June 2000, pp. 56-63.

[25] S. Vercauteren, B. Lin, and H. De Man, "A strategy for real-time kernel support in application-specific HW/SW embedded architectures," in *Proceedings, 33rd Design Automation Conference*, ACM Press, 1996, pp. 678-683.

[26] Wayne Wolf, "An architectural co-synthesis algorithm for distributed embedded computing systems," *IEEE Transactions on VLSI Systems*, 5(2), June 1997, pp. 218-29.

[27] Jiang Xu, Wayne Wolf, Joerg Henkel, and Srimat Chakradhar, "A design methodology for application-specific networks-on-chips," *ACM Transactions on Embedded Computing Systems*, to appear in the Special Issue on Multiprocessor Systems-on-Chips.

SOME ISSUES IN MODEL-BASED DEVELOPMENT FOR EMBEDDED CONTROL SYSTEMS

Paul Caspi
caspi@imag.fr
Verimag-CNRS
www-verimag.imag.fr

Abstract This presentation aims to discuss the needs for better and more solid foundations of model-based development in embedded control systems. Three particular points are discussed: a comparison between model-based development in control and in computer sciences, the need for a sampling theory of discrete event systems and the need for precise implementation methods based on preemptive scheduling.

Keywords: model-based development, embedded control, approximation, sampling, voting, distances

1. INTRODUCTION

Model-based development is widely recognised as a method of choice for efficiently and safely designing computing systems. After all, isn't it the way other branches of engineering have followed for achieving such a goal? Just think of how bridges and buildings are designed. Yet, though the need for model-based development is widely recognised, it is true that advances in this direction are quite slow and charges are put on both the youth of computer science and the intrinsic complexity of computing to account for this state of affairs.

There is however a particular subdomain, the embedded control domain, where things have progressed faster. For instance, automatic code generation from high level model have been in use at Airbus in the fly-by-wire department for more than twenty years [7]. Since the beginning of the nineties, the Simulink/Stateflow tool-box also allows automatic code generation (RealTime Workshop) and has achieved an impressive diversity of possible implementation platforms ranging from simple cyclic monoprocessor ones to multi-threaded ones, based on preemptive scheduling and to distributed ones based on specialised CAN

Please use the following format when citing this chapter:

Caspi, P., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 9–13.

or TTA libraries. It is not unfair to say that embedded control is by now one (if not the only one) computing subdomain that has reached the highest possible level of model-based development.

However, as it is often the case, this fast progress has been achieved rather empirically, without taking much care of foundations. Basically, it is the accomplishment of practitioners rather than of theoreticians and the latter have cast very little attention to it. The thesis we would like to support in this presentation is that times have come to strengthen the foundations of the method. Not only this effort can be expected to be fruitful for intellectual purposes but it is also likely that practitioners can benefit from it by getting better, with wider scope and simpler development tools.

So the aim of this presentation is to discuss this issue: which are the foundation needs? Three points will be more precisely considered:

1 What is the use of models in control and how this use differs from what is currently considered in computer science?

2 Is there a well-admitted theory of computer implementation for control models, in particular concerning the sampling of discrete event systems?

3 How can we guarantee behaviour equivalence between models and implementations in case of preemptive scheduling?

2. MODEL-BASED DESIGN IN COMPUTER SCIENCE AND CONTROL

Model-based design is advocated in both theories as a method of choice for efficiently and safely building systems. However these theories differ in the way of achieving this goal:

In computer science, the proposed method (see for instance [1]) is based on successive refinements: a large specification is designed first, imprecise (non deterministic) in general, but sufficient for meeting the desired system properties. Then implementation details are brought in progressively, making the specification more and more precise, while keeping the properties, up to a point when it can be implemented. Clearly, this is an ideal scheme which is seldom fulfilled in practice, but which has a paradigmatic value.

In control science, on the contrary, an exact model is built first, which allows a control system to be designed. Then the various uncertainties that may affect the system behaviour are progressively introduced and it is checked that the designed controller is robust enough to cope with these uncertainties.

Clearly, these two schemes are not, in practice, too far from each other. But, as control systems are mostly implemented by now on computers, some effort is needed if these two schemes have to match more closely. This can be valuable in the perspective of achieving an easier communication between computer and control cultures. A way to reach this goal would be to see the initially precise control model as representing a large class of models, those models which fall within some given "distance" from this model. This distance would then represent the maximally admissible uncertainty around the model and further refinements would make this uncertainty smaller. This goal requires thus some notion of control system distance and approximation.

3. SAMPLING DISCRETE EVENT AND HYBRID SYSTEMS

Large modern control systems mix very closely continuous and discrete event systems. This is due for instance, to mode changes, alarms, fault tolerance and supervisory control. From a theoretical point of view, computer implementation techniques for these two kinds of activity are quite different. Continuous control is dealt with through periodic sampling (time-triggered computations as defined by [5]) while discrete event systems use event-triggered implementations. However, in practice, many mixed continuous control and discrete event control systems are implemented through periodic sampling. This is the case, for instance, in Airbus fly-by-wire systems [7] and many other safety-critical control systems. The problem is that there are no solid foundations to periodically sampling discrete event systems and practitioners rely on in-house "ad-hoc" methods. Building a consistent sampling theory for mixed continuous control and discrete event systems would help in strengthening these practices.

A situation where such lack of theory is particularly critical concerns fault-tolerance: though the theory of distributed fault-tolerant systems [8; 5] advocates the use of clock synchronisation, still many critical real-time systems are based on the GALS (globally asynchronous, locally synchronous) and, more precisely, the "Quasi-Synchronous" [3] paradigm: in this framework, each computer is time-triggered but the clocks associated with each computer are not synchronised and communication is based on periodic sampling: each computer has its own clock and periodically samples its environment, *i.e.*, the physical environment but, also, the activities of the other computers with which it communicates. When such an architecture is used in critical systems, there is a need for a thorough formalisation of fault tolerance in this framework.

4. FAITHFUL IMPLEMENTATIONS BASED ON PREEMPTIVE SCHEDULING

A key question in model based development is the possible discrepancy between models and their computer implementations. As a matter of fact, if this discrepancy is too large, the benefits gained from the use of models can be spoiled. This is the general question investigated in section 2 where this question is considered in terms of distances and topologies. Yet there are particular situations where other approaches can be used. A typical example is found when implementations are based on multiple theads and preemptive scheduling. This kind of implementation is mandatory in several cases, for instance:

- in multi-periodic models for efficiency reasons;
- in event-triggered systems when urgent events have to be handled.

In such systems, inter-task communication is likely to be strongly non deterministic [2]. In some cases, for instance when discrete events are considered, critical races can take place and the "distance" between models and implementations may become too large. There is thus a need for more precise implementation techniques, which do not spoil the benefits of model-based development such as those described in [4; 6].

REFERENCES

[1] Abrial, J.-R. (1995). *The B-Book.* Cambridge University Press.

[2] Caspi, P. and Maler, O. (2005). From control loops to real-time programs. In Hristu, D. and Levine, W., editors, *Handbook of Networked and Embedded Computing Systems.* Birkhäuser.

[3] Caspi, P., Mazuet, C., Salem, R., and Weber, D. (1999). Formal design of distributed control systems with Lustre. In *Proc. Safecomp'99*, volume 1698 of *Lecture Notes in Computer Science.* Springer Verlag.

[4] Henzinger, T. A., Horowitz, B., and Kirsch, Ch. M. (2003). Giotto: A time-triggered language for embedded programming. *Proceedings of the IEEE*, 91:84–99.

[5] Kopetz, H. (1997). *Real-Time Systems Design Principles for Distributed Embedded Applications.* Kluwer.

[6] Scaife, N. and Caspi, P. (2004). Integrating model-based design and preemptive scheduling in mixed time- and event-triggered systems. In Pushner, P., editor, *Euromicro Conference on Real-Time Systems, ECRTS04.*

[7] Traverse, P., Lacaze, I., and Souyris, J. (2004). Airbus fly-by-wire: A total approach to dependability. In *IFIP World Congress, Toulouse.* IFIP.

[8] Wensley, J.H., Lamport, L., Goldberg, J., Green, M.W., Lewitt, K.N., Melliar-Smith, P.M., Shostak, R.E, and Weinstock, Ch.B. (1978). SIFT: Design and

analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255.

MDE BENEFITS FOR DISTRIBUTED, REAL TIME AND EMBEDDED SYSTEMS

François Terrier and Sébastien Gérard
CEA-List, {francois.terrier, sebastien.gerard}@cea.f

Abstract: Embedded systems envelopment are currently being challenged to provide global solutions that reconcile three conflicting agendas: enrichment/refinement of system functionalities, reduction of time-to-market and production costs, and compliance with nonfunctional requirements. Model-Driven Engineering (MDE) can help development master these complexities by both separating concerns and systematically automating the production, integration and validation process. This paper draws on the Accord$|_{UML}$ research project to illustrate the benefits of model-driven engineering for embedded real time systems development.

Keywords: Model-Driven Engineering, UML profile, Embedded System, Real Time

1. INTRODUCTION

On today's sharply competitive industrial market, engineers must focus on their core competencies to produce ever more innovative products, while also reducing development times and costs. This has further heightened the complexity of the development process. At the same time, industrial systems, and specifically embedded systems, have become increasingly software-intensive. New software development approaches and methods must therefore be found to free engineers from the technical constraints of development and allow them to concentrate on their core specialties. One possible solution is to provide them with development models adapted to these specialties instead of asking them to write code.

The purpose of this paper is to describe the advantages expected from MDE for embedded real time systems. The paper begins with an overview

Please use the following format when citing this chapter:

Terrier, F., Gérard, S., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 15–24.

of development problems, and then describes the various stages in MDE and their advantages, using the example of a CEA-List experiment conducted for the Accord$|_{\text{UML}}$ research project.

2. MODEL MANIPULATION

Model-driven engineering relies on mastery of two essential mechanisms: abstraction and refinement. Both mechanisms partially satisfy the need for top-down and bottom-up approaches, while providing the different points of view required for system development (e.g. design and validation).

2.1 MODEL ABSTRACTION

The concept of "abstraction" is intrinsic to that of modeling, which by definition consists of representing a real world object in simplified form. This involves two possible types of abstraction – vertical and horizontal.

- *Vertical abstraction* – it follows development process flow, producing models that focus on the pertinent level of detail. There is a recurrent need, in system development, for models of standardized software (RTOS and/or middleware) and of hardware implementation platforms (e.g. POSIX, OSEK) that identify dependencies between application models and implementation choices/constraints.
- *Horizontal abstraction* – it takes place at a same level of definition and emphasizes certain system facets or complementary viewpoints. Examples include task models for RT analysis, architectural models centering on system functions and scenarios models for system testing.

For refinement purposes, the goal is to master and automate the process of building one specialized model from another. This typically means producing executable applications for example by model compilation, formalization, use of design patterns for the domain.

2.2 EXECUTABLE MODELS

Fast prototyping, evaluation and validation are vital to the development of real time embedded (RT/E) systems. To this effect, engineers need "executable" models, i.e. models whose behavior (dynamics) can be executed or simulated. More specifically, they must also be:

- *Deterministic behavior models* – RT/E applications must always behave in the same way in a given, same initial context. The semantics of the models used, and therefore, of the underlying modeling language, must be precisely defined and above all unambiguous.

- *Complete models* – A complete model contains all the data required to analyze its behavior and to generate an executable view of this data.

A model is executable if it exhibits both the above properties: *determinism and completeness.* It can be executed in different ways, e.g. via automatic code generation and simulation of the resulting code, or through formal model analysis tools that trace system operation and verify behavior.

2.3 UML FORMALISM

This section provides a brief description of the UML standard from an RT/E perspective. It then identifies the reasons why extensions to this standard, in the form of language-dedicated MDE artifacts, are necessary. Following this quick overview, the high level abstractions required to model RT concerns, i.e. concurrency and RT constraints, are presented.

2.3.1 UML PROFILE FOR EMBEDDED SYSTEMS

The Object Management Group has standard profiles (e.g. for CORBA), to model "schedulability, performance and time" (SPT [1]) and "quality of service and fault tolerance" (QoS&FT [2]). However, these profiles cannot fully support the needs of the real time domain. OMG has therefore launched a new profile, definition for this domain (MARTE, [3]), which includes the previous SPT profile and affords:

- generic concepts required to model real time aspects in both qualitative and quantitative terms (concurrency, resources, time);
- concepts required for schedulability or performance analysis on a model;
- a complete set of modeling elements to build specification and design models of embedded systems, and support the various (asynchronous and synchronous) computation models used in the RT domain;
- extensive models of standard platforms (POSIX, Arinc, OSEK, etc.).

2.3.2 OPEN POINTS

The UML2 standard deliberately leaves open a number of issues for which solutions differ, depending on domain and software design approach. These points need to be clarified to obtain complete and unambiguous models [4]. The three most important of them are:

- *Concurrency models* – the basic concept of UML is the "active object", whose only given characteristic is the ability to autonomously process certain messages, thereby introducing parallel execution.
- *Message management policy for state machines* – the basic management scheme calls for messages to be placed in a queue, then selected one at a

time to trigger a transition and undergo processing. The next message is only taken once the first has been processed. Neither the message selection criterion nor the exact system behavior on receipt of an unexpected message is defined here.

- *Description of algorithms* – UML2 proposes a conceptual framework for describing all actions and their sequences. It does not, however, define a standardized notation that is common to and directly usable for them.

Practical implementation of UML for embedded systems also requires well-defined semantic variation points and a modeling method adapted to each development stage, to ensure selection of the right modeling level, usable concept subset and writing rules.

3. BACKGROUND ON MDE RESEARCH

The CEA-List conducts studies on new methods and techniques to facilitate development of distributed real-time embedded systems (or DREs). Through regular collaboration with major industrial firms (PSA, Thales, EdF, EADS, CS-SI, etc.). Among the needs expressed there is a strong demand for assistance in dealing with the increasing complexity of systems, a wide variety of implementation options, constantly evolving functionalities and shorter times-to-market. Strong attention is also being paid to "capturing" and generalizing company development knowhow to afford and enhance reusability, not only in the last stages of implementation but also throughout the system development cycle. DREs are therefore an ideal case for studying possible automation of development steps or model analysis:

1. They intrinsically involve various points of view (e.g. functional, real-time, security, fault-tolerance), with the drawback that separation of product line generic requirements from those of particular applications is difficult (due to *a priori* implementation choices or constraints or frequent expression of requirements based on specific real time values).
2. Target implementation options vary widely: they may use different execution models for a same specification, (e.g. multitask models with RTOS, synchronous models, loop programming); they may be mapped on various platforms (single/multiple processors, shared/distributed memory...). In addition, these options use largely proprietary and ad hoc solutions and benefit only from a few viable standards.
3. Performance is often a "sensitive" issue, which cannot be dealt with practically by conventional software encapsulation techniques (multi-level interfaces may be inefficient). This can lead to strong interleaving of real time and functional codes, a critical factor where component

models for DREs must be defined without knowing the intricacies of real time and functional code operation inside the component.

4. They are often critical to testing or validation and require complete and accurate specifications and intensive use of system analysis techniques.
5. They generally require very skilled developers (specification, design, implementation, validation, integration), with a high level of expertise.

All of these requirements are an incentive for intensive use of MDE techniques throughout the system development and validation process. This means use of complementary MDE artifacts tested, implemented and evaluated in the Accord$_{|UML}$ methodology tool kit as discussed below.

4. PREVIOUS EXPERIENCE WITH MDE

Accord$_{|UML}$ [5-13] is both a conceptual framework and a method whose purpose is to assist in developing RTE applications, whereby specific design and implementation aspects are abstracted as much as possible, so that:

- Developers can concentrate on the specialist aspects of these systems (e.g. functionalities and performance constraints).
- Porting of applications is made easier by use of a first modeling level that is independent from implementation platforms.

To achieve this, Accord$_{|UML}$ is adapted to an MDE context. Accord$_{|UML}$ is based on use of UML models for abstraction and automatic and/or assisted transformation for refinement. Another of its objectives is to provide the method with modeling guidance tools. This method involves four main modeling phases: preliminary analysis, detailed analysis, validation/testing and prototyping. Progression from one to another of these phases is achieved by a continuous and iterative process of refining UML models.

The preliminary analysis phase thus consists essentially of rewriting system specifications in unambiguous form [8, 11]. Requirements are translated into use cases that are themselves broken down into scenarios depicted by sequence diagrams; these scenarios model interactions between the system, considered as a black box, and its environment. This requires very few concepts: the modeling rules (e.g. choice of diagrams and consistencies) are formalized and incorporated into the development process model as an initial profile, identified as the "Preliminary Analysis Rules".

The second phase consists of switching from a black box to a white box view in which emphasis is then placed on system content. The initial version of the resulting detailed analysis model is obtained by refining the preliminary analysis model. This model affords precise modeling of system behavior using interaction diagrams and state diagrams [6, 7, 9]. It can be transformed into a dedicated RT validation model and annotated to perform either performance analysis or scheduling analysis [14].

The prototyping model is then obtained on the same basis, by automatic generation from the previous model. This operation is performed specifically through systematic application of design patterns to key elements of the embedded system model (real time objects, automata management, real time constraints, asynchronous or synchronous communications, etc.). The result is a multitask model that can be implemented on the Accord platform [11, 12]. This platform is a "framework" for implementing the high level concepts (such as "real time object") that are manipulated via the application models. It enables complete separation of the specialty models from the implementation constraints of the target platforms.

To assist users in this modeling process, the methodology is backed up by a set of Eclipse modules (modeling and model transformation profiles), which were integrated for experimentation purposes into the IBM-Rationale modeling tool "Rationale Software Architect" (RSA).

4.1 HIGH-LEVEL MODELING CONCEPTS

To facilitate the expression of RT requirements from the onset of the embedded system modeling process, modeling concepts must be suitably detailed and independent from implementation techniques. Two types of abstractions are especially vital to RT/E modeling: parallelism and RT characteristics. To integrate these abstractions, Accord$|_{\mathrm{UML}}$ proposes [11]:

- The "RealTimeObject" stereotype: Model elements that have this stereotype are in fact considered as concurrent entities that encapsulate concurrency and state controls for the messages received for processing. This is an extension of the UML *active object* concept. Such an approach views the real time object as a task server whose RT characteristics (deadline, periodicity, etc.) are defined by the received messages triggering the tasks. The operation calls received by the object are processed by a concurrent activity.

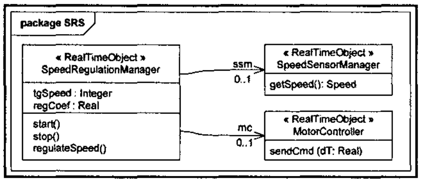


Figure 1. Typical use of RealTimeObject.

- The "RealTimeFeature" stereotype reifies the quality of service concept for modeling the RT constraints of an application. The different tag definitions associated with this concept allow modeling of qualitative RT characteristics such as deadlines, periods, or ready times. Figure 2 describes the behavior of the *SpeedRegulationManager* class, using an executable-protocol-type state machine (standard specialization of a common state machine). Addition of the "RTF" stereotype with the tagged value *period = 100 Hz* on the cyclic transition of the *On* state with itself causes this cyclic treatment to take place periodically.

Figure 3 illustrates a *startRegulating* interaction. It has the «RTF» stereotype that specifies an end-to-end deadline for said scenario, which here is *100 ms*. This means that all the activities executed in the system on receipt of the *startRegulating* message must end no later than 100 ms after the message has been received.

All of the UML extensions proposed by Accord$_{|\text{UML}}$ are thus grouped together in a language-dedicated MDE artifact – the Accord$_{|\text{UML}}$ profile.

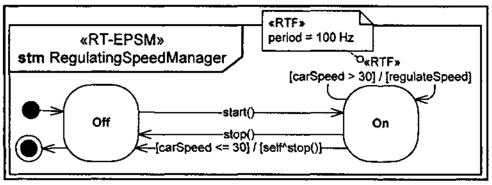

«RTF»
period = 100 Hz
«RT-EPSM»
stm RegulatingSpeedManager
«RTF»
[carSpeed > 30] / [regulateSpeed]
Off
On
start()
stop()
[carSpeed <= 30] / [self^stop()]

Figure 2. Periodic RTF applied to a state machine.

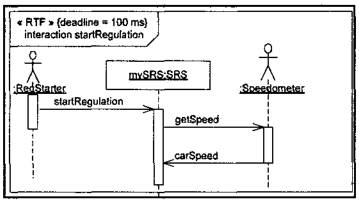


Figure 3. End-to-end Deadline.

4.2 EXECUTABLE MODELS

4.2.1 DETERMINISTIC MODELS

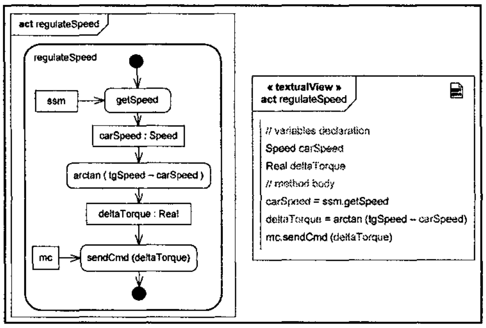
To enable construction of a deterministic model, the underlying modeling language must have both well-defined grammar (syntax) and unambiguous, deterministic semantics. The following paragraphs use three examples based on the Accord$_{|\text{UML}}$ approach to illustrate how the indeterminisms remaining in the standard are corrected by specializing the latter in a dedicated profile.

One open variation point in UML semantics is the policy set for dispatching the messages present in a state machine queue. In Accord$_{|\text{UML}}$, this variability has been eliminated, by assigning each message an RT constraint, i.e. by considering that each state machine stores received events in a mailbox and that messages are selected by comparing their RT constraints (e.g. the first message selected is the one with the earliest deadline, which is tantamount to "earliest deadline first" scheduling).

4.2.2 COMPLETE MODELS

Once model determinism has been confirmed, the second requirement for RT/E-dedicated models – completeness – must likewise be met. In our case, a complete model not only describes system structure and communications but also fully models behavior, control mechanisms and data processing actions. Our approach proposes to separate control (life cycle) aspects from data processing aspects. Control mechanisms are modeled using state machines as described above. Data processing actions are modeled using UML activity diagrams supplemented by various basic actions. Mathematical actions are modeled using MathML language syntax [15].

To meet the needs of all users, Accord$|_{AL}$ – the action language associated with Accord$|_{UML}$, proposes two formalisms to describe data processing actions, textually or graphically [13] (Figure 4). They are equivalent, and the user can switch back and forth between them according to his needs or working habits. Each action is defined as follows: semantics, textual notation (in EBNF), graphic equivalent and examples.



```
// variables declaration
Speed carSpeed
Real deltaTorque
// method body
carSpeed = ssm.getSpeed
deltaTorque = arctan (tgSpeed – carSpeed)
mc.sendCmd (deltaTorque)
```

Figure 4. Graphic and textual views of algorithm modeling.

4.3 MODEL EXECUTION

An executable model is a model that can be ran on a computer. There are two main techniques for making a model "executable":

- By a machine that incorporates model elements and can run them on a computer (this is known as "model interpretation") [14].
- By transformation, i.e. conversion of the model into a formalism that is itself executable (i.e. code generation).

5. HOW COULD MDA HELP DREs?

Five requirements have been defined for DRES development. Experience with Accord$|_{UML}$ shows that use of model-driven engineering techniques and development more than cover these requirements, as shown for each here:

- *DREs intrinsically require various points of view*: process definitions and tooling associated with UML extensions that provide high-level concepts can formalize the content of the models as well as their interdependency. It is then possible, on a given model, to ensure visibility of the requirements corresponding to implementation concerns and to easily extract or modify them without changing the rest of the model.
- *DREs implementation choices vary widely*: definition of Computation Description Models and Platform Description Models allow separation of these elements from the rest of the application model. Final implementation can then be done though dedicated transformations.
- *Performance is often a "sensitive" issue*: Code generation heuristics have shown that it is possible to both manipulate high-level concepts in the model and ensure effective implementation.
- *DREs are often critical to testing or validation*: Definition of UML extensions eliminates ambiguities and provides full semantics, thus enabling, by example, use of formal analysis tools to derive test sequences or determine the feasibility of scheduling.
- *DREs generally require highly skilled developers*: generic implementation patterns and architectures are widely used for such systems. Introduced into the development process as reusable elements, they allow easy reuse of developer know-how.

It should be noted that transition from code-oriented to model-oriented development will not alleviate the need for answers to the usual problems of traceability, configuration and version management, etc. For MDE to be successful in industry, solutions to these issues will have to be available for all tools claiming to be MDE-compliant!

Definition and development of a set of MDE artifacts lies at the core of new large-scale joint research programs as for example:

- The *CARROLL* program involving CEA, INRIA and Thales (www.carroll-research.org), whose goal is to provide tools for model-driven engineering and component-based middleware.
- The *Software Factory* project of the System@tic competitiveness cluster (www.systematic-paris-region.org) federates more than 40 partners in research on MDE, validation and execution. This project will provide an open source platform supporting MDE for embedded systems.

REFERENCES

[1] OMG, "UML Profile for Schedulability, Performance, and Time, v1.1," formal/05-01-02, 2005.

[2] OMG, "UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics & Mechanisms," ptc/04-09-01, 2004.

[3] OMG, "UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP", realtime/05-02-06.

[4] S. Gérard and F. Terrier, "UML for Real-Time," chapter in "UML for Real: Design of Embedded Real-Time Systems," L. Lavagno, G. Martin, and B. Selic, editors, Kluwer Academic Publishers, Boston, 2003, p. 369.

[5] F. Terrier and S. Gérard, "Real Time System Modeling with UML: Current Status and Some Prospects," in 2nd Workshop on SDL and MSC, 2000, Grenoble, France.

[6] A. Lanusse, S. Gérard, and F. Terrier, "Real-Time Modeling with UML : The ACCORD Approach," in "UML'98 : Beyond the Notation," 1998, Mulhouse, France, J. Bezivin et P.A. Muller eds.

[7] S. Gérard, N. S. Voros, C. Koulamas, and F. Terrier, "Efficient System Modeling of Complex Real-time Industrial Networks Using The *ACCORD/UML* Methodology," presented at Architecture and Design of Distributed Embedded Systems (DIPES 2000), B. Kleinjohann, Kluwer Academic Publishers, p. 10, Paderborn University, Germany, October 18-19 2000.

[8] S. Gérard, F. Terrier, and Y. Tanguy, "Using the Model Paradigm for Real-Time Systems Development: ACCORD/UML," presented at OOIS'02-MDSD, J.-M. Bruel and Z. Bellahsène eds., Springer, pp 260-269, Montpellier, September 2002.

[9] C. Mraidha, S. Gérard, F. Terrier, and J. Benzakki, "A Two-Aspect Approach for a Clearer Behavior Model," presented at the 6th IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC'2003), T. N. P. Puschner, A. Ghafoor eds., IEEE Computer Society, ISBN 0-7695-1928-8, pp 213-220, Hakodate, Hokkaido, Japan, 14-16 May 2003.

[10] P. Tessier, S. Gérard, C. Mraidha, F. Terrier, and J.-M. Geib, "A Component-Based Methodology for Embedded System Prototyping," presented at 14th IEEE International Workshop on Rapid System Prototyping (RSP'03), IEEE Computer Society, ISBN 0-7695-1943-1, pp 9-15, San Diego, USA, 9-11 June 2003.

[11] S. Gérard, C. Mraidha, F. Terrier, and B. Baudry, "A UML-Based Concept for High Concurrency: the Real-Time Object," presented at The 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'2004), T. A. a. I. L. J. Gustafsson, IEEE Computer Society, ISBN 0-7695-2124-X, pp 64-67, Vienna, Austria, 12-14 May 2004.

[12] N. Guelfi, A. Schoos, S. Gérard, and F. Terrier, "EUDEMES: Component-Based Development Methods for Small-Size Embedded Systems," ERCIM NEWS, 2003, vol. 52 (Embedded Systems), p. 64

[13] C. Mraidha, S. Gérard, Y. Tanguy, H. Dubois, and R. Schneckenburger, "Action Language Notation for Accord/UML," CEA DTSI/SOL/LLSP/04-163/HD, 2004.

[14] D. Lugato, N. Rapin, and J.-P. Gallois, "Verification and tests generation for SDL industrial specifications with the AGATHA," presented at Workshop on Real-Time Tools, CONCUR'01, pp, 2001.

[15] W3C, "Mathematical Markup Language (MathML) Version 2.0 (Second Edition)," http://www.w3.org/TR/2003/REC-MathML2-20031021/, 2003.

REIFYING THE SEMANTIC DOMAINS OF COMPONENT CONTRACTS

Jean-Marc Jézéquel
Irisa (INRIA & University of Rennes 1)[*]

Abstract In domains such as automotive or avionics, software cannot any longer be produced as a single chunk, and engineers are contemplating the possibility of componentizing it. A component only exhibits its provided or required interfaces, which must be enriched to take into account extra-functional aspects. This defines multi-level *contracts* between components allowing one to properly wire them. Instead of defining an integrated language only making available a limited set of concepts for modeling extra-functional aspects, we propose to handle open-ended modeling of extra-functional aspects of real-time and embedded systems, based on meta-modeling techniques and Model Driven Engineering (MDE) for reifying their semantics. Then the designer can use off-the-shelf tools to perform various kinds of design time analysis.

1. INTRODUCTION

In domains such as automotive or avionics, products are characterized by high performance, high dependability, outstanding quality demands, and exponentially increasing complexity. Since these real-time and embedded systems are getting ever more software intensive, their software cannot any longer be produced as a single chunk. Automotive or avionics engineers are thus contemplating the possibility of componentizing it, along the lines of Szyperski's [12] ideas, where

> *a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third-party.*

In real-time and embedded systems however, we have to take into account many extra-functional aspects, such as timeliness, memory con-

*This work has been partially supported by the Artist2 Network of Excellence on Embedded Systems Design (IST-004527).

sumption, power dissipation, reliability, performances, and generally speaking Quality of Service (QoS). These aspects can also be seen as *contracts* [9] between the system, its environment and its users. These contracts must obviously be propagated down to the component level. One of the key desiderata in component-based development for embedded systems is thus the ability to capture both functional and extra-functional properties in component contracts, and to verify and predict corresponding system properties [11].

To master the complexity of modern real-time and embedded systems, engineers must rely on *modeling* for representing several aspects of reality for some purposes, such as thorough quality and stability assessment at early design stages or active treatment of design assumptions to guide system development. However the extra-functional aspects that must be taken into account are so various and domain specific (and even vary very much company-wide inside the same domain) that there is no integrated modeling language encompassing them all.

We propose an alternate way to handle open-ended modeling of extra-functional aspects of real-time and embedded systems, based on meta-modeling techniques and Model Driven Engineering (MDE) [3]. Instead of defining an integrated language only making available a limited set of concepts for modeling extra-functional aspects, we propose to let the designer define as many modeling sub-languages as needed for describing QoS contracts. These sub-languages are defined as executable meta-models: their abstract syntax are defined as plug-in extensions to the UML2.0 meta-model, their static semantics are given with a set of OCL constraints, and their dynamic semantics are expressed with Kermeta [10]. Once the semantic domains of component contracts has been reified this way, the designer can use off-the-shelf tools (such as model-checkers or constraint solvers) to perform various kinds of design time analysis.

The rest of the paper is organized as follows. Section 2 details the notion of *contract* along the four levels defined in [2]. Section 3 presents how they can be integrated at meta-modeling level. Section 4 discusses the problem of composing the components and computing the QoS properties of the assembly. Section 5 discusses how various kinds of design time analysis can then be performed. Section 6 discusses related works, and we finish by some conclusions and perspectives.

2. FOUR LEVELS OF COMPONENT CONTRACTS

The term contract can very generally be taken to mean "component specification" in any form. This specification should tells us what the component does without entering into the details of how.

A contract is in practice taken to be a constraint on a given aspect of the interaction between a component that supplies a service, and a component that consumes this service [9]. Component contracts differ from object contracts in the sense that to supply a service, a component often explicitly requires some other service, with its own contract, from another component. So the expression of a contract on a component-provided interface might depend on another contract from one of the component-required interfaces.

That is also known as the *assume/promise* approach where each component has a black-box model, which explicates assumptions about its environment and state corresponding promises on the service offered by the component to the environment (e.g.; promising state-dependent latencies of component services as a function of assumed service times of invoked services). Similarly, bounds on the occurrence of critical events can be promised only based on failure rates for invoked services, assumptions on failure rates for an execution platform, and knowledge of the ways the component itself can induce and propagates failures.

A now widely accepted classification of different kinds of contracts has been proposed in [2], where a contract hierarchy is defined consisting of four levels.

Level 1 : Syntactic interface, or signature (i.e. types, fields, methods, signals, ports etc., which constitute the interface). The syntactic interface of a component is a list of operations or ports, including their signatures (the types of allowed inputs and outputs), by means of which communication with this component is performed. These Level 1 contracts allow static type checking, that is a verification that there is no possibility of interaction errors (i.e. messages not understood).

Level 2 : Constraints on values of parameters and of persistent state variables, expressed, e.g., by pre- and post-conditions and invariants. Functional properties are used to achieve more than just interoperability. Level 2 contracts are about the actual values of data that are passed between components through the interfaces, whose syntax is specified at Level 1. Typical properties of interest are constraints on their ranges, or on the relation between the parameters of a method call and its return value.

Level 3 : Synchronization between different services and method calls (e.g., expressed as constraints on their temporal ordering). Level 3 contracts are about the actual ordering between different interactions at the component interfaces. They allow one to express explicit control information, which makes the expression of complex, history dependent input/output relations much easier.

Level 4 : Extra-functional properties, such as performance, memory consumption, constraints on response times, throughput, etc..

A quality of a system (e.g.; memory consumption) can in general be considered as a function mapping a given system instance with its full behavior onto some scale. The scale may be qualitative, in particular it may be partially or totally ordered, or the scale can be quantitative (as for memory consumption), in which case the quality is a measure. The problem of realizing systems that have certain guaranteed qualities, also known as their quality of service (QoS), involves the representation of such qualities in design models or languages and techniques to implement and analyze them as properties of implemented system instances.

There exist many QoS contracts languages which allow the designer to specify the extra-functional properties and their constraints on the provided interfaces only. However, few of them allow specifying dependency relationships between the provided and required services of a component (e.g.; the memory consumption of a component may depend on some function of its required interfaces). If we want an open-ended way of letting the designer specify such QoS dimensions, we need to provide a meta-model infrastructure allowing for both:

- the description of the various types of contracts and their dependencies at the modeling level,
- the specification of a mapping function from the syntactic domain of the contract to its semantic domain.

For example, to continue with our simple example of memory consumption, in the case of a simple centralized memory system, the operational semantic of memory can be defined as a class with an integer attribute modeling the amount of memory already consumed, and two operations, *alloc* and *free*, each taking an integer parameter.

3. META-MODELING COMPONENT CONTRACTS

These four levels of contracts should fully describe all the visible properties of components, whatever their actual implementation. Model based engineering is the idea that working with models of such component can be useful to perform a range of engineering tasks, such as prototyping, dimensioning, validation, and even code or test generation. While the levels 1 to 3 are supported in a number of ways in many modeling languages (e.g.; SDL, Lotos, as well as various flavors of automata-based languages) the aspects that must be taken into account at level 4 are so various and domain specific that there can probably be no integrated modeling language encompassing them all.

We thus propose an alternate way to handle open-ended modeling of extra-functional aspects of real-time and embedded systems, based on meta-modeling techniques and Model Driven Engineering (MDE) [3]. Instead of defining an integrated language only making available a limited set of concepts for modeling extra-functional aspects, we propose to let the designer define as many modeling sub-languages as needed for describing QoS contracts.

At the abstract syntax level, these open-ended sub-languages are linked to a component meta-model playing the role of a backbone. We use a subset of the UML2.0 meta-model for describing component-related notions, as follows:

Level 1 The structural part of a component type is defined by a set of port types. Each port type is identified by a name and a set of provided and required UML2 interfaces. Each interface groups a set of services. Composite component types also contain a slot for each sub component they contain.

Level 2 For describing functional aspects, we can just reuse the OCL (Object Constraint Language), hence providing means for describing partial functions or relations by means of invariants, pre- and postconditions.

Level 3 In the description of a primitive component type we include an abstraction of its behavior, based on the UML2.0 State-Chart formalism. Since composite component types must delegate all their ports, they do not contain any behavior.

Level 4 Since our extra-functional contracts would be used on software components with explicit dependency specification, we need means to express a provided contract in terms of required contracts. In the most general case, a component may bind together its provided contracts with its required contracts as an explicit set of equations (i.e. how offered QoS is related to required QoS).

Therefore, the meta-model for Level 4 contracts is made of the following concepts:

- expression of QoS spaces (dimensions, units);
- primitives bindings between these spaces and the levels 1-3 modeling elements (bindings to observable events, conversion from discrete event traces to continuous flows, definition of measures);
- constraint languages on the QoS spaces (defining the operations that can be used in the equations, form of these equations).

The declarative nature of Level 4 contract will make them suitable to various kinds of design-time analysis, including solving them with Constraint Logic Programming techniques [4].

At each level, these meta-models can be enriched with well-formedness rules expressed with the OCL, hence providing some elements of static semantics. Note that all these definitions are made at the component *type* level. UML2.0 indeed rightly distinguishes between component *types* and component *instances*, which are deployed into particular configurations, called *component assemblies.*

4. COMPONENT COMPOSITION

At the syntactic level, component composition is easy: the designer just has to wire required interfaces to provided interfaces of component into a particular component assembly. The overall system can even be closed if we are able to provide a model of the environment, which is formally seen as just another component. Things are getting a little bit more interesting at the semantic level, which should answer the following question "what is the global behavior of the assembly on each of the 4 levels that have been defined above?". At the behavioral level, that's not too difficult a question, because the global behavior of a component assembly can often be described as the parallel composition of the component state-charts, with some synchronization on input/outputs. However the question is more complex for level 4. Ideally, for any components (C_x, C_y), and for any interesting property P of the components, the composition operator o should have the following property: $P(C_x o C_y) = P(C_x) o P(C_y)$

This ideal composition operator might however be difficult to define for two different kinds of reasons.

- real system level composition operators are non trivial. This might be overcome by modeling these composition operators as components relying on a few set of primitive operators. In practice however, it is tedious to reverse engineer complex component frameworks such as .NET or real-time buses.

- the mere nature of the composition meaning depends on the property of interest. For instance, while memory consumption MC is clearly additive among components ($MC(C_x o C_y) = MC(C_x) + MC(C_y)$), this is seldom the case for other quality attributes (e.g; reliability). Furthermore, if we want to allow open-ended level 4 contracts, we need the designer to express the meaning of composition on new quality attributes.

We thus prefer to keep all aspects separate in the semantics, or more precisely to define one semantic domain for each aspect, and then let the designer explicitly define the meaning of the composition with a set of projections o_i of the composition operator on each aspect: $\forall i \in aspect, P_i(C_x o_i C_y) = P_i(C_x) o_i P_i(C_y)$

We propose an operational way of handling this projection, which is based on the reification of the semantic domain of each QoS dimension. As we have seen before with our simple example of memory consumption, the operational semantic of memory can be defined as an integer that can be incremented or decremented as memory is allocated or freed. Then we can use an Aspect-Oriented Programming (AOP) kind of approach to "weave" this memory consumption aspect into the global behavior of the component assembly which is given by the parallel composition of component state-charts. Each time a component service with a memory consumption related contract is called, we get a side effect which is a call to the relevant operation on the reification of the memory QoS dimension. In AOP terms, the contract plays the role of a point-cut, while the state-chart transition holding the service call is the join point.

We then need a tool for (1) describing the operational semantics of extra-functional aspects and (2) implementing this weaving at modeling time, for model analysis purposes. We have developed Kermeta exactly for this kind of problems.

Kermeta [10] is an open source meta-modeling language developed by the Triskell team at IRISA. It has been designed as an extension to the EMOF 2.0 to be the core of a meta-modeling platform. Kermeta extends EMOF with an action language that allows specifying semantics and behavior of meta-models. The action language is imperative and object-oriented. It is used to provide an implementation of operations defined in meta-models. As a result the Kermeta language can, not only be used for the definition of meta-models but also for implementing their semantics, constraints and transformations, as well as weave extra-functional aspects [8] into base models (e.g. memory consumption within the component model).

5. APPLICATIONS

Since the projection of a component assembly onto any QoS dimension can be seen at the semantic level as a system of non-linear constraints that must be satisfied, we can foresee several ways of exploiting this information at design time, with (1) Constraint Logic Programming (CLP) techniques (2) Model Checking and (3) Simulation.

For any QoS dimension, the constraints attached to a component can be translated into a specific CLP-compliant language, using a model transformation techniques as described in [5]. Then for a single component, we can use a CLP(R) based constraint solver to get ranges for admissible values for the QoS properties of the component. It might also allow an early detection of incompatibilities among component with respect to QoS properties. Similarly, since the component assembly is being seen a a complex system of constraints, it can be solved in two directions, either bottom-up or top-down. Knowing the value ranges for QoS properties of the deployment platform, the system of non-linear constraints can be solved bottom-up to obtain end-to-end QoS value ranges. Conversely, based on wanted operational value ranges for QoS properties of the component assembly, the system of non-linear constraints can be solved top-down to obtain dimensioning information for the deployment platform.

The idea of using model-checking techniques is to run an exhaustive co-simulation of the various semantic domains. Since the semantic domains of the QoS dimensions have been reified and woven into the global behavior of the component assembly, QoS dimensions such as memory consumption are now part of the global state of the system as seen by a model checker. Since the model checker needs no knowledge at all of this fact, we can easily reuse off-the-shelf model checkers and obtain results on e.g.; the exact bounds on memory consumption of the component assembly. However, it is clear that if our model checking tool relies on enumeration techniques for the state space exploration, we are bound for trouble if our QoS semantic domains are based on unbounded integers or even worse, floating point real numbers. This limitation can be somehow overcome either by bounding QoS semantic domains, or by using recent progress on symbolic model checking. In the latter case, it would however limit our level 4 contracts to constraint expressed with simple arithmetics.

Simulation can be seen as a non-exhaustive exploration of the component assembly semantic domain, again including the reification of the QoS dimensions. Then, given a sets of operational profiles, we can get interesting statistics on typical distributions of end-to-end QoS values.

We actually see these various techniques as complementary. Once the designer has described her component assembly, then a set of complementary analysis can be performed depending on the characteristics of the application, as well as the availability of tools.

6. RELATED WORKS

In the Component-Based Software Engineering community, the concept of predictability [6] is getting more and more attention, and is now underlined as a real need (see for instance Predictable Assembly from Certifiable Components (PACC) initiative promoted at the SEI [7]). At modeling level, the Object Management Group (OMG) has developed its own UML profiles for QoS and for schedulability, performance and time specification, and it is still working in this domain with the MARTE RFP. In these works however, the semantic domain of extra-functional properties is either hard-coded or implicit. The interest of our approach is to make their semantics both open-ended and explicit at the meta-model level.

In most of these approaches, a QoS property is specified as a constant: they do not allow the specification of QoS properties dependency relationships. In contrast, Reussner proposes parameterized contracts [11]: the set of available services provided by a component depends on its required services that the context can provide. We actually follow the same line, just making it more flexible an open-ended with executable meta-modeling techniques.

The Metropolis meta-model [1] also allows capturing extra-functional aspects of design by so-called quantity managers, and provides means for declarative specification of extra-functional constraints through its constraints logic. Our approach follows the same line, but starts from a different context where component contracts are explicitly modeled to allow assume/promise reasoning.

7. CONCLUSION

In the Component-Based Software Engineering community, the concept of predictability is getting more and more attention: how component technology can be extended to achieve predictable assembly, enabling runtime behavior to be predicted from the properties of components. We have proposed to handle open-ended modeling of extra-functional aspects of real-time and embedded systems, based on meta-modeling techniques and Model Driven Engineering. We let the designer define as many modeling sub-languages as needed for describing QoS contracts. These sub-languages are defined as executable meta-models: their abstract syntax are defined as plug-in extensions to the UML2.0 meta-model, their static semantics are given with a set of OCL constraints, and their dynamic semantics is expressed with Kermeta in order to be woven into the base behavioral model of the component assembly. Once the semantic domains of component contracts has been reified this way, the designer can use off-the-shelf tools (such as model-checkers or constraint solvers) to perform various kinds of design time analysis.

REFERENCES

[1] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Paserone, and A. Sangiovanni-Vincentelli. Metropolis: an integrated electronic system design environment. *IEEE Computer*, 36(4), April 2003.

[2] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins. Making components contract aware. *IEEE Computer*, 13(7), July 1999.

[3] Jean Bézivin, Nicolas Farcet, Jean-Marc Jézéquel, Benot Langlois, and Damien Pollet. Reflective model driven engineering. In G. Booch P. Stevens, J. Whittle, editor, *Proceedings of UML 2003*, volume 2863 of *LNCS*, pages 175–189, San Francisco, October 2003. Springer.

[4] Olivier Defour, Jean-Marc Jézéquel, and Nol Plouzeau. Applying CLP to predict extra-functional properties of component-based models. In J. S. de Boer, editor, *Proceedings of Logic Programming: 20th International Conference, ICLP 2004*, number 3132 in LNCS. Springer Heidelberg, September 2004.

[5] Olivier Defour, Jean-Marc Jézéquel, and Nol Plouzeau. Extra-functional contract support in components. In *Proc. of International Symposium on Component-based Software Engineering (CBSE7)*, May 2004.

[6] Aagedal J.O. *Quality of service support in development of distributed systems.* PhD thesis, University of Oslo, Dept. Informatics, March 2001.

[7] Wallnau K. Volume iii: A technology for predictable assembly from certifiable compo-nents. Technical Report CMU/SEI-2003-TR-009, SEI, 2003.

[8] Jacques Klein, Loic Hélouet, and Jean-Marc Jézéquel. Semantic-based weaving of scenarios. In *proceedings of the 5th International Conference on Aspect-Oriente d Software Development (AOSD'06)*, Bonn, Germany, March 2006. ACM.

[9] B. Meyer. Applying design by contract. *IEEE Computer (Special Issue on Inheritance & Classification)*, 25(10):40–52, October 1992.

[10] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented meta-languages. In S. Kent L. Briand, editor, *Proceedings of MODELS/UML'2005*, volume 3713 of *LNCS*, pages 264–278, Montego Bay, Jamaica, October 2005. Springer.

[11] Reusnerr R.H., Schmidt H.W., and Poernomo I.H. Reliability prediction for component-based software architecture. *Journal of Systems and Software*, 66:241–252, 2003.

[12] C. Szyperski. *Component software, beyond object-oriented programming.* Addison-Wesley, 2nd edition, 2002.

MODEL-BASED TEST SELECTION FOR INFINITE STATE REACTIVE SYSTEMS *

Thierry Jéron
Irisa/Inria Rennes, Campus de Beaulieu, 35042 Rennes, France
Thierry.Jeron@irisa.fr

Abstract We address the problem of off-line selection of test cases for testing the conformance of a black-box implementation with respect to a specification of a reactive systems. Efficient solutions to this problem have been proposed in the context of finite-state models, based on the **ioco** conformance testing theory. We extend them in the context of infinite state specifications, modelled as automata extended with variables. We consider the selection of test cases according to test purposes describing abstract scenarios that one wants to test. The selection of program test cases then consists in syntactical transformations of the specification model, using approximate analysis.

1. INTRODUCTION AND MOTIVATION

Testing is the most used validation technique to assess the correctness of reactive systems. For more than a decade, *model-based testing* (see e.g. [1]) advocates the use of models to formalize this validation activity. The formalization relies on precise models of specifications, implementations and test cases, a formal definition of correctness, required properties of test cases with respect to correctness, and test generation algorithms.

In this paper we address the generation of test cases in the framework of conformance testing of reactive systems [6]. Conformance testing consists in checking that a black-box implementation of a system, only known by its interactions with the environment, behaves correctly with respect to its specification. Conformance testing then relies on experimenting the system with test cases, with the objective of detecting some faults with respect to the specification's external behaviour.

We consider models of reactive systems, called Input/Output Symbolic Transition Systems (ioSTS), which are automata extended with variables, with dis-

*This paper is partly based on [11, 8, 12].

Please use the following format when citing this chapter:

Jéron, T., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 35–44.

tinguished input and output actions, and corresponding to reactive programs without recursion. Their semantics can be defined in terms of infinite state Input/Output Labelled Transition Systems (ioLTS). For ioLTS, the **ioco** testing theory [13] defines conformance as a partial inclusion of external behaviours (suspension traces) of the implementation in those of the specification. Several research works have considered this testing theory and propose test generation algorithms. We focus on off-line test selection where a test case is built from a specification and a test purpose (representing abstract behaviours one wants to test), and further executed on the implementation. Test cases are built directly from the ioSTS model rather than constructing test cases from the enumerated ioLTS semantic model. This construction relies on syntactic transformations of the specification model, guided by an approximate analysis of the set of states co-reachable from a set of final state.

2. ioSTS: A MODEL OF REACTIVE SYSTEMS

Syntax of the ioSTS model. We propose a model called ioSTS for *Input/Output Symbolic Transition Systems.* It extends labelled transition systems for modelling imperative programs without recursion and communicating with their environment. An ioSTS is made of variables, input and output actions carrying communication parameters carried by actions, guards and assignments. In ioSTS, a data d (variable or communication parameter) has a type $type(d)$, with values in $Dom(d)$. For a data set $B = \{d_1, \ldots, d_n\}$, we note $Dom(B) = Dom(d_1) \times \cdots Dom(d_n)$. A predicate ϕ (e.g. a guard) on a data set B defines the subset of vectors in $Dom(B)$ satisfying ϕ.

DEFINITION 1 (IOSTS) *An input/output symbolic transition system (*ioSTS*) is a tuple* $\mathcal{M} = \langle D, \Theta, L, l^0, \Sigma, \mathcal{T}\rangle$ *where*

- $D = V \cup P$ *is a finite set of* data *partitionned into* variables V *and* communication parameters P. *We note* $\mathcal{V} = Dom(V)$ *and* $\Pi = Dom(P)$.
- Θ *called the* initial condition *is a predicate on variables* V.
- L *is a finite set of* locations, *with* $l^0 \in L$ *the* initial location.
- $\Sigma = \Sigma^? \cup \Sigma^!$ *is the* finite alphabet of actions *partitionned into* inputs $\Sigma^?$ *and* outputs $\Sigma^!$ [1] . *An action* $a \in \Sigma$ *is characterized by its* signature $sig(a) = \langle p_1, \ldots, p_k\rangle \in P^k$ *specifying types of communication parameters carried by the action* a. *We note* $\Pi_a = Dom(sig(a))$.
- $\mathcal{T}$ *is a finite set of* symbolic transitions. *A transition is a tuple* $t = \langle l, a, G, A, l'\rangle$ *defined by: its* origin *and* destination *locations* l *and* $l' \in L$; *an* action $a \in \Sigma$; *a* guard G *is a predicate on* $V \cup sig(a)$; *an* assignment A, *of the form* $(x := A^x)_{x \in V}$ *such that, for each* $x \in V$, A^x *is an*

[1] The general model also considers internal actions

expression on $V \cup sig(a)$ *defining the evolution of variables* [2]. *We note* Id_V *the identity assignment* $(x := x)_{x \in V}$.

Semantics of ioSTS. The semantics of $\mathcal{M} = \langle D, \Theta, L, l^0, \Sigma, \mathcal{T} \rangle$ is an input/ouput labelled transition system (ioLTS) $[\![\mathcal{M}]\!] = \langle Q, Q^0, \Lambda, \rightarrow \rangle$, where:

- $Q = L \times V$ is the set of states and $Q^0 = l^0 \times \Theta$ its subset of initial states;
- $\Lambda = \Lambda^? \cup \Lambda^!$ s.t. for $\# \in \{?, !\}$, $\Lambda^\# = \{\langle a, \pi \rangle | a \in \Sigma^\#, \pi \in \Pi_a\}$ is the set of *valued actions* partitionned into *valued inputs* $\Lambda^?$, and *outputs* $\Lambda^!$,
- $\rightarrow \subseteq Q \times \Lambda \times Q$ is the smallest relation defined by the following rule:

$$\frac{\langle l, \nu \rangle, \langle l', \nu' \rangle \in Q \quad \langle a, \pi \rangle \in \Lambda \quad t = \langle l, a, G, A, l' \rangle \in \mathcal{T} \quad G(\nu, \pi) = true \quad \nu' = A(\nu, \pi)}{(\langle l, \nu \rangle, \langle a, \pi \rangle, \langle l', \nu' \rangle) \in \rightarrow}$$

Intuitively, the ioLTS semantics of an ioSTS enumerates all possible *states* (pairs $q = \langle l, \nu \rangle$ composed of a location and the vector of values of variables) and *valued actions* (pairs $\alpha = \langle a, \pi \rangle$ composed of an action and the vector of values of its communication parameters) between states. The rule means that in a state $\langle l, \nu \rangle$, a transition $t = \langle l, a, G, A, l' \rangle$ is fireable if there exists a valuation π of $sig(a)$ such that G evaluates to $true$ for ν and π. The system then moves from with the action $\langle a, \pi \rangle$ to a state $\langle l', \nu' \rangle$ where ν' is the new valuation of variables obtained from ν and π by the assignment A.

As usual for ioLTS, we note $q \xrightarrow{\alpha} q'$ for $(q, a, q') \in \rightarrow$. For a sub-alphabet $\Lambda' \subseteq \Lambda$, we say that M is Λ'-*complete* in a state q if $\forall \alpha \in \Lambda' : q \xrightarrow{\alpha}$. An ioSTS is *deterministic* if Θ has a unique solution and in each location l, for all action a, for all pairs of transitions starting in l and carrying a, the conjonction of their guards is empty.

A *run* of an ioSTS $\mathcal{M}$ is an alternate sequence of states and valued actions $\rho = q_0\alpha_0q_1 \ldots \alpha_{n-1}q_n \in Q^0.(\Lambda.Q)^*$ s. t. $\forall i, q_i \xrightarrow{\alpha_i} q_{i+1}$. ρ is *accepted in* $F \subseteq Q$ if $q_n \in F$. *Runs*$(\mathcal{M})$ (resp. *Runs*$_F(\mathcal{M})$) denotes the set of runs (resp. accepted runs in F) of $\mathcal{M}$. When modelling the testing activity, we need to abstract away states which are not observable from the environment. A *trace* of a run $\rho \in Runs(\mathcal{M})$ is the projection $proj_\Lambda(\rho)$ of ρ on actions.b *Traces*$(\mathcal{M}) \triangleq proj_\Lambda(runs\mathcal{M})$ denotes the set of traces of $\mathcal{M}$ and *Traces*$_F(\mathcal{M}) \triangleq proj_\Lambda(Runs_F(\mathcal{M}))$ is the set of traces of runs accepted in F.

Visble behaviour for testing. During conformance testing, the tester stimulates inputs of the system under test, and observes not only its outputs, but also its *quiescences* (absence of output) using timers, assuming that timeout values are large enough such that, if a timeout occurs, the system is indeed

[2]The scope of parameters is limited to one transition

quiescent. The tester should be able to distinguish between specified and unspecified quiescence. But as trace semantics does not preserve quiescence in general, possible quiescence should be made explicit on the specification by a transformation called *suspension* [13]. This consists in adding a self-loop labelled with a new output δ in each quiescent state. We define suspension for ioSTS as follows. For an ioSTS $\mathcal{M} = \langle D, \Theta, L, l^0, \Sigma = \Sigma^! \cup \Sigma^?, \mathcal{T}\rangle$, the *suspension* of $\mathcal{M}$ is the ioSTS $\Delta(\mathcal{M}) = \langle D, \Theta, L, l^0, \Sigma^\delta = (\Sigma^! \cup \{\delta\}) \cup \Sigma^?, \mathcal{T}_\delta, \rangle$ with $\mathcal{T}_\delta = \mathcal{T} \cup \{\langle l, \delta, G_{\delta,l} Id_V, l\rangle \mid l \in L\}$ and

$$G_{\delta,l} = \neg \bigvee_{\langle l,a,G,A,l'\rangle \in \mathcal{T},\ a \in \Sigma^!} \exists \pi \in \Pi_a . G(a, \pi)$$

For an ioSTS $\mathcal{M}$ modelling a system, the behaviour considered for testing is then $\mathit{STraces}(\mathcal{M}) \triangleq \mathit{Traces}(\Delta(\mathcal{M}))$.

3. CONFORMANCE TESTING THEORY

We now reformulate the **ioco** testing theory from [13]. It mainly consits in defining models for specifications, implementations and test cases, defining conformance, test executions and verdicts.

Conformance relation. We assume that the specification is an ioSTS $\mathcal{S}$, and that the behaviour of the unknown implementation could be modelled by an (non-deterministic) ioLTS $I = \langle Q_I, Q_I^0, \Lambda^! \cup \Lambda^?, \rightarrow_I\rangle$ with same interface. We also assume that I is $\Lambda^?$-complete [3]. The conformance relation then defines the set of correct implementation models:

I **ioco** $\mathcal{S} \triangleq \mathit{STraces}_{\neg ioco}(\mathcal{S}) \cap \mathit{STraces}(I) = \emptyset$

where $\mathit{STraces}_{\neg ioco}(\mathcal{S}) = \mathit{STraces}(\mathcal{S}) \cdot (\Lambda^! \cup \{\delta\}) \setminus \mathit{STraces}(\mathcal{S})$.

$\mathit{STraces}_{\neg ioco}(\mathcal{S})$ exactly represents the set of non-conformant behaviours: I is non-conformant as soon as it may exhibit a suspension trace of $\mathcal{S}$ prolongated with an unspecified outputs or quiescence. Interestingly, our formulation of **ioco** explicits the fact that conformance is a safety property of I: conformance is violated if one exhibits a finite trace of I in $\mathit{STraces}_{\neg ioco}(\mathcal{S})$. If I was known, verifying conformance would then amount to building a deterministic *non-conformance observer* $\mathit{can}(\mathcal{S})$ equiped with a set of states **Fail** such that $\mathit{Traces}_{\mathbf{Fail}}(\mathit{can}(\mathcal{S})) = \mathit{STraces}_{\neg ioco}(\mathcal{S})$, computing the synchronous product of I and $\mathit{can}(\mathcal{S})$ and checking whether **Fail** is reachable.

However, as I is unknown, one can only experiment it with selected test cases, providing inputs and checking that outputs and quiescences of I are specified in $\mathcal{S}$. This entails that, except in simple cases, conformance cannot be proved by testing, only non-conformance witnesses can be exhibited.

[3]This ensures that the composition of I with a test case TC never blocks because of non-implemented inputs.

Test cases, test executions and verdicts. In our modelling framework, we aim at building test cases in the form of ioSTSA test case for the specification ioSTS $\mathcal{S}$ is a deterministic ioSTS $\mathcal{TC} = \langle D = V \cup P, \Theta_{TC}, L_{TC}, l^0_{TC}, \Sigma_{TC} = \Sigma^!_{TC} \cup \Sigma^?_{TC}, \mathcal{T}_{TC}\rangle$ with $\Sigma^!_{TC} = \Sigma^?$ and $\Sigma^?_{TC} = \Sigma^! \cup \{\delta\}$ (actions are mirrored w.r.t. $\mathcal{S}$) with semantics $[\![\mathcal{TC}]\!] = TC$. $\mathcal{TC}$ is equipped with a collection of sets of verdict locations Verdict partitionned into **Fail** (meaning rejection), **Pass** (meaning that targetted behaviours have been reached) and **Inconc** (meaning that targetted behaviours cannot been reached anymore). We also call Verdict the collection of sets of states of TC where the location is in Verdict. We assume that Verdict states are trap states (with no transitions) and that all states except Verdict ones are $\Lambda^?_{TC}$-complete.

We model the execution of a test case $\mathcal{TC}$ on an implementation I by the *parallel composition* of $TC = [\![\mathcal{TC}]\!]$ with $\Delta(I)$ (quiescences of I are observed) with synchronization on common actions. Let $\Delta(I) = \langle Q_I, Q^0_I, \Lambda^! \cup \{\delta\} \cup \Lambda^?, \rightarrow_{\Delta(I)}\rangle$ and $TC = \langle Q_{TC}, q^0_{TC}, \Lambda^? \cup \Lambda^! \cup \{\delta\}, \rightarrow_{TC}\rangle$, $\Delta(I)\|TC$ is the ioLTS $(Q_I \times Q_{TC}, Q^0_I \times \{q^0_{TC}\}, \Lambda^! \cup \{\delta\} \cup \Lambda^?, \rightarrow_{\Delta(I)\|TC})$ where, for $\alpha \in \Lambda^! \cup \{\delta\} \cup \Lambda^?$, $(q_1, q'_1) \xrightarrow{\alpha}_{\Delta(I)\|TC} (q_2, q'_2)$ iff $q_1 \xrightarrow{\alpha}_{\Delta(I)} q_2$ and $q'_1 \xrightarrow{\alpha}_{TC} q'_2$.

The possible rejection of I by TC is defined by the fact that $\Delta(I)\|TC$ may lead to **Fail** in TC: TC *mayfail* $I \triangleq Traces_{Q_I\times\mathbf{Fail}}(\Delta(I)\|TC) \neq \emptyset$ which is equivalent to $Traces(\Delta(I)) \cap Traces_{\mathbf{Fail}}(TC) \neq \emptyset$.

Now, test generation algorithms should produce test cases with properties relating rejection with non-conformance. Formally, let TS be a set of test cases. We say that TS is *complete* if it is both *correct* and *exhaustive* where:

TS is *correct* $\triangleq \forall I, (I$ **ioco** $S \implies \forall TC \in TS, \neg TC$ *mayfail* $I)$.

i.e. only non-conformant implementations can be rejected by a test case in TS.

TS is *exhaustive* $\triangleq \forall I, (\neg(I$ **ioco** $S) \implies \exists TC \in TS, TC$ *mayfail* $I)$.

i.e. every non-conformant implementation can be rejected by a test case in TS.

Using the definitions of I **ioco** S and TC *mayfail* I, one can now prove:

TS is correct $\iff \bigcup_{TC\in TS} Traces_{\mathbf{Fail}}(TC) \subseteq STraces_{\neg ioco}(S)$ and

TS is exhaustive $\iff \bigcup_{TC\in TS} Traces_{\mathbf{Fail}}(TC) \supseteq STraces_{\neg ioco}(S)$.

Interestingly, if one considers the non-conformance observer $can(S)$ as a test case (by mirroring its actions), as $Traces_{\mathbf{Fail}}(can(S)) = STraces_{\neg ioco}(S)$, it immediately follows that the singleton $\{can(S)\}$ is a complete test suite, in some sense the most general testing process for conformance w.r.t. S. Moreover, all correct test cases should be sub-observers of $can(S)$, while an exhaustive test suite must reject all implementations rejected by $can(S)$. In fact, all test generation algorithms for **ioco** producing complete test suites can be understood as producing an infinite number of unfoldings of $can(S)$. But in practice, $can(S)$ cannot be used directly as a test case. One wants to select individual test cases focussed on some particular behaviour. Selection of a sound test suite will then be based on the selection of sub-behaviours of $can(S)$. The selection algorithm should remain *limit exhaustive*: for any non-conformant implementation, one could generate a test case that could reject it.

4. TEST SELECTION FOR ioSTS

At the end of the section an example illustrates the principles of test selection. As explained previously, test selection consists in extracting a sub-observer of the non-conformance observer $can(\mathcal{S})$. The first operation consists in constructing the ioSTS $can(\mathcal{S})$ such that $[\![can(\mathcal{S})]\!] = can([\![\mathcal{S}]\!])$. When $\mathcal{S}$ is deterministic (or determinized) [4], this is easily done by adding, in every location l and for all output a, a new transition $\langle l, a, G_{\mathbf{Fail}}, Id_V, \mathbf{Fail}\rangle$ where $G_{\mathbf{Fail}} = \neg \bigvee_{\langle l,a,G,A,l'\rangle \in \mathcal{T}} G$ and **Fail** is a new location.

In this paper we focus on the selection of test cases by test purposes describing some abstract behaviour one wants to test. We define test purposes as ioSTS equipped with a set of accepting locations playing the role of a non intrusive observer. Its set of variables consists of its set of *proper* variables and the set of variables of the specification that it may observe, but cannot modify.

A *Test Purpose* for a specification ioSTS $\mathcal{S} = \langle V \cup P, \Theta, L, l^0, \Sigma, \mathcal{T}\rangle$ is an ioSTS $\mathcal{TP} = \langle V_p \cup V \cup P, \Theta_{TP}, L_{TP}, l^0_{TP}, \Sigma \cup \{\delta\}, \mathcal{T}_{TP}\rangle$ equipped with a distinguished set of locations $Accept \subseteq L_{TP}$. We assume that $\mathcal{TP}$ is complete in all locations except *Accept* (for each action a the conjunction of guards of all transitions carrying a is *true*) and cannot modify variables in V (assignments to these variables are the identity assignment).

The role of the test purpose is to select suspension traces of $can(\mathcal{S})$ accepted by $\mathcal{TP}$. The usual way to define this intersection for ioLTS is to perform a *synchronous product*. We define a corresponding syntactic operation on ioSTS where transitions with same actions synchronize on the conjunction of their guards. Formally, the *synchronous product* of $can(\mathcal{S}) = \langle V \cup P, \Theta, L \cup \{\mathbf{Fail}\}, l^0, \Sigma, \mathcal{T}_c\rangle$ and $\mathcal{TP} = \langle V \cup V_p \cup P, \Theta_{TP}, L_{TP}, q^0_{TP}, \Sigma \cup \{\delta\}, \mathcal{T}_{TP}\rangle$ is the ioSTS $can(\mathcal{S}) \times \mathcal{TP} = \langle V \cup V_p \cup P, \Theta \wedge \Theta_{TP}, L \cup \{\mathbf{Fail}\} \times L_{TP}, (l^0, l^0_{TP}), \Sigma, \mathcal{T}'\rangle$ where $\langle (l_1, l_2), a, G_1 \wedge G_2, A_1; A_2, (l'_1, l'_2)\rangle \in \mathcal{T}'$ if and only if $\langle l_1, a, G_1, A_1, l'_1\rangle \in \mathcal{T}_c \wedge \langle l_2, a, G_2, A_2, l'_2\rangle \in \mathcal{T}_{TP}$ and $A_1; A_2$ is the sequential composition of assignments affecting disjoint sets of variables.

$\mathcal{TP}$ is non-intrusive, thus $Traces(can(\mathcal{S}) \times \mathcal{TP}) = Traces(can(\mathcal{S}))$ and $Traces_{\mathbf{Fail} \times L_{TP}}(can(\mathcal{S}) \times \mathcal{TP}) = Traces_{\mathbf{Fail}}(can(\mathcal{S})) = STraces_{\neg ioco}(\mathcal{S})$ meaning that $can(\mathcal{S}) \times \mathcal{TP}$ is a non-conformance observer. We also have $Traces_{L \times Accept}(can(\mathcal{S}) \times \mathcal{TP}) = STraces(\mathcal{S}) \cap Traces_{Accept}(\mathcal{TP})$ meaning that $can(\mathcal{S}) \times \mathcal{TP}$ is an observer of traces accepted by $\mathcal{TP}$, restricted to

[4]For the sake of simplicity, we restrict here to deterministic ioSTSspecifications. Non-deterministic ioSTScan be handled at least for a sub-class of ioSTS where non-determinism can be solved with bounded lookahead [9].

suspension traces of S. Thus, depending on the considered distinguished locations **Fail** $\times L_{TP}$ or $L \times$ *Accept*, the ioSTS observer $can(S) \times \mathcal{TP}$ can play two different roles.

But $can(S) \times \mathcal{TP}$ is just an unfolding of $can(S)$ from which we now need to select traces by focussing on traces accepted in *Accept*. Ideally, we want to select exactly $STraces(S) \cap Traces_{Accept}(\mathcal{TP})$, plus unspecified outputs prolongating these traces in $STraces_{\neg ioco}(S)$. However we consider *non-controllable* system models, for which an input does not determine an unique output. After a trace, the tester should then consider all possible outputs: those from which *Accept* is reachable or **Fail** is reached, but also those after which *Accept* is not reachable anymore. In this last case, we want to detect this divergence as soon as possible, and set the *Inconc* verdict. This reduces to the problem of computing the set $coreach(Accept)$ of states co-reachable from $L \times Accept$. This is easy for finite state systems and solved with graph algorithms. However, this problem is undecidable for $ioSTS$ models.

Our solution, implemented in the STG tool [3], consists in computing an over-approximation $coreach^{\alpha} \supseteq coreach(Accept)$ represented by a predicate. This is provided by an interface with the NBac tool [7] using abstract interpretation [4]. For any assignment A of a transition $t \in \mathcal{T}'$, we also compute an over-approximation of the set of values for variables and parameters allowing to stay in $coreach^{\alpha}$ when firing t, noted $pre^{\alpha}(A)(coreach^{\alpha})$. In other words it is a *necessary condition* to go in $coreach(Accept)$ by t. Its negation is thus a *sufficient condition* to leave $coreach(Accept)$. The selection of a test case $\mathcal{TC}$ from $can(S) \times \mathcal{TP}$ then consists in mirroring actions, transforming *Accept* locations into **Pass** and modifying the transitions in $\mathcal{T}'$ into $\mathcal{T}_{TC}$ with the two following rules:

$$\text{Keep: } \frac{\langle l, a, G, A, l' \rangle \in \mathcal{T}'}{\langle l, a, G \wedge pre^{\alpha}(A)(coreach^{\alpha}), A, l' \rangle \in \mathcal{T}_{TC}}$$

$$\text{Inconc: } \frac{\langle l, a, G, A, l' \rangle \in \mathcal{T}' \quad a \in \Sigma_!}{\langle l, a, G \wedge \neg pre^{\alpha}(A)(coreach^{\alpha}), A, Inconc \rangle \in \mathcal{T}_{TC}}$$

The effect of rule (Keep) is to discards all (semantic) transitions labeled by a (controllable) input that *certainly* exit $coreach(Accept)$, and rule (Inconc) "redirects" to a new location *Inconc* all transitions labelled by an (uncontrollable) output that *certainly* exit $coreach(Accept)$. The test case can be further simplified (without modifying its semantics) with an over-approximation of its reachable states $reach^{\alpha}(\Theta \wedge \Theta_{TP})$. Notice that these analysis can be improved using the dynamic partitionning facility of NBac, allowing to separate locations with respect to the analysis.

Test case properties. As $can(S)$ is sound and is not modified by the synchronous product and selection, all test cases are sound. Limit exhaustiveness comes from the following construction: for any non-conformant implementation, there exists a trace $\sigma.a$ in $TracesI \cap STraces_{\neg ioco}(\mathcal{S})$. It then suffices to construct a test purpose $\mathcal{TP}$ such that the trace $\sigma.a$ leads to *Accept*. The test case obtained from $\mathcal{S}$ and $\mathcal{TP}$ then may reject I.

What is lost by the over-approximation of *coreach*(*Accept*), compared with an (hypothetical) exact computation, is the hability to detect infeasible traces to *Accept* as soon as this happens. Of course, the more precise is the approximation, the sooner is the detection [8].

Simple example.

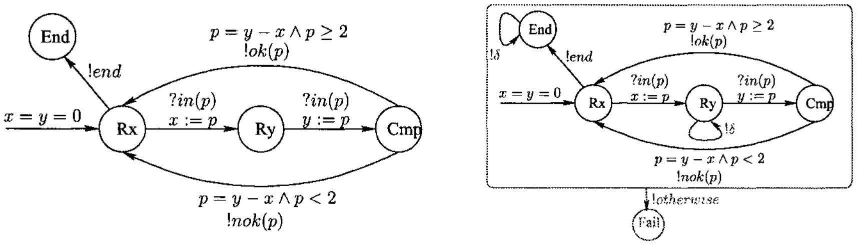


Figure 1. (Left) ioSTS $\mathcal{S}$ reading and comparing two values. (Right) canonical tester *can*($\mathcal{S}$).

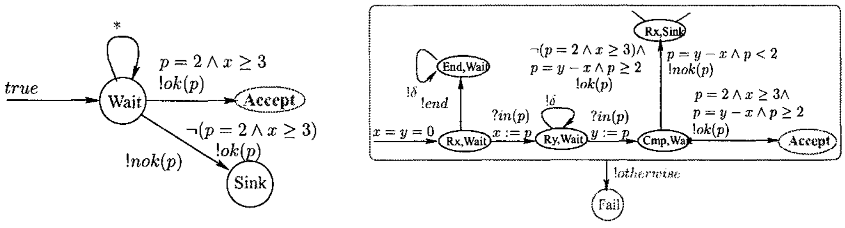


Figure 2. (Left) ioSTS test purpose $\mathcal{TP}$. (Right) Synchronous product *can*($\mathcal{S}$) $\times$ $\mathcal{TP}$.

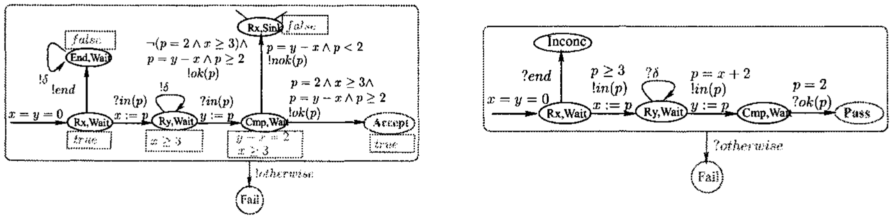


Figure 3. (Left) Computation of *coreach*$^{\alpha}$. (Right) Resulting test case $\mathcal{TC}$.

Test execution:. Test cases produced so far are ioSTS. In particular the values of communication parameters of test cases are not instanciated. During test execution, values of communication parameters have to be chosen for outputs of the test cases, among values satisfying the guard (e.g. $p = 5$ for $p \geq 3$ in the example). This is simply done by a constraint solver. Conversely, when receiving an input from the implementation, as the test case is input complete and deterministic, one has to check which transition can be fired, by checking the guard with the value of the received communication parameter (e.g. go to **Pass** is $p = 2$, and $Fail$ otherwise).

5. CONCLUSION AND PERSPECTIVES

There is still very few research work on model-based test generation which are able to cope with models containing both control and data without enumerating data values. Some exist however in the context of the **ioco** testing theory. In [10] the authors use selection hypotheses combined with operation unfolding for algebraic data types and predicate resolution to produce test cases from Lotos specifications. The paper [5] lifts the **ioco** theory from LTS to STS (Symbolic Transition Systems) but addresses the on-line test generation problem where next actions of test cases are computed during execution. In [2] the authors start with a specification model similar to ioSTS, abstract the model in a finite state one, use our TGV tool to generate test cases in the abstract domain, and then solve a constraint programming problem in the concrete model.

In the present paper, we have presented an approach to the off-line generation of test cases from specification models with control and data (ioSTS) and test purposes in the same model. The main advantage of this test generation technique is to avoid the state explosion problem due to the enumeration of data values. Test generation reduces to syntactic operations on these models and an over-approximate analysis of the co-reachable states to a target location. Test cases are generated in the form of ioSTS, thus representing uninstanciated test programs. During execution of test cases on the implementation, constraint solving is used to choose output data values. For simplicity, the theory exposed in this paper is retricted to deterministic specifications. However non-determnistic specifications can be taken into account if ioSTS have no loops of internal actions and have bounded lookahead.

Among the perspectives of this work, we expect to consider more powerful models of systems with features such as time, recursion and concurrency. For test generation, one problems to address in these models is partial observability, which entails the identification of determinizable sub-classes corresponding to applications. We also think that the ideas of this technique can also be used in other contexts, in particular for structural white box testing where test cases are generated from the source code of the system. One of the main problems of these techniques which is to avoid infeasible paths, could be partly solved by techniques similar to ours.

ACKNOWLEDGMENTS

I wish to thank the Organizing Committee of DIPES for this invitation, as well as all my colleagues who participated in this work.

REFERENCES

[1] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors. *Model-Based Testing of Reactive Systems: Advanced Lectures*, volume 3472 of *LNCS*. Springer, 2005.

[2] J. R. Calamé, N. Ioustinova, J. van de Pol, and N. Sidorova. Data abstraction and constraint solving for conformance testing. In *Proc. of the 12th Asia-Pacific Software Engineering Conference (APSEC 2005), Taipei, Taiwan*, pages 541–548. IEEE Computer Society, December 2005.

[3] D. Clarke, T. Jéron, V. Rusu, and E. Zinovieva. STG: a symbolic test generation tool. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *LNCS*, pages 470–475, Grenoble, France, avril 2002.

[4] P. Cousot and R. Cousot. Abstract intrepretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[5] L. Frantzen, J. Tretmans, and T. Willemse. Test generation based on symbolic specifications. In *4th International Workshop on Formal Approaches to Testing of Software (FATES 2004), Linz, Austria*, volume 3395 of *LNCS*. Springer-Verlag, 2004.

[6] ISO/IEC 9646. Conformance Testing Methodology and Framework, 1992.

[7] B. Jeannet. Dynamic partitioning in linear relation analysis. *Formal Methods in System Design*, 23(1):5–37, 2003.

[8] B. Jeannet, T. Jéron, V. Rusu, and E. Zinovieva. Symbolic test selection based on approximate analysis. In *11th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05), Edinburgh, Scottland*, volume 3440 of *LNCS*. Springer, april 2005.

[9] T. Jéron, H. Marchand, and V. Rusu. Symbolic determinisation of extended automata. In *4th IFIP International Conference on Theoretical Computer Science, 2006, Santiago, Chile*. SSBM (Springer Science and Business Media), August 2006.

[10] G. Lestiennes and M.-C. Gaudel. Testing processes from formal specifications with inputs, outputs and data types. In *13th International Symposium on Software Reliability Engineering (ISSRE'02), Annapolis, Maryland*. IEEE Computer Society Press, 2002.

[11] V. Rusu, L. du Bousquet, and T. Jéron. An approach to symbolic test generation. In *International Conference on Integrating Formal Methods (IFM'00)*, volume 1945 of *LNCS*, pages 338–357. Springer Verlag, November 2000.

[12] V. Rusu, H. Marchand, and T. Jéron. Automatic verification and conformance testing for validating safety properties of reactive systems. In John Fitzgerald, Andrzej Tarlecki, and Ian Hayes, editors, *Formal Methods 2005 (FM05)*, volume 3582 of *LNCS*. Springer, juillet 2005.

[13] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software—Concepts and Tools*, 17(3):103–120, 1996.

CONTINUOUS ENGINEERING OF EMBEDDED SYSTEMS

Bernhard Steffen [1], Tiziana Margaria[2]
[1] *Chair of Programming Systems, Universität Dortmund (Germany)*
[2] *Chair of Service and Software Engineering, Universität Potsdam (Germany)*

Abstract We investigate the late phases of the embedded systems' life cycles, in particular the treatment of change requests, the integration of legacy components, and the problem of emerging platforms. We propose to tackle these issues in a model-driven design paradigm, on the behavioral models, and to employ techniques from automata theory, model checking and automata learning. The main practical impact of our approach is its support of the systematic *completion* and *update* of user/customer requirements, which by their nature are quite partial and concentrate on the most prominent scenarios. Our technique generalizes these typical requirement skeletons by extrapolation and it indicates via automatically generated traces where the requirement specification is too loose and additional information is required. This works in the initial phases of system development, but also in case of change requests, where our technique hints at possible problems with their realization (feature interactions), and helps to keep the requirement model in synchrony along the chain of new releases.

1. MOTIVATION

The bulk of research concerning embedded systems focusses of the early stages of the systems' life cycles. Today's systems require unacceptable efforts just for deployment, typically caused by incompatibilities, feature interactions, and by the sometimes catastrophic behavior of component upgrades, which no longer behave as expected. This is particularly true for embedded systems, with the consequence that some components' lifetimes are 'artificially' prolonged far beyond a technological justification, for fear of problems once they are substituted or eliminated.

The time after the first deployment causes the majority of the overall costs of the system, but is hardly addressed, as is the integration of legacy components. It is a major research challenge to provide systematic and

Please use the following format when citing this chapter:

Steffen, B., Margaria, T., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 45–54.

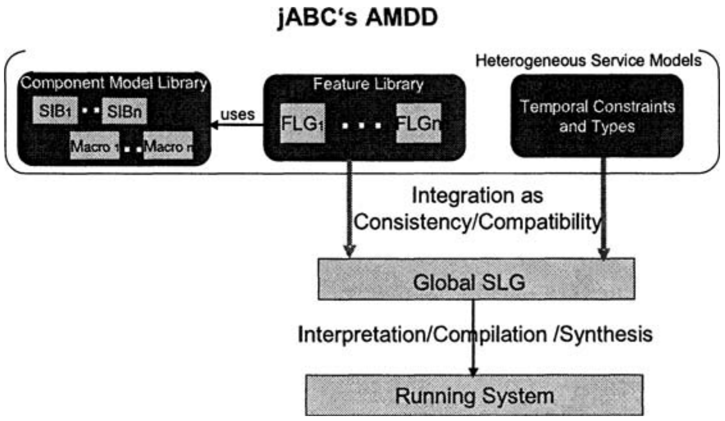

jABC's AMDD
Heterogeneous Service Models
Component Model Library
SIB1 • • SIBn
Macro 1 • • Macro n
uses
Feature Library
FLG1 • • • FLGn
Temporal Constraints and Types
Integration as Consistency/Compatibility
Global SLG
Interpretation/Compilation /Synthesis
Running System

Figure 1. The AMDD Process in the jABC

consistent support to the later phases of the life cycle as well as system construction with legacy components. Prerequisite in this direction is the development of modelling levels that capture these aspects.
Here we investigate this situation under three perspectives, that treat

1 *changing requests*: in particular, the consistent integration of new requirements.

2 *legacy components*: how can we increase our confidence when dealing with components lacking specification.

3 *emerging platforms*: how can we separate the two issues above from the technological details required for effective technology mapping.

Starting point for our analysis is an *aggressive* version of model-driven development (AMDD) [14], which moves most of the recurring problems of compatibility and consistency of system design, implementation, and evolution from the coding and integration levels to the modelling level (Fig. 1). Being a paradigm of *system design*, it inherently leaves a high degree of freedom in the design of adequate settings. We propose to treat the first two issues at the level of *behavioral models*, and use automata theory, model checking, and automata learning to consistently deal with looseness of requirements, uncertainties of legacy components, and the consequences of change requests. The third point coincides with the technology mapping issue discussed in the context of AMDD [14].

The main practical impact of the technique proposed in this paper is its ability to support the systematic *completion* and *update* of user/customer requirements, which by their nature are typically very partial and concentrate on the most prominent scenarios. Our technique generalizes these typical requirement skeletons by learning-based extrapolation, and it indicates via automatically generated traces where the requirement specification is too loose and additional information is required. This technique can also be used for the construction of behavioral models for third party/legacy components [5; 12].

This works in the initial phases of system development, but also with change requests, where our technique hints at possible problems with their realization (feature interactions), and helps to semi-automatically keep the requirement model in synchrony along the chain of new releases.

In the following, Sect. 2 briefly sketches our tool landscape, based on the jABC Modelling Framework, and Sect. 3 the essential features of automata learning. Then Sect. 4 presents our method of learning-based long-term requirement engineering: the principle, the simple monotonic case of requirement completion, and the non-monotonic case of requirement updates. Finally Sect. 5 gives our conclusions and perspectives.

2. BASICS CONCEPTS OF THE jABC

jABC [6] is a Java-based framework for service development along the AMDD paradigm [14]. Its central model structure are hierarchical, flow chart-like graphs called Service Logic Graphs (SLGs) [13]. They model the application behavior in terms of the intended process flows, based on coarse granular building blocks called SIBs (Service-Independent Building blocks) which are to be understood directly by the application experts – independently of the structure of the underlying code, which in our case is typically written in Java/C/C++. The component models (single SIBs or hierarchical subservices), the feature-based service models called Feature Logic Graphs (FLGs), and the Global SLGs modelling applications are all hierarchical SLGs. Additionally, the jABC supports several *model specification* styles, including

1 two modal logics, mu-calculus and monadic second order logic on strings [19], to abstractly and loosely characterize valid behaviors of finite and parametric systems, resp., which come with plugins for the graphical, pattern-based definition of constraints [7],

2 a classification scheme for building blocks and types, useful e.g. for service discovery in distributed service environments [11], and

3 model-level and source code-level analysis and consistency verification mechanisms based on these specifications [2].

In this sense, the jABC is an instance of *actor-based* modelling environments according to [3].

Predecessors of jABC have been used since 1995 to design, among others, industrial telecommunication services [15], Web-based distributed decision support systems [8], and test automation environments for Computer-Telephony integrated systems [5], and most recently to equip the Ricoh AFICIO printer series with a flexible process management.

jABC allows users to develop services and applications by composing reusable building-blocks into (flow-)graph structures. An extensible set of plugins provides additional functionalities to adequately support all the activities needed along the development lifecycle, like e.g. animation, rapid prototyping, formal verification, debugging, code generation, monitoring, and evolution. This process does not substitute but rather enhance other modelling practices like the UML-based RUP 17, which is in fact used in our process to design the single components.

jABC offers a number of advantages that play a particular role when integrating off-the-shelf, possibly remote functionalities.

- **Agility**. We expect requirements, models, and artifacts to change over time, therefore the process supports evolution as a normal process phase by various means, like, e.g., model checking, monitoring-based consistency checking, and requirement completion/update.
- **Consistency**. The same modelling paradigm underlies the whole process, from the very first steps of prototyping up to the final execution, guaranteeing traceability semantic consistency.
- **Verification**. With techniques like model checking and local checks we support the user to consistently modify his model. The basic idea is to provide automatic checking mechanisms for previously defined local or global properties that the model must satisfy.
- **Service orientation**. Legacy or external features, applications, or services can be easily integrated into a model by wrapping the functionality into building blocks to be used inside the models.
- **Executability**. The models can be executed in various modes: executions can be as abstract as guided documentation browsing and as complex as the concrete run of the final implementation.

Several applications have shown how these properties are exploited in different application areas, like e.g. [9; 4; 8; 10].

3. AUTOMATA LEARNING

Machine learning deals in general with the problem how to automatically generate a system's description. Besides the synthesis of static soft-

and hardware properties, in particular invariants, the field of *automata learning* is of particular interest for soft- and hardware engineering.

Automata learning tries to construct a deterministic finite automaton (see below) that matches the behavior of a given target automaton on the basis of observations of the target automaton and perhaps some further information on its internal structure. The interested reader may refer to [18] for details, here we only summarize the basic aspects of our realization, which is based on Angluin's learning algorithm L^*[1].

L^*, which is an *active* learning algorithm, learns finite automata by *actively* posing *membership* queries and *equivalence* queries to the target automaton in order to extract behavioral information, and refining successively an own *hypothesis automaton* (HA) based on the answers. Membership queries test whether a string (a potential run) is contained in the target automaton's language (its set of runs), and equivalence queries compare the HA with the target automaton for language equivalence, in order to determine whether the learning procedure has (already) successfully completed and the experimentation can stop.

In its basic form, L^* starts with the one state HA that treats all words over the considered alphabet (of elementary observations) alike, and refines it on the basis of query results iterating two steps. Here, the dual way of how L* characterizes (and distinguishes) states is central:

- from *below*, by words reaching them. This characterization is too fine, as different words may well lead to the same state.
- from *above*, by their future behavior wrt. a dynamically increasing set of words. These future behaviors are essentially bitvectors: a '1' means that the corresponding word of the set is guaranteed to lead to an accepting state and a '0' captures the complement. This characterization is typically too coarse, as the considered sets of words are typically rather small.

The second characterization directly defines the HAs: each occurring bitvector corresponds to one state.

The initial HA is characterized by the outcome of the membership query for the empty observation. Thus it accepts any word if the empty word is in the language, and no word otherwise. Next, the learning procedure (1) iteratively establishes local consistency, after which it (2) checks for global consistency.

Local Consistency. This is an automatic *model completion* loop, that iterates two phases: (a) checking whether the constructed HA is *closed* under the one-step transitions, i.e., each transition from each state of the HA ends in a well defined state of this very automaton. And (b) checking *consistency* according to the bitvectors characterizing the future behavior as explained above, i.e., whether all reaching words with an identical characterization from above have the same one step transitions. If this fails, a distinguishing transition is taken as an additional *distinguishing future* that resolves the inconsistency, splitting the state.

Global Equivalence. After local consistency, an equivalence query checks whether the language of the HA coincides with that of the target automaton. If this is true, the learning procedure successfully terminates. Otherwise the equivalence query returns a counterexample, i.e., a word which distinguishes the hypothesis and the target automaton. This counterexample gives rise to a new cycle of modifying the HA and starting the next iteration.

4. LEARNING-BASED REQUIREMENTS MANAGEMENT

Key towards continuous engineering is the treatment of new/changing requirements. Our learning-based approach to requirement management is organized to capture three dimensional requirement evolution:

- Use cases: individual ‘runs’ of the intended system.
- Temporal properties: global constraints that capture safety and liveness properties of the considered system.
- Structure requirements: system properties like symmetry between technical components, determinacy of individual behaviors,

Change request may concern all three dimensions, although individual change requests typically belong to one. This means in particular that the other dimensions remain unchanged, which drastically enhances the stability of upgrading procedures.

It is the central data structure of active automata learning, the *observation table*, which, slightly enhanced, enables the incremental and evolutionary model construction. It does not only comprise the actual HA, but also the concrete evidence which lead to its construction. Thus it allows us to distinguish between the model structure based on concrete observations and the model structure which arose in the course of extrapolation: the HAs are the state-minimal automata consistent with the actual observations. As such, they are neither an under- nor an over-approximation: they may as well allow (extrapolated) behavior, which the considered system will never engage in, as also refuse behavior the considered system is capable of.

The enhanced observation tables are adequate as a means for change management at the requirement level: they indicate how to cover new or changing requirements without violating the primary model structure. The basic intuition behind our approach is the following technique for completing and changing requirement specifications.

4.1 COMPLETING REQUIREMENT SPECIFICATIONS

Requirement specifications in terms of individual traces are by their nature very partial and represent only the most prominent situations. This partiality is one of the major problems in requirement engineering: it often causes errors in the system design that are difficult to fix. Thus techniques for systematically completing such partial requirement specifications are of major practical importance.

We propose a method for requirements completion based on automatic (active) automata learning that in essence

- *initializes* the learning algorithm with the set of traces constituting the requirement specifications, and
- constructs a *consistent behavioral model* by establishing the local consistency introduced in the previous section.

This way, we build a finite state behavioral model which *extrapolates* the given requirement specification: it comprises *all* 'positive' traces of the specification, and rejects all forbidden traces. All the other potential traces are consider as 'don't cares', in order to construct a corresponding state minimal HA. In particular, although the learning procedure by its nature will only investigate finitely many traces, the constructed HA will typically accept infinitely many traces, since the extrapolation process introduces loops.

For this method to work, a number of membership queries need to be answered. Both, establishing closure of the model, as well as establishing the consistency of the abstraction of reaching words into states (i.e., of the characterization from above introduced in the previous section) can only be achieved on the basis of additional information about the intended/unknown system. This is not unexpected, rather even desired (at least to some extent): the posed membership queries directly hint at the places where the given requirement specification is partial. On the other hand, it is not practical: the number of such membership queries is the major bottleneck of active learning, even when its generation is fully automated. The execution of membership queries is interactive, and the effort unacceptable.

We observed that the number of membership queries can be drastically reduced on the basis of orthogonally given expert knowledge about the intended/unknown system. We could show that already the following three very general structural criteria, *prefix closure*, *independence* of actions and *symmetry*, were sufficient to reduce the number of membership queries by several orders of magnitude [5; 12].

This idea, originally designed for the optimization of the treatment of 'use case'-based requirement completion, allows us to also capture the other two dimensions of requirement specification:

Temporal properties:. global constraints that capture safety and liveness properties of the system. Besides typical example runs, application experts are also able to formulate many necessary safety conditions, be it on the basis of required protocols, or the exclusion of catastrophic states. Adding these safety requirements in terms of temporal logics to our specification can automatically answer a huge number of membership and equivalence queries, via model checking. This way, these properties are automatically taken care of during model construction.

Structure requirements:. concern the shape of the overall system. The nature of the system to be learned often leads to structural constraints, like symmetry between technical components, or determinacy of individual behaviors. These constraints can be automatically taken care of by corresponding operations on automata, which add this structure to the observed behavioral skeleton.

4.2 CHANGING REQUIREMENT SPECIFICATIONS

So far our learning scenario was *monotonic*: observations made once are guaranteed to remain valid. Thus it might only happen that

- new observations revive the learning process, and/or
- assumed temporal of structural properties turn out to be false

forcing us to revise the hypothesis model. In both cases we can incrementally deal with membership queries – the major bottleneck – and in fact the classical observation table supports this incremental treatment.

In contrast to requirement completion, requirement update is non-monotonic, and may have a destructive effect on the observations made. It may force us to discharge some results of previously answered membership queries. There are two extreme approaches (and of course numerous compromises) to do so:

- start the learning procedure from scratch, which is of course not incremental, but inherits all the nice properties of automata learning, like 'the HA is a state-minimal representative of all automata consistent with the made observations'.
- continue the learning process as if it were monotonic, only question previous query results in case there is a conflict, and in case of conflict give the new observation precedence.

  This approach is incremental, but unfortunately conflict resolution is not as easy as it might seem: a new trace may well be in conflict to quite a number of traces. Perhaps even worse, an old (no longer valid) observation may not be conflicting with the new observation but make the hypothesis model 'explode'. We are currently investigating various strategies to overcome these problems.

On the other hand, if we maintain knowledge about which queries were answered by model checking or the assumption of structural properties (our enhancement of the observation table), one can rather comfortably deal with the change of these kinds of requirements.

5. CONCLUSIONS AND PERSPECTIVES

We have presented an approach to support the systematic *completion* and *update* of user/customer requirements along a system's life cycle. This method, mainly based on automata learning, elegantly complements our behavioral model construction for legacy systems, and provides a powerful support for the late phases of the systems' life cycles.

Currently, we are investigating how to efficiently deal with non-monotonic updates. Due to the three dimensional structure of the considered space of requirement specification, we do not expect a unique answer. In fact, we believe that, in practice, one will very much depend on additional information about which kind of changes have been made, in order to derive an efficient strategy for learning-based model update.

REFERENCES

[1] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 2(75):87–106, 1987.

[2] A.L. Lamprecht, T. Margaria, B.Steffen: *Data-Flow Analysis as Model Checking within the jABC*, Proc. CC'06, 15th Int. Conf. on Com-piler Construction, Vienna (A), March 2006, LNCS, 3923, Springer Verlag, pp. 101-104.

[3] E. Lee, S. Neuendorffer, M. Wirthlin. *Actor-oriented design of embedded hardware and software systems* Journal of circuits, systems, and computers. 2002.

[4] M. Hörmann, T. Margaria, T. Mender, R. Nagel, M. Schuster, B. Steffen, H. Trinh: *The Jabc Approach To Collaborative Development of Embedded Applica-*

tions, CCE'06, Worksh. on Challenges In Collaborative Engineering (Industry day), Prag, April 2006.

[5] H. Hungar, T. Margaria, B. Steffen: *Test-Based Model Generation for Legacy Systems*, Proc. IEEE ITC'03, Charlotte, 2003, IEEE CS Press, pp.971–980.

[6] jABC Webseite: www.jabc.de

[7] S. Jörges, T. Margaria, B. Steffen: *FormulaBuilder: A Tool for Graph-based Modelling and Generation of Formulae*, Proc. ICSE 2006, 28th ACM-IEEE Int. Conf. on software Engineering, Shanghai (CHN), May 2006, to appear.

[8] M. Karusseit, T. Margaria: *Feature-based Modelling of a Complex, Online-Reconfigurable Decision Support Service*, WWV'05, 1st Int. Worksh. Automated Specification and Verification of Web Sites, Valencia, March 2005, ENTCS 1132.

[9] C. Kubczak, R. Nagel, T. Margaria, B. Steffen: *The jABC Approach to Mediation and Choreography*, Semantic Web Services Challenge 2006, Phase I Workshop, DERI, Stanford University, Palo Alto, March 2006.

[10] T. Margaria, C. Kubczak, M. Njoku, B. Steffen: *Model-based Design of Distributed Collaborative Bioinformatics Processes in the jABC*, IEEE ICECCS 2006, Stanford, Aug. 2006, to appear.

[11] T. Margaria, R. Nagel, B. Steffen: *Remote Integration and Coordination of Verification Tools in JETI*, Proc. IEEE ECBS 2005, April 2005, Greenbelt (USA), IEEE CS Press, pp. 431–436.

[12] T. Margaria, H. Raffelt, B. Steffen: *Knowledge-based relevance filtering for efficient system-level test-based model generation*, Innov. in System and Software Engineering - a NASA Journal, Vol. 1(2), pp.147-156, Springer Verl., Sept. 2005.

[13] T. Margaria, B. Steffen: *Lightweight Coarse-grained Coordination: A Scalable System-Level Approach*, STTT, Int. Journal on Software Tools for Technology Transfer, Special section on Formal Methods for Industrial Critical Systems, Vol.5, N.2-3, 2004, Springer Verlag, pp.107-123.

[14] T. Margaria, B. Steffen: *Aggressive Model-Driven Development: Synthesizing Systems from Models viewed as Constraints*, Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation, The Monterey Workshop Series, Chicago, September 2003.

[15] T. Margaria, B. Steffen, M. Reitenspieß: *Service-Oriented Design: The Roots*, ICSOC 2005: 3rd ACM SIGSOFT/SIGWEB Int. Conf. on Service-Oriented Computing, Amsterdam, Dec. 2005, LNCS 3826, pp. 450-464, Springer V..

[16] H. Raffelt, B. Steffen, T. Berg: *LearnLib: A Library for Automata Learning and Experimentation*, Proc. FMICS 2005, 10th ACM Workshop on Formal Methods for Industrial Critical Systems, Lisbon, Sept. 2005, ACM Press, pp.62–71.

[17] Rational Unified Process. http://www-306.ibm.com/software/awdtools/rup/

[18] B. Steffen and H. Hungar, Behavior-based model construction. In S. Mukhopadhyay and L. Zuck, editors, *Proc. 4th Int. Conf. on Verification, Model Checking and Abstract Interpretation*, LNCS 2575, Springer 2003, pp.5–19.

[19] C. Topnik, E. Wilhelm, T. Margaria, B. Steffen: *jMosel: A Stand-Alone Tool and jABC Plugin for M2L(Str)*, Proc. SPIN'06, 13th Int. SPIN Works. on Model Checking of Software, Vienna, April 2006, LNCS 3925, Springer V., pp.293-298.

PROTOTYPING AN AMBIENT LIGHT SYSTEM - A CASE STUDY

Henning Zabel and Achim Rettberg
University of Paderborn/C-LAB, Germany
{henning.zabel, achim.rettberg}@c-lab.de

Abstract: This paper describes an indirect room illumination system which is called Ambient Light System (ALS). By illumination an object or room in a deliberate manner a certain mood or emotion could be evoked. For example by watching a music video on a television (TV) the room illumination can support the spirit of the music song. The realized room lightning is based on LEDs and enables the illumination with a wide range of RGB colors. The use of LEDs enforces special requirements for power supply, cooling and controlling. In our approach the colors depends on a TV video signal. We propose a hardware setup for driving and controlling a set of LED based lights and a software for capturing and analysing video signals to calculate a representative color, that is used as illumination color.

1. THE AMBIENT LIGHT SYSTEM

The development of an ambient light for room lightning is an interesting feature for modern devices, like televisions or music players. Ambient light offers to visualize more visual effects of videos or the spirit of a music song. The ambient light system (ALS) is able to grab a video and light a room with power LEDs in the color shown in the video. The calculation of the lightning is based on the majority of pixels shown in the video. This approach is similar to the idea of the *Ambilight*© (see [2]) developed by Philips for televisions. *Ambilight* is an ambient lighting feature that projects a soft light onto the wall behind the TV. The user is able choose from different preset colors and white tones, or it can be fully personalized using the custom settings. Furthermore, *Ambilight* simply allows adjusting the back light color based upon the action on the screen. In contradiction to the *Ambilight*, we are able to illuminate the entire room with a specific color. Besides this, due to the flexibility of our ALS it is adaptable to other systems.

Please use the following format when citing this chapter:

Zabel, H., Rettberg, A., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 55–64.

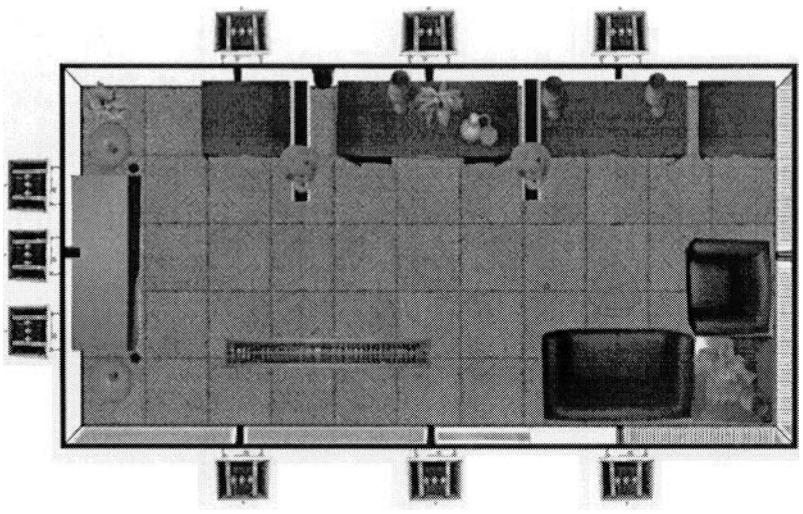


Figure 1. Top view of the ACLAB.

The developed ALS illuminates an entire room, see figure 1, with a specific RGB color. As shown in the figure the room is equipped with nine LED-based lights. Seven of them are mounted on the wall and illuminates the room by indirect lightning. The two other lights are mounted behind our Plasma television to illuminate the left and right side of the wall behind the TV.

The ALS is realized partly in software and in hardware. The top-level view is depicted in figure 2. One software part of the ambient light system (ALS) runs on a standard PC with a PCI TV-card. The ALS is able to grab the video and analyses the pixels from the left and the right area of the captured frame. This introduces stereo lightning within ALS. The data for the lights generated by the ALS is sent by over CAN interface to control boards.

2. PC SOFTWARE CONTROLLER

The software of ALS is split into two parts. The first one run on a PC and the second on a set off microprocessors, see section 3. The graphical user interface of the software part running on a PC is presented in figure 3. This component calculates the color mean value. This is sent to the lights to illuminate a room. The Java Media Framework, see [4] is used to grab the video signal.

The frames are grabbed with the smallest possible resolution (80x60) supported by the hardware. At least only a small resolution is enough to extract an average color from the frame and reduces the processing costs. By the implementation of a *VideoRenderer* class the single frames are directed to the frame processing. The frame processing consist of several stages (see figure 4),

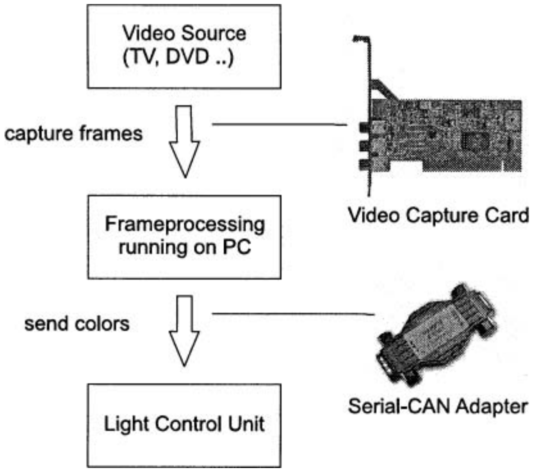
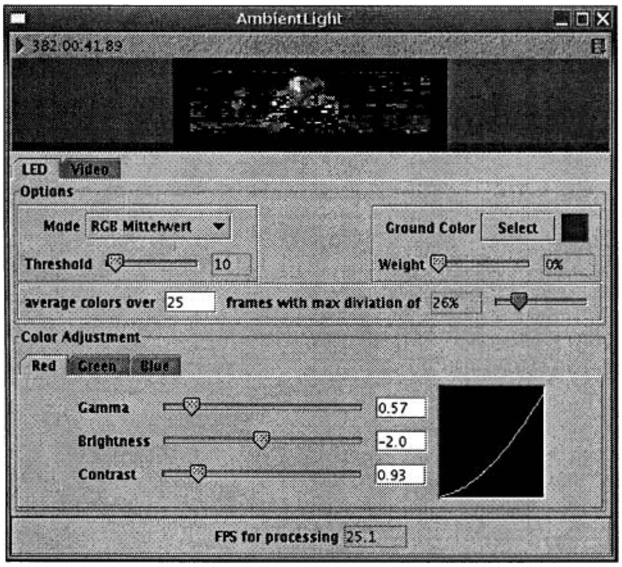


Figure 2. Ambient Light System (ALS).



Options

Mode RGB Mittelwert

Threshold 10

Ground Color

Weight 0%

average colors over 25 frames with max diviation of 26%

Color Adjustment

Gamma 0.57

Brightness -2.0

Contrast 0.93

FPS for processing 25.1

Figure 3. Mainwindow of ALS PC Software.

where as each of them can be parameterized by the graphical user interface (see figure 3). The last stage results in an average color value that is transmitted via a RS232/CAN interface to the control boards with a communication API (see [1]) for further processing.

The frame processing stages perform different analysis on the frames that are presented in detail in the following:

- In the first stage brightness, saturation and gamma corrections of the frames are applied depending on the settings given by the user in the graphical user interface. Dark or to bright videos can be corrected to achieve a proper room illumination.

- From the complete frame masks are calculated to extract significant pixels. Criteria for significant pixels are: (1) high differences (frame difference), (2) high saturation and (3) high brightness. Firstly, all these characteristics for each pixel are evaluated and in a second path the mask is constructed. Within this mask only those pixels are marked that are above a threshold, the others are set to zero. The threshold results from a constant and the mean value for the specific characteristic. The mask for the characteristics are depicted in the control window (see figure 5, right up: difference, left down: saturation, right down: brightness and left up: combination of all three).

  Depending on the mask selection all pixels are kept that are marked by one of the masks. Unmarked pixels are set to the color black.

- Out of the frame a representative color value is calculated. This can be either the mean value of all colors within the picture or one from a predefined palette. This palette contains gray values and colors that consists of not more than two base colors. These colors can be mixed

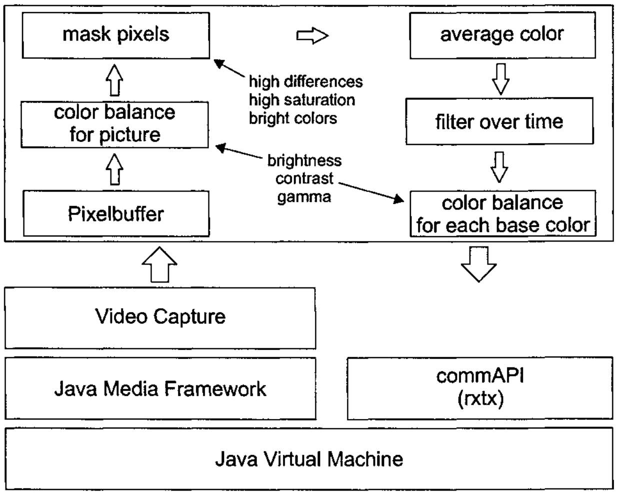
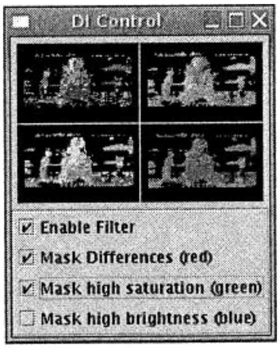


Figure 4. ALS PC Software Architecture.



☑ Enable Filter
☑ Mask Differences (red)
☑ Mask high saturation (green)
☐ Mask high brightness (blue)

Figure 5. ALS Picture Analysis.

easier as all others. Each pixel of a frame is assigned to one of the palette colors. The occurrence of each palette color is counted and than the most common color is used as representative. These calculation is performed for the left and right side of the frame.

During mean value calculation dark areas decrease the brightness of the representative color very strong. This reduces the illumination of the room and with it the effect of the ambient light. Therefore, it is necessary that the color has a specific brightness to be involved in the above mentioned calculation. By this, the color black is ignored in general and with it the above unmarked pixels.

- A video card can introduce noise in the captured frames. To filter this noise it is possible to calculate a mean value of the representative color over a specified time period (fading). If the current representative color differs from the mean value more than by a user defined value then this value will be taken. In this case the calculation of mean value resets. This allows on the one hand the reduction of noise and jitter that results from weak color changes and on the other hand direct color switching at scene changes.

- To realize a color balance between the calculated representative color value and the real color value that illuminates the room it is necessary to adjust each base color. This can be done by adjusting the brightness, saturation and most important the gamma value. The calculated color values are transmitted via a serial interface to an RS232/CAN converter. This serial interface is accessed with an implementation of the communication API (see [1]) from SUN.

3. HARDWARE CONTROLLER BOARDS

Besides the software part on a PC, we use a small controller running on a board (see [3]) with an ATMEL microprocessor AT90CAN128 to control the lights. Each board can handle up to two lights, therefore, to illuminate the room with nine lights five of these controller boards are needed. These boards are built in a central control unit. The lights consists of three high-power LEDs mounted on heat sink and a driver board.

The control unit is connected to the PC with a CAN interface. The controller transmit the brightness for each color by a pulse width modulated (PWM) signal to the driver boards in the lights. The driver boards modulates the power supply of the LED with this signal. The energy for the high-power LEDs come from a 12V power supply unit. This is depicted in figure 6. Furthermore, the

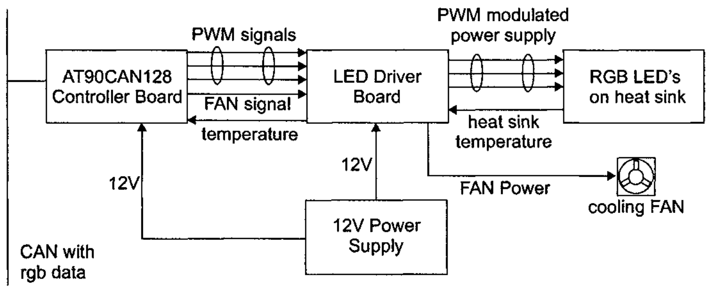


Figure 6. ALS Controller Board.

controller observes the heating of the lights by switching a fan on and off. If the temperature rises above 50° Celsius an emergency shutdown is executed and the light is completely switched off.

3.1 CONTROLLER BOARDS

The controller boards (see figure 7) are assembled on a backplane, that supplies them with power and connects them to the CAN bus. Each controller board has two serial interfaces. Additional, the backplane connects the controller boards in a daisy chain by the serial interfaces. This communication channel is used for enumerating the boards. Thus, the boards are given a unique ID and with it the lights can be identified. The enumeration is initiated by the first board in the chain. It is identified by a shortened input pin. This allows using the same software on all boards. During the development of the control software color and control data is transmitted via this serial connection. Whereas, commands that don't belong to a locally connected light are forwarded to the next board in the chain. Theoretically it is possible to extend this system by further controller boards and lights. In practice, the latency

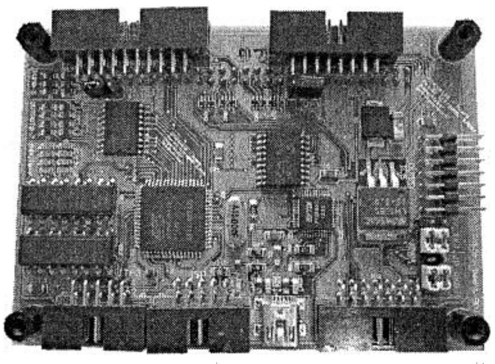


Figure 7. Controller boards with Atmel Processor.

of the communication through different controller boards limits the maximal number.

3.2 DRIVER BOARDS AND LIGHTS

As mentioned before the light uses three high-power LEDs as light source. Each LED has a power of 5 W. To achieve the major part of the RGB color palette one red, one green and one blue LED is used. Not all power is emitted as light and, therefore, the lights has to be cooled. For this the LEDs are mounted on a big heat sink, see figure 8. At the bottom of each light case a driver board is installed.

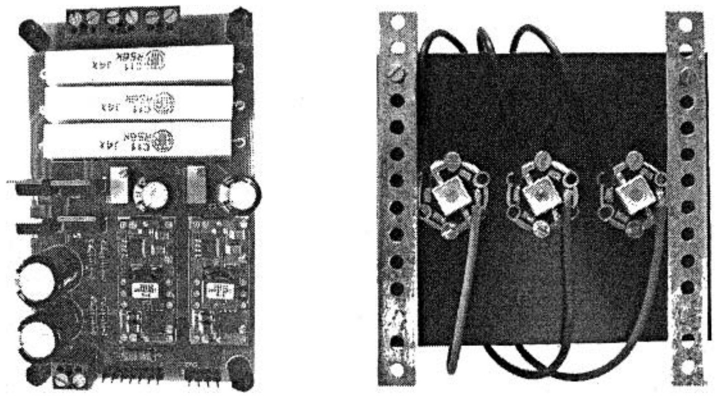


Figure 8. Light components: left the driver board and right the LEDs mounted on a heat sink.

The driver board adjusts the 12 V power supply to the power supply of the LEDs, 3.4 V for green and blue and 2.5 V for red. This is efficiently done by the use of a DC-DC switching power supply. The power supply for each LED is switched on/off via the PWM modulated signal of the controller by a BUZ11a MOSFET. Although a single diode consumes 1.4 A current, only 1.5 A current on the 12 V power supply is needed because of the switching power supply. Therefore, the voltage drop in the supply wires is kept low, that means, we achieve a less power dissipation by powering the LEDs. Additionally, a fan and temperature sensor is installed in the lights to measure and control the temperature of the diodes during operation. At $40°C$ the fan is activated and at $50°C$ the LEDs are switched off. This helps to extend the life time of the diodes. The fan is controlled by a signal of the controller board.

4. EVALUATION

The system is evaluated based on three problem areas, that are namely: latency, color balance and color matching.

4.1 LATENCY

The latency is the time between capturing the frame and the illumination of the room with the representative color. This time includes the time that is needed for the PC software to capture and analyse the frame and to transmit it to the CAN bus and finally the time needed by the controller boards for reacting on the CAN message.

The software on the controller consists of one main loop that handles the communication and executes the transmitted commands. With the help of the hardware timer the time consumption of one loop is measured and the maximum value is stored. This value converges over time and represents the worst-case execution time of the loop. The measured clock rate is 4900 Hz. The registers for PWM modulation are updated at a clock rate of 1 Khz, so the maximal latency for reaction is 1.2 ms, which is sufficient enough for the application.

The time-shift of the PC software is about 5 pictures, that means, the illumination data arrive 5 pictures later than the frames shown on the TV. Nevertheless, the frame rate of 25 fps for PAL is achieved. For testing a video is used that switch between different base colors every 5 sec. shown in full screen. The value of 5 pictures is a subjective impression when watching this video compared with the illumination of the room. A more precise measurement, for example by recording TV and the back light, is required.

4.2 COLOR BALANCE

The RGB value of the representative color is represented by three 8 bit values, one for each color. Also the PWM register on the controller boards have a width of 8 bit. But in normal the RGB value can not directly be used, because a color balance has to be applied, similar as it has to be done by crt monitors. For this, it is possible to correct the brightness, saturation and the gamma value for each base color. Adjusting the gamma value is most important.

The balance is adjusted manual by comparing different colors on a crt monitor with their relating illumination of the room. The easiest way is to start adjusting colors that are a combination of only two base colors. If reducing their saturation step by step the color tone should not change until the color white is reached. The colors yellow and orange are problematic, because they are both a mix ofred and green.

Simply by appliing the gamma correction to each base color, good results in color representation are achieved. Because each value is only 8bit wide, this correction can be precalculated and stored in arrays. The adjustment during runtime then simply is an access to this array and avoids further calculations.

4.3 COLOR MATCHING

The problem of color matching is the question is the representative color really representative, that means, does it really enhance the impression of the video picture. This is very hard to answer, because it heavily depends on the subjective impression of the viewer and also on the presented video.

When watching life music videos with a stage illumination containing strobe lights good results could be achieved by masking the frame with high difference and saturation. When disabling the mean value of the representative color over time the stage illumination is transfered in the entire room. In that case, the viewer gets the impression of being on stage. During repeating color switches that takes less than 5 frames the above mentioned latency leads to a phase shift between the room illumination and the TV. At the worst case the illumination alternates between the TV and the lights. This stresses the eyes extremely.

When watching movies it is more pleasant if averaging the representative color over time for about a second. For scenes with less movements (means less color changes) this reduces flickering of the ambient light and also avoids noise produced by the frame grabber card. On the other side, lightnings and fast color switches are desired on rapidly changing scenes. For example laser gun shots in science fiction movies should be immediately visible for the audience. Finding an adequate level at which the calculation of the representative color mean value is aborted, is a complicated challenge.

5. CONCLUSION AND OUTLOOK

This paper presents the ambient light system (ALS). ALS allows the grabbing of a video stream and the calculation of the color mean value for lightning a room with power LEDs. The mean value calculation is adaptable by different mask computations. Therefore, with ALS the user is able to adapt the lightning to his needs. ALS consists of a hardware and software part. For the hardware two boards are developed, the driver board for the low level control of the LEDs and the backplane in the ALS control unit. The software is divided into a real-time controller and a PC process which analyses the captured frame from video signal to calculate a representative color. On the hardware a maximal latency of 1.2 ms is achieved, which is noticeable smaller than the time for one frame (40 ms). The latency of 5 frames of the PC process is only a subjective measurement. Therefore, the overall latency of our ALS is not precisely determinable. Nevertheless, the latency could be measured by recording the TV together with the illuminated wall. The difference of the time between the TV shows a single color in full screen and the time point when this color is visible on the wall is more precise value for this latency. This test should be done by displaying colors from a small palette to clearly assign the recorded colors to one of the palette. Additionally, the PC process supports color balance and color matching. With our approach of manual adjustment the color balance achieved sufficient results. The color matching is still an open problem.

As further work, it is planned to replace the PC process step by step with suitable hardware, like FPGAs for frame processing and when needed additional controller boards. For example calculation of mean value of the representative color over time and performing of the color balance can be integrated into the software part running on the controller board.

ACKNOWLEDGMENTS

The following people are involved in the development and assembling process and gave some really useful hints and ideas (mentioned in alphabetical order): Philip Adelt, Dirk Hopmann, Bernd Kleinjohann, Alexander Krupp, Christoph Löser, Wolfgang Müller, Stefan Rose and Robbie Schäfer.

REFERENCES

[1] Keane Jarvi. *Communication API for Java: RXTX.* http://users.frii.com/jarvi/rxtx/download.html.

[2] Philips Electronics N.V. *Learn about Picture and Sound.* http://www.flattv.philips.com/index.cfm?$event = main\&cat_id = 1\&subcat_id = 2\&page = pg3$.

[3] Paderkicker. *ATMEGA128-Board, Rev. 0.2.* University of Paderborn/C-LAB, 2005.

[4] Inc. Sun Microsystems. *Java Media Framework API (JMF), Version 2.1.1e.* http://java.sun.com/products/java-media/jmf/index.jsp, 2006.

THE PADERKICKER TEAM: AUTONOMY IN REALTIME ENVIRONMENTS

Willi Richert, Bernd Kleinjohann, Markus Koch, Alexander Bruder, Stefan Rose, and Philipp Adelt
Faculty of Computer Science, Electrical Engineering and Mathematics, University of Paderborn, Germany
richert@c-lab.de

Abstract: The Paderkickers are a robot soccer team that makes heavy use of automotive technology like C167 micro-controllers or communication over CAN bus. All sensor data is processed on these decentralized embedded nodes to yield a high degree of reliability and hardware layer abstraction. In this paper, we describe how the complex system copes with perception and action in real-time and integrates it in the higher strategy layer to achieve autonomous behavior.

1. INTRODUCTION

The Paderkicker team [8] consists of five robots (Fig. 1) that already participated successfully in the German Open competition in 2004 and 2005, and the Dutch Open 2006. Currently, last preparations for the RoboCup 2006 World Championships are in progress.

Our platform asks for the whole range of research issues needed for a successful deployment in the real world. This includes embedded real-time architectures [2, 5, 14–17], real-time vision [2, 14–17], learning and adaptation from limited sensor data, skill learning and methods to propagate learned skills and behaviors in the robot team [13, 12, 9]. However, our goal is not to carry out research for specific solutions in the robotic soccer domain, but to use and test advanced techniques from different research projects. The Paderkicker platform serves as a test bench for the collaborative research center 614 (funded by the Deutsche Forschungsgesellschaft). Furthermore, the knowledge in vision, motion and object tracking is currently used in the AR PDA (Bundesministerium für Bildung und Forschung) project [11].

Please use the following format when citing this chapter:

Richert, W., Kleinjohann, B., Koch, M., Bruder, A., Rose, S., Adelt, P., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 65–74.

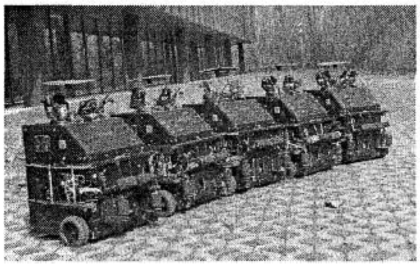


Figure 1. The Paderkicker team.

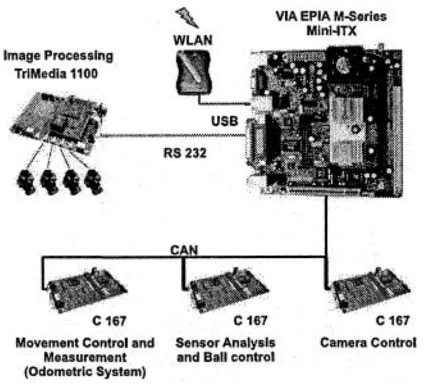

VIA EPIA M-Series Mini-ITX
WLAN
Image Processing TriMedia 1100
USB
RS 232
CAN
C 167
C 167
C 167
Movement Control and Measurement (Odometric System)
Sensor Analysis and Ball control
Camera Control

Figure 2. The Paderkicker architecture.

2. ROBOT OUTLINE

The robots (Fig. 1) have differential drive (2*75W) allowing for a maximum speed of 2.5 m/s. Instead of omni-vision we use four standard cameras for improved recognition of far away objects. As the colored marker objects at the corner will be removed in the near future the robots will have to distinguish far away line markers. We use a slide driven by an elastic band to shoot the ball. For the proper positioning of the ball prior to shooting the robot can use two side wise rolls and one above the ball, all of them electrically driven in both directions if needed. The central processing unit in our robot architecture (Fig. 2) is a PC compatible board which boots real-time Linux. The main process on the PC is Paderkicker's *Brain* module written in Java, which is the central instance to process the accumulated perception and choose the correct actions. This is possible, because all time-consuming processing is sourced out to the diverse distributed controllers. The system still has very low processor load. The Java process is assisted by a Particle Filter process (C++) which sits between the vision sensor and *Brain*, calculating the global robot position, relative ball position, and obstacles. Thereby, we have two ball positions the behavior system can operate with: a fast inaccurate one for approaching the near ball for better reactivity and a delayed, accurate and more stable one for moving towards the ball that is more than 2,5m away. Furthermore, *Brain* connects to a separate Jess [7] process, a rule engine similar to CLIPS but written in Java. The Jess instance has the task to switch between strategies (Attack, Defend, etc.) dependent on team variables like e.g. "our team possesses the ball" or "enemy in goal area".

2.1 HARDWARE

Paderkicker robots make use of automotive technology like C167 16bit micro controllers and communication over CAN bus. Furthermore dedicated hardware is used for image processing. Self designed PCBs are integrated for voltage and temperature control. All incidental sensor data is processed on these decentralized embedded nodes for better reliability and hardware layer abstraction. The extracted data is reported to the central processing node, a Mini-ITX board with a VIA EPIA 1GHz CPU running real-time Linux.

Research is done mainly in the area of real-time image processing. An optimized algorithm for low latency real-time color segmentation [16] which even performs well on a PDA is implemented on a Trimedia TM 1100 video processing board running at only 100 MHz. Four rotatable analog cameras are used to cover the whole 360° view instead of omnivision resulting in an overall higher resolution. Hence, we are capable of seeing distant objects much better. Higher visual coverage of the environment will become more important in the near future when the field will likely be extended in size. All four cameras are connected to the Trimedia Board. The data stream of one camera is evaluated at a time and the extracted objects are delivered at a speed of up to 20 fps over RS232 to the Mini-ITX board. Additionally, object recognition and tracking [2] are a prerequisite for accurate self localization. Combining our real-time color segmentation algorithm and the implemented edge vectorizer [17] we can easily feed our real-time particle filter [10] with the crucial data needed to provide our world model with the absolute robot position.

The central node of our hardware layer is the Mini-ITX board running a Timesys Linux kernel (2.6 series) on a linux system built from scratch. The board is connected to the robot's actuators via the CAN bus. With three C167 boards we handle the drive control, odometry, ball control (rolls), and camera positioning. Voltage and temperature sensors are read out over USB. The base system is placed on a compact flash card, mutable configuration data resides on a USB stick. Finally, we use a WLAN USB module to communicate with a central server that routes the messages to the proper robot peer. With this hardware architecture, the system is able to provide the behavior and strategy level with all the necessary information it needs.

2.2 SOFTWARE

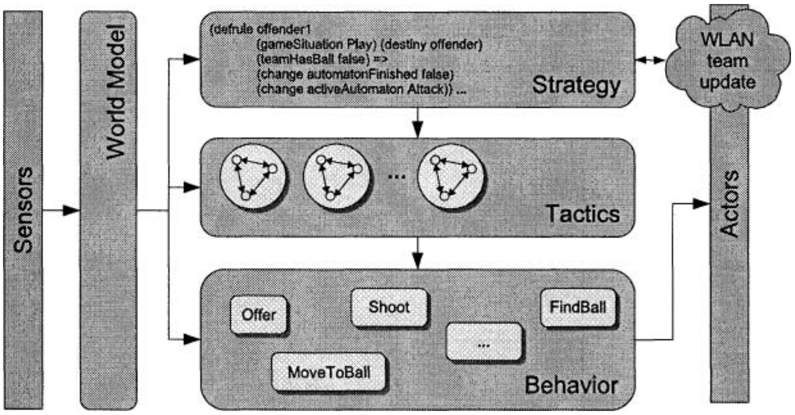
Our software behavior system breaks down into four independent modules (Fig. 3):

World model module We use particle filters for every robot and communication between the team members to update their world state, that includes the state of the robot and some features from the team members.

Strategy module Depending on the world model the strategy module decides the current strategy independently for every robot. Strategies include e.g. *Defend*, *Attack*, but also standard situations like *Kick off* or *Penalty*. This is done by the aforementioned rule engine (cf. Section 3.1).

Tactics module Every strategy is realized by a finite state machine that carries out the tactics. Its states are, e.g., *Ball facing*, or *Stay between goal and opponent.*

Behavior module This module executes the actual actions. It is behavior based in terms of Arkin's Motor Schemes [1]. Our behavior system [4] allows for a distinction between cooperative and competitive behaviors and behavior control through time excited evaluation functions. It consists of a set of low-level behaviors that have to be combined in order to result in a vector that can be sent to the actuators.



```
(defrule offender1
        (gameSituation Play) (destiny offender)
        (teamHasBall false) =>
        (change automatonFinished false)
        (change activeAutomaton Attack)) ...
```

Figure 3. Information flow in the Paderkicker.

Every major tactic like attacking the opponent or defending the own goal has been modelled as a separate automaton. The selection of the proper automaton is the task of a rule engine implemented in Jess that keeps track of robot and team state changes (Fig. 3). In this way, the RoboCup soccer rules could be implemented very quickly and we could concentrate on the behavior details.

2.3 THE MESSAGE FORMAT: CONNECTING HARDWARE AND SOFTWARE

With the decentralized approach the systems autonomy is delegated to the dedicated hardware boards, every board has one task to accomplish and is responsible for it. This leads to a more complex information exchange mechanism needed to provide all subsystems with needed information in time. In addition, there are different bus systems to pass information over — CAN bus, RS 232, USB and TCP/IP for communication between robots. For this domain, a special message format has been designed that can be used on all bus systems in the Paderkicker architecture (Fig. 4). This message format turned out to be robust and

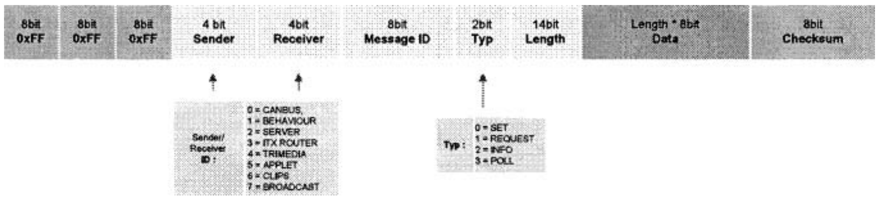


Figure 4. The message format.

computationally cheap. Every message is preceded by three bytes 0xFF as a message start delimiter. Thereby, every subsystem is able to figure out the starting point of messages in continuous data streams with low processing overhead. With the *Message ID* field it is possible to extend the current message set by not yet foreseen message possibilities.

3. TEAM OUTLINE

The mission of the RoboCup project is to "develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer" by 2050 [3]. To decrease the gap of robots ability today and the mentioned honor future goal we have to focus on different substantial skills. A robot on one hand must be capable of acting fully autonomously under a dynamic environment and on the other hand needs to cooperate with other team members to have a chance to win a soccer game. We will focus on these two important points on the layer of strategy. The complexity of such a project will spread quickly to different other problems like a union world model, exchange of perceptions between teammates, role negotiation, to play by the competition rules and to handle uncertainty. The strategy level of both bases on the rule-based system Jess [7] which is used as an expert system for RoboCup and interacts closely with other components of the robot and even over the whole team.

3.1 STRATEGY LAYER

For the strategy layer of the Paderkicker we use a rule-based system for tactics on a higher level of the robotic architecture. The planning on that layer is very intuitive by using expert system declarative programming methodology. So the expert knowledge is well human readable and acts in a manner of what's to be solved and not procedural how this should happen. We prefer to program like first mentioned and rely on the ability of the expert system to reason even under incomplete world information [6]. First we want to mention the interaction between the rule-based system and the other components of the robotic software architecture.

Architecture and Interfaces. The overall rule-based system architecture is shown in Fig. 5. Beginning at the left side of the figure the expert system core architecture is shown. It consists of different modules. The working memory holds the facts and variables of the actual world model (Here: It is unknown if the ball is in the perception range of the robot). The rule-base holds the domain specific expert knowledge coded into rules. (Here: If the ball is in range then change the role of the robot to offender.) The pattern matcher matches the rule premises against the facts in working memory and creates an agenda for execution of the activated rules. Now the interface to the rest of the robot comes into account on the right side of the figure. The rule-based system is fed for example by the perception module of the behavior system. The perceptions are preprocessed and appended into the working memory as facts independently of the inference machine. This is solved by an observer in Java. For example if the perception system recognizes a ball in range then the observer reflects this as a fact into the working memory. The way back to the behavior system and later on to the actuators is done by extending Jess with so called User-Functions which implement a Transmitter to the behavior system. In our example if the ball is in range the rule will fire and change the role of the robot via the change-transmitter to offender. This closes the circle of interaction.

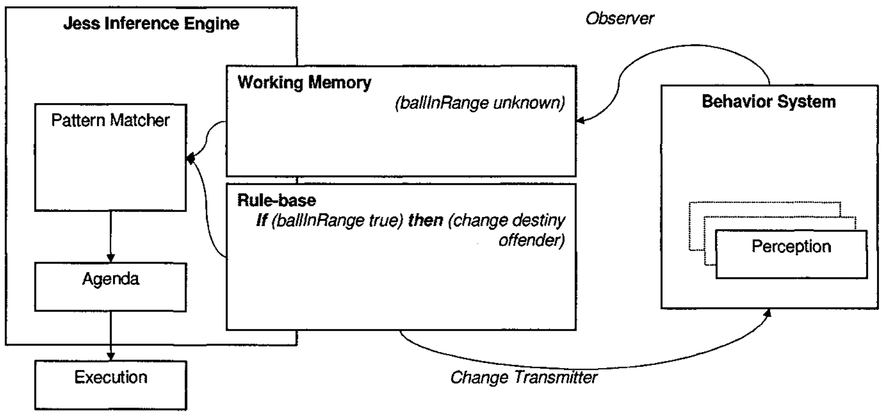


Figure 5. The rule engine allows for autonomy at the strategy level.

In future enhancements we think about using the capabilities of shadowed facts in Jess, which are automatically updated by core rule system mechanism. The way from the expert system to the lower layers is done by extending the Jess functionality by special user-functions which transmits the automation activation and other commands.

Teammate Knowledge. The teammate knowledge base holds the facts and rules which are relevant for one autonomous robot on its own. The rule-base reasons on states of the robot world that are mapped to facts of the expert system. In addition, it reasons on requirements which can be created inside the expert system itself or outside, like other team members or a team server. The interaction with a team server is particularly important for mixed teams like the mixed team of Paderkicker and Tech United participating in the world championship in Bremen this year. The teammate expert system activates and monitors the different tactics (finite state machines) which are required to handle typical situations in RoboCup.

Team Knowledge. There are different ways to form a team out of individual soccer robots. One way is to create one dedicated server process that distributes roles and commands to the teammates. Another way is to design the expert system in a distributed way, so that every robot has the same rule base and there are special rules for team decisions that every teammate makes an independent decision in a given situation. We started with the second solution, implementing our team play as a distributed expert system. That way, we have to handle a lot of special rules for rule assignment and additionally have to hold every piece of information redundant in every database. For the RoboCup world championship in Bremen we need to build a mixed team where both teams — the Paderkickers and TechUnited Eindhoven — have completely different hard- and software. This led to the introduction of a team server that holds a separate Expert System for cooperation of robots from different universities. The team server acts on a coach level like in real soccer games. The different robots register at the team server and transmit facts which are important for a coach. Examples of these facts are the robots' positions and the position of the ball on the playground.

3.2 CONNECTING TEAMS

In order to be able to connect with other soccer robot teams to form one team, a team server serves as a central point, featuring a simple and easy-to-adopt team message format. This was necessary, since other teams should not have to bother with the message format peculiarities that stem from our robots' unified communication including CAN-bus, a serial line, etc. The new team message format thus only supports messages that are dedicated to inter-team communication, world model exchange und routing referee commands between the different teams. It is now being evaluated with TechUnited, the Robocup midsize league team of Eindhoven and Delft.

4. OUTLOOK

At the moment we are redesigning our platform (Fig. 6) having now omni-wheels, allowing for better vision capabilities and supporting inter-team connection for building mixed teams.

Concerning the demanding needs for line detection and higher frame rates the Trimedia processor and the RS232 connection are bottlenecks. Therefore current research takes place in the field of image processing on FPGA where color segmentation and line detection is to be integrated to deliver extracted 2D features easily at speeds of more than 50 fps. A self designed PCB with a FPGA and USB as well as the image processing circuit meet these needs and will be integrated in the near future.

Another purpose of our Paderkicker soccer team is the investigation of appropriate means to propagate learned knowledge in teams of robots [13, 12, 9]. Currently, we develop a framework that enables robots in a team to find their most natural skills. As every robot has a different perception stream the sensorimotorical couplings will be learned much

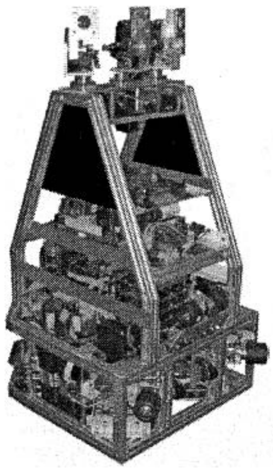
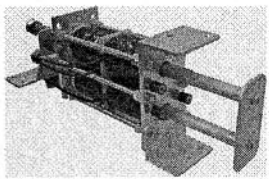



Figure 6. The new platform with a more robust chassis, better vision capabilities, omnidirectional drive, a stronger shooting device (right side), and a software architecture that supports inter-team cooperation.

faster by decentralizing the “babbling phase”, in which they find out basic behaviors by trial and error. These can then be propagated to other team members.

5. CONCLUSION

With the platform described in this paper we achieve the two conflicting goals in the soccer domain needed to reach true autonomy: quick response rates for high reactivity and more complex but less frequent deliberation processes. Components that are subject to quick changes in the environment are arranged in a decentralized manner and are located on specialized hardware. Those processes that do not need that fast update cycles like team communication, planning or processing at the higher levels are located at the Mini-ITX, allowing for faster development cycles but slower execution time. This architecture has evolved naturally out of the needs to combine a fault tolerant embedded system with fast development cycles for behavior exploration.

REFERENCES

[1] R. C. Arkin. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *Proceedings of the IEEE Conference on Robotics and Automation*, 1987.

[2] D. Beier, R. Billert, B. Brüderlin, Bernd Kleinjohann, and Dirk Stichling. Marker-less vision based tracking for mobile augmented reality. In *Proceed-*

ings of the Second International Symposium on Mixed and Augmented Reality (ISMAR 2003), 2003.

[3] H.-D. Burkhard, D. Duhaut, M. Fujita, P. Lima, R. Murphy, and R. Rojas. The road to robocup 2050. *IEEE Robotics and Automation Magazine*, 9(2):31–38, 2002.

[4] Natascha Esau, Bernd Kleinjohann, Lisa Kleinjohann, and Dirk Stichling. MEXI - machine with emotionally extended intelligence: A software architecture for behavior based handling of emotions and drives. In *Proceedings of the 3rd International Conference on Hybrid and Intelligent Systems (HIS'03)*, 2003.

[5] Natascha Esau, Bernd Kleinjohann, Lisa Kleinjohann, and Dirk Stichling. Visitrack - video based incremental tracking in real-time. In *6th IEEE International Symposium on Object-oriented Real-time Computing (ISORC '03)*, 2003.

[6] Ernest Friedman-Hill. *Jess in Action : Java Rule-Based Systems (In Action series)*. Manning Publications, December 2002. ISBN 1930110898.

[7] Ernest Friedman-Hill. Web site for the software Jess, 2005. http://herzberg.ca.sandia.gov/jess/.

[8] Bernd Kleinjohann. The Paderkicker Team, 2006. http://paderkicker.upb.de.

[9] Markus Koch, Willi Richert, and Alexander Saskevic. A self-optimization approach for hybrid planning and socially inspired agents. In *Second NASA GSFC/IEEE Workshop on Radical Agent Concepts*, 2005.

[10] Cody C. T. Kwok, Dieter Fox, and Marina Meila. Real-time particle filters. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *NIPS*, pages 1057–1064. MIT Press, 2002. ISBN 0-262-02550-7.

[11] Christian Reimann. Kick-Real - a mobile mixed reality game. In *ACE2005, ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, 2005.

[12] Willi Richert, Bernd Kleinjohann, and Lisa Kleinjohann. Evolving agent societies through imitation controlled by artificial emotions. In M. Huang, X.-P. Zhang, and M. Huang, editors, *ICIC 2005*, number 3644 in LNCS, pages 1004–1013. Springer-Verlag Berlin, 2005.

[13] Willi Richert, Bernd Kleinjohann, and Lisa Kleinjohann. Learning action sequences through imitation in behavior based architectures. In *Systems Aspects in Organic and Pervasive Computing - ARCS 2005*, number 3432 in LNCS, pages 93–107. Springer-Verlag Berlin, 14 - 17 March 2005.

[14] Dirk Stichling. *VisiTrack - Inkrementelles Kameratracking für mobile Echtzeitsysteme*. PhD thesis, Universität Paderborn, Fakultät für Elektrotechnik, Informatik und Mathematik, 2004.

[15] Dirk Stichling and Bernd Kleinjohann. CV-SDF - a model for real-time computer vision applications. In *IEEE Workshop on Application of Computer Vision*. IEEE, December 2002.

[16] Dirk Stichling and Bernd Kleinjohann. Low latency color segmentation on embedded real-time systems. In Bernd Kleinjohann, K.H. Kim, Lisa Kleinjohann, and Achim Rettberg, editors, *Design and Analysis of Distributed Embedded Systems*. Kluwer Academic Publishers, 2002.

[17] Dirk Stichling and Bernd Kleinjohann. Edge vectorization for embedded real-time systems using the CV-SDF model. In *Proceedings of the 16th International Conference on Vision Interfaces (VI 2003)*, June 2003.

MODULAR COMPILATION OF SYNCHRONOUS PROGRAMS

Klaus Schneider, Jens Brandt, and Eric Vecchié
University of Kaiserslautern
Department of Computer Science
Reactive Systems Group
P.O. Box 3049, 67653 Kaiserslautern, Germany
http://rsg.informatik.uni-kl.de

Abstract: We present a new method for modular compilation of synchronous programs given in imperative languages like Quartz or Esterel. The main idea of our approach consists of computing sequential jobs that correspond with control flow locations of the program. Each job encodes that part of an instantaneous reaction that is triggered by the activation of the corresponding control flow location. The special consideration of the initial job that is executed at initial time yields a simple method for modular code generation.

Keywords: synchronous languages, modular compilation

1. INTRODUCTION

Synchronous languages [13], [2] like Esterel [3] and its variants [14], [19] are particularly interesting for system design: First, it is possible to generate both efficient software and hardware from the same synchronous program. Second, it is possible to determine tight bounds on the reaction time by a simplified worst-case execution time analysis [15]. Third, the formal semantics of these languages allows one to formally prove (1) the correctness of the compilation and (2) the correctness of particular programs with respect to given formal specifications [18], [19], [21].

Although several success stories have been reported [12], there is still a need for further research on efficient and modular compilation of synchronous languages. In the past years, several different compilation techniques have been developed [9], [17], [11]:

Please use the following format when citing this chapter:

Schneider, K., Brandt, J., Vecchié, E., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 75–84.

- The first compilers translated the program to an extended finite state machine whose transitions are endowed with corresponding code fragments [6]. The disadvantage is the potential state-explosion problem; the advantage is the very fast execution time of the generated code.
- Polynomial compilation was first achieved by a translation to equation systems [16], [18], [21] that symbolically encode the automata. The idea behind this approach is to consider control flow locations instead of entire control states[1]. This approach is successfully used for hardware synthesis and it is still the core of commercial tools [12], although the generated software is sometimes comparably slow.
- A third approach has been followed by the Saxo-RT compiler [8], [7] of France Telecom, which translates the program into an event graph. Hence, an event driven simulation scheme can be used to generate code, which is compiled into efficient C code.
- A fourth approach is based on the translation of programs into concurrent control data flow graphs [11], [17], [9], [10], [23], whose sizes depend linearly on the given program. At each instant, the control flow graph is traversed until active nodes are found to trigger the execution of the corresponding subtree.

All of the above approaches have been developed to optimize the compilation time, as well as the size and the execution time of the generated code. However, with the exception of [23], essentially none of the above compilation techniques considered a modular compilation, which is standard for all sequential programming languages.

Modular compilation of synchronous programs is not at all straightforward: A previously compiled module may start or end with an incomplete macro step whose micro steps can interact with the micro steps of later added modules. Hence, to achieve a modular compilation, the *surface* of each module must be known: The surface [4], [22] of a statement consists of those micro steps that are executed at initial time before the first control flow location is reached. Surfaces are the essential information for combining pre-compiled statements.

For this reason, we have developed a completely new compilation technique, which has different advantages [20]. Our compiler splits the given program into so-called jobs that correspond with the control flow locations of the program. Hence, we simply execute those jobs that correspond with the currently active control flow locations. To this end, we have to take care of mutual dependencies that have to be checked by causality analysis. An important sim-

[1]We distinguish between a control flow state that consists of a boolean vector of control flow locations. A control flow location is a statement of the program that can hold the control flow for an instant of time. In case of Esterel, control flow locations are essentially *pause* statements.

plification is obtained by our compilation technique since each job consists of purely sequential code.

For modular compilation, the job-based compilation technique has the advantage that the surface of the compiled module is explicitely given as the unique initial job. Thus, modular compilation is basically achieved by taking the union of the set of jobs and declaring the new initial job as the new surface.

The paper is organized as follows: in the next section, we briefly describe the Esterel/Quartz language that we consider in this paper. We then define the syntax and semantics of an intermediate language that we use to represent the sequential jobs, and we explain the key idea of our job-based compiler. After this, we illustrate the job-based compilation by means of a small example. Then, we explain in detail how modular compilation can be achieved with the job-based code. Finally, we discuss the advantages of our new compilation technique for modular compilation and conclude with a short summary. Details of the compiler are given in [20].

2. THE SYNCHRONOUS LANGUAGE Quartz

Quartz [18], [19], [20] is a descendant of Esterel that shares its basic model with its ancestor Esterel. In this paper, we rely on the common statements and therefore only consider the following:

DEFINITION 1 **[Statements of Quartz]** *The set of statements of* Quartz *is the smallest set that contains the following statements, provided that S, S_1, and S_2 are also statements of* Quartz, *ℓ is a location variable, x is an event variable, y is a state variable, σ is a Boolean expression, and α is a type:*

- *nothing (empty statement)*
- *emit x and emit next(x) (boolean event emissions)*
- *$y = \tau$ and next(y) $= \tau$ (assignments)*
- *ℓ : pause (consumption of time)*
- *if (σ) S_1 else S_2 (conditional)*
- *S_1; S_2 (sequential composition)*
- *S_1 || S_2 (synchronous concurrency)*
- *do S while(σ) (iteration)*
- *[weak] suspend S when [immediate](σ) (suspension)*
- *[weak] abort S when [immediate](σ) (abortion)*
- *$\{\alpha\ y;\ S\}$ (local variable y with type α)*

There are two kinds of (local and output) variables in Quartz, namely *event* and *state variables*: State variables y are persistent, i.e., they store their current value until an assignment changes it, while event variables take a default value if no assignment is made. Executing a delayed assignment *next*$(y) = \tau$ means to evaluate τ in the current macro step (environment) and to assign the obtained value to y in the following macro step. Immediate assignments update y in the current macro step and are therefore rather equations than assignments. As most events are of Boolean type, we use the statements *emit* x and *emit next*(x) as macros for $y =$ true and *next*(y) $=$ true, respectively.

There is only one basic statement that defines a control flow location, namely the *pause* statement[2]. For this reason, we endow *pause* statements with unique Boolean valued *location variables* ℓ that are true iff the control is currently at location ℓ : *pause*.

The semantics of the statements is the same as in Esterel. Due to lack of space, we do not describe their semantics in detail, and refer instead to [19], [18], and, in particular, to the Esterel primer [5], which is an excellent introduction to synchronous programming.

3. COMPUTING JOBS FOR PROGRAMS

In this section, we describe the computation of an equivalent set of jobs for a given Quartz program. As already outlined, the overall idea of the proposed code generator is as follows: For each control flow location ℓ of the program, a job S_ℓ is computed that has to be executed iff the control flow resumes the execution from location ℓ. Of course, several jobs may have to be executed in one macro step since several locations can be active at once.

3.1 THE JOB LANGUAGE

In principle, a job S_ℓ consists of a set of guarded actions and guarded schedule statements (see below) to implement the data flow and the control flow of the program, respectively. However, we do not compute simple sets of guarded statements. Instead, we additionally use conditional and sequential statements to allow sharing of common conditions. Moreover, we use statements for barrier synchronization to implement the concurrency of synchronous programs.

DEFINITION 2 **[Job Language]** *The set of* Job *statements is the smallest set that contains the following statements, provided that S, S_1, and S_2 are also* Job *statements, ℓ is a location variable, x is an event variable, y is a state variable, σ is a Boolean expression, and λ is a lock variable (having integer type):*

- `nothing` *(empty statement)*
- `emit` x *and* `emit next`(x) *(event emissions)*
- $y = \tau$ *and* `next`$(y) = \tau$ *(assignments)*
- `init`(x, τ_0) *(initialize local variable)*
- `schedule`(ℓ) *(resumption at next reaction)*

[2]To be precise, immediate forms of suspend also have this ability.

- `reset`(λ) *(reset a barrier variable)*
- `join`(λ) *(apply for passing barrier)*
- `barrier`(λ, c) *(declare barrier λ)*
- `if`(σ) S_1 `else` S_2 *(conditional)*
- $S_1; S_2$ *(sequential composition)*

Note that there is no longer a parallel statement and also the abort/suspend statements are no longer required. Moreover, there are no loops, since we can implement them by the help of schedule statements (explained below). Furthermore, all job statements are instantaneous[3].

The atomic statements `nothing`, `emit` x, `emit next`(x), $y = \tau$, and `next`$(y) = \tau$ have the same meaning as in Quartz programs. The meaning of conditionals and sequences is also the same as in Quartz. The statement `init`(x, τ_0) replaces a local variable declaration as follows: when executed, it first removes x from the current context as well as pending (delayed) assignments to x, and then gives x the initial default value τ_0.

The `schedule`(ℓ) statement corresponds with a control flow location ℓ of the Quartz program. When executed, it simply puts the label ℓ in the schedule, so that the runtime environment will execute the corresponding job S_ℓ in the next reaction step. Note, however, that `schedule`(ℓ) is instantaneous, so that `schedule`(ℓ_1); `schedule`(ℓ_2) will at once put both ℓ_1 and ℓ_2 to the schedule for the next reaction step.

The statements `reset`(λ), `join`(λ), and `barrier`(λ, c) are used to implement concurrency based on *barrier synchronization*. `barrier`(λ, c) declares a barrier with an integer lock variable λ and an integer constant c as threshold. Executing this statement checks whether $\lambda \geq c$ holds, and if so, it immediately terminates, so that a further statement S can be executed in a sequence like `barrier`$(\lambda, c); S$. If $\lambda < c$ holds, the execution stops, so that the control thread terminates.

Executing `reset`(λ) simply resets $\lambda = 0$, and `join`(λ) first increments λ and then invokes a function f_λ that is associated with the barrier whose lock variable is λ. Usually (and in our compiled jobs always), it is the case that the code of function f_λ is a sequence `barrier`$(\lambda, c); S$ with some statement S.

Using the statements for barrier synchronization, it is straightforward to execute parallel code on a uniprocessor machine: We associate with each parallel statement a barrier with lock variable λ and threshold $c = 2$ that is reset when the parallel statement is started. When a thread of the parallel statement terminates, it executes a `join`(λ) statement. If both threads have executed their final `join`(λ) statements, the barrier will be passed, so that the code following

[3]The job language is therefore also a synchronous language on its own, which is however not meant to be offered to the programmer. Instead, it is used as an intermediate language that could, in principle, be the target for many synchronous languages.

the associated `barrier`(λ, c) statement in the function f_λ associated with the barrier can be executed.

The implementation of the barrier synchronization for other architectures may (and must) be different. Hence, it depends on the platform that is used to execute the program, while our jobs remain architecture-independent. Different implementations for barrier synchronization already exist [1] for hardware, software on multiprocessors, and software on uniprocessors, so that our jobs can be executed on all of these platforms.

3.2 COMPUTING JOBS

The computation of the jobs of a statement is done in a single pass using a recursive function $\mathsf{Jobs}(\cdot,\cdot,\cdot)$. To compile a statement S, we start the function call $\mathsf{Jobs}(S, \textit{nothing}, \{\})$, which computes a tuple $(S_\alpha, \mathcal{P}, \mathcal{F})$ with the following meaning:

- S_α is the surface statement of S, i.e., that code that is executed when S is initially started (which is often viewed as being started from an invisible 'boot' control flow location ℓ_α).
- $\mathcal{P}$ is a set of pairs (ℓ, S_ℓ) such that S_ℓ is the job that is associated with control flow location ℓ.
- $\mathcal{F}$ is a set of pairs (λ, S_λ), where λ is the lock variable of a barrier and S_λ is of the form `barrier`$(\lambda, c); S'$ with some threshold c (hence, S' is the job that is executed when the barrier is passed).

The execution of the initial call $\mathsf{Jobs}(S, \textit{nothing}, \{\})$ will produce subsequent calls $\mathsf{Jobs}(S, S_\eta, J)$ with statements S, S_η with the following meaning: During the function calls, the statement that has to be compiled has been transformed to an equivalent one that is now of the form $S; S_\eta$. Moreover, the set J is either $\{\}$ or a singleton set $\{\lambda\}$. In the latter case, we have to immediately execute `join`(λ) to apply for passing the barrier λ as soon as $S; S_\eta$ terminates. If λ is large enough, the barrier can be passed and the job S_λ associated with the barrier will be immediately executed.

In principle, our compilation procedure performs a symbolic execution of the statement, and each recursive call corresponds with a SOS rule that defines the semantics of Quartz, which allows us to easily verify its correctness. The recursion is made primarily on S, and secondarily on S_η. Details of the compilation are given in a forthcoming publication and also in [20].

4. AN ILLUSTRATING EXAMPLE

A difficult example program (with event input `i` and event outputs `a`, `b`, and `c`) is given in Figure 1. This program suffers from a schizophrenia problem, since the scope of the declaration of the local variable x can be left and re-entered in

```
module Schizo(event i,&a,&b,&c) {
  loop
    {bool x;
      if(i) {
        next(x) = true;
        q1:pause;
      }
      abort {
       emit a; || if(not(x)) emit b;
                  else q2:pause;
       emit c;
       q3:pause;
     } when(not(i));
    }
}
```

Figure 1. A Challenging Example with a Schizophrenic Local Declaration.

```
void f__start() {
 init(x,false);
 if(i) {
    next(x) = true;
    schedule(q1);
 } else {
    reset(_lmb4);
    emit a;
    join(_lmb4);
    if(~x) {
       emit b;
       join(_lmb4);
    } else
       schedule(q2);
 }
}
```

```
void f_q1() {
 reset(_lmb4);
 emit a;
 join(_lmb4);
 if(~x) {
    emit b;
    join(_lmb4);
 } else
    schedule(q2);
}


void g__lmb4() {
 barrier(_lmb4,2);
 emit c;
 schedule(q3);
}
```

```
void f_q2() {
 if(~i) {
   init(x,false);
   reset(_lmb4);
   emit a;
   join(_lmb4);
   if(~x) {
      emit b;
      join(_lmb4);
   } else
      schedule(q2);
 } else
   join(_lmb4);
}
```

```
void f_q3() {
 if(~i) {
   init(x,false);
   reset(_lmb4);
   emit a;
   join(_lmb4);
   if(~x) {
      emit b;
      join(_lmb4);
   } else
      schedule(q2);
 } else {
   init(x,false);
   next(x) = true;
   schedule(q1);
 }
}
```

Figure 2. Sequential Jobs for Module Schizophrenia (Figure 1).

the same macro step. It is well-known that a statement may be entered more than once in a single macro step if the module is called in a surrounding context where the module is nested in several loops.

Figure 2 shows the resulting jobs that are obtained by compilation of this module. As can be seen, our code generator has constructed functions `f_q1`, `f_q2`, and `f_q3` for each control flow location as well as for the boot location (function `f__start`). Moreover, there is a continuation function `g__lmb4` to implement the termination of the parallel statement.

Note that the schizophrenic local declaration is correctly implemented due to the initialization statements that are called in the correct order.

5. MODULAR COMPILATION

Since a previously compiled module may start or end with incomplete macro steps, it is possible that these micro steps can interact with the micro steps of of a surrounding calling module. Hence, we have to consider the potentially incomplete initial and final macro steps of the modules in order to compile them in a modular way. In particular, we have to combine the incomplete macro steps to a complete macro step of the entire module.

The job-based compilation technique presented above lends itself well for this purpose: Assume we have to compile a module M with body statement S for later use. To this end, we first replace the usually used initial function call $\mathsf{Jobs}(S, \mathit{nothing}, \{\})$ for the module's body statement S with the extended function call $\mathsf{Jobs}(S, \mathit{nothing}, \{\lambda_S\})$ with a new lock variable λ_S. Hence, when M terminates, it immediately calls a corresponding continuation function g_{λ_S} with job statement S_{λ_S}. Since g_{λ_S} is not available in the compiled code of M, the runtime environment has to add such a function (with $S_{\lambda_S} :=$ `nothing`) when M is executed without a further context module.

Now assume M is instantiated in a surrounding module M'. Then, the job-based compilation function works as follows: The function call $\mathsf{Jobs}(S, S_\eta, J)$ is replaced by (1) $\mathsf{Jobs}(S, \mathit{nothing}, \{\lambda_S\})$ and (2) $\mathsf{Jobs}(S_\eta, \mathit{nothing}, J)$. Since (1) is what we already compiled in the previous compilation run for module M, we can simply read[4] the compilation result $(S_\alpha, \mathcal{P}, \mathcal{F})$ from the file that contains the results of the previous compilation run. Call (2) is obtained by normal compilation and will thereby generate a triple $(S_\alpha^\eta, \mathcal{P}^\eta, \mathcal{F}^\eta)$. The final result is then $(S_\alpha, \mathcal{P} \cup \mathcal{P}^\eta, \mathcal{F} \cup \mathcal{F}^\eta \cup \{(\lambda_S, \mathtt{barrier}(\lambda_S, 1); S_\alpha^\eta)\})$, i.e., we use the initial job S_α^η of S_η as the continuation function for the barrier λ_S.

Hence, modular compilation can be simply integrated with the job-based compilation technique. The only fact we have to verify is that $\mathsf{Jobs}(S, S_\eta, J)$ is equivalent to $(S_\alpha, \mathcal{P} \cup \mathcal{P}^\eta, \mathcal{F} \cup \mathcal{F}^\eta \cup \{(\lambda_S, \mathtt{barrier}(\lambda_S, 1); S_\alpha^\eta)\})$, where (1) λ_S is a new barrier variable, (2) $(S_\alpha, \mathcal{P}, \mathcal{F}) = \mathsf{Jobs}(S, \mathit{nothing}, \{\lambda_S\})$, and (3) $(S_\alpha^\eta, \mathcal{P}^\eta, \mathcal{F}^\eta) = \mathsf{Jobs}(S_\eta, \mathit{nothing}, J)$ holds.

Note that during the compilation of the context module M', the jobs $\mathcal{P}$ that have been generated by the previous compilation run of M may be modified due to surrounding abortion or suspension statements. These statements have to abort or suspend the job's execution whenever the corresponding abortion or suspension condition holds. Since this is done in the usual job-based compilation as well, we need not discuss this issue further, but we want to note that it may be necessary to modify the jobs $\mathcal{P}$. Moreover, several module calls to M may even require copies of the jobs $\mathcal{P}$.

Another problem is posed by causality analysis: Although all modules can be checked independently of each other, a complete causality analysis can be only performed after all modules have been linked together. Hence, causality analysis has to be done after the complete compilation. Nevertheless, it may

[4]Clearly, substitutions may be necessary due to the given arguments of the module instantiation.

be additionally done as well on single modules after their local compilation in order to speed up the final causality analysis.

Equation-based code can be integrated in the modular compilation scheme as well: The compiler just wraps the equations into two jobs: All initial equations define the initial job, and the transition equations define the main job j_{main}. Both jobs conclude with a check whether at least one control flow location is active: If such a location exists, j_{main} is scheduled again, otherwise, the exit continuation function is joined.

6. SUMMARY

In this paper, a very simple code generation scheme has been presented that is based on splitting the given program into sequential jobs that correspond with the control flow locations of the program. Additionally, continuation functions are required in order to avoid an exponential blow-up of the code, and to efficiently execute parallel statements on uniprocessor systems.

In particular, we have shown in this paper that our compilation technique is suited for modular compilation, since the jobs explicitly contain the surface of the program given as the initial job `f_start`. Modular compilation is not as simple as known from sequential programming languages, since a reprocessing of the compiled module cannot be avoided. However, the main benefits of modularization remain: Compiled and potentially highly-optimized components can be distributed and reused. Moreover, they can be shared without revealing their source codes, since the generated jobs are a rather low-level (but still adaptable) description from which the original code cannot be reconstructed.

REFERENCES

[1] ANDREWS, G. *Concurrent Programming – Principles and Practice.* The Benjamin/Cummings Publishing Company, Redwood City, California, 1991.

[2] BENVENISTE, A., CASPI, P., EDWARDS, S., HALBWACHS, N., LE GUERNIC, P., AND DE SIMONE, R. The synchronous languages twelve years later. *Proceedings of the IEEE 91*, 1 (2003), 64–83.

[3] BERRY, G. The foundations of Esterel. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, G. Plotkin, C. Stirling, and M. Tofte, Eds. MIT, 1998.

[4] BERRY, G. The constructive semantics of pure Esterel. http://www-sop.inria.fr/esterel.org, July 1999.

[5] BERRY, G. The Esterel v5_91 language primer, June 2000.

[6] BERRY, G., AND GONTHIER, G. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming 19*, 2 (1992), 87–152.

[7] CLOSSE, E., POIZE, M., PULOU, J., SIFAKIS, J., VENTER, P., WEIL, D., AND YOVINE, S. TAXYS: A tool for the development and verification of real-time embed-

ded systems. In *Conference on Computer Aided Verification (CAV)* (Paris, France, 2001), vol. 2102 of *LNCS*, Springer, pp. 391–395.

[8] CLOSSE, E., POIZE, M., PULOU, J., VENIER, P., AND WEIL, D. SAXO-RT: Interpreting Esterel semantics on a sequential execution structure. *Electronic Notes in Theoretical Computer Science (ENTCS) 65*, 5 (2002). Workshop on Synchronous Languages, Applications, and Programming (SLAP).

[9] EDWARDS, S. An Esterel compiler for large control-dominated systems. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 21*, 2 (February 2002), 169–183.

[10] EDWARDS, S. ESUIF: An open Esterel compiler. *Electronic Notes in Theoretical Computer Science (ENTCS) 65*, 5 (2002). Workshop on Synchronous Languages, Applications, and Programming (SLAP).

[11] EDWARDS, S., KAPADIA, V., AND HALAS, M. Compiling Esterel into static discrete-event code. In *Synchronous Languages, Applications, and Programming (SLAP)* (Barcelona, Spain, 2004).

[12] ESTEREL TECHNOLOGY. Website. http://www.esterel-technologies.com.

[13] HALBWACHS, N. *Synchronous programming of reactive systems*. Kluwer, 1993.

[14] LAVAGNO, L., AND SENTOVICH, E. ECL: A specification environment for system-level design. In *International Design Automation Conference (DAC)* (New Orleans, Louisiana, USA, 1999), ACM, pp. 511–516.

[15] LOGOTHETIS, G., AND SCHNEIDER, K. Exact high level WCET analysis of synchronous programs by symbolic state space exploration. In *Design, Automation and Test in Europe (DATE)* (Munich, Germany, March 2003), IEEE Computer Society, pp. 196–203.

[16] POIGNÉ, A., AND HOLENDERSKI, L. Boolean automata for implementing pure Esterel. Arbeitspapiere 964, GMD, Sankt Augustin, 1995.

[17] POTOP-BUTUCARU, D., AND DE SIMONE, R. Optimizations for faster execution of Esterel programs. In *Formal Methods and Models for Codesign (MEMOCODE)* (Mont Saint-Michel, France, 2003), IEEE Computer Society, pp. 227–236.

[18] SCHNEIDER, K. A verified hardware synthesis for Esterel. In *Workshop on Distributed and Parallel Embedded Systems (DIPES)* (Schloß Ehringerfeld, Germany, 2000), F. Rammig, Ed., Kluwer, pp. 205–214.

[19] SCHNEIDER, K. Embedding imperative synchronous languages in interactive theorem provers. In *Conference on Application of Concurrency to System Design (ACSD)* (Newcastle upon Tyne, UK, June 2001), IEEE Computer Society, pp. 143–156.

[20] SCHNEIDER, K. The synchronous programming language Quartz. Internal Report to appear, Department of Computer Science, University of Kaiserslautern, 2006.

[21] SCHNEIDER, K., BRANDT, J., AND SCHUELE, T. A verified compiler for synchronous programs with local declarations. *Electronic Notes in Theoretical Computer Science (ENTCS)* (2006).

[22] SCHNEIDER, K., AND WENZ, M. A new method for compiling schizophrenic synchronous programs. In *Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)* (Atlanta, USA, November 2001), ACM, pp. 49–58.

[23] ZENG, J., AND EDWARDS, S. Separate compilation of synchronous modules. In *International Conference on Embedded Software and Systems (ICESS)* (Xian, China, 2005).

TRENDS IN TIMING ANALYSIS

Björn Lisper
Dept. of Computer Science and Electronics
Mälardalen University
P.O. Box 883
SE-721 23 Västerås
Sweden
bjorn.lisper@mdh.se

Abstract Static Worst-Case Execution Time (WCET) analysis aims to find safe upper bounds to the execution time of a program. We give a brief status report on the field of static WCET analysis, and we then present a personal perspective on the current and anticipated forthcoming trends in the area.

Keywords: Real-time System, Timing Analysis, Program Analysis

1. INTRODUCTION

A *Worst-Case Execution Time* (WCET) analysis finds an upper bound to the largest possible execution time of a computer program, or a time-critical part of a program. Reliable WCET estimates are crucial when designing and verifying embedded and real-time systems, especially safety-critical ones.

The WCET is often estimated through measurements. Estimates obtained in this way are, however, not reliable in general. An alternative is *static WCET analysis*, which determines a timing bound from mathematical models of the software and hardware. If the models are correct, then the analysis will derive a timing bound that is *safe*, i.e., greater than or equal to the true WCET.

In this paper, we discuss WCET analysis, its current status, and trends. What can be achieved today? How will trends in hardware and software affect the field? What is needed to turn WCET analysis into a widespread technique? This is a personal perspective, and we make no claims of completeness and scientific rigor.

The rest of this paper is organized as follows. In Section 2 we describe briefly what WCET analysis is, which basic approaches there are, and

Please use the following format when citing this chapter:

Lisper, B., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 85–94.

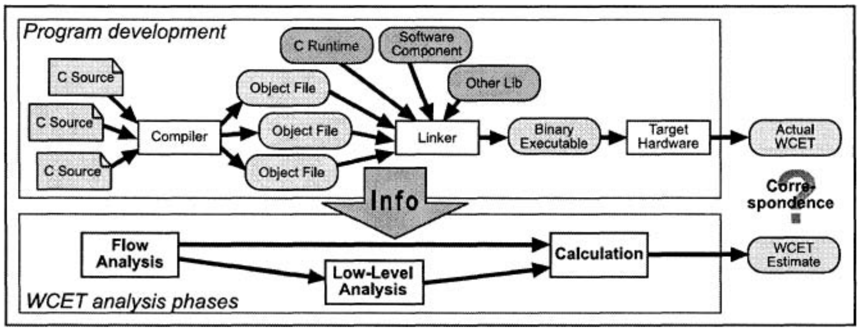


Figure 1. Embedded program development and static WCET analysis.

give a short account for the current status of the field. In Section 3 we discuss how trends in hardware architecture will affect WCET analysis. Section 4 provides a similar discussion as regards software, and requirements on WCET analysis. In Section 5, we monitor the research trends in WCET analysis. In Section 6, finally, we wrap up.

2. WCET ANALYSIS

Static WCET analysis is usually divided into three phases: a (fairly) machine-independent *flow analysis* of the code, where information about the possible program execution paths is derived, a *low-level analysis* where the execution times for sequences of instructions are decided from a performance model for the target architecture, and a final *calculation* phase where the flow and timing information are combined to yield a WCET estimate. See Figure 1.

The purpose of the flow analysis phase is to find constraints on the possible execution paths of the program, like upper limits to loop iterations, infeasible path constraints, etc. *Automatic* flow analysis calculates this information with little or no manual intervention. It is in general impossible to derive exact information. Automatic methods therefore calculate approximate flow information, and allow additional information as *manual annotations* (3, 13, 14).

Flow analysis can be done on source code, on compiler-intermediate code, or on binary code. In the first two cases, the flow information must somehow be mapped to the binary code. Approaches include abstract interpretation to bound the values of execution counters (7), pattern-matching (8, 24), symbolic execution (17), and the use of Presburger Arithmetics to identify loop parameters (3).

Today, most processors use performance-enhancing features such as *pipelines*, *caches*, *branch predictors*, and *instruction-level parallelism.*

These features unfortunately yield complex timing models. Low-level analysis research mostly concerns methods to deal with such timing models.

The timing effects of pipelines are mostly local, and can be handled quite well (4). Caches are harder to analyze, since they yield a global, history-dependent timing model, where memory access times depend on the current contents in the cache. A *cache analysis* attempts to predict the contents of the cache (6). Instruction streams often are quite easy to predict, and thus instruction caches can often be handled quite well, whereas data caches often are subject to more dynamic memory accesses, and thus are harder to analyze. Branch predictors, like caches, yield history-dependent performance models, and low-level analyses for branch predictors attempt to estimate their states (2).

Modern high-end processors use instruction-level parallelism, often through a superscalar architecture. Such architectures are hard to analyze w.r.t. timing properties, due to their dynamic instruction scheduling. Nevertheless, attempts have been made (15).

The final step in WCET analysis is to calculate a WCET estimate from flow and timing information. Three common methods are *tree-based calculation* (18), *path-based calculation* (8, 23), and *IPET* (Implicit Path Enumeration Technique) (14, 21). IPET is the most common calculation method today, and it calculates a WCET bound by solving an integer linear programming problem.

WCET analysis is currently taking the step from research to industrial use. Three commercial tools exist, which are mainly used in automotive and avionics industry to analyze hard real-time systems. The technique is, however, not yet widely adopted.

For reasonably simple embedded processors, with pipeline but without cache, the current tools can provide WCET estimates which are typically 5–10% larger than the largest measured execution time. However, most programs today require extensive manual annotations, constraining the program flow, to get there. The annotations often require deep understanding of the code, and they are cumbersome to make. More complex processor architectures can also be handled, but for a more limited set of programs, and analysis times grow rapidly. The practical limit today seems to be at the level of processors like Motorola ColdFire 5307 and PowerPC 755 (24).

3. HARDWARE

We see a trend towards even more complex processor architectures, with more advanced features enhancing the overall performance. Embedded systems tend to migrate towards more complex processors. Therefore, future WCET analysis methods must be able to keep up with the development of high-performance architectures.

There is an inherent conflict between analyzability and modern performance-enhancing features. These features typically make the processor adapt dynamically to run the code at hand faster, through caches, branch predictors, etc. They typically record the execution history into a complex internal processor state. The more complex the internal state, the harder the timing analysis. Not only does it become harder to estimate the WCET with good precision, the difference between the true WCET and the average execution time also tends to increase.

On the other hand, a trend in embedded systems is to put more of the architecture under software control, in order to cut corners in chip cost, and to allow for smart optimizations. Two examples are *software-controlled cache locking*, and *scratchpad memories*. Both allow a more explicit handling of on-chip memory, which is beneficial for analyzability (26, 25).

Unwanted interaction between hardware features can hurt the analysis. A modularized WCET analysis is often less costly. For instance, the analysis is simplified if the cache behaviour can be analyzed separately, to provide information about hits and misses, which then is used in the further low-level analysis. If more misses always implies longer execution time, then the cache analysis can safely assume that a memory access is a miss if it is not surely found to be a hit, which makes the analysis less complex. However, for superscalar processors, a cache miss may yield a shorter execution time (17). This is due to the dynamic instruction scheduling, where a delayed release of an instruction in the end may give a better schedule. Unwanted interactions of this kind necessitate a more integrated low-level analysis, which can be very detrimental to performance since the sets of possible hardware states grow rapidly.

Another kind of unwanted interaction is due to *sharing of resources which have a stateful timing model.* For instance, if a processor has separate instruction and data caches, then the analysis of one cache may be precise even if the other is not. However, some processors have a shared cache: a poor cache hit/miss estimation for one of the access streams will then typically pollute the information about the whole cache contents, even if the other access stream could have been accurately analyzed. As a result, the total cache analysis will be less precise.

A very important paradigm shift in processor architecture is the introduction of *multicore processors*, where a single chip hosts a multiprocessor. High-end PC's already have dual-core Pentiums, and in a few years multicore processors are likely to prevail. This changes the rules of the game radically.

On the positive side, the processor cores will be simpler than current high-performance processors. Thus, that part of the low-level analysis will be simpler. On the other hand, the processors will often have shared resources, like buses, and cache memories. Since different threads will access these resources, the concurrency will lead to a combinatorial explosion of possible state transitions, which then yields both long analysis time and poor precision.

A big problem is that all current WCET analysis methods assume a strictly sequential execution model. In general, the real-time research community seems to assume that parallel hardware is to be utilized by sequential tasks running on different processors. This assumption is very doubtful: computationally heavy tasks are advantageous to parallelize, and most likely this will happen. WCET analysis for parallel programs must then be developed in order to analyze such tasks.

4. SOFTWARE AND REQUIREMENTS

Traditionally, time-critical embedded applications have been programmed in C or assembler, and current WCET analysis methods mostly target C or code generated from C. This is imperative code, which often has a reasonably simple structure. Such code is relatively easy to analyze. Whole programs are typically analyzed, which means that the tool has full information about the code.

However, there is a migration to higher levels of abstractions. *Higher level programming languages*, such as object-oriented (OO) languages, become more used. *Model-based design*, where code is generated from models, is rapidly gaining ground in many application areas. *Component-based software engineering*, where program components are reused and combined, is also a strong trend.

Higher-level programming languages give rise to new problems. The control structure typically becomes more dynamic. For OO languages, with methods rather than functions, methods are accessed indirectly through a method table. This makes it harder to reconstruct the control flow. Data structures also tend to be more dynamic, and memory management might be automatic. It is then harder to decide the adresses of memory accesses, which makes it much harder to predict the memory access times.

Some high-level languages, like Java, are implemented on virtual machines. These machines introduce an additional abstraction layer between the program and the hardware, which makes analysis harder. If the implementation uses just-in-time compilation, then it will be very hard to predict the actual instructions executed. WCET analysis of JVM code has been attempted (19), but the results have not been too encouraging.

However, many programs written in high-level languages do have a quite static structure. The abstraction mechanisms may be used to support reuse over different static configurations, rather than for truly dynamic run-time behaviour. For such programs, program analyses may be able to uncover the static structure, as well as other properties like sizes of computed data structures (9). Program specialization techniques, such as *partial evaluation* (10), may also yield more analyzable (and also faster) programs. Declarative languages are particularly amenable to program analysis and transformation, due to their clean semantics.

Model based design, where code is generated from models, poses new challenges and possibilities as regards WCET analysis. Since different tools, for different purposes, generate very different code, the challenges will be very tool-specific. For instance, code generated for a state machine is likely to be highly unstructured, whereas control system code generated from a simulation model should have a more regular structure. The way a tool chooses to implement a model feature, like a FSM, can also have a great impact on the WCET analyzability.

Obviously, WCET analysis of generated code should use tool-specific information about how the code is generated whenever possible. It may, for instance, be the case that a tool generates code where the loop iteration bounds are directly related to the size of a matrix in the model. Annotations bounding the loop iterations can then be automatically generated from the model. Experiments in this direction have been done for Simulink (12).

Component-based software engineering (CBSE) is gaining ground. It emphasizes the structuring of software into reusable components, with well-defined interfaces. There are many different component models, ranging from quite static models, where components are statically configured, to very dynamic models where components can be swapped at runtime.

A key feature is communication. Since the component model typically is independent of the component implementation language, and even the host processor, the communication often includes marshaling of data between different formats. This is especially noticeable for distributed component models, where middleware like CORBA is common.

There is an increasing interest in CBSE for embedded systems. This raises the issue of how to analyse such software w.r.t. timing. Again, there are a number of problems. First, components may be "black boxes", with little information about the code. Second, reusable components may be heavily parameterized. If the WCET is parameter-dependent, then a single upper bound may be very untight. Third, complex communication protocols may make the component communication very hard to analyze.

These problems must be solved to make WCET analysis useful for component-based software systems. Conversely, component models for hard real-time systems should be designed to be analyzable.

Little attention has been paid to the role of WCET analysis in the *software development process.* The current tools require compiled binaries, and can thus be applied only late in the development process. However, timing-aware software design is often done with *time budgets*, where different software parts are given maximal execution times. It is then very interesting to estimate the execution time early, before the complete binaries are available, to help set up these budgets. Thus, there seems to be a market for early, approximate WCET analysis, using, say, incomplete source code and crude performance models.

For high performance processors, the current WCET estimates tend to be very pessimistic compared with the average execution time. Many applications, especially in areas like telecommunications and multimedia, have Quality of Service (QoS) requirements. Infrequent deadline violations are then not harmful, and some violations are typically allowed in order to utilize the hardware better. For such systems, it would be more appropriate to derive statistical estimates for violations of given deadlines, rather than absolute upper WCET bounds. A step in that direction is the pWCET framework for probabilistic WCET analysis (1).

5. WCET ANALYSIS TECHNIQUES

What developments can be expected in WCET analysis? We believe that there is a need for approximate WCET analyses, with some kind of probabilistic guarantees. This is due to the demands from the large set of QoS-oriented applications, as well as the wish to introduce timing analysis early in the tool chain. We also believe that hybrid analyses, involving both static analysis and measurements, will be further developed. Such analyses can avoid the costly and pessimistic low-level analysis, at the price of not obtaining absolutely safe upper WCET bounds. For single-path programs, measurements can give safe and accurate WCET estimates, and a single-path programming style has been advocated (20). The aforementioned probabilistic WCET analysis (1) also includes measurements.

Traditional, static low-level analysis is also developing. A very interesting trend is the incorporation of methods from model checking to handle very large state spaces. For instance, BDD's have been proposed (27).

WCET-aware compilation attempts to improve the WCET, or its analyzability, rather than average performance. An example is dynamic cache locking to keep the cache contents predictable (25).

An important aspect of WCET analysis tools is their *usability.* Current tools require many manual annotations, in particular to constrain the program flow. To give flow information manually is cumbersome, and requires a deep understanding of the code. This restricts the usability of the tools (5). The level of automation must be raised, which requires better methods in automatic flow analysis.

However, to find tight flow information is difficult. Loop bounds may depend in complex ways on inputs, pointers, etc. Flow analysis of binary code is especially difficult. If the compiler can map flow information from the source code to the binaries, then this problem is alleviated. In particular, manual annotations are best given on source code level, and they cannot be completely avoided in general. Studies in this direction have been made (11), but production compilers must adopt this kind of technique if it is to have any impact on real practice.

Another option is *parametric* WCET analysis (16). Such an analysis computes a formula for the WCET bound rather than a single number. Code with input-dependent WCET is common in many applications (22). A parametric analysis can also help analyze the sensitivity of the WCET w.r.t. different parameters, which is useful when developing code with time budgets.

6. CONCLUSIONS

WCET analysis is a maturing technology that is currently being introduced in industry. However, it is not widely established yet. While promising and important, many challenges remain to meet before the technique will be adopted on a wider basis. The most important challenges are to keep up with the hardware development, to increase the level of automation of the analysis, and to widen the scope to include also soft real-time systems with QoS requirements.

The ability to meet these challenges depends on hardware, compilers, and other tools which generate code. If one were to make a wish list, it would include: WCET-aware compilers, with the ability to map program flow information from source code to binary code, and hardware which avoids shared resources and hidden stateful features to improve average performance. For future multicore processors, we wish for mechanisms to partition shared resources, like caches or scratchpad memories, between tasks.

ACKNOWLEDGMENTS

I want to thank Jan Gustafsson, Andreas Ermedahl, and Christer Sandberg for their valuable comments.

REFERENCES

[1] Guillem Bernat, Antoine Colin, and Stefan M. Petters. WCET analysis of probabilistic hard real-time systems. In *Proc. 23rd IEEE Real-Time Systems Symposium*, 2002.

[2] François Bodin and Isabelle Puaut. A WCET-oriented static branch prediction scheme for real time systems. In *Proc. 17th Euromicro Conference of Real-Time Systems*, pages 33–40, July 2005.

[3] Bound-T tool homepage, 2006. `www.tidorum.fi/bound-t/`.

[4] Jakob Engblom. *Processor Pipelines and Static Worst-Case Execution Time Analysis.* PhD thesis, Uppsala University, Sweden, April 2002.

[5] Andreas Ermedahl, Jan Gustafsson, and Björn Lisper. Experiences from industrial WCET analysis case studies. In Reinhard Wilhelm, editor, *Proc. 5th Int. Workshop on Worst-Case Execution Time Analysis*, pages 19–22, Palma de Mallorca, July 2005.

[6] Christian Ferdinand and Reinhard Wilhelm. Efficient and precise cache behavior prediction for real-time systems. *Real-Time Systems*, 17:131–181, 1999.

[7] Jan Gustafsson, Andreas Ermedahl, and Björn Lisper. Towards a flow analysis for embedded system C programs. In *Proc. 10th IEEE Int. Workshop on Object-oriented Real-time Dependable Systems*, Sedona, USA, February 2005.

[8] C. Healy, R. Arnold, Frank Müller, David Whalley, and M. Harmon. Bounding pipeline and instruction cache performance. *IEEE Transactions on Computers*, 48(1), January 1999.

[9] John Hughes, Lars Pareto, and Amr Sabry. Proving the correctness of reactive systems using sized types. In *Proc. 23rd ACM Symposium on Principles of Programming Languages*, pages 410–423, 1996.

[10] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice Hall, Hertfordshire, UK, 1993.

[11] Raimund Kirner. *Extending Optimising Compilation to Support Worst-Case Execution Time Analysis.* PhD thesis, Technische Universität Wien, Austria, 2003.

[12] Raimund Kirner, R. Lang, G. Freiberger, and Peter Puschner. Fully automatic worst-case execution time analysis for matlab/simulink models. In *Proc. 14th Euromicro Conf. of Real-Time Systems*, pages 31–40, June 2002.

[13] Raimund Kirner and Peter Puschner. Transformation of path information for WCET analysis during compilation. In *Proc. 13th Euromicro Conf. of Real-Time Systems*, pages 29–36, Delft, June 2001. IEEE Computer Society Press.

[14] Y-T. S. Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. In *Proc. 32:nd Design Automation Conf.*, pages 456–461, 1995.

[15] S. Lim, J. Han, J. Kim, and S. L. Min. A worst case timing analysis technique for multiple-issue machines. In *Proc. 19th IEEE Real-Time Systems Symposium*, December 1998.

[16] Björn Lisper. Fully automatic, parametric worst-case execution time analysis. In Jan Gustafsson, editor, *Proc. 3rd International Workshop on Worst-Case Execution Time Analysis)*, pages 77–80, Porto, July 2003.

[17] Thomas Lundqvist and Per Stenström. An integrated path and timing analysis method based on cycle-level symbolic execution. *Journal of Real-Time Systems*, May 2000.

[18] Chang Yun Park and Alan C. Shaw. Experiments with a program timing tool based on a source-level timing schema. *IEEE Computer*, 24(5):48–57, 1991.

[19] Peter Puschner and Guillem Bernat. WCET analysis of reusable portable code. In *Proc. 13th Euromicro Conf. of Real-Time Systems*, pages 45–52, Delft, June 2001.

[20] Peter P. Puschner and Alan Burns. Writing temporally predictable code. In *Proc. 7th IEEE Int. Workshop on Object-oriented Real-time Dependable Systems*, pages 85–94, San Diego, January 2002. IEEE Computer Society.

[21] Peter P. Puschner and Anton V. Schedl. Computing maximum task execution times – a graph-based approach. *Journal of Real-Time Systems*, 13(1):67–91, July 1997.

[22] D. Sandell, A. Ermedahl, J. Gustafsson, and B. Lisper. Static Timing Analysis of Real-Time Operating System Code. In *Proc. 1st Int. Symposium on Leveraging Applications of Formal Methods*, October 2004.

[23] Friedhelm Stappert, Andreas Ermedahl, and Jakob Engblom. Efficient longest executable path search for programs with complex flows and pipeline effects. In *Proc. 4th Int. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, November 2001.

[24] S. Thesing. *Safe and Precise WCET Determination by Abstract Interpretation of Pipeline Models.* PhD thesis, Saarland University, 2004.

[25] Xavier Vera, Björn Lisper, and Jingling Xue. Data Cache Locking for Higher Program Predictability. In *Proc. Int. Conf. on Measurement and Modeling of Computer Systems*, pages 272–282, San Diego, CA, June 2003. ACM Press.

[26] Lars Wehmeyer and Peter Marwedel. Influence of onchip scratchpad memories on WCET prediction. In Isabelle Puaut, editor, *Proc. 4th Int. Workshop on Worst-Case Execution Time Analysis*, pages 15–18, Catania, June 2004.

[27] Stephan Wilhelm. Efficient analysis of pipeline models for WCET computation. In Reinhard Wilhelm, editor, *Proc. 5th Int. Workshop on Worst-Case Execution Time Analysis*, pages 19–22, Palma de Mallorca, July 2005.

TRAFFIC SCHEDULING ANOMALIES IN TEMPORAL PARTITIONS

Luís Almeida, Paulo Pedreiras, Ricardo Marau
LSE – IEETA / DET, Universidade de Aveiro, 3810-193 Aveiro, Portugal

Abstract: Many network protocols rely on temporal partitions to provide isolation between different nodes (TDMA slots) or different traffic classes (multi-phase cyclic frameworks). Typically, the duration of the slots or phases is not correlated with the duration of packet transmissions, which is variable and non-preemptive. Thus, it is possible that the limit of the slot or phase be overrun by an on-going packet transmission or, if this cannot be tolerated, idle-time must be inserted at the end of the slot or phase whenever a packet does not fit in. Nevertheless, both situations lead to scheduling anomalies in which the worst-case network delay does not occur necessarily with the synchronous release of all other packets, or just the higher priority ones. This paper highlights two such anomalies showing their origin and indicating that, in such circumstances, it is not possible to determine the worst-case network delay with exactitude in the general case. However, it is still possible to upper bound the network delay and the paper shows non-optimal solutions for those cases.

Key words: real-time systems, real-time scheduling, real-time communication, distributed real-time systems

1. INTRODUCTION

The expression *scheduling anomaly* in real-time processor scheduling is typically used to refer to a counter intuitive phenomenon in which increasing the systems resources or relaxing the application constraints can lead to increased schedule length. In a real-time scope, this means that previously schedulable sets may then become non-schedulable. Probably the first such anomalies being identified were the multiprocessor scheduling anomalies described in (Graham, 1976) and known as Richard's anomalies (Stankovic

Please use the following format when citing this chapter:

Almeida, L., Pedreiras, P., Marau, R., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 95–104.

etal., 1995). The cause for such anomalies was related to the use of mutually exclusive shared resources. A simple illustrative example is shown in (Stankovic *etal.*, 1995) in which the reduction of the execution time of one task inverts the order by which two subsequent competing tasks, allocated to different processors, access a shared resource. Such inversion may cause an extra delay in the global schedule, possibly violating deadlines.

In off-line scheduled applications, the anomalies were often hidden by optimization algorithms that would implicitly avoid them by acting upon the task offsets and inserting idle-time where needed to delay the execution of certain tasks. However such kind of optimization algorithms are too complex to be used online and thus, under dynamic conditions, e.g. sporadic arrivals, anomalies can indeed occur and generate worst-case response times that are worse than expected. Buttazzo (2002) has shown that, when using variable speed processors to reduce energy consumption, the anomalies can occur even in uniprocessor systems whenever tasks synchronize in the access to shared resources. To prevent such situation, he proposes using non-synchronized task interactions through asynchronous buffers. He also refers to a previous work by Mok (2000) that shows that anomalies can also occur in variable speed uniprocessor systems with non-preemptive tasks. Recently, Chen *etal* (2005) showed more anomalies in uniprocessor systems arising from hardware configuration changes, such as processor upgrading, with mutually exclusive shared resources and active I/O devices. In their work, they propose three rules to prevent scheduling anomalies, namely inserting idle-time, enforcing the same order when accessing resources and dealing separately with passive and active resources.

Another form of anomaly can occur when using temporal partitions in hierarchical scheduling frameworks. These have been recently studied by Shin and Lee (2003), Bini and Lipari (2004), Almeida and Pedreiras (2004), Davis and Burns (2005) and others. In these studies, it has become clear that the response time of a task within a partition depends not only on the set of interfering tasks that run within the same partition but also on the sequence of availability periods of the respective partition. Particularly, Davis and Burns (2005) show that the partition utilization as a function of the partition period, subject to assuring the schedulability of a given task set, presents local minima that can be very sharp in special cases. These cases correspond to the situations in which the tasks periods are integer multiples of the partition period and the tasks are released synchronously with the partition instances. This leads to a good packing of the tasks within the partition and any slight variation, either reduction or increase, in the partition period will require a significant increase in the partition size to fulfill the same task requirements. This is also an anomaly since reducing the partition period, i.e. increasing computing resources, may turn schedulable tasks unschedulable.

In this paper, we will address a similar case to the previous one, but considering non-preemptive execution. This model is typical in real-time networks used for control applications, in which communication is achieved through single packet transfers over shared broadcast buses. In this scope, temporal partitions have long been used either to control the access to the shared medium, e.g. TDMA, or to separate different classes of traffic, e.g. the multi-phase cyclic framework.

We will start by presenting our reference model and then highlighting two anomalies that may occur in these systems. We will terminate by presenting one technique for each case that allows bounding the anomaly duration and consequently its worst-case impact on network induced delay.

2. SCHEDULING MODEL

We consider a communication system based on a shared broadcast bus. The bus conveys messages which, for the purpose of this work, are considered as single packet and thus are transmitted non-preemptively. This is common in control networks or networks of sensors and actuators. Moreover, we consider the bus to be time-partitioned, meaning that specific types of communication can only occur within pre-defined intervals of time, being suspended outside those intervals. Examples of time-partitions are TDMA slots as in TT-CAN and TTP/C, and periodic/aperiodic phases, sometimes referred to as synchronous/asynchronous, as in WorldFIP, FlexRay, Ethernet POWERLINK, or the FTT protocols. Token passing protocols, e.g. PROFIBUS, also constrain the intervals during which nodes may access the bus, i.e., the token holding intervals. When the token holding time expires the nodes do not issue further transmissions no matter the number or priority of their ready messages. In this sense, the token holding intervals of a given node also form a temporal partition.

Therefore, we consider a time-partitioned system characterized by a set M of messages that will be scheduled within a partition Π. The message set M is defined as in (1), containing N periodic messages. Message m_i has maximum transmission time C_i, period T_i, initial offset Φ_i and priority Pr_i either fixed or variable depending on the particular scheduling policy.

$$M \equiv \{m_i\,(C_i,\ T_i, \Phi_i, Pr_i),\ \ i=1..N\} \qquad (1)$$

The partition Π is formed by an infinite periodic sequence of time intervals or windows, during which the bus is available for transmitting the messages of set M. The period of the windows, also called the partition period, is π, and the windows duration, also called partition capacity, is called w (2). In TDMA protocols, π is the TDMA round and w is the slot duration. In multi-phase cyclic frameworks, π is the micro-cycle and w is the phase duration. Moreover, as expressed in (2), we consider two types of partitions depending on whether the partition capacity is constant (type 1) or variable (type 2). TDMA protocols use partitions of type 1 only, while multi-phase cyclic frameworks normally use partitions of both types, the periodic phase being typically of type 1 and the aperiodic phase of type 2 since it reclaims the capacity not used by the periodic phase (Fig. 1).

$$\Pi^1 = \Pi^1(\pi, w=\text{cons}^{\text{te}}) \; ; \quad \Pi^2 = \Pi^2(\pi, w_{min}<w<w_{max}) \qquad (2)$$

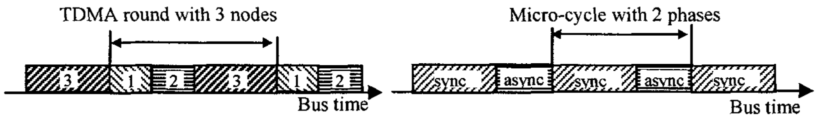


Figure 1. Examples of time-partitioned buses.

Moreover, we consider the following constraints on the message set, which are typically found in the communication systems referred before. Firstly, the message transmission time is always substantially shorter than the partition capacity ($\forall m_i,\ C_i \ll w$). Secondly, the message periods and initial offsets are integer multiples of the partition period ($\forall m_i,\ \exists n \in \{1,2,\ldots\}$: $P_i = n * \pi$ and $\exists l \in \{0,1,\ldots\}: \Phi_i = l * \pi$) and the messages are released synchronously with the start of partition windows. This model is typically found in TDMA and master-slave networks that are based on cyclic frameworks.

A helpful definition that applies to both types of partitions is to consider the infinite sequence of partition windows, each characterized by a start and finish instant as in (3).

$$\Pi \equiv \{w_k(w_k^s, w_k^f),\ k=0..\infty\} \qquad (3)$$

Then, we can define the capacity of each partition window as $w_k = w_k^f - w_k^s$. Notice that $w_k = w\ \forall_k$ for type 1 partitions. Also, we can define the load of each partition window as in (4), corresponding to the total transmission time of the messages scheduled within w_k. Notice that w_k will be used indistinctively to refer to the $(k+1)^{th}$ partition window as well as to its length.

$$load(w_k) = \sum_{m_i \in w_k} C_i \qquad (4)$$

An important aspect is that w_k is normally uncorrelated with $\{C_i, i=1..N\}$ and thus two situations can occur when scheduling messages within a given partition window, depending on whether an overrun of the partition capacity is allowed or not. If it is not allowed, as in TDMA systems in which there must be a strict temporal isolation between slots, a message m_i is transmitted within window w_k if its transmission completes before or at w_k^f. Otherwise, it is delayed to the following windows. This can lead to the insertion of idle-time at the end of window w_k (Figure 2, top line). In other systems, typically in periodic phases, any message that can start transmission before w_k^f is actually transmitted, thus overrunning the w_k limit (Figure 2, lower line). Despite seeming different, this latter approach can be easily converted to the former inserted idle-time approach by considering an equivalent $w_k^{f'}$ that equals w_k^f plus the longest overrun and an equivalent $w_k' = w_k^{f'} - w_k^s$.

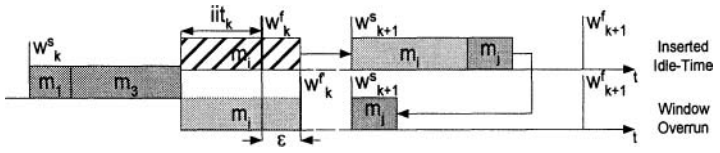


Figure 2. Inserting idle-time (top line) or overrunning the window limit (lower line).

Therefore, whenever idle-time is inserted in window w_k, we have $load(w_k) < w_k$ even when there are more messages ready for transmission. The maximum inserted idle-time depends on the combinations of messages that are scheduled within each window. Calculating its exact value requires building a traffic timeline over an entire macro-cycle. However, it can be upper bounded by the maximum transmission time among all messages or, in some special cases, among a subset (Almeida and Fonseca, 2001).

Finally, we define the network delay nd_i of message m_i as the time span between message release and end of the respective transmission.

3. SYNCHRONOUS RELEASE ANOMALY

The scheduling anomalies that we show in this section refer to the definition of *critical instant.* This is a key concept in schedulability analysis of task or message sets with arbitrary offsets. By definition (Liu and Layland, 1973), the critical instant is such that releasing a message at that instant will result in its largest network delay. When such critical instant can be determined it is sufficient to compute the message network delay for such instant to assess the schedulability of the message set for any phasing condition. Liu and Layland (1973) proved that, for preemptive tasks running on a single non-idling processor, the critical instant happens when a task is released simultaneously with all others (Earliest Deadline First scheduling), or with all higher priority ones (Rate-Monotonic scheduling). This property also holds for Fixed-Priorities scheduling in general, as well as for time partitioned systems, as long as preemption is allowed. However, for the non-preemptive case, as in our model, such property does not hold.

In this paper we will present two anomalies related to the critical instant. The former one is called *direct synchronous release anomaly* and refers to a situation in which the network delay of a message within a given partition increases when the phasing of the remaining messages is non-synchronous. The latter is called *indirect synchronous release anomaly* and refers to a situation in which the partition capacity is variable and dependent on the effective use of the capacity of another partition that has higher priority in using the bus. This is typical in aperiodic phases with respect to periodic ones, which are normally given higher priority. In this case, the maximum network delay of an aperiodic message might not occur when the message is released together with all the periodic messages. These two counter intuitive phenomena are the motivation for this paper and will be described next.

3.1 DIRECT SYNCHRONOUS RELEASE ANOMALY

Proposition 1: When scheduling a message set M (non-preemptively) within a temporal partition $\Pi(\pi, w)$ of type 1, i.e., with fixed capacity, using fixed-priorities or deadlines-based criteria, the critical instant may not coincide with the synchronous release of the messages in M.

Proof: We prove the proposition with an example. Consider a communication system with a partition $\Pi(\pi, w)$ with capacity $w = 5$ time units and period $\pi > 5$ time units. Consider a set $M=\{m_i, i=1..5\}$ of periodic messages, sorted by decreasing priorities or later deadlines, with transmission times $C_i=\{2,2,1,3,2\}$ time units and period $T_i > 3*\pi$. Let us consider the first 3 availability windows $\{w_0, w_1, w_2\}$

i) When the messages are released synchronously ($\Phi_i=0$, $i=1..5$) messages m_1 to m_3 fit in w_0, and messages m_4 and m_5 fit in slot w_1 (Figure 3). Thus, $nd_4= w_1^s+C_4$ and $nd_5= w_1^s+C_4+C_5$.

ii) Consider now that message m_3 has initial offset $\Phi_3=\pi$. After the transmission of message m_2, messages m_4 and m_5 are ready but are not scheduled since they do not fit in w_0. Then, message m_3 becomes ready at w_1^s and is scheduled before m_4 and m_5 because it has higher priority or an earlier deadline. Message m_4 still fits in w_1 but m_5 is pushed forward to w_2 (Figure 3). Thus, $nd_4= w_1^s+C_3+C_4$ and $nd_5= w_2^s+C_5$.

In this scenario, messages m_4 and m_5 experience a longer response time when $\Phi_i=\{0,0,\pi,0,0\}$ than with the synchronous release $\Phi_i=0$, $i=1..5$ thus proving the proposition. ■

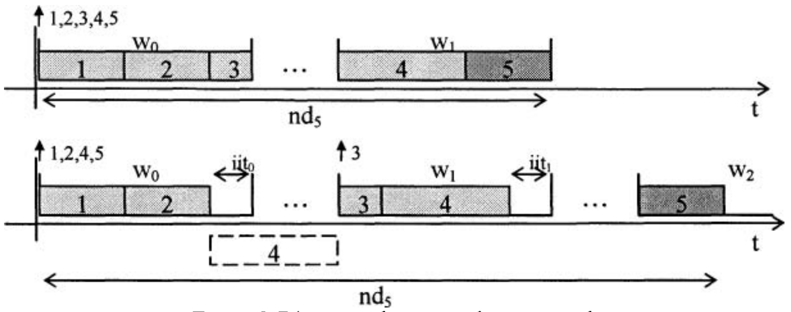


Figure 3. Direct synchronous release anomaly.

The reason for the anomaliy is the idle-time inserted in w_0 and w_1, where the difference w_k-$load(w_k)>0$ is added to the schedule length. This inserted idle-time can vary significantly with just small variations in any C_i. Moreover, it is virtually impossible to compute it exactly for all partition windows w_k unless generating the message schedule for the whole macro-cycle, i.e., for a number of windows equal to $lcm(T_i)$, i=1..N. Finally, different messages may exhibit their worst-case network delays for different phasings not just one particular critical instant.

3.2 INDIRECT SYNCHRONOUS RELEASE ANOMALY

Proposition 2: When scheduling a message set M^L (non-preemptively) within a temporal partition $\Pi^L(\pi, w_{min}, w_{max})$ of type 2, i.e., with variable capacity resulting from reclaiming unused capacity from another partition $\Pi^H(\pi, w^H)$ of type 1 that has higher priority, using fixed-priorities or deadlines-based criteria, the critical instant for the set M^L may not coincide with the synchronous release of the set M^H scheduled within the Π^H.

Proof: Again, we will use an example to prove this proposition. Consider the set $M^H=\{m_i^H, i=1..5\}$, with $C_i^H=\{2,2,3,2,2\}$ and period $T_i^H>3*\pi$, scheduled within a partition $\Pi^H(\pi=6, w^H=5)$. Now consider a partition $\Pi^L(\pi=6, w^L_{min}=1, w^L_{max}=6)$ within which we schedule a message set $M^L=\{m_1^L(C_1^L=2, T_1^L>3*\pi)\}$ with just one message. Notice that w^L stretches and shrinks to reclaim the unsued capacity of Π^H, including any inserted idle-time. Let us consider the 3 first partition periods, with the first 3 windows of each partition $\{w_0^H, w_0^L, w_1^H, w_1^L, w_2^H, w_2^L\}$.

When the messages of M^H are released synchronously ($\Phi_i^H=0$, i=1..5) messages m_1^H and m_2^H fit in w_0^H, messages m_3^H and m_4^H fit in slot w_1^H and message m_5^H fits in w_2^H (Figure 4). Thus, the effective capacity of partition Π^L in its 3 first windows is $w_0^L=2$, $w_1^L=1$ and $w_2^L=4$.

i) Suppose we release message m_1^L also synchronously, i.e., with $\Phi_1^L=0$. Since $C_1^L=w_0^L=2$, it fits in w_0^L and its network delay is given by $nd_1^L=w_0^{L,s}+C_1^L$ (which equals π).

ii) Now suppose we release message m_1^L with $\Phi_1^L=\pi$. Since $C_1^L>w_1^L=1$, it does not fit in w_1^L being pushed to w_2^L. Thus, its network delay is now given by $nd_1^L=w_2^{L,s}+C_1^L-\pi$ (which equals $\pi+C_5^H+C_1^L$).

Since nd_1^L is larger in the second case, i.e., when the message is released in an instant different from the synchronous release of the M^H set in the partition Π^H the proposition is proved. ■

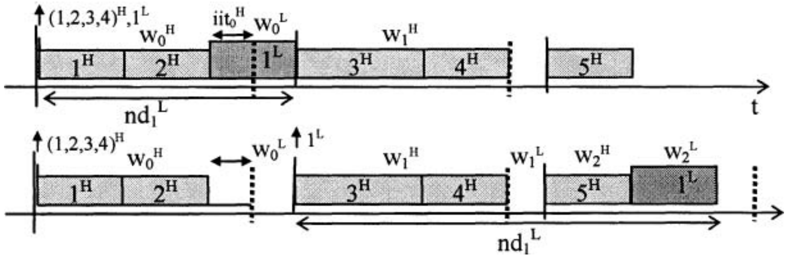


Figure 4. Indirect synchronous release anomaly.

This anomaly is also caused by idle-time inserted within Π^H, whose effect is propagated to Π^L because of the reclaiming mechanism. If M^L included more messages we would obtain a superposition of both anomalies, i.e., the effect of the idle-time inserted directly in Π^L because of the scheduling of set M^L, and the idle-time inserted in Π^H caused by the scheduling of set M^H and propagated to Π^L via the reclaiming mechanism.

Moreover, it is also virtually impossible to determine the exact phasing for a message within Π^L that will cause its worst-case network delay without actually building the schedule of M^H within Π^H for the respective macro-cycle. Finally, there is not one single phasing that causes the worst-case network delay for all messages in M^L.

4. WORST-CASE NETWORK DELAY

In the previous section we have seen that using inserted idle-time, either directly or indirectly, within temporally partitioned networks may lead to situations in which we lose the ability to determine an exact critical instant. This has a drastic consequence in terms of schedulability analysis, and it shows that it is not possible, in the general case, to determine exact worst-case network delays under the referred conditions.

However, the duration of the anomalies can be upper bounded and so can the network delays. In recent years, a few analysis were developed that already overcome these anomalies. On the other hand, the anomalies were not recognized as such and were not formally presented, thus not illustrating their full impact. Other analysis were presented that fail in the cases shown in this paper, because the anomalies were not taken into account.

For example, Almeida *etal.* (2002) present a simple approach for the direct anomaly considering the maximum idle-time being inserted every window and noticing that its magnitude cannot be larger than the longest message. This corresponds to considering one extra message $m_{iit}(C_{iit}=C_{max}, T_{iit}=\pi)$, i.e., occurring every partition window and with a length equal to the transmission time of the longest message in the set. This was proposed for scheduling synchronous messages under EDF within FTT-CAN. They also present another approach that is more efficient with utilization-based schedulability tests for rate-monotonic scheduling. It consists in inflating the transmission times of the messages to $C_i*w/(w-C_{max})$. It is also shown that, with fixed-priorities scheduling, the maximum inserted idle-time can be determined from within a subset of messages. With the adapted C_i and the referred scheduling model, any existing analysis for preemptive fixed-priorities scheduling can be used in this case but always generating sufficient non-necessary schedulability conditions (Almeida and Fonseca, 2001).

For the indirect anomaly, it can be avoided when analysing the worst-case network delay by not reclaiming the capacity left free within the higher priority partition whenever idle-time could have been inserted. Thus, whenever $w_k \leq load(w_k) \leq w_k - C_{max}$ the analysis considers $load(w_k)=w_k$. This has been proposed in (Almeida *etal.*, 2002) to analyse the network delay of asynchronous messages in FTT-CAN. With these approaches, for the direct and indirect anomalies, the worst-case network delay obtained considering the synchronous release will always be longer than the network delays occurring at run-time with any phasing, thus resulting in a safe upper-bound.

Examples of analysis that may fail to provide a safe upper-bound for the worst-case network delay are the timeline analysis for cyclic frameworks (Almeida and Fonseca, 2001), the analysis for PROFIBUS in (Cavalieri *etal.*, 2002) as noted in (Ferreira, 2005), and the analysis for WorldFIP in (Tovar and Vasques, 2000).

5. CONCLUSION

Scheduling messages non-preemptively within temporaly-partitioned networks, e.g. in TDMA, token-passing or multi-phase cyclic buses, will generally lead to the insertion of idle-time at the end of the partitions, either slots or phases. This inserted idle-time may create scheduling anomalies in which the critical instant is not coincident with the synchronous release of the scheduled messages. In this paper we exposed and formalized two anomalies of this kind, one related to scheduling messages within a partition of constant width and another one within a partition of variable width. The paper also showed existing approaches for traffic schedulability analysis that are robust with respect to the anomalies, as well as approaches that may fail because the anomalies were not considered.

REFERENCES

L. Almeida, J.A. Fonseca (2001). "Analysis of a Simple Model for Non-Preemptive Blocking-Free Scheduling". In Proceedings of the 13th Euromicro Conference on Real-Time Systems, Delft, The Netherlands.

L. Almeida, P. Pedreiras, J.A. Fonseca (2002). "The FTT-CAN Protocol: Why and How". In IEEE Transactions on Industrial Electronics, 49(6).

L. Almeida, P. Pedreiras (2004). "Scheduling within temporal partitions: response-time analysis and server design", ACM Int Conf on Embedded Software (EMSOFT 2004), Pisa.

G. Buttazzo (2002). "Scalable Applications for Energy-Aware Processors". 2nd Int. Conf. on Embedded Software (EMSOFT 2002), Grenoble, France, LNCS 2491, Springer-Verlag.

S. Cavalieri, S. Monforte, E. Tovar, F. Vasques (2002). Multi-Master Profibus-DP Modelling and Worst-Case Analysis Based Evaluation. 15th IFAC World Congress, Barcelona, Spain.

Y.-S. Chen, L.-P. Chang, T.-W. Kuo, A.K. Mok (2005). "Real-time task scheduling anomaly: observations and prevention", ACM Symp on Applied Computing, Santa Fe, New Mexico.

R.I. Davis, A. Burns (2005). "Hierarchical Fixed Priority Scheduling", Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS). Miami, USA.

L. Ferreira (2005). "A Multiple Logical Ring Approach to Real-time Wireless-enabled PROFIBUS Networks". PhD Thesis, Universidade do Porto, Portugal.

R. Graham (1976). "Bounds on the performance of scheduling algorithms". In E. G. Coffman, Jr., ed, Computer and Job-Shop Scheduling Theory. John Wiley & Sons, New York.

Giuseppe Lipari, Enrico Bini (2004). "A Methodology for Designing Hierarchical Scheduling Systems". In Journal of Embedded Computing 1(2).

C. L. Liu, J. W. Layland (1973). "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". In Journal of the ACM, 20(1).

A. Mok (2000). "Scalability of Real-Time Applications". Keynote address at the 7th International Conference on Real-Time Computing Systems and Applications.

I. Shin, I. Lee (2003). "Periodic Resource Model for Compositional Real-Time Guarantees". Proceedings of 24th IEEE Real-Time Systems Symposium (RTSS), Cancun, Mexico.

J. A. Stankovic, M. Spuri, M. Di Natale, G. C. Buttazzo (1995). "Implications of Classical Scheduling Results for Read-Time Systems". In Computer, June, 1995, pp. 1625.

E. Tovar, F. Vasques (2000). "Distributed Computing for the Factory-floor: a Real-Time Approach Using WolrdFIP Networks", Computers in Industry, 44(1), Elsevier Science.

PULSED DATA STREAMS

Hermann Kopetz
Institut für Technische Informatik, Technische Universität Wien, A 1040 Wien, Treitlstrasse 3

Abstract: This paper proposes a new communication primitive for distributed embedded control systems: *the pulsed data stream*. A pulsed data stream is a time-triggered cyclic unidirectional data stream that is transmitted for a short duration at a defined phase of every cycle of a periodic control system. Since the duration of a cycle and the phase and duration of the transmission within each cycle are known a priori, the transmission of pulsed data-streams can be scheduled by the network ahead of their activation in order to minimize the end-to-end latency that is provided to an application. This paper introduces the concept of a pulsed data stream, presents a number of practical examples that demonstrate the utility of pulsed data streams in distributed monitoring and control applications and hints at implementation issues of pulsed data streams in local and wide-area networks.

Keywords: embedded system, real-time, distributed system, distributed control.

1. INTRODUCTION

The widespread availability of powerful, low-cost and dependable silicon devices is dramatically impacting the field of distributed control in embedded systems. The integration of MEMS sensing and actuating elements with a micro-controller on a single silicon die makes it possible to build an intelligent sensor/actuator node that performs front-end signal conditioning and calibration directly at the sensor and present the sensor data in a uniform digital format via a standardized communication protocol to a remote processing component that executes the control algorithm. Real-time communication issues have thus entered distributed control applications at the lowest control-loop level. The temporal performance of the

Please use the following format when citing this chapter:

Kopetz, H., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 105–114.

communication protocols within a control loop has thus a direct influence on the quality of control.

Large control systems are organized in a hierarchical manner where control loops are closed at multiple levels. Today it is common practice that different communication protocols are used at the different levels[1]. For example, while at the lowest level field bus protocols, such as CAN[2], are extensively deployed, standard Ethernet is widely used at the higher levels within a given location. In geographically distributed control systems, such as *in power grid control*, standard Internet protocols, such as TCP [3], are deployed. At any level the temporal characteristic of the communication protocol, such as latency and jitter, are determining parameters for the quality of the overall control system.

In this paper a single new communication primitive, the *pulsed data stream*, for the exchange of real-time data in distributed real-time control systems is proposed. *Pulsed data streams* can be used at all levels of the control hierarchy, from low-level local sensor communication up to wide-area distributed control systems, such as the control of the power-grid[4]. *Pulsed data streams* form a uniform mechanism for the exchange of real-time data among the components of a distributed control system.

This paper is organized as follows. In Section two we analyze the communication requirements of distributed control system. Section three introduces the concept of a *pulsed data stream* as a new communication primitive for distributed control systems at all levels of the control hierarchy. Section four presents a number of application scenarios from the field of distributed control. Section five looks at implementation issues and discusses three different scenarios: networks-on-chip, local-area networks and wide-area networks.

2. TIMING AND COMMUNICATION REQUIREMENTS OF CONTROL SYSTEMS.

In this Section we analyze the timing and communication requirements of local and wide area digital control systems. We are looking at three different tasks of a control system: (i) process monitoring, (ii) continuous control, (iii) discrete control.

2.1 PROCESS MONITORING

A *controlled object*, e.g., a car or an industrial plant process, changes its state as a function of time. If we freeze the time, we can describe the current state of the controlled object by recording the values of its state variables at that moment. Possible state variables of a controlled object "car" are the position of the car, the speed of the car, the position of switches on the dash board, and the position of a piston in a cylinder. We are normally not interested in *all* state variables, but only in the *subset* of state variables that is *significant* for our purpose [5]. In a distributed control system, e.g., a nation-wide power grid, care must be taken that the controlled object is observed by all sensors at the same instant of time. A necessary prerequisite for monitoring a large controlled object is thus the availability of a global time of sufficient accuracy at every sensor node. Other than that, monitoring does not require real-time communication protocols that exhibit controlled delay and jitter.

In the Final Report of the US-Canada Task force on the NE American power outage in 2003, the following remarks are contained about synchronization and timing of the collected data [6] p. 162: *A valuable lesson from the August 14 blackout is the importance of having time-synchronized system data recorders. The Task Force's investigators labored over thousands of data items to determine the sequence of events, much like putting together small pieces of a very large puzzle. That process would have been significantly faster and easier if there had been wider use of synchronized data recording devices. . . . Today at a relatively modest cost, all digital fault recorders, digital event recorder and power system disturbance recorders can and should be time-stamped at the point of observation using a Global Positioning System (GPS) synchronizing signal.*

2.2 CONTINUOUS CONTROL

In *continuous control* a continuous process is sampled periodically by a digital system—giving rise to a *hybrid system.* From the viewpoint of the distributed digital control system, the progress of time is partitioned into an (infinite) set of consecutive cycles, where within every cycle the same sequence of periodic activities is performed: waiting for the beginning of the next cycle, sample the inputs at the (often physically dispersed) sensors, send the data to a component that executes the control algorithm, and finally distribute the data to the actuators that act on the physical parameters of the process. The instants, when these activities are started, should be highly predictable [1], p. 19-4: *Periodicity is not mandatory, but often assumed as it leads to simpler algorithms and more stable and secure systems. Most of the algorithms developed with this assumption are very sensitive to period duration variations, jitter in the starting instant. This is especially the case of motor controllers in precision machines. Simultaneous sampling of inputs is also an important stability factor.*

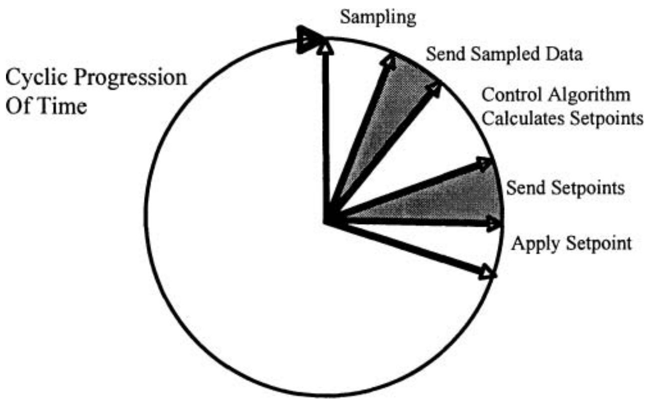

Sampling
Send Sampled Data
Control Algorithm Calculates Setpoints
Send Setpoints
Apply Setpoint
Cyclic Progression Of Time

Fig. 1: Cyclic Model of Time and phases of a control system.

Fig. 1 suggest a *cyclic*, not a *linear* model time. Cyclic systems are very common in control systems and in biological systems [7]. The progress in cyclic systems is inherently *time-triggered*. The progression of time (and activities) within each cycle can be depicted by the concept of *phase*, ranging from 0 degree to 360 degree for the full cycle. The trigger for an activity is thus the instant when the time progresses to a defined phase start point. Fig. 1 depicts five phase start points: the start of sampling, start sending the sampled data, start the execution of the control algorithm, start sending the set-points to the actuators and finally apply the set-points to the controlled object. The interval between the *start of sampling* and the *application of the set-points* to the controlled object should be minimized in order to reduce the dead-time of the control loop and thus increase the quality of control. Within every cycle of activities there are two communication phases: the *sending of input data* to the control algorithm and the *sending of output data* to the actuators as depicted by the shaded areas in Fig. 1. Reducing the duration of these communication phases reduces the dead-time in the control loop and thus increases the achievable quality of control. Since many of the widely used communication protocols, such as CAN, Ethernet and TCP do not minimize delay and jitter, the quality of control in these systems is unnecessarily degraded.

2.3 DISCRETE CONTROL

Discrete control systems are basically event-driven system. A significant state change causes control actions by the computer system. In many of the event-driven systems, the significant state changes are relayed to the computer system by the interrupt mechanism. However, every interrupt driven system is in danger of overload in case the minimum time between interrupts is not bounded. In order to exclude the possibility of overload by design, a number of discrete control systems sample the significant states cyclically and thus convert the discrete control system to a *quasi continuous* system. In such a design care must be taken that the sampling interval is smaller than the smallest duration between correct events that have to be sensed. Such a solution rejects events at the source that are closer together than the specified minimum interval between events and thus protects the control system from overload caused by sensor failure.

3. PULSED DATA STREAMS

Before introducing the pulsed data stream concept, we would like to elaborate on a fundamental conflict in the design of real-time communication protocols.

3.1 FUNDAMENTAL CONFLICTS IN REAL-TIME PROTOCOL DESIGN

Before designing a protocol for the transfer of real-time data between the components of a control system, one has to make a decision between the following two alternatives: (i) the components are assumed to be *competing* with each other for communication bandwidth or (ii) the components are assumed to be *cooperating*. The first alternative is the only alternative for *open system*, where the set of communicating components is not known *a priori*.

Independently of the characteristics of the detailed network protocol, there is always the possibility that in alternative (i)--*competition of the components*--two or even *n* components try to start sending messages to the same receiver at the same instant. Since only one message can be received at a sequential receiving port at an instant, there is the possibility that a conflict among the *n* components will occur. The two possible ways to resolve this conflict are: (a) either one message is sent immediately and the other *n-1* messages *are **stored** in the network* and sent to the receiver sometimes later when the link to the receiver is free again (this is the solution of *switched Ethernet*) or (b) the network *exercises **dynamic** back-pressure flow control* to the *n-1* components that did not succeed in sending their message (this is the solution of *CAN* or *bus-based Ethernet*). Both alternatives are *unsatisfactory* from the point of view of the real-time properties of the protocol, such as *delay* and *jitter*, because the *worst-case transmission time* for a message depends on the momentary traffic generated by *all* components and is much larger than the *minimal transmission time,* when no other component is active. Furthermore, the jitter becomes a *global property* that depends on the behavior of all components, violating some principles of composability [8]. Jitter-sensitive applications, such as control applications and high-quality multi-media applications, do have problems with a communication infra-structure that introduces significant jitter [9].

In alternative (ii), a *closed-world system,* the number of clients is known *a priori*. The clients *cooperate* with each other or with a central scheduler in order to establish a coordinated schedule, such that the communication system is in the position to meet the requests of all clients within specified temporal bounds. Temporal guarantees for *all* time-critical messages can only be given in a *closed-world system.*

3.2 THE PULSED DATA STREAM CONCEPT

The *pulsed data stream* is a new communication concept for closed-world distributed real-time systems. It assumes that a set of *a priori* known components that are aware of a global notion of time agree on a cooperative cyclic communication schedule, such that all transmission requests of all partners can be satisfied without conflict.

A *pulsed data stream* is a cyclic data stream that transports data uni-directionally in *pulses* from *one* sender to *n a priori* identified receivers at a specified phase of the cycle for a specified duration. We call the duration between the *start of transmission* and the *termination of transmission* at the sender, i.e., the duration of the *pulse* at the sender, the *active phase at the sender.* We call the duration between the *start of reception* and the *termination of reception* at a receiver the *active phase at the receiver.* Pulsed data streams are *bursty,* but the bursts are not sporadic, but regular and known in advance. This common knowledge is instrumental for the scheduling of the pulsed data stream by the network.

The *pulse* at the sender is characterized by the following parameters

- the set of receivers
- the duration of the cycle
- the start-instant of the *pulse*, expressed in the metric of global time
- the termination-instant of the pulse
- the reliability class
- security class

We call this data set the *pulse characterization.* The amount of data that can be transmitted during a pulse at the sender depends on the transmission bandwidth of the sender to the network.

Before the start of a pulsed data stream, the network must be informed about the *pulse characterization* in order that the required resources can be reserved. Depending on the characteristics of the network (size, bandwidth, internal protocol structure) and the current commitments to other pulsed data streams, the network will respond with the performance parameters that can be guaranteed for the transport of this newly requested pulsed data stream: the *transmission delay*, i.e., the interval between the *start of the active phase* at the sender and the start of the *active phase at a receiver* and the duration of the delivery phase. The *transmission delay* can be very small in an on-chip-network (in the order of nanoseconds) and substantial in a wide area network (in the range of hundreds of milliseconds, or even seconds). The *transmission delay* depends on the propagation delay of the channels and the scheduling strategies within the network.

In many cases the data transmitted within a pulse is *state data*, i.e., it informs about the current value of state variables. At the receiver, a new version of the state data replaces the previous version (update in place). State data is not consumed by a receiver (similar to a variable in computer memory). There are no queues needed to handle state data. In many control applications, the loss of a single pulse is not critical, because the receiver will automatically take the state at the previous cycle as a replacement for the lost pulse. The retransmission of a corrupted pulse by the network, which will impact the timing of the delivery, is thus not a standard option in pulsed data streams. The fact that corrupted pulse data does not have to be retransmitted has deep implications for the design of the network protocols.

Fig. 1 depicts two pulsed data streams within a typical control loop. Since a properly designed local area real-time communication network has a very small *transmission delay*, it is not depicted in Fig. 1. The most important property of the pulsed data stream concept concerns the *a priori* knowledge about the phase of the recurring send instants. This a *priori* knowledge must be used by the communication system to plan for the provision of the predictable transport service for the pulsed data streams within a network.

4. APPLICATION OF PULSED DATA STREAMS

In this Section we sketch a number of examples for the use of pulsed data streams in distributed control systems.

4.1 REAL-TIME MONITORING

Many real-time monitoring applications require the periodic transmission of sensor values to a central process monitoring facility. Two parameters are critical for the proper operation of this application: the synchronized sampling of the data and a small delay between the sense instant and the delivery instant of the sensor data at the monitoring facility. The *a priori knowledge* about the parameters of the pulsed data streams enables the communication system to plan and reserve the required transmission capacity for the pulsed data streams.

4.2 CLOSED LOOP CONTROL

A typical local control loop has the cycle structure of Figure 1. Two pulsed data streams, the *pulsed data stream* from the sensors to the component that executes the control algorithm and the *pulsed data stream* from the control-algorithm component to the actuators must be supported by the communication system.

In a wide area control scenario, such as power-grid control, a number of different networks are involved in the transport of the real-time data from the sensor components to the control center. The a-priori knowledge of the parameters of the pulsed data streams makes it possible to align the transmission phases of the different networks *a priori*, such that the overall latency of the real-time data transmission can be optimized.

4.3 TRIPLE MODULAR REDUNDANCY

In a TMR (triple modular redundant) system three components have to exchange *pulsed data streams* in order to vote on the results of the computations and the inner state of the components. These *pulsed data streams* can be coordinated *a priori* to minimize the latency and jitter that is introduced by the transport service.

4.4 MULTI-MEDIA SYSTEMS

In some multi-media applications a complete frame has to be transmitted from one component to another component before the frame processing can be initiated. This puts a high bursty load on the communication system that can be mitigated if the regularity of the *a priori* known parameters of the pulsed data stream is taken into account in scheduling the communication.

5. IMPLEMENTATION HINTS

The implementation of *pulsed data streams* requires the establishment of a system-wide global time base and a scheduler that coordinates the pulsed data streams requested by the involved components. The establishment of a global time of known precision is a well-understood problem that has been implemented in a number commercial applications[10-13]. Communication systems that are controlled by time-triggered communication protocols provide an ideal environment for the implementation of pulsed data streams. The scheduling problems is substantially simplified if the cycle durations are harmonic, e.g., positive powers of a smallest units. External synchronization can be eased if one of the cycle durations is exactly the duration of the physical second.

5.1 ON-CHIP NETWORKS

The communication network within a deeply embedded multi-computer SoC, such as the Cell chip [14], can be implemented as a time-triggered network. Such an on-chip time-triggered network can be expected to support a very high bandwidth (the bandwidth of the interconnect on the cell chip is more than 100 Gbits/second). The implementation of pulsed data streams is trivial if such a time-triggered network is available as a network-on-chip (NoC). In order to be able to synchronize the pulsed data streams within a chip with off-chip networks, an external synchronization of the chip-internal time base must be provided.

5.2 LOCAL AREA NETWORKS

TT-(time-triggered) Ethernet [13] provides an ideal environment for the implementation of pulsed-data stream in a local area context. TT Ethernet distinguishes between two traffic categories, i.e. open-world ET (event-triggered) Ethernet traffic and the closed-world time-triggered traffic. ET traffic is handled in full compliance with the Ethernet standard [15], while TT traffic is coordinated by a scheduler and transported with a-priori known constant latency and minimal jitter. For a detailed description of TT Ethernet refer to [13].

5.3 WIDE AREA NETWORKS

The implementation of *pulsed data streams* in wide-area networks, such as the Internet is a research challenge, since the present Internet is driven by flexible dynamic routing algorithms that are optimized for average performance, but do not support real-time guarantees that are required in real-time control systems.

6. CONCLUSION

Pulsed data-streams are a new communication primitive for the exchange of real-time data among the components of a distributed control system. A pulsed data stream is a powerful abstraction that is an exact fit with the requirements of many real-time control applications.

7. ACKNOWLEDGEMENTS

This work has been supported in part by the EU integrated research project DECOS and by the NoE Artist II.

REFERENCES

[1] Decotignie, J., D., *Which Network for Which Application*, in *The Industrial Communication Technology Handbook*, R. Zuwarski, Editor. 2005, Taylor and Francis: Boca Raton. p. 19/1-19/15.

[2] CAN, *Controller Area Network CAN, an In-Vehicle Serial Communication Protocol*, in *SAE Handbook 1992*. 1990, SAE Press. p. 20.341-20.355.

[3] Furrer, F., J, *Industrieautomation mit Ethernet TCP/IP and Web -Technologie*. 2003, Heidelberg: Hüthig Verlag.

[4] Birman, K.P., et al., *Overcoming Communication Challenges in Software for Monitoring and Controlling Power Systems*. Proc. of the IEEE, 2005. **93**(5): p. 1028-1041.

[5] Kopetz, H., *Real-Time Systems, Design Principles for Distributed Embedded Applications; ISBN: 0-7923-9894-7, Seventh printing 2003*. 1997, Boston: Kluwer Academic Publishers.

[6] Force, U.-C.T., *Final Report on the August 14, 2003 Blackout in the United States and Canada*. 2004.

[7] Winfree, A.T., *The Geometry of Biological Time*. 2001: Springer Verlag New York.

[8] Kopetz, H. and N. Suri. *Compositional Design of Real-Time System: A Conceptual Basis for the Specification of Linking Interfaces*. in *ISORC 2003--The 6th International Symposium on Object Oriented Real-Time Computing*. 2003. Hakodate, Japan: IEEE Press.

[9] Reinder, J., et al. *Dynamic Behaviour of Consumer Multimedia Terminals: System Aspects*. in *IEEE International Conference on Multimedia*. 2001: IEEE Press.

[10] Kopetz, H. and W. Ochsenreiter, *Clock Synchronisation in Distributed Real-Time Systems*. IEEE Trans. Computers, 1987. **36**(8): p. 933-940.

[11] Kopetz, H., *Specification of the TTP/C Protocol*. 1999, TTTech, A 1040 Wien, Schönbrunnerstraße 13.

[12] IEEE, *1588 Standard for a Precision Clock Synchronization Protocol for Network Measurement and Control Systems*. 2002.

[13] Kopetz, H., et al. *The Design of TT Ethernet*. in *ISORC 2005*. 2005. Seattle: IEEE Press.

[14] Wang, D.T., Proc. of the *ISSCC 2005: The CELL Microprocessor*. 2005.

[15] Ethernet, *IEEE Ethernet Standard 802.3 at URL: http://standards.ieee.org*. 2002.

FROM TIME-TRIGGERED TO TIME-DETERMINISTIC REAL-TIME SYSTEMS

Peter Puschner and Raimund Kirner
Vienna University of Technology, A-1040 Vienna, Austria
{peter, raimund}@vmars.tuwien.ac.at

Abstract With the increased use of powerful, performance-optimized hardware components in embedded systems, timing prediction is getting more and more complex. Thus while the execution speed of software is generally increasing, it is getting more and more difficult (if not infeasible) to perform an accurate and safe timing analysis of software that runs on those high-end embedded computer systems.

This paper presents a very rigid software execution model for building distributed hard real-time subsystems that are time predictable. The software model is based on the time-triggered communication model. It uses a purely time-triggered input-output interface and relies on single-path code (code that is free from input-data dependent control flow) in both the operating system and application software. Tasks are only preempted at pre-planned task preemption points and a simple clock synchronization keeps the operations of the hard real-time subsystem in synchrony with the real-time environment.

The proposed execution model yields software that is time-predictable by construction. Verifying temporal correctness and tracing the timing behavior of this software is trivial.

Keywords: Real-time systems, time-triggered architecture, determinism, time predictability.

1. INTRODUCTION

The computer architectures used in embedded systems are becoming increasingly complex. Modern microcontrollers for embedded systems are built around powerful superscalar CPU cores that use a number speedup features including instruction pipelines, caches and branch prediction, whose actual impact on the speedup of a particular block of code strongly depends on the processor's execution history, i.e., the stream of instructions the processor executed before the block. As the number of different execution histories at each point of a program can be enormous, an analysis of all possible behaviors of a piece of software running on such a high-end computer is generally extremely difficult. Consequently, worst-case execution time (WCET) analysis (i.e., assessing the timing of single tasks assuming non-preempted execution) is dif-

Please use the following format when citing this chapter:

Puschner, P., Kirner, R., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 115–124.

ficult for such systems, and analyzing the timing of the whole system, i.e., including preemptions and their effects on overall timing, requires unmanageably high efforts.

While the mentioned complexity problem might seem intuitive for highly dynamic systems with event-triggered task activation and scheduling, it even applies to very simple systems, e.g., time-triggered systems that use a very simple table-driven task activation. It thus seems to become more and more difficult to argue about timing guarantees and the system safety of embedded systems. To avoid the risk of missing a critical deadline, the use of very defensive estimates about resource needs and an over-dimensioned resource planning are becoming a necessity, unless simpler execution models are found.

In the light of this unsatisfactory situation we started to search for a real-time systems architecture which would have time-predictability as its number one property, i.e., temporal correctness (the absence of timing faults) should be easy to validate. All other properties, including performance in the classical sense should come second. This paper presents a software/hardware architecture for safety-critical hard real-time systems that came out of this work.

The only interface of the proposed architecture is a time-triggered state message interface that blocks all asynchronous external control signals that would otherwise disrupt the deterministic timing of the subsystem (Section 2). The software of the proposed architecture builds on a simple task model and table-driven static scheduling (Section 3.1) as well as on a deterministic code execution scheme (single-path code) in applications and in the operating-system code (Section 4). The architecture further uses deterministic task preemption, i.e., the number of instructions executed between each pair of preemptions points is planned offline and therefore exactly known (Section 5). A simple clock synchronization keeps the operations of the hard real-time subsystem synchronized with the clock of the environment (Section 6).

2. THE SAFETY-CRITICAL SUBSYSTEM INTERFACE

In this section we describe the interface of the proposed architecture. As a prerequisite for building a time-predictable (sub)system, the interface of this system has to be predictable as well.

The following description uses the model and terminology of the DECOS integrated architecture (6) for which our work was originally conceived. This does, however, not mean that our work is only useful in the context of DECOS. On the contrary, the architecture model can be adopted to any architecture that provides a time-triggered state message interface.

2.1 THE CONNECTOR UNIT

A DECOS component consists of two separated subsystems, the safety-critical subsystem for executing all safety-critical tasks and the non safety-critical subsystem for performing all other, non-critical services. Both types of subsystems are connected to the rest of the distributed computer system via so-called connector units. The connector units realize the architectural services of the distributed architecture, comprising the predictable transportation of messages, the fault-tolerant clock synchronization, fault isolation, and the consistent diagnosis of node failures (6).

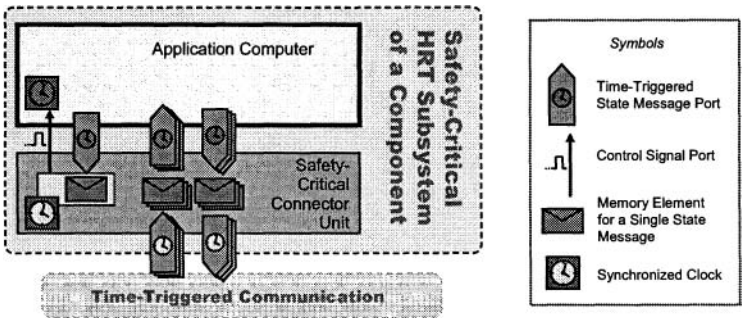

Application Computer
Safety-Critical HRT Subsystem of a Component
Safety-Critical Connector Unit
Time-Triggered Communication
Symbols
Time-Triggered State Message Port
Control Signal Port
Memory Element for a Single State Message
Synchronized Clock

Figure 1. Interfacing between the Safety-Critical Hard Real-Time Subsystem of a Component and the Time-Triggered Communication Channel.

Within this paper our focus is on the safety-critical subsystem of a component (see Figure 1). This is where time predictability is needed. The application computer of this subsystem communicates with its environment solely via the safety-critical connector unit. The connector unit provides the following services in support of the time-predictable software architecture of the application computer.

- The connector unit implements a temporal firewall interface (5) for all data elements exchanged between the application computer and the communication subsystem. The read and write operations of the communication subsystem access the memory elements of the temporal firewalls only at predefined times, according to the a-priory known time-triggered communication schedule (in Figure 1 envelopes represent the memory elements and the arrows marked with light clocks show the accesses of the communication subsystem to the firewalls).

- The time windows during which the communication system accesses the memory elements of the connector unit are known for each of temporal firewall.

- The communication system provides a time-signal service to the application computer. A dedicated memory element in the connector unit can be written to set the timer (Figure 1, left). When the global system time reaches the timer value, the connector unit sends an interrupt signal over the signal port to the application processor.

3. A TIME-PREDICTABLE APPLICATION COMPUTER

The timing of the actions performed by a computer system depends on both, the software running on the computer and the properties of the hardware executing the software (8). We therefore list the hardware and software features that in combination allow us to make an application computer time-predictable.

3.1 HARDWARE ARCHITECTURE

A central idea of our approach is to obtain time predictability by using a software architecture that has an invariable control flow (see below). As a consequence of using this restrictive software model, we can allow for the use of hardware features that are otherwise considered as being "unpredictable" (e.g., instruction caches) and yet build systems whose timing is invariable. So the idea is to keep hardware restrictions and modifications within limits (e.g., we restrict caches to direct-mapped caches but do not demand special hardware modifications as, for example, needed for the SMART cache (2)). To support our execution model, the following hardware properties have to be fulfilled:

- The execution times of instructions do not depend on the values of the operands.

- The CPU supports a conditional move instruction or a set of predicated instructions that have invariable execution times.

- Instruction caches are either direct mapped or set-associative with LRU replacement strategy.

- Memory access times for data are invariable for all data items. (In our view, this is the strongest limitation at the moment. We will try to relax this in future work).

- CPU has a counter that counts the number of instructions executed. The counter can be reset and used to generate an interrupt when a given number of instructions has been completed.

3.2 THE SOFTWARE ARCHITECTURE

To construct a time-predictable computer system while being not more restrictive about the hardware than explained above, we need to be very strict about the software structure. In fact, the proposed software architecture does not allow for any decisions in the control flow whose outcome has not already been determined before the start of the system. This property is true for both the application tasks and the operating system. Even task preemptions are implemented in a way that does not allow for any timing variation between different task invocations.

Task Model. The structure of all tasks follows the simple-task model found in (4). Tasks never have to wait for the completion of an input/output operation and do never block. There are no statements for explicit input/output or synchronization within a task. It is assumed that the static schedule of application tasks and kernel routines ensures that all inputs for a task are available when the task starts and that outputs are ready in the output variables when the task completes. The actual data transfers for input and output are under control of the operating system and are scheduled before respectively after the task execution.

An important and unique property of our task model is that all tasks have only a single possible execution path. By translating the code of all real-time tasks into single-path code we ensure that all tasks follow the only possible, pre-determined control flow during execution and have invariable timing. For more details about the single-path translation see Section 4.

Operating System Structure. If not properly designed, the activities of the operating system can create a lot of indeterminism in the timing of a computer system. We have therefore been very restrictive in the design of the operating system and its mechanisms.

Predictability in the code execution of the operating system is achieved by two mechanisms. First, single-path coding is used wherever possible. Second, all data that are relevant for run-time decisions of the operating system are computed at compile time. These data include the pre-determined times for I/O, task communication, task activation, and task switching. They are stored in static decision tables that the operating system interprets at runtime.

Task communcation and I/O is implemented by simple read and write operations to specific memory locations. As these memory accesses are prescheduled together with the application tasks, no synchronization and no waiting is necessary at run time.

The two greatest challenges in building a fully predictable operating system were in maintaining time-predictability in case of task preemptions and keeping the activities of the application computer in synchrony with its environment (the rest of the system).

- To maintain the deterministic timing in the presence of preemptions it was necessary to introduce a mechanism that allows for a precise preemption when a given number of instructions have finished execution, i.e., planning preemptions at specific times of the CPU clock turned out to be insufficient (see Section 5).
- The programmable time interrupt provided by the communication system is used to synchronize the operation of the application computer with the global time base (see Section 6).

3.3 TOOL SUPPORT

The software structure of our architecture is very specialized. Code generation for an application therefore needs to be supported by a number of tools:

- To generate single-path code, either a special compiler or a code conversion tool that converts branching code into single-path code is needed.
- A tool for worst-case execution-time analysis (either a static analyzer or a measurement tool) returns the execution times of the tasks and the operating system routines.
- An off-line scheduler generates the tables that control all operations of the application computer. The scheduler has to resolve all precedence and mutual exclusion constraints between task pairs as well as tasks and the communication system. It further has to plan all preemptions, thereby taking into account the effects of the preemptions on the system timing.

4. DETERMINISTIC SINGLE-PATH TASK EXECUTION

As all branches in the control flow of a task may potentially cause variable timing, we translate the code of all tasks into so-called single-path code (9). The code resulting from the single-path translation has only a single execution trace, hence the name single-path translation.

The strategy of the single-path translation is to remove input-data dependencies in the control flow. To achieve this, the single-path translation replaces all input-data dependent branching operations in the code by predicated code. It serializes the input-dependent alternatives of the code and uses predicates (instead of branches) and, if necessary, speculative execution to select the right code to be executed at runtime.

For pieces of code with an if-then-else semantics, a similar transformation, called if-conversion, has been used before to avoid pipeline stalls in processors with deep pipelines (1). In addition to code with if-then-else semantics the single-path translation transforms loops with input-data dependent control conditions. This transformation yields loops with constant iteration counts, again with a single execution path (7).

As a prerequisite for the single-path translation of a piece of code, the upper bounds for the number of iterations of all loops have to be available. These numbers can either be computed by a semantic analysis of the code or can be provided by the programmer in the form of annotations, in case an automated analysis is not possible or available.

5. PREDICTABLE TASK PREEMPTION

The idea of predictable task preemption is to preempt each task that needs to be preempted at the same points in time in each execution cycle of the static schedule. By doing so, the overall timing of all repetitive executions of the cyclic schedule would also be invariable.

Our original plan was to implement the predictable task preemption by using the CPU clock for task preemptions, i.e., preempt tasks always when the CPU clock assumed one of the values given in the preemption-time tables of the operating system. It turned out, however, that on hardware with instruction cache this simple preemption strategy does not guarantee temporal predictability. It may lead to oscillating task execution times, see (3).

Figure 2 shows the execution of two tasks, T_1 and T_2. T_1 preempts T_2 at the pre-scheduled time marked by the dashed line on the left. T_2 resumes after T_1 has completed its execution. Let us assume that T_1 and T_2 execute instructions that map to the same cache line of a direct mapped cache (T_2 executes these instructions twice, e.g., in a loop). The execution of these conflicting instructions is marked by the dark boxes; upon a cache miss, the execution time of the instruction increases, which is shown by the striped boxes.

Let us assume the very first activation of our schedule leads to the execution shown in Scenario A. The first access of T_2 to the conflicting cache line leads to a cache miss, and so do the other accesses by T_1 respectively T_2 (The latter misses are due to the order in which the tasks access memory).

When the schedule is repeated, T_2 has a cache hit on the first memory access. So T_2 makes faster progress and the second access to the conflicting address occurs before T_2 is preempted, thus resulting in a hit, too. T_1 then executes with a cache miss, and as T_2 has already completed its two critical memory accesses, the instruction of T_1 remains in cache (see Scenario B). On the next execution of the schedule, T_2 has a cache miss when accessing the conflicting

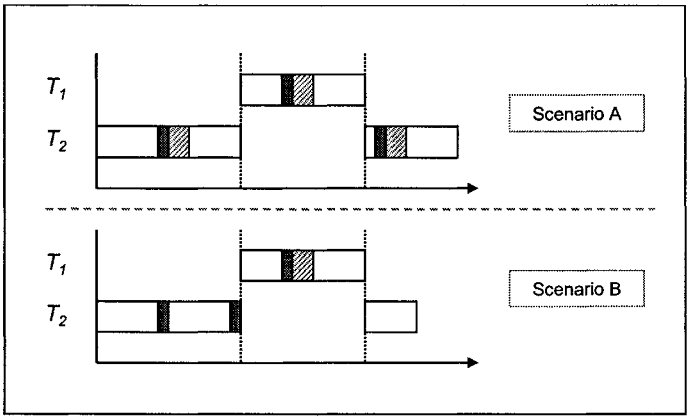


Figure 2. Task preemption by clock interrupt.

address, and so Scenario A is repeated. Following Scenario A, Scenario B happens again, and so on. The timing does not stabilize.

We found that preempting tasks based on the number of instructions executed yields the desired time-predictable behavior. So instead of using a clock we count the number of instructions completed. Preemptions happen when the value of this counter matches an entry in the scheduling table.

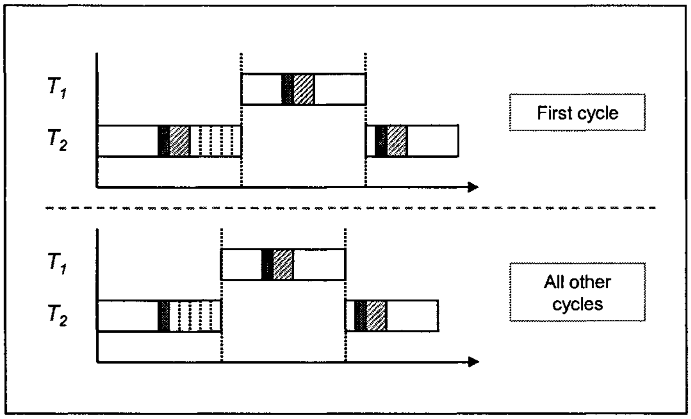


Figure 3. Task preemption by instruction counter.

Figure 3 shows the schedule for our example, using an instruction counter. Still, the timing of the second execution of the schedule differs from the first, in which we have an initial cache miss. From the second execution on, however, the execution always starts from the same cache state and has a constant execution time.

6. MAINTAINING SYNCHRONY WITH THE GLOBAL CLOCK

To keep in phase with the rest of the system, the execution of all actions of the application computer has to be synchronized to the global time reference that is provided by the time-triggered communication system. Deviations of the local clock from the global time are corrected. This clock synchronization uses the clock signal from the communication subsystem. Whenever the programmed timer expires the clock of the application computer is reset and a new round of the execution cycle is started.

In order to maintain the time predictability of the software, the clock synchronization must not interfere with the software execution on the application computer (i.e., the clock signal, that it is not in synchrony with the CPU clock, must not preempt any software running). As a consequence, the application computer has to be idle when a clock signal arrives. This is achieved by using schedules that consist of alternative intervals of task activity and inactivity, where the latter are used as synchronization windows. Clock synchronization interrupts have to be configured such that the clock signals arrive inside the synchronization windows even in case of the worst-case deviation of the local time from the global time.

7. SUMMARY AND CONCLUSION

In this paper we described a software architecture for safety critical hard real-time systems. This software architecture relies on time-triggered communication and uses the static task-activation scheme of a cyclic executive. Further, the operating system design and the single-path translation of code, together with a task preemption mechanism that triggers preemptions based on the number of instructions executed and the simple master clock synchronization make it possible to build fully time-deterministic computer systems on powerful, state-of-the-art hardware. These computer systems are easy to analyze for their timing and their timing properties and correctness can easily be traced. So they can safely be used in time-critical systems for which temporally correct behavior has to be guaranteed.

ACKNOWLEDGMENTS

This work has been supported in part by the European IST project ARTIST2 under project No. IST-004527 and the European IST project DECOS under project No. IST-511764.

REFERENCES

[1] Allen, J., Kennedy, K., Porterfield, C., and Warren, J. (1983). Conversion of Control Dependence to Data Dependence. In *Proc. 10th ACM Symposium on Principles of Programming Languages*, pages 177–189.

[2] Kirk, D. B. (1989). Smart (strategic memory allocation for real-time) cache design. In *Proc. 10th Real-Time Systems Symposium*, pages 229–237, Santa Monica, CA, USA.

[3] Kirner, Raimund and Puschner, Peter (2006). Time-Predictable Task Preemption in Real-Time Systems with Instruction Cache. Research Report 27/2006, Technische Universitat Wien, Institut fur Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria.

[4] Kopetz, H. (1997). *Real-Time Systems*. Kluwer Academic Publishers.

[5] Kopetz, Hermann and Nossal, Roman (1997). Temporal Firewalls in Large Distributed Real-Time Systems. In *Proc. 6th IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 310–315.

[6] Obermaisser, Roman, Peti, Philipp, and Kopetz, Hermann (2005). Virtual Networks in an Integrated Time-Triggered Architecture. In *Proc. 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 241–253.

[7] Puschner, Peter (2002). Transforming execution-time boundable code into temporally predictable code. In Kleinjohann, Bernd, Kim, K.H. (Kane), Kleinjohann, Lisa, and Rettberg, Achim, editors, *Design and Analysis of Distributed Embedded Systems*, pages 163–172. Kluwer Academic Publishers. IFIP 17th World Computer Congress - TC10 Stream on Distributed and Parallel Embedded Systems (DIPES 2002).

[8] Puschner, Peter and Burns, Alan (2000). A review of worst-case execution-time analysis. *Journal of Real-Time Systems*, 18(2/3):115–128.

[9] Puschner, Peter and Burns, Alan (2002). Writing temporally predictable code. In *Proc. 7th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 85–91.

LAZY SCHEDULING FOR ENERGY HARVESTING SENSOR NODES

C. Moser[1], D. Brunelli[2], L. Thiele[1], L. Benini[2]

[1] *Computer Engineering and Networks Laboratory*
Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

[2] *Department of Electronics, Computer Science and Systems*
University of Bologna, Italy

Abstract The paper studies the case of a sensor node which is operating with the power generated by an environmental source. We present our model of an energy driven scheduling scenario that is characterized by the capacity of the node's energy storage, the deadlines and the power dissipation of the tasks to be performed. Since the execution of these tasks requires a certain amount of energy as well as time, we show that the complexity of finding useful scheduling strategies is significantly increased compared to conventional real-time scheduling. We state online scheduling algorithms that jointly account for constraints arising from both the energy and time domain. In order to demonstrate the benefits of our algorithms, we compare them by means of simulation with the classical Earliest Deadline First Algorithm.

1. INTRODUCTION

Wireless sensor networks have been the subject of intensive research over the past several years. As for many other battery-operated embedded systems, a sensor's operating time is a crucial design parameter. As electronic systems continue to shrink, however, less energy is storable on-board. Research continues to develop higher energy-density batteries and supercapacitors, but the amount of energy available still severely limits the system's lifespan. Recently, energy harvesting has emerged as viable option to power sensor nodes: If nodes are equipped with energy transducers like e.g. solar cells, the generated energy may increase the autonomy of the nodes significantly.

In [6], technologies have been discussed how a sensor node may extract energy from its physical environment. Moreover, several prototypes (e.g. [2, 3]) have been presented which demonstrate both feasibility and usefulness of sensors nodes which are powered by solar or vibrational energy.

Please use the following format when citing this chapter:

Moser, C., Brunelli, D., Thiele, L., Benini, L., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 125–134.

The authors of [4] propose algorithms for tuning a node's duty cycle dependent on the parameters of the energy source. Nodes switch between active and sleep mode and try to achieve perpetual operation. Other approaches addressed offline scheduling with regenerative energy by means of Dynamic Voltage Scaling (DVS) [1, 7]. In contrast to this work, we present online algorithms to dynamically schedule arriving tasks and thereby, we are not restricted to a certain technique like Dynamic Voltage Scaling.

In [5], Lazy Scheduling Algorithms (LSA) have been presented for the first time. The latter work primarily focuses on proving the optimality of LSA and derives schedulability conditions from that proof. This paper, on the other hand, presents a detailed description of the algorithms as well as extensive simulative studies revealing the benefits of this new scheduling discipline.

The Earliest Deadline First (EDF) algorithm has been proven to be optimum with respect to the schedulability of a given taskset in traditional time-driven scheduling. The following example shows why greedy scheduling algorithms (like EDF) are not necessary optimal in the context of this paper.

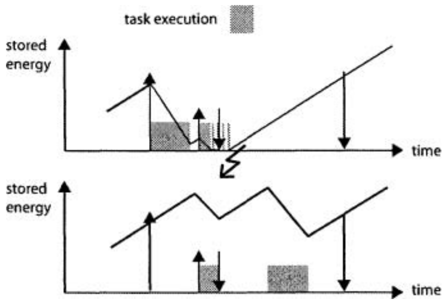


Figure 1. Greedy vs. Lazy Scheduling.

Imagine a sensor node with an energy harvesting unit that replenishes a battery with constant power. Now, this node has to perform an arriving task that has to be finished until a given deadline. In Figure 1, the arrival time and deadline of this task are depicted by "long" -up and down- arrows respectively. Meanwhile, a less energy-intensive task has to be executed within a short time interval that is again given by an arrival time and a deadline (indicated by the "short" arrows). As depicted in the top diagram, the EDF scheduler violates the deadline of the second task since it uses greedily the stored energy to drive the first, energy-intensive task. When the energy is required to execute the second task, the battery level is not sufficient to meet the deadline. In this example, however, a scheduling strategy that hesitates to spend energy meets both deadlines. The bottom plot illustrates how the *Lazy Scheduling* paradigm described in this paper outperforms a naive, greedy approach like EDF in the described situation.

2. PROBLEM STATEMENT

Let us consider a sensor node as depicted in Fig. 2. In the following, the single components of this node will be explained in detail.

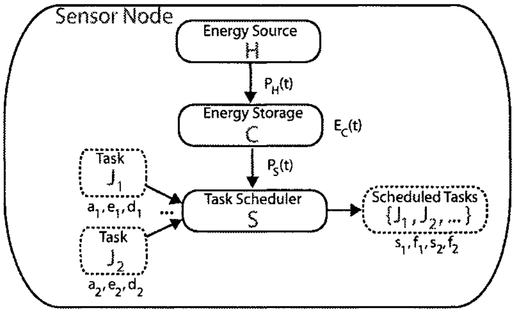


Figure 2. Scheduling Scenario.

2.1 ENERGY SOURCE

We denote $P_H(t)$ the charging power that is actually fed into the energy storage and hence incorporates all losses due to power conversion. Next, the corresponding energy E_H scavenged in the time interval $[t_1, t_2]$ is given by the integral $E_H(t_1, t_2) = \int_{t_1}^{t_2} P_H(t)\mathrm{d}t$.

2.2 ENERGY STORAGE

We assume an ideal energy storage (e.g. a battery) that may be charged up to its capacity C , i.e., $E_C(t) \leq C$. According to the scheduling policy of the sensor node, power $P_S(t)$ and the respective energy $E_S(t_1, t_2)$ is drained from the storage to execute tasks. In particular, if the node decides to assign power $P_i(t)$ to the execution of task J_i during the interval $[t_1, t_2]$, we denote the corresponding energy $E_i(t_1, t_2)$. If no tasks are executed and the storage is consecutively replenished by the energy source, an energy overflow occurs.

2.3 TASK SCHEDULING

As illustrated in Fig. 2, we use the notion of a task scheduler that assigns energy E_C from the storage to arriving tasks. Only one task is executed at the same time and preemptions are allowed. For the sake of simplicity, we bound the power consumption of all tasks to the maximum value p_d. In other words, we introduce the abstraction of a single processing device that determines how much power $P_S(t)$ it uses at any moment in time, i.e.

$$0 < P_S(t) < p_d \, .$$

A task is characterized by its arrival time a_i, its energy demand e_i and its deadline d_i. The effective starting time s_i and finishing time f_i of a task are dependent on the scheduling strategy used: A task started at time s_i will finish as soon as the required amount of energy e_i has been consumed by it. We can write

$$f_i = \min \{t : E_i(s_i, t) = e_i\} .$$

Tasks are considered to be preemptive i.e. the currently active task may be interrupted at any time and continued at a later time. If the full processing power p_d is continuously assigned to a single task J_i, the task is finished after a minimum execution time $w_{i,min} = \frac{e_i}{p_d}$.

3. LAZY SCHEDULING WITH $p_d = \infty$

We start with a node that executes tasks with infinite power $p_d = +\infty$. This theoretical model of a node which runs a task in zero time can be a good approximation for many practical scenarios. If processing times w_i are negligible compared to the time to recharge the battery (i.e. $p_d \gg P_H(t)$), the assumed model can be regarded as reasonable.

Moreover, we assume the processing device on the sensor node to select between three power modes. The node may process tasks with the maximal power $P_S(t) = p_d$ or not at all ($P_S(t) = 0$). In between, the node may choose to spend only the currently incoming power $P_H(t)$ from the harvesting unit on tasks. Altogether, we consider a node that decides between $P_S(t) = P_H(t)$, $P_S = 0$ and $P_S = +\infty$ to advance arriving tasks.

As already indicated in the introduction, the naive approach of scheduling tasks with the EDF algorithm may result in unnecessary deadline violations. Given a node with $p_d = \infty$, LSA avoids spending energy on tasks too early by executing all tasks at their deadline. At time d_j, task J's remaining amount of unprocessed energy $(e_j - E_j(a_j, d_j))$ is drained from the energy storage with $P_S = \infty$. Only if we hit the capacity limit ($E_C(t) = C$) at some time t, we execute the task with the currently earliest deadline using power $P_S(t) = P_H(t)$. The above two rules formulated as pseudo-code are shown in Alg. 1.

In the next section we will see that Alg. 1 is an optimal algorithm for respecting the deadlines of an arbitrary taskset. Note that it degenerates to an *earliest deadline first* (EDF) policy, if $C = 0$. On the other hand, we find an *as late as possible* (ALAP) policy for the case of $C = +\infty$.

4. LAZY SCHEDULING WITH FINITE p_d

Using a device with finite power consumption p_d, one has to take into account finite execution times w_i, too. Obviously, starting at a task's deadline d_j is not appropriate anymore and also determining straightforward starting times $s_j = d_j - \frac{e_j}{p_d}$ does not help: Already a second task J_n arriving shortly

Algorithm 1 (Lazy Scheduling for $p_d = \infty$)

```
Require: maintain a set of indices $i \in Q$ of all ready but not finished tasks $J_i$
  $P_S(t) \Leftarrow 0$;
  while (true)
    $d_j \Leftarrow \min\{d_i : i \in Q\}$;
    process task $J_j$ with power $P_S(t)$;
    $t \Leftarrow$ current time;
    if $t = a_k$    then add index $k$ to $Q$;
    if $t = f_j$    then remove index $j$ from $Q$;
    if $t = d_j$    then $E_C(t) \Leftarrow E_C(t) - e_j + E_j(a_j, t)$;
                         remove index $j$ from $Q$;
                         $P_S(t) \Leftarrow 0$;
    if $E_C(t) = C$ then $P_S(t) \Leftarrow P_H(t)$;
```

after s_j and having an earlier deadline $d_n < d_j$ inevitably causes a deadline violation. Clearly, this kind of timing conflict can be solved by starting tasks earlier. In doing so, however, we risk to run into energy conflicts as pointed out in the introduction. In the following, we focus on finding optimal starting times s_j for a task J_j that balance the *time* and *energy* constraints for our scheduling scenario.

In order to find an optimal starting time s_j for task J_j, LSA requires the knowledge of the incoming power flow $P_H(t)$ for all future times $t \leq d_j$. In addition we make the realistic assumption that $P_H(t) < p_d$, that is, the incoming power $P_H(t)$ from the harvesting unit never exceeds the power consumption p_d of a busy node. Finding useful predictions for the power $P_H(t)$ can be done for example by analyzing traces of the harvested power over a finite duration. If these measurements of the past are also representative for the future, the prediction will be close to the real value of $P_H(t)$.

At first, we consider task J_j illustrated in Fig. 3. We calculate the starting time s_j^* as

$$s_j^* = d_j - \frac{E_C(a_j) + E_H(a_j, d_j)}{p_d}.$$

Once again, we assume that a second task J_n arrives after J_j but has to finish before J_j ($d_n < d_j$). Since at the time of its arrival task J_n has an earlier deadline, it is reasonable to adhere to the well-known EDF policy and interrupt execution of task J_i. At this point, we demonstrate that starting task J_j before or after s_j^* may lead to unnecessary deadline violations.

On the one hand, starting before s_j^* ensures the completion of task J_i whereas task J_n may "starve" because of missing energy and thus finishes after its deadline. Fig. 3 illustrates that a too early starting time may cause this conflict since the nested task J_n has to process with the harvested power flow $P_H(t)$ instead of p_d. Note that in the diagrams also the energy/power assigned to the respective tasks is displayed. On the other hand, starting after s_j^* can result in deadline violations due to lack of time while a sufficient amount of energy is stored on the node. As depicted in Fig. 3, task J_j violates its deadline if the complete energy $E_C(a_j) + E_H(a_j, d_j)$ is needed to process tasks J_j and J_n.

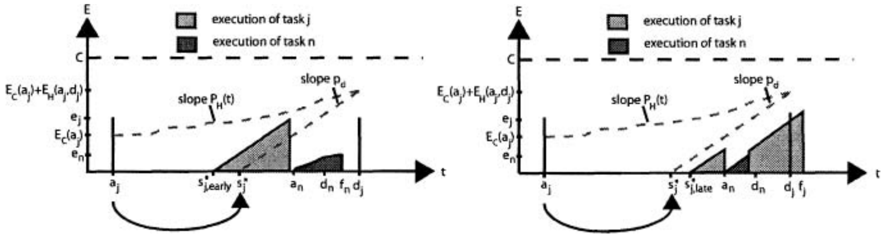


Figure 3. Starting execution of task J_i before or after s_j^*.

If the complete energy $E_C(a_j) + E_H(a_j, d_j)$ is available during the interval of full utilization, the starting time s_j^* is certainly optimal considering both energy and time constraints. But what happens if the stored energy E_C reaches its maximum value C before the starting time s_j^*? – Of course, the overflowing part of the energy $E_H(a_j, s_j^*)$ is used in an EDF-manner to execute task J_j. As a consequence, only energy $C + E_H(s_j^*, d_j)$ can be processed continuously with p_d and it is not possible to maintain full utilisation of the device until the deadline d_j. A better, lazier starting time s'_j can be found by numerically solving the following equation (see Fig. 4):

$$E_H(a_j, s'_j) - C = E_H(a_j, d_j) + (s'_j - d_j)p_d$$

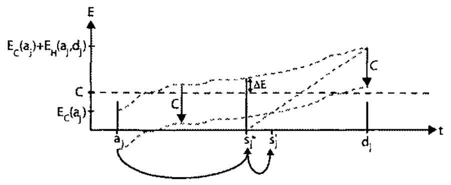


Figure 4. Recalculation of the starting time due to storage limitation.

If we now choose the maximum starting time $s_j = \max\left(s'_j, s_j^*\right)$ we have finally found the optimal starting time. The calculation of s_j must be performed once the scheduler selects the task with the earliest deadline. Then, LSA either executes task J_j with power p_d if the current time $t \geq s_j$, or it adheres to the EDF strategy with $P_S(t) = P_H(t)$ if the storage is full. Alg. 2 shows the pseudo-code for LSA with constant power consumption p_d.

If the scheduler is not energy-constraint, i.e. if the available energy is more than the device can consume with power p_d within $[a_j, d_j]$, the starting time s_j

Algorithm 2 (Lazy Scheduling with $p_d = const.$)

```
Require: maintain a set of indices $i \in Q$ of all ready but not finished tasks $J_i$
  $P_S(t) \Leftarrow 0$;
  while (true)
    $d_j \Leftarrow \min\{d_i : i \in Q\}$;
    calculate $s_j$;
    process task $J_j$ with power $P_S(t)$;
    $t \Leftarrow$ current time;
    if $t = a_k$        then add index $k$ to $Q$;
    if $t = f_j$        then remove index $j$ from $Q$;
    if $E_C(t) = C$ then $P_S(t) \Leftarrow P_H(t)$;
    if $t \geq s_j$       then $P_S(t) \Leftarrow p_d$;
```

will always be before the current time t. Then, the resulting scheduling policy is EDF, which is reasonable, because only time-constraints have to be satisfied. On the other hand, whenever the sum of stored energy E_C and generated energy E_H is small, the scheduling policy changes towards an ALAP policy. In doing so, LSA avoids spending scarce energy on the "wrong" tasks too early. In summary, LSA can be regarded as an adaptive, energy-clairvoyant algorithm that schedules tasks in an Earliest Deadline First fashion.

THEOREM 1 (OPTIMALITY OF LAZY SCHEDULING, $P_S = const.$) *If the Lazy Scheduling Algorithm cannot schedule a given taskset, then no other scheduling algorithm can. This holds even if the other algorithm knows the complete taskset in advance.*

The proof of Theorem 1 is omitted due to space constraints, but can be found in [5]. As a direct consequence of its optimality, LSA successfully schedules a taskset with the minimum possible capacity C.

5. SIMULATION RESULTS

We implemented the Lazy Scheduling Algorithm LSA as well as the Earliest Deadline First EDF algorithm in a simulation framework. Since LSA and EDF may exhibit identical behaviour for a taskset in dependency of p_d, we run all simulations in this section with power $p_d = \infty$. Figure 5 shows the harvested power $P_H(t)$ generated by a random number generator for 1000 time units.

Since the performance of a given algorithm will be severely affected by the properties of the arriving tasks, we performed all simulations in this section for two different tasksets: Taskset T_1 consists of 30 periodic tasks with a common period of 300 time units. Initial phases, energy demands and relative deadlines of the tasks are randomly assigned by a random number generator. Taskset T_2 consists of 8 periodic tasks, also with a common period of 300 time units. In contrast to taskset T_1, phases, energies and deadlines of taskset T_2 are manually assigned. Figure 5 displays the respective values of taskset T_2 within one period.

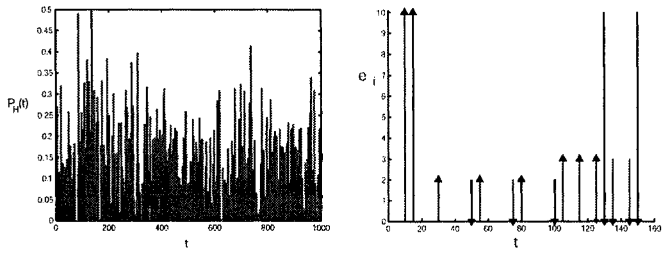


Figure 5. Randomly generated power curve $P_H(t)$ and taskset T_2.

5.1 TIME UNTIL FIRST DEADLINE VIOLATION

We are now interested in the first deadline that cannot be hold with a certain scheduling strategy. With the help of some offline analysis, we tuned the amplitude of the power source $P_H(t)$ to enforce early deadline violations of tasksets T_1 and T_2 respectively. We assume $E_C(0) = C$, i.e. at the beginning of the simulation the battery is fully charged. After a deadline violation is detected, the simulation terminates.

A first simulation result is depicted in Fig. 6 for taskset T_1. Obviously, both curves are monotonically increasing since a longer time is needed to deplete the battery if the initial energy $E_C(0) = C$ is higher. Due to the optimality of LSA, the time of the first deadline violation with EDF is always earlier than with LSA. Though, for some values of the capacity C, the difference between LSA and EDF is very low.

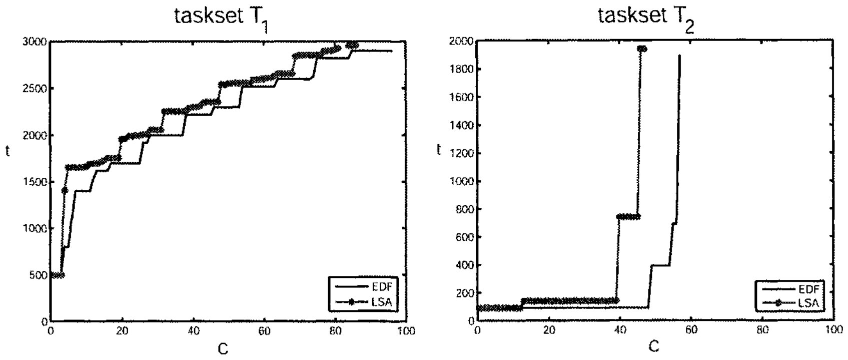


Figure 6. Time t until the first deadline of taskset T_1/T_2 is violated.

The same simulations have been repeated for taskset T_2 (see Fig. 6). For values of the capacity over 60, both algorithms schedule taskset T_2 without deadline violations in the simulated time ($t < 2000$). For capacities between

40 and 60, however, we find significant differences between LSA and EDF. Clearly, taskset T_2 is tailored to the weak points of EDF. But the principal arrangement of taskset T_2 is not unreasonable if tasks for radio communication, sensor activity or data processing have to be executed on the same device. Large differences of the tasks' energy demands and overlapping arrival times and deadlines are the ideas underlying taskset T_2.

5.2 NUMBER OF VIOLATED DEADLINES

Another approach is to disregard missed deadlines and to count only the number of violations. We decided to use an extended version of the LSA algorithm that continues task execution even after a deadline violation. To this end, the scheduler drains $P_H(t)$ from the storage until the task is finally finished.

Fig. 7 presents the number of recorded deadline violations that occurred during a period of 3000 time units for taskset T_1. Since Lazy Scheduling makes the best use of the scavenged energy, it outperforms EDF for all capacities. The difference between LSA and EDF is varying between 0 and 9 deadline violations in the considered interval. For high values of C, no deadline violations could be detected for both algorithms because of the high energy availability.

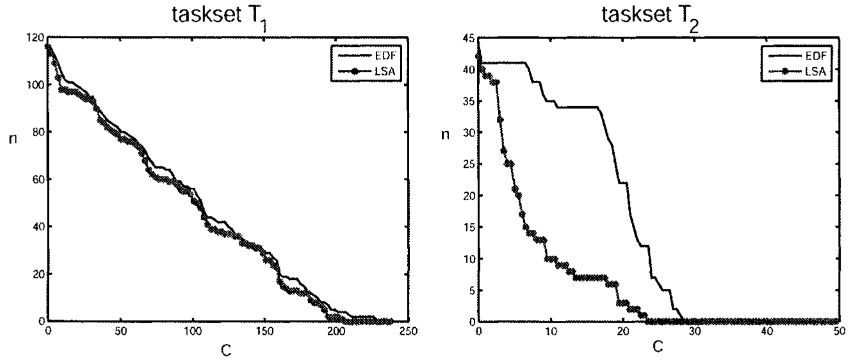


Figure 7. Number n of violated deadlines of taskset T_1/T_2.

For taskset T_2, LSA performs significantly better in terms of violated deadlines than EDF. Especially for values of C between 10 and 20, LSA's energy management pays off. On the other hand, for a given number n of violated deadlines, a much higher capacity C is necessary under EDF scheduling. For example, to obtain $n = 0$ deadline violations, LSA requires a capacity $C = 24$ while EDF needs a 25% higher capacity ($C = 30$) to respect all deadlines.

6. CONCLUSION

In this paper, we studied the case of an energy harvesting sensor node that has to schedule a set of real-time tasks. These tasks require a certain amount of energy as well as time to complete. We have discussed Lazy Scheduling Algorithms for online scheduling. However, LSA algorithms are energy-clairvoyant, i.e., the profile of the energy generated in the future has to be known to the scheduler in advance. Finally, simulation results demonstrate how LSA outperforms the Earliest Deadline First Algorithm and that significant reductions of the battery size are possible when running LSA.

ACKNOWLEDGMENTS

The work presented in this paper was partially supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322. In addition, this research has been founded by the European Network of Excellence ARTIST2.

REFERENCES

[1] A. Allavena and D. Mosse. Scheduling of frame-based embedded systems with rechargeable batteries. In *Workshop on Power Management for Real-Time and Embedded Systems (in conjunction with RTAS 2001)*, 2001.

[2] Y. Ammar, A. Buhrig, M. Marzencki, B. Charlot, S. Basrour, K. Matou, and M. Renaudin. Wireless sensor network node with asynchronous architecture and vibration harvesting micro power generator. In *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*, pages 287–292, New York, NY, USA, 2005. ACM Press.

[3] X. Jiang, J. Polastre, and D. E. Culler. Perpetual environmentally powered sensor networks. In *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005*, pages 463–468, UCLA, Los Angeles, California, USA, April 25-27 2005.

[4] A. Kansal, D. Potter, and M. B. Srivastava. Performance aware tasking for environmentally powered sensor networks. In *Proceedings of the International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS 2004*, pages 223–234, New York, NY, USA, June 10-14 2004. ACM Press.

[5] C. Moser, D. Brunelli, L. Thiele, and L. Benini. Real-time scheduling with regenerative energy. In *18th Euromicro Conference on Real-Time Systems, ECRTS 2006*, Dresden, Germany, July 5-7 2006.

[6] S. Roundy, D. Steingart, L. Frechette, P. K. Wright, and J. M. Rabaey. Power sources for wireless sensor networks. In *Wireless Sensor Networks, First European Workshop, EWSN 2004, Proceedings*, Lecture Notes in Computer Science, pages 1–17, Berlin, Germany, January 19-21 2004. Springer.

[7] C. Rusu, R. G. Melhem, and D. Mosse. Multi-version scheduling in rechargeable energy-aware real-time systems. In *15th Euromicro Conference on Real-Time Systems, ECRTS 2003*, pages 95–104, Porto, Portugal, July 2-4 2003.

TRANSIENT PROCESSOR/BUS FAULT TOLERANCE FOR EMBEDDED SYSTEMS

With hybrid redundancy and data fragmentation

Alain Girault[1], Hamoudi Kalla[2], and Yves Sorel[3]

[1]*INRIA Rhône-Alpes, 655 avenue de l'Europe, 38334 Saint-Ismier cedex, FRANCE*
Alain.Girault@inrialpes.fr

[2]*IRISA, Campus Universitaire de Beaulieu, 35042 Rennes Cedex France Cedex, FRANCE*
Hamoudi.Kalla@irisa.fr

[3]*INRIA Rocquencourt, B.P.105 – 78153 Le Chesnay Cedex, FRANCE*
Yves.Sorel@inria.fr

Abstract We propose an approach to build fault-tolerant distributed real-time embedded systems. From a given system description (application algorithm and architecture) and a given fault hypothesis (type and number of faults to be tolerated), we generate automatically a static fault-tolerant multiprocessor schedule of the algorithm components on the target architecture, which minimizes the schedule length, and tolerates transient faults of both processors and communication media. Our approach is dedicated to heterogeneous architectures with multiple processors linked by several shared buses. It is based on hybrid redundancy and data fragmentation strategies, which allow fast fault detection and handling. This scheduling problem is NP-hard and we rely on a heuristic algorithm to obtain efficiently an approximate solution. Our simulation results show that our approach generally reduces the schedule length overhead.

Keywords: real-time embedded systems, safety-critical systems, transient faults, scheduling heuristics, hybrid redundancy, data fragmentation, heterogeneous architectures.

1. INTRODUCTION

Today, embedded real-time systems invade many sectors of human activity, such as transportation, robotics, and telecommunication. The progresses achieved in electronics and data processing improve the performances of these systems. As a result, the new systems are increasingly small and fast, but also more complex and critical, and thus more sensitive to faults. Due to catastrophic consequences (human, ecological, and/or financial disasters) that could result from a fault, these systems must be fault-tolerant. This is why fault-tolerant techniques are necessary to make sure that the system continues to deliver a correct service in spite of faults [1]. A fault can affect either the hardware or the software of the system. Thanks to formal validation techniques, such as model-checking and theorem proving, a lot of software faults can be prevented. Although software faults are still an important issue, we chose to concentrate on hardware faults. More particularly, we consider processor and bus faults. A bus is a multipoint connection characterized by a physical medium that connects all the processors of the architecture. As we are targeting embedded systems with limited resources (for reasons of weight, volume, energy consumption, or price constraints), we investigate only software redundancy solutions based on scheduling algorithms.

The paper is organized as follows. Sections 2 and 3 describe respectively related work and system models. Section 4 states the faults assumptions and our fault-tolerance problem. Section 5 presents our approach for providing fault-tolerance, and Section 6 details the performances of our approach. Finally, Section 7 concludes the paper and proposes future research directions.

2. RELATED WORK

The literature about fault tolerance of distributed embedded real-time systems is very abundant. Yet, there are very few methods that manage to tolerate both processor and bus faults. Here, we present related work involving scheduling heuristics to tolerate processor faults, bus faults, or both.

Processor faults. Several scheduling heuristics have been proposed to tolerate exclusively processor faults. They are based on active software redundancy [2, 3] or passive software redundancy [4–6]. In active redundancy, multiple replicas of a task are scheduled on different processors, which are run in parallel to tolerate a fixed number of processor faults. [2] presents an off-line scheduling algorithm that tolerates a single processor faults in multiprocessor systems, while [3] tolerates multiple processor faults. In passive redundancy, also called primary/backup approach, a task is replicated into one primary and several backup replicas, but only the primary replica is executed. If it fails, one of the backup replicas is selected to become the new primary. For instance, [5] presents a scheduling algorithm that tolerates one processor fault.

Bus faults. Techniques proposed to tolerate exclusively buses faults are based on proactive or reactive schemes. In the proactive scheme [7, 8], multiple redundant copies of a message are sent along distinct buses. In contrast, in the reactive scheme [9], only one copy of the message, called primary, is sent; if it fails, another copy of the message, called backup, will be transmitted.

Processor and bus faults. Few techniques have been proposed to tolerate both processor and bus faults [10–12]. In [10], faults of buses are tolerated using a TDMA (Time Division Multiple Access) communication protocol and an active redundancy approach, while faults of processors are tolerated using a hardware redundancy approach. The approach proposed in [11] tolerates only a specified set of processor and bus permanent faults. The method proposed in [12] is only suited to one class of algorithms called fan-in algorithms. Our approach is more general since it uses *only software redundancy solutions*, i.e., no extra hardware is required, because hardware resources in embedded systems are limited. Moreover, our approach can tolerate up to a fixed number of *arbitrary processor and bus transient faults*. This is important since transient faults [13] are increasingly the majority of faults in logic circuits, due to radiation, energetic particles, and so on.

3. SYSTEM DESCRIPTION

In this section, we present the system models (algorithm and architecture), and define the execution characteristics of the algorithm on the architecture.

Algorithm model. The algorithm is modeled by a data-flow graph, called algorithm graph and noted Alg. Each vertex of Alg is an operation and each edge is a data-dependency. A data-dependency $(o_1 \triangleright o_2)$ corresponds to a data transfer from a producer operation o_1 to a consumer operation o_2, defining a partial order on the execution of operations. We say that o_2 is a successor of o_1, and that o_1 is a predecessor of o_2. An operation of Alg can be either an external input/output operation or a computation operation. Operations with no predecessor (resp. no successor) are the input interfaces (resp. output), handling the events produced by the sensors (resp. actuators). The inputs of a computation operation must precede its outputs. Moreover, computation operations are side-effect free, i.e., the output values depend only of the input values.

Figure 1(left) is an example of Alg, with seven operations: In_1 and In_2 (resp. Out_1) are input (resp. output) operations, while A, B, C and D are computation operations. The data-dependencies between operations are depicted by arrows. For instance the data-dependency $(A \triangleright D)$ can correspond to the sending of some arithmetic result computed by A and needed by D.

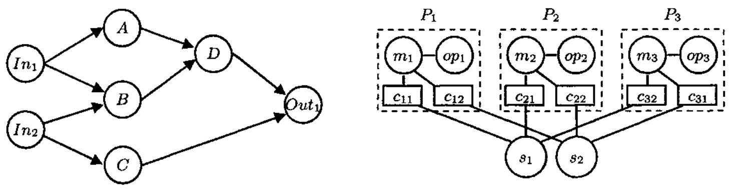


Figure 1. Example of an algorithm graph (left) and an architecture graph (right).

Architecture model. The architecture is composed of two principal components: a processor and a bus. A processor P_i consists of an operator op_i, a memory resource m_i of type RAM (Random Access Memory), and several communicators c_{ij}. A bus B_i consists of one communicator for each existing processor and one memory resource s_i of type SAM (Sequential Access Memory). Each operator executes sequentially a set of operations of $\mathcal{A}lg$, and reads and writes data from and into its local memory. Each communicator of each processor cooperates with each other in order to execute sequentially transfers of data stored in the memory between processors through a SAM.

The architecture is modeled by a non-directed graph, called architecture graph and noted $\mathcal{A}rc$. Vertices of $\mathcal{A}rc$ are: operators, communicators, and memory resources. Edges of $\mathcal{A}rc$ are connections between these components. Figure 1(right) gives an example of $\mathcal{A}rc$, with three processors P_1, P_2, and P_3, and two buses $B_1{=}\{s_1, c_{11}, c_{21}, c_{31}\}$ and $B_2{=}\{s_2, c_{12}, c_{22}, c_{32}\}$, where each processor P_i is made of one operator op_i, one local memory m_i, and two communicators c_{i1} and c_{i2}.

Execution characteristics. We target systems based on a cyclic execution model; this means that a fixed schedule of the operations of $\mathcal{A}lg$ is executed cyclically on $\mathcal{A}rc$ at a fixed rate. This schedule must satisfy one real-time constraint $\mathcal{R}tc$ and a set of distribution constraints $\mathcal{D}is$. In our execution model $\mathcal{E}xe$, we associate to each operator op a list of pairs $\langle o, d/op\rangle$, where d is the worst case execution time (WCET) of the operation o on op. Also, we associate to each communicator c a list of pairs $\langle dpd, d/c\rangle$, where d is the worst case transmission time (WCTT) of the data-dependency dpd on c. Since we target heterogeneous architecture, WCET (resp. WCTT) for a given operation (resp. data-dependency) can be distinct on each operator (resp. communicator). Specifying the distribution constraints $\mathcal{D}is$ amounts to associating the value "∞" to some pairs of $\mathcal{E}xe$: $\langle o, \infty/op\rangle$ meaning that o cannot be executed on op. Finally, since we produce static schedules, we can compute their length and compare it to the real-time constraint $\mathcal{R}tc$.

4. FAULT MODEL AND SCHEDULING PROBLEM

In our fault hypothesis, we assume only hardware faults and a fault-free software. We consider only transient processor and bus faults. Transient faults, which persist for a "short" duration, are significantly more frequent than other faults in systems [13]. Permanent faults are a particular case of transient faults. We assume at most $\mathcal{N}pf$ processor faults and $\mathcal{N}bf$ buses faults can occur in the system, and that the architecture includes at least $\mathcal{N}pf{+}1$ processors and $\mathcal{N}bf{+}1$ buses. Our problem is therefore formally stated as:

PROBLEM 1 *Given:*

- *a distributed heterogeneous architecture $\mathcal{A}rc$ composed of a set $\mathcal{P}$ of processors and a set $\mathcal{B}$ of buses: $\mathcal{P} = \{\dots, P_i, \dots\}$, $\mathcal{B} = \{\dots, B_j, \dots\}$*
- *an algorithm $\mathcal{A}lg$ composed of a set $\mathcal{O}$ of operations and a set $\mathcal{E}$ of data-dependencies: $\mathcal{O} = \{\dots, o_i, \dots, o_j, \dots\}$, $\mathcal{E} = \{\dots, (o_i \triangleright o_j), \dots\}$*
- *all the execution characteristics $\mathcal{E}xe$ of the algorithm components of $\mathcal{A}lg$ on the architecture components of $\mathcal{A}rc$,*

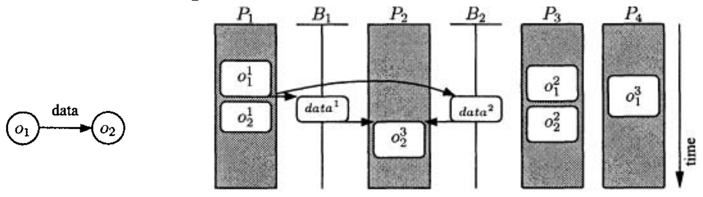
- *a real-time constraint $\mathcal{R}tc$ (schedule length), and several distribution constraints $\mathcal{D}is$,*
- *a number $\mathcal{N}pf < |\mathcal{P}|$ of processor faults that may affect the system,*
- *a number $\mathcal{N}bf < |\mathcal{B}|$ of bus faults that may affect the system,*

find a multiprocessor static schedule of $\mathcal{A}lg$ on $\mathcal{A}rc$, which minimizes the schedule length, and tolerates up to $\mathcal{N}pf$ processor and $\mathcal{N}bf$ bus faults with respect to $\mathcal{R}tc$, $\mathcal{E}xe$, and $\mathcal{D}is$.

5. THE PROPOSED APPROACH

Our solution is based on hybrid redundancy and data fragmentation techniques. In the aim to minimize communication overhead, we use active redundancy to tolerate processor faults, and passive redundancy to tolerate bus faults. The reason why to use data fragmentation is to minimize the fault detection latency, i.e, the time it takes to detect a fault.

Hybrid redundancy and data fragmentation. In order to tolerate $\mathcal{N}pf$ processor and $\mathcal{N}bf$ bus faults, each operation is replicated in $\mathcal{N}pf+1$ replicas scheduled on $\mathcal{N}pf+1$ distinct processors. The replica with the earliest ending time is the primary replica, while the other ones are the backup replicas. The earliest ending time is the sum of the earliest starting time (computed in absence of faults) plus the operation's WCET. The data of each data dependency is fragmented into $\mathcal{N}bf+1$ packets, sent by the primary replica of the data-dependency source via $\mathcal{N}bf+1$ distinct buses to each of the $\mathcal{N}pf+1$ replicas of the data-dependency destination. For example, in the schedule of Figure 2b, operations o_1 and o_2 of Figure 2a are replicated into three replicas to tolerate two processors faults ($\mathcal{N}pf=2$), and the data of the data-dependency $(o_1 \triangleright o_2)$ are fragmented into two packets to tolerate one bus fault ($\mathcal{N}bf=1$).

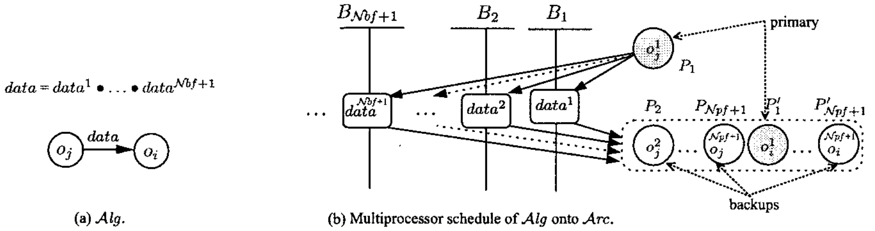


(a) $\mathcal{A}lg$. (b) Multiprocessor schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$.

Figure 2. Tolerating two processors and one bus faults.

Figure 3 illustrates these principles in the general case where $\mathcal{N}pf \geq 1$ and $\mathcal{N}bf \geq 1$. Only the primary replica of each operation o_j sends all the fragmented data "$data^m$", of each of its data outputs, in parallel via $\mathcal{N}bf+1$ buses to all the replicas of all its successor operations in $\mathcal{A}lg$.

Communication mechanism. Each operation receives each of its data inputs via $\mathcal{N}bf+1$ buses; when it has received all the packets of each data input, it defragments these packets and starts its execution. In some cases, the replica of an operation will only receive some of its inputs once, through an intra-processor communication; this will occur whenever one of its predecessor operations has one of its replicas scheduled on the *same* processor.



(a) $\mathcal{A}lg$. (b) Multiprocessor schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$.

Figure 3. Tolerating $\mathcal{N}pf$ processors and $\mathcal{N}bf$ buses faults.

Transient fault recovery and handling. In Figure 3, three cases can occur:

1. All the packets $data^m$ sent by o_j^1 are received: in this case, each replica of o_i defragments these packets and starts its execution. Also, each replica of o_j receives a copy of these packets, which it ignores.

2. None of the packets $data^m$ sent by o_j^1 are received: this concerns $\mathcal{N}bf+1$ packets, and as no more than $\mathcal{N}bf$ buses faults may occur in the system (by hypothesis), this means the failure of the processor P_1 executing the replica o_j^1. To deal with this failure, one backup replica among the $\mathcal{N}pf$ other replicas of o_j is selected to re-send all the packets $data^m$ via the same buses. Since the fault of processor P_1 can be transient, it is not marked as faulty by the other processors. This scheme can be improved by deciding that, if a processor remains faulty during some number of consecutive executions of the schedule (e.g., 5), then its fault is permanent and this processor is permanently removed from the schedule.

3. Some packets $\{data^m, \ldots, data^k\}$ sent by o_j^1 are not received: let $data^-$ be this set of missing packets, and $\mathcal{B}^- = \{B^m, \ldots, B^k\}$ be the set of the buses that were supposed to transmit them. Since other packets have been received, it means that P_1, the processor executing o_j^1 is not faulty, and hence that the buses of $\mathcal{B}^-$ are faulty. Therefore, the same replica o_j^1 re-sends the packets $data^-$ via other buses chosen among the set $\mathcal{B} \setminus \mathcal{B}^-$. Since the fault of the buses of $\mathcal{B}^-$ can be transient, they are not marked as being faulty. This scheme can be improved with a similar approach as in step **2.**

In summary, this communication mechanism yields three advantages: ① fast fault detection; ② fast distinction between processor and bus faults; and ③ fast fault recovery.

We have implemented these principles in a greedy list scheduling heuristic, called FT-AAA (Fault-Tolerant Adequation Algorithm Architecture). In the following algorithm of FT-AAA, the superscript numbers in parentheses refer to the steps of the heuristic, e.g., $O_{sched}^{(n)}$:

```
ALGORITHM FT-AAA
- Inputs = $\mathcal{A}lg$, $\mathcal{A}rc$, $\mathcal{N}pf$, $\mathcal{N}bf$, $\mathcal{E}xe$, $\mathcal{R}tc$, and $\mathcal{D}is$;
- Output = a fault-tolerant multiprocessor static schedule;
                                                        INITIALIZATION
Initialize the sets of candidate operations $O_{cand}$ and scheduled operations $O_{sched}$:
    $O^{(1)}_{cand} := \{$operations of $\mathcal{A}lg$ without predecessors$\}$;
    $O^{(1)}_{sched} := \emptyset$;
While $O^{(n)}_{cand} \neq \emptyset$ do
                                                        SELECTION
    - Select for each candidate operation $o_{cand}$ of $O^{(n)}_{cand}$ a set $\mathcal{P}_{best}$ of $\mathcal{N}pf$+1 processors that
      minimizes the dependable schedule pressure (Equation (1));
    - Select for each candidate operation $o_{cand}$ of $O^{(n)}_{cand}$, among the processors $\mathcal{P}_{best}(o_{cand})$,
      the best processor $P_{best}$ that maximizes the dependable schedule pressure;
    - Select, among all the pairs $(o_{cand}, P_{best})$, the best pair $(o_{best}, P_{best})$ that maximizes the
      dependable schedule pressure;
                                                        DISTRIBUTION AND SCHEDULING
    - Let $\mathcal{P}_{best}(o_{best})$ be a best set of $\mathcal{N}pf$+1 processors of $o_{best}$ computed at the "Selection"
      step;
    - For each $o_j$, predecessor of $o_{best}$, fragment the data of the data-dependency $(o^1_j \rhd o_{best})$
      into $\mathcal{N}bf$+1 packets $data^m$;
    - Schedule the packets $data^m$ of each data-dependency on $\mathcal{N}bf$+1 distinct buses;
    - Add $\mathcal{N}pf$ replicas of $o_{best}$ into $\mathcal{A}lg$;
    - Schedule each replica $o^k_{best}$ on the processor $P^k_{best}$ of $\mathcal{P}_{best}(o_{best})$.
                                                        UPDATE SETS
    - Update the sets of candidate and scheduled operations for the next step $(n+1)$:
      $O^{(n+1)}_{sched} := O^{(n)}_{sched} \cup \{o_{best}\}$;
      $O^{(n+1)}_{cand} := O^{(n)}_{cand} - \{o_{best}\} \cup$
          $\left\{o_{new} \in \{\text{successors of } o_{best}\} \mid \{\text{predecessors of } o_{new}\} \subseteq O^{(n+1)}_{sched}\right\}$;
end While
                                                        END OF THE ALGORITHM
```

The algorithm of FT-AAA is divided in four main steps:

Initialization step. The set of candidate operations $O^{(1)}_{cand}$ is initialized as the operations without predecessor. Later, an operation is said to be a candidate if all its predecessors are already scheduled. The set of scheduled operations $O^{(1)}_{sched}$ is initially empty.

Selection step. For each candidate operation $o_{cand} \in O^{(n)}_{cand}$, a set $\mathcal{P}_{best}$ of $\mathcal{N}pf$+1 processors is selected among all the processors of $\mathcal{P}$ to schedule $\mathcal{N}pf$+1 replicas of o_{cand}. The selection rule is based on the dependable schedule pressure function, noted $\sigma^{(n)}$. It is computed, for each operation $o_i \in O^{(n)}_{cand}$ and each processor $P_j \subset \mathcal{P}$, as follows:

$$\sigma^{(n)}(o_i, P_j) := S^{(n)}_{o_i,P_j} + \overline{S}^{(n)}_{o_i} - R^{(n-1)} \quad (1)$$

where $S^{(n)}_{o_i,P_j}$ is the earliest time at which operation o_i can start its execution on processor P_j, $\overline{S}^{(n)}_{o_i}$ is the latest start time from end of o_i (defined to be the length of the longest path from the output operations to o_i), and $R^{(n-1)}$ is the schedule length at step $(n-1)$. The set $\mathcal{P}_{best}$ of each $o_{cand} \in O^{(n)}_{cand}$ is composed of the $\mathcal{N}pf$+1 processors that minimize $\sigma^{(n)}$. Then, among all $O^{(n)}_{cand}$, the most urgent candidate o_{best}, with a processor $P_{best} \in \mathcal{P}_{best}(o_{best})$ that maximizes this function, is selected to be replicated and scheduled.

Distribution and scheduling step. This step involves first replicating the best candidate o_{best} into $\mathcal{N}pf + 1$ replicas, and second scheduling each replica o^k_{best} of o_{best} respectively on the processor P^k_{best} of $\mathcal{P}_{best}$. Before scheduling each of these replicas, the data of each data-dependency are fragmented into $\mathcal{N}bf$+1 packets that are scheduled on $\mathcal{N}bf$+1 distinct buses.

Updating step. The scheduled operation o_{best} is removed from $O^{(n)}_{cand}$, and the operations of $\mathcal{A}lg$ which have all their predecessors in the new set of scheduled operations are added to this set.

6. SIMULATIONS

To evaluate FT-AAA, we have implement it in SYNDEX, a CAD tool for optimizing and implementing real-time embedded systems (`http://www.syndex.org`). Then, we have applied the FT-AAA heuristic to a set of randomly generated algorithm graphs and an architecture graph composed of five processors ($|\mathcal{P}| = 5$) and four buses ($|\mathcal{B}| = 4$). In our simulations, we study the impact of $\mathcal{N}pf$, $\mathcal{N}bf$, the number of operations N, and CCR (Communication to Computation Ratio) on the schedule length overhead introduced by FT-AAA, computed by Equation (2):

$$\text{overhead} = \frac{\text{length}(\text{FT-AAA}(\mathcal{N}pf, \mathcal{N}bf)) - \text{length}(\text{AAA})}{\text{length}(\text{AAA})} \qquad (2)$$

where FT-AAA takes as parameter the numbers of processor and bus faults $(\mathcal{N}pf, \mathcal{N}bf)$, AAA is exactly FT-AAA$(0,0)$, and "length" is a function that computes the schedule's length.

Impact of $\mathcal{N}bf$ and N. We have plotted in Figure 4 the average overheads on the schedule length of 100 random algorithm graphs for each N, $\mathcal{N}pf$=0, CCR=1, and $\mathcal{N}bf$=1, 2, 3. This figure shows that the average overhead is very low (between 6% and 18%) and increases slightly with N. This is due first to $\mathcal{N}pf$=0, i.e., operations of $\mathcal{A}lg$ are not replicated, and second to the use of passive redundancy of communication. Also, for the three values of $\mathcal{N}bf$, the heuristics FT-AAA(0,1), FT-AAA(0,2) and FT-AAA(0,3) bear almost similar results with no significant advantage between the three variants.

Impact of $\mathcal{N}pf$ and N. We have plotted in Figure 5 the average overheads on the schedule length of 100 random $\mathcal{A}lg$ for each N, $\mathcal{N}bf$=0, CCR=1, and $\mathcal{N}pf$=1, 2. This figure shows that the average overhead when $\mathcal{N}pf$=1 is 45%, while for $\mathcal{N}pf$=2 it is 75%. These figures are much lower than the expected 100% when all computations are scheduled twice, and 200% when all computations are scheduled thrice. It also shows that the performances of FT-AAA decrease when $\mathcal{N}pf$ increases. This is due to the fact that FT-AAA uses the active redundancy of operations. However, for the two values of $\mathcal{N}pf$, FT-AAA($\mathcal{N}pf$,0) produces almost no significant difference between the overheads obtained for the different values of N.

Impact of CCR**.** We have plotted in Figure 6 the average overheads on the schedule length of 100 random $\mathcal{A}lg$ for N=40, $\mathcal{N}pf$=1, $\mathcal{N}bf$=1,2,3, and each CCR. Thanks to the data fragmentation, this figure shows that, when the communications are less expensive than the computations (CCR <1), the performances are almost identical for $\mathcal{N}bf$=1 to 3. In contrast, when the communications are more expensive (CCR >1), the performances decrease when $\mathcal{N}bf$ increases. Also, for $\mathcal{N}bf \leq 2$, CCR has no significant impact on the performances of FT-AAA; again this is due to the data fragmentation. It is not true anymore when $\mathcal{N}bf \geq 3$, because the number of buses, 4, becomes limitative.

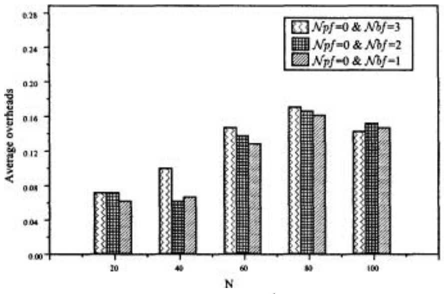


Figure 4. Impact of $\mathcal{N}bf$ and N.

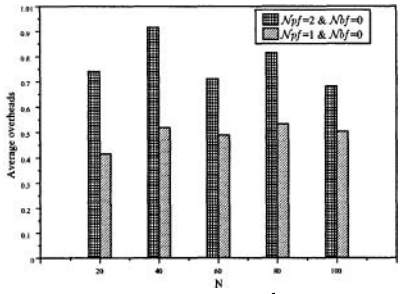


Figure 5. Impact of $\mathcal{N}pf$ and N.

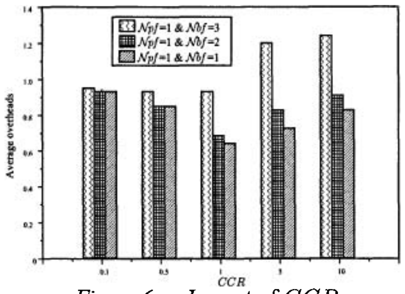


Figure 6. Impact of CCR.

7. CONCLUSION

We have proposed in this paper a solution to tolerate transient faults of both processors and communication media in distributed heterogeneous architectures with multiple-bus topology. Our solution, based on hybrid redundancy and data-fragmentation strategies, is a list scheduling heuristic, called

FT-AAA. It generates automatically a multiprocessor static schedule of a given algorithm on a given architecture, which minimizes the schedule length, and tolerates up to $\mathcal{N}pf$ processors and up to $\mathcal{N}bf$ buses faults, with respect to real-time and distribution constraints. The communication mechanism, based on data-fragmentation, allows the fast distinction between processor and bus faults, the fast detection of faults, and the fast handling of faults. Simulations show that our approach can generally reduce the schedule length overhead. Currently, we are working on an improved solution to take sensors/actuators faults into account.

REFERENCES

[1] P. Jalote. *Fault-Tolerance in Distributed Systems*. Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[2] K. Hashimoto, T. Tsuchiya, and T. Kikuno. Effective scheduling of duplicated tasks for fault-tolerance in multiprocessor systems. *IEICE Trans. on Information and Systems*, E85-D(3):525–534, March 2002.

[3] A. Girault, H. Kalla, M. Sighireanu, and Y. Sorel. An algorithm for automatically obtaining distributed and fault-tolerant static schedules. In *International Conference on Dependable Systems and Networks, DSN'03*, San-Francisco, USA, June 2003. IEEE.

[4] K. Ahn, J. Kim, and S. Hong. Fault-tolerant real-time scheduling using passive replicas. In *Pacific Rim International Symposium on Fault-Tolerant Systems*, Taipei, Taiwan, December 1997.

[5] X. Qin, H. Jiang, and D. R. Swanson. An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems. In *International Conference on Parallel Processing*, pages 360–386, Vancouver, Canada, August 2002.

[6] Y. Oh and S. H. Son. Scheduling real-time tasks for dependability. *Journal of Operational Research Society*, 48(6):629–639, June 1997.

[7] N. Kandasamy, J.P. Hayes, and B.T. Murray. Dependable communication synthesis for distributed embedded systems. In *International Conference on Computer Safety, Reliability and Security, SAFECOMP'03*, Edinburgh, UK, September 2003.

[8] S. Dulman, T. Nieberg, J. Wu, and P. Havinga. Trade-off between traffic overhead and reliability in multipath routing for wireless sensor networks. In *Wireless Communications and Networking Conference*, 2003.

[9] B. Kao, H. Garcia-Molina, and D. Barbara. Aggressive transmissions of short messages over redundant paths. *IEEE Trans. on Parallel and Distributed Systems*, 5(1):102–109, January 1994.

[10] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, October 2003.

[11] C. Dima, A. Girault, and Y. Sorel. Static fault-tolerant scheduling with "pseudo-topological" orders. In *Joint Conference FORMATS-FTRTFT'04*, volume 3253 of *LNCS*, Grenoble, France, September 2004. Springer-Verlag.

[12] R. Vaidyanathan and S. Nadella. Fault-tolerant multiple bus networks for fan-in algorithms. In *International Parallel Processing Symposium*, pages 674–681, April 1996.

[13] M. Pizza, L. Strigini, A. Bondavalli, and F. Di Giandomenico. Optimal discrimination between transient and permanent faults. In *3rd IEEE High Assurance System Engineering Symposium*, pages 214–223, Bethesda, MD, USA, 1998.

DISTRIBUTED TIMED MULTITASKING - A MODEL OF COMPUTATION FOR HARD REAL-TIME DISTRIBUTED SYSTEMS

Christo Angelov[1] and Jesper Berthing[2]
[1]Mads Clausen Institute for Product Innovation, University of Southern Denmark, Grundtvigs Alle 150, 6400 Soenderborg, Denmark; [2]Danfoss Drives A/S, Ulsnaes 1, 6300 Graasten, Denmark

Abstract: The paper presents a new type of system architecture for distributed embedded systems, whereby a system is composed from embedded actors that communicate transparently by exchanging labeled messages (signals), independent of their allocation onto network nodes. Signals are exchanged at precisely specified time instants in accordance with the concept of Timed Multitasking, whereby actors can be viewed as real-time tasks with event-triggered input and output drivers activated by timing or external events. The combination of actors, signal-based communication and timed multitasking has resulted in a new operational model – *Distributed Timed Multitasking (DTM)*. The latter has been used to derive a task execution model and requirements specifications for a real-time kernel, which have been used to develop the timed-multitasking version of the *HARTEX* kernel. In this context, the paper presents a discussion of specific implementation issues related to the execution of periodic and sporadic tasks under DTM.

Keywords: hard real-time distributed systems; actor-based architecture; signal-based communication; timed multitasking

1. INTRODUCTION

There are essentially two approaches to task scheduling for dependable embedded systems: static scheduling and predictable dynamic scheduling, i.e. rate-monotonic or deadline-monotonic scheduling[1,2]. Static scheduling provides for deterministic and highly predictable behaviour under hard real-time constraints. That is why it is widely used in safety-critical systems, e.g. aerospace and military systems, automotive applications, etc. However, static scheduling has a major disadvantage: its application results in closed systems that are generally difficult to re-configure and maintain. This is in contradiction to the requirement for an open and flexible system architecture that ought to support software reuse and reconfiguration.

The second approach is inherently more flexible but unfortunately, dynamic scheduling is plagued by another problem, i.e. task execution jitter, which is largely due to interrupts and task preemption. That is ultimately demonstrated as input/output jitter, which is detrimental to control system operation.

The above problem can be overcome through a new scheduling paradigm known as *Timed Multitasking (TM)*[3]. This is a generalized event-driven execution model that accommodates timing, external and internally generated events, which goes beyond pure time-triggered models such as *Giotto*[4]. Timed Multitasking eliminates task execution jitter via *split-phase execution* of tasks, whereby task I/O drivers are executed atomically at precisely specified time instants (i.e. task release and deadline instants), whereas application tasks are executed in a dynamic scheduling environment. Hence, task jitter is of no consequence, as long as the tasks finish execution before their deadlines. Ultimately, timed multitasking makes it possible to engineer real-time systems combining high flexibility inherent to dynamic scheduling and predictable operation, which is typical for statically scheduled systems.

Unfortunately, the existing implementation has a number of limitations:

- It supports only local communication via interconnected ports implemented as shared data structures[3], whereby an output driver of the producer task has to explicitly invoke the input driver of the receiver task in order to send an event (message) into the input port of the receiver task.
- Consequently, the communication pattern is "hardwired" in the code of I/O drivers and cannot be reconfigured without reprogramming. Obviously, this technique is also not quite suitable for implementing one-to-many multicast or broadcast communication.
- The implementation outlined in the paper[3] uses conventional (blocking) techniques for exclusive access to input port data structures. These are not appropriate for hard real-time tasks, which are often implemented as non-blocking basic tasks.
- Another limitation comes from the fact that each task has to use its own set of timers (e.g. period and deadline timers), which could result in considerable kernel overhead. That overhead might be substantially reduced by developing an integrated time manager, featuring simultaneous execution of identical operations involving different tasks (e.g. release simultaneously multiple tasks at a particular time instant)[7].

The above observations have motivated research that resulted in an extended version of timed multitasking - *Distributed Timed Multitasking (DTM)*, which has been developed in the context of the *COMDES-II* framework. The latter is characterized by an actor-based component model featuring transparent signal-based communication between actors, which is independent of their allocation onto network nodes.

This paper presents Distributed Timed Multitasking and its implementation in *HARTEX*$_{TM}$ - a timed-multitasking version of the *HARTEX* kernel. The rest of the paper is structured as follows: Section 2 presents Distributed Timed Multitasking, and the related task execution model, which has been used to derive the requirements specifications for the *HARTEX*$_{TM}$ kernel. Section 3 presents a discussion of implementation issues, focusing on event management for periodic and sporadic tasks executed in a timed multitasking environment. The concluding section presents a summary of the main features of Distributed Timed Multitasking and its implications for embedded software design.

2. DISTRIBUTED TIMED MULTITASKING

Distributed Timed Multitasking has been developed in the context of *COMDES-II* - an actor-based version of the *COMDES* framework[5]. The latter defines a hierarchy of executable models, such as function blocks, activities and function units that can be used to configure embedded control systems.

Under *COMDES-II*, the control system is composed from a number of (possibly distributed) embedded actors, whereby system structure is modeled by a data flow diagram specifying constituent actors and their interactions (see Fig. 1). Actors are logically grouped together into subsystems (function units), such as *Sensor Unit, Control Unit*, etc., that may reside in one or more network nodes. This solution provides for greater flexibility in comparison with the original version of the framework where a function unit is conceived as high-level component (software integrated circuit) that has to be always allocated on a particular network node.

Actors interact by exchanging labeled messages (global signals) within various types of distributed transactions. Communication between actors is *transparent*, i.e. independent of their allocation on network nodes. It follows the producer-consumer model of communication within one-to-one and one-to-many interactions involving local and/or distributed actors.

Signals are exchanged at precisely-specified time instants in accordance with the concept of *Timed Multitasking*, whereby actors can be viewed as application objects (tasks) with event-triggered inputs and outputs that can be activated by various types of periodic and sporadic events, depending on the type of transaction (e.g. Fig. 2). This figure shows a distributed phase-aligned transaction, involving actors *Sensor (S), Controller (C)* and *Actuator (A)*, whereby the process variable is sampled at the start of the period and the output is actuated upon the deadline of the transaction.

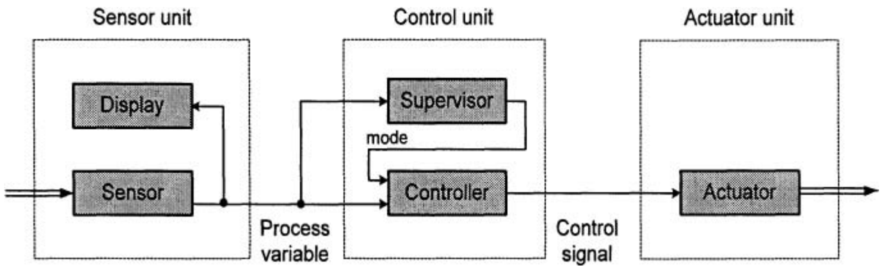


Figure 1. Actor diagram of a distributed embedded system.

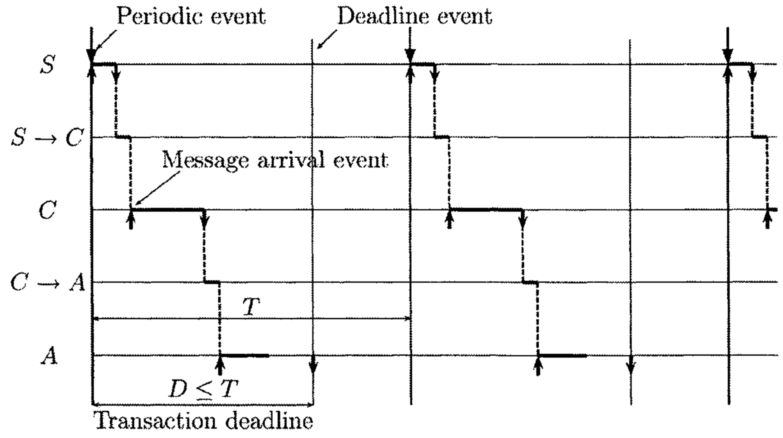


Figure 2. Periodic phase-aligned transaction with precisely specified I/O actions.

Accordingly, an actor is composed from input signal drivers, signal-processing block (SPB) and output signal drivers. An input driver transforms an external signal (i.e. physical input signal or global signal) into a subset of local input signals that are sent to the SPB. The latter processes local signals supplied by one or more input drivers, and generates local output signals, which are passed on to one or more output drivers. An output driver transforms a subset of internal output signals into an external - physical output or global signal.

The signal-processing block is mapped onto a non-blocking (basic) task, which is executed in a dynamic preemptive-priority scheduling environment, whereas input and output drivers are executed atomically at precisely specified time instants. This mode of operation is known as *split-phase execution* of real-time tasks (see Fig. 3), and it follows from the adopted timed multitasking model of computation.

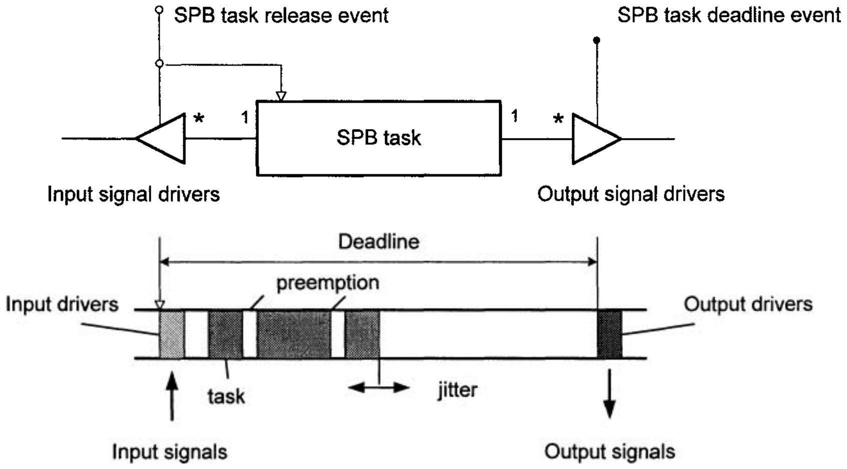


Figure 3. Split-phase execution of actor under Distributed Timed Multitasking.

In particular:

- The actor task is released by an *execution trigger*; this may be any kind of event, e.g. periodic timing event, external interrupt, message-arrival event, etc., depending on the type of transaction executed.
- Input drivers are executed when the task is released, whereas output drivers are executed when the task deadline arrives or when the task comes to an end, if no deadline has been specified.

Consequently, task jitter is effectively eliminated, provided the task is schedulable and always comes to an end before its deadline. That technique can be extended to distributed transactions as illustrated by Fig. 2, which shows a distributed phase-aligned transaction executed under Distributed Timed Multitasking. Such transactions usually suffer from input and output jitter, which is due to task release and termination jitter, as well as communication jitter. This is a serious problem, which complicates schedulability analysis and affects system operation. However, in our case I/O jitter is eliminated, as long as the transaction comes to an end before its deadline. That is because the process variable is always sampled at the start of the period, and the output signal is generated at the transaction deadline instant.

The DTM model has been used to develop a task execution model that might be implemented within a timed-multitasking kernel. In particular, such a kernel must support the split-phase execution of tasks and I/O drivers in the context of distributed transactions. This model can be represented as a *virtual node*, which resides in a physical node and is executed under the node-resident real-time kernel (see Fig. 4).

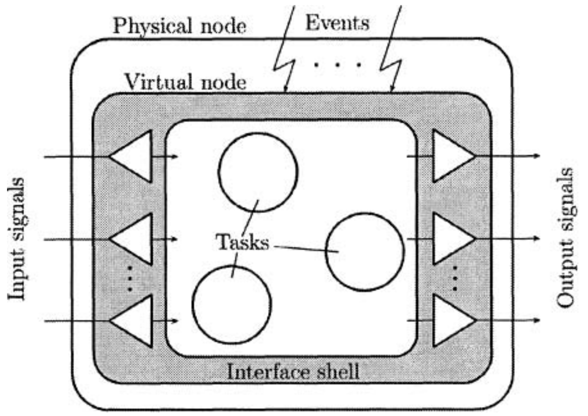


Figure 4. Task execution model: the *HARTEX*$_{TM}$ virtual node.

The virtual node contains a subset of actor tasks, which communicate with the outside world via an *interface shell* consisting of input and output signal drivers. Input drivers are used to receive input signals that may be physical or communication signals (i.e. labeled messages). These are decomposed into local variables that are processed by actor tasks. Output drivers are used to generate output signals from local variables that are computed by actor tasks. These may be either physical or communication signals (messages).

Communication signals are exchanged transparently between actors residing in the same or different virtual/physical nodes by means of kernel primitives *broadcast (SIGNAL_NAME)* and *receive (SIGNAL_NAME)*, which are invoked within output and input signal drivers, respectively[7]. The above primitives constitute the task interaction layer of a real-time communication protocol built on top of a Controller Area Network. The underlying layers of the protocol provide a deterministic priority-driven environment supporting predictable communication between remote actors. This is largely due to specific CAN features, such as priority-driven bus arbitration, content-oriented message addressing and acceptance filtering, which provide adequate hardware support for signal-based communication.

3. IMPLEMENTATION OF DISTRIBUTED TIMED MULTITASKING IN THE *HARTEX$_{TM}$* KERNEL

The task execution model has been used to derive requirements specifications for the timed multitasking version of the *HARTEX* kernel. In particular, it is obvious from the above model that the kernel has to manage several types of software entity: events, activity tasks and task I/O drivers, and it must provide support for task interaction via signal-based communication.

Accordingly, the kernel is built from dedicated components implementing subsystems such as *Event Manager, Task Manager, Task I/O Manager* and *Software Bus*. This is the minimum configuration needed in order to implement the presented task execution model. Another component that has been included in the kernel configuration is the so called *Static Time Manager*, which can be used to efficiently implement timed multitasking for multiple periodic tasks (transactions) using Boolean vector processing techniques[7].

Boolean vector processing is a common feature of all *HARTEX* subsystems. Boolean vectors (bit-strings) have been substituted for conventional linked-list queues throughout the kernel, resulting in considerable reduction of system overhead and constant execution time of kernel functions, independent of the number of tasks involved[6,7]. In particular, the kernel operates with the following vectors:

The *Event Vector (EV)* is used to register primary events generated by interrupt service routines, which are subsequently processed by the Event Manager. Events are recognized only if they are enabled by the corresponding bit-mask stored in the *Enabled Events Vector (EEV)*. The processing of events results in the generation of task execution requests, which is accomplished by setting the corresponding task bits in the *Active Tasks Vector (ATV)*. Released tasks are then executed following a task state transition diagram implemented by the Task Manager, using two additional vectors - the *Enabled Tasks Vector (ETV)* and the *Preempted Tasks Vector (PTV)*[7]. The *Input Drivers Vector (IDV)* and the *Output Drivers Vector (ODV)* are used to register I/O driver execution requests that may be generated by either the Event Manager or the Static Time Manager. In addition, the Software Bus employs Boolean vector semaphores that can be used to broadcast an event to one or more receiver tasks[6].

Distributed Timed Multitasking is an event-driven model, which can be implemented by means of standard event-processing techniques supported by the *HARTEX* Event Manager. The latter provides a unified mechanism of processing external and timing events via event counters and event control blocks (see Fig. 5).

```
Event_manager()  {
  for all enabled events registered in Event Vector  {
    reset Event Vector bit;
    decrement event_counter;
    if(event_counter != 0) return;
    else  {
      if(not_free_running) disable event_counter;
      switch(event_type) {
        case 'release_event'
          if(task_has_deadline)start deadline timer;
                        // enable timer event counter
          release task triggered by event;
                             // register task in ATV
          register task input drivers in IDV;
          release Task I/O Manager;
          break;
          .........
        case 'deadline_event'
          generate a deadline violation vector (DVV) and
          disable task if it has not finished execution;
          send  DVV  to  deadline  violation  handler  (if
          non-empty);
          register  task  output  drivers  in  ODV  if  the
          task has finished before its deadline;
          release Task I/O Manager;
          break;
      } //end switch
      event_counter = threshold; //reinitialize counter
    } //end else
  }  //end for all
  preempt();  //invoke the Task Manager;
} //end Event Manager
```

Figure 5. Event Manager operation.

The event control block (ECB) specifies an event threshold, i.e. the number of primary events that must be counted before an event-related operation is executed, such as release task(s), signal task deadline violation, signal-and-release task(s) via a Boolean vector semaphore, etc., depending on event type. Event control blocks are processed by the Event Manager, which is invoked from within interrupt service routines after the primary events have been registered in the EV. Events are processed in priority order, which is indicated by the bit positions of event flags within the Event Vector.

The Task I/O Manager is implemented as a system task whose priority is higher than the priority of actor tasks. When invoked, it executes atomically the output drivers of tasks registered in ODV and/or input drivers of tasks registered in IDV before exiting, whereupon the Task Manager (re)activates the highest priority actor task registered in the ATV.

The presented technique makes it possible to execute task output drivers at deadline instants. However, if no deadline is specified the drivers must be invoked at the end of task execution. This option is implemented in the primitive *task_exit()*, which is invoked when the task comes to an end.

```
task_exit()   {
  reset task bit in ATV;//remove task from system queue
  if(! task_has_deadline) {
    register task output drivers in ODV;
    release Task I/O Manager; }
}
```

The techniques discussed above outline a general solution that can be used to implement timed multitasking in a single-computer environment. It can be extended to distributed systems if local clocks are properly synchronized, e.g. by using a sync message generated by a synchronization master node or globally synchronized clocks.

The presented solution provides for a unified event-driven approach towards the implementation of timed multitasking. Unfortunately, its use may result in increased overhead when the kernel has to execute multiple periodic tasks, since each task needs at least two timers, i.e. a period and deadline timer. This problem has been solved by developing another type of kernel component – the Static Time Manager (alternatively called the *Drum Sequencer*). It provides an operational environment for multiple periodic tasks executing under Distributed Timed Multitasking, using only one timer per network node and sophisticated Boolean vector processing techniques[7].

4. CONCLUSION

The paper has presented a new operational model for distributed embedded systems, which has been developed in the context of the *COMDES-II* framework. The latter specifies a software architecture for distributed control systems, whereby the system is conceived as a composition of embedded actors that communicate transparently by exchanging labeled messages (signals), independent of their allocation onto network nodes.

Signal-based communication is combined with timed multitasking, whereby actors are mapped onto real-time tasks and signals are exchanged by means of signal drivers. The latter invoke kernel primitives that constitute the task interaction layer of a real-time communication protocol, which is used to transparently exchange signals between local and/or remote tasks. Signal drivers are triggered at precisely specified time instants, e.g. task release and deadline instants, within a distributed transaction involving two or more communicating actors.

The combination of actors, signal-based communication and timed multitasking has resulted in a new operational model – *Distributed Timed Multitasking (DTM)*. The latter has been used to derive a task execution model and requirements specifications for the timed-multitasking version of the *HARTEX* kernel.

Distributed Timed Multitasking presents a unique combination of features, such as flexible task scheduling, transparent communication and predictable behaviour, which can be used to engineer open, and at the same time - safe and predictable embedded systems.

REFERENCES

[1] Liu, J.W.S.: Real-Time Systems. Prentice Hall (2000)

[2] Kopetz, H.: Real-Time Systems. Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers (1997)

[3] Liu, J., Lee, E.A.: Timed Multitasking for Real-Time Embedded Software. IEEE Control Systems Magazine: Advances in Software Enabled Control, February (2003), pp. 65-75

[4] Henzinger, T.A., Horowitz, B., and Kirsch, C.M.: Embedded Control Systems Development with Giotto. Proc. of the Conference on Languages, Compilers and Tools for Embedded Systems LCTES'01, Salt Lake City, USA (2001)

[5] Angelov, C., and Sierszecki, K.: A Software Framework for Component-Based Embedded Applications. Proc. of the Asia-Pacific Software Engineering Conference APSEC'2004, Busan, Korea, Dec. 2004, pp. 655-662

[6] Angelov, C., Ivanov, I., and Burns, A.: HARTEX - a Safe Real-Time Kernel for Distributed Computer Control Systems. Software: Practice and Experience, vol. 32, N3, (2002), pp. 209-232

[7] Angelov C., Berthing J., and Sierszecki K.: A Jitter-Free Operational Environment for Dependable Embedded Systems, in: A. Rettberg et al. (Eds.): From Specification to Embedded Systems Application. Springer (2005), pp. 277-288

ITERATIVE REFINEMENT APPROACH FOR QOS-AWARE SERVICE CONFIGURATION

Luís Nogueira, Luís Miguel Pinho
IPP Hurray Research Group, Polythecnic Institute of Porto, Portugal
luis@dei.isep.ipp.pt, lpinho@dei.isep.ipp.pt

Abstract In heterogeneous environments, diversity of resources among the devices may affect their ability to perform services with specific QoS constraints, and drive peers to group themselves in a coalition for cooperative service execution. The dynamic selection of peers should be influenced by user's QoS requirements as well as local computation availability, tailoring provided service to user's specific needs. However, complex dynamic real-time scenarios may prevent the possibility of computing optimal service configurations before execution. An iterative refinement approach with the ability to trade off deliberation time for the quality of the solution is proposed. We state the importance of quickly finding a good initial solution and propose heuristic evaluation functions that optimise the rate at which the quality of the current solution improves as the algorithms have more time to run.

1. INTRODUCTION

The amount of data produced by a variety of data sources and sent to end systems to further processing is growing significantly. There are several examples of sensors being installed to continuously measure environmental properties and disseminate data streams. These applications are pushing the limits of traditional data processing infrastructures [15]. The challenges become even more critical when coordinated content analysis of stream data from multiple sources is necessary [3]. This calls for an architecture that supports the distribution of the processing task to different nodes in order to be able to cope with increasing resource requirements.

At the same time, quality-aware processing of those data streams is increasingly being considered an important user demand, receiving wide attention in real-time research. Unfortunately, in most systems, users do not have any real influence over the QoS they can obtain, since service

Please use the following format when citing this chapter:

Noguera, L., Pinho, L.M., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 155–164.

characteristics are fixed when the systems are initiated. Furthermore, users can differ enormously in their service requirements as well as applications in the resources which need to be available to perform a service with a specific level of quality. Therefore, there is an increasing need for customisable services that can be tailored to user's specific requirements [14]. A QoS negotiation model is the key to build predictable, gracefully degradable services for real-time applications [1].

This paper addresses the growing demand on resources and performance requirements by allowing resource constrained devices to cooperate with more powerful or less congestioned neighbour nodes to meet resource allocation requests and handle stringent constraints, opportunistically taking advantage of global network resources and processing power.

We are primarily interested in dynamic scenarios where new tasks can appear while others are being executed, the processing of those tasks has associated real-time execution constraints, and service execution can be performed by a coalition of neighbour nodes. Such scenarios may prevent the possibility of computing optimal resource allocations before execution. Instead, nodes should negotiate partial, good-enough service proposals that can be latter refined if time permits. Moreover, taking the cost of decision-making into account is not an easy task, since the "optimal" level of deliberation varies from situation to situation. It is therefore beneficial to build systems that can trade off computational resources for quality of results.

We propose and evaluate new anytime algorithms for coalition formation and service proposal formulation with the ability to trade off deliberation time by the quality of the solutions. The proposed algorithms can be interrupted at any time and provide a solution and a measure of its quality. This quality is expected to improve as the run time of the algorithms increase. A higher adaptation to changing conditions in dynamic environments is thus introduced by allowing flexibility in the execution times of the algorithms.

The conformity of both algorithms with the desired properties of anytime algorithms and the validation through extensive simulations of the design decisions of our approach is detailed in [11]. The achieved results emphasise our believe that use of anytime algorithms for coalition formation and service proposal formulation significantly improve the ability of our framework to adapt to changes in a dynamic heterogeneous environments.

2. RELATED WORK

The quality of the outputs may depend on the available amount of resources. For example, in multimedia applications, higher network and CPU bandwidth produces better audio and video quality, at higher resolutions and/or higher frame rates. As such, researchers have been proposing and optimising several techniques for resource management in resource constrained devices.

Computation offloading to a remote machine has been explored to achieve power and performance gains [6, 7, 13]. The authors conclude that the efficiency of an application execution can be improved by careful partitioning the workload between a device and a fixed neighbour. Optimal application partitioning depends on the trade off between the computation workload and the communication cost. However, a method for finding and selecting the best subset of service providers among the set of neighbour nodes is still missing. Also, to the best of our knowledge, previous work in offloading do not take into consideration QoS constraints imposed by users in their service requests. Since different users can access multiple devices at the same time, supporting users' QoS preferences in service execution is a key issue.

Work on applications' decomposition into tasks has, for example, been reported in [3, 9, 16]. Interpretation of QoS constraints and consequent mapping on resource parameters as been described, for example, in [12, 4, 5]. We focus on proposing a generic model that enables a distributed QoS-aware service allocation, with the ability to adapt to dynamically changing system conditions.

Our preliminary work [10] proposes a system where heterogeneous nodes organise themselves into a coalition for cooperative service execution, dictated by computational capabilities. However, the assumption that the algorithms can have all the time they need to compute their outputs was used.

The work on anytime algorithms [2, 17] recognises that the computation time needed to compute optimal solutions will typically reduce the overall utility of the system. An anytime algorithm is an iterative refinement algorithm that can be interrupted and asked to provide an answer at any time. It is expected that the quality of the answer will increase (up to some maximum quality) as the anytime algorithm is given increasing time to run, offering a trade off between the quality of the results and computational requirements. Associated with an anytime algorithm is a performance profile, a function that maps the time given to an anytime algorithm (and in some cases input quality) to the quality of the solution produced by that algorithm.

3. TIME-BOUNDED COALITION

A coalition formation process should enable the selection of individual nodes that, based on their own resources and availability, will constitute the best group to satisfy user's QoS requirements Q associated with service S. The anytime approach proposed here extends the algorithm introduced in [10] by allowing it to return many possible approximate answers and a measure of their qualities for a given input of service proposals to evaluate. Those service proposals are sent by neighbour nodes, in reply to a cooperative service execution request with associated user's QoS constraints that this node is not able to fulfil by itself.

We consider a user's service request to be formulated through the relative decreasing importance of a set of QoS dimensions [10]. Furthermore, for each dimension a relative decreasing importance order of attributes, and possible values for each attribute, is also specified. As a result, the user is able to express acceptable compromises in QoS and their relative importance.

All admissible proposals are evaluated according to user's QoS preferences, measuring the distance between requested and proposed values [10]. The best proposal is the one that contains the attributes' values more closely related to user's preferences, in all QoS dimensions.

Time-bounded coalition formation implies trying to quickly find a good initial solution and gradually improve that solution if time permits. The selection of the next candidate proposal to be evaluated from the set of available proposals should be done in a order that maximises the expected improvement in solution quality. It is necessary to make a trade off between search effort and solution quality explicitly in the heuristic selection of the next candidate proposal so that we can optimise search effort directly, rather than relying in arbitrary proposal evaluation. As such, for each task T_i we select the next candidate proposal P_{ki} from the set of received proposals P_i to be evaluated for task T_i, as *the one sent by node N_k that has the greatest local reward R_k.*

The local reward R_k is an indicator of node's local QoS optimisation, according to the set of tasks being locally executed and their QoS constraints. We claim that the local reward achieved by a node should be used to guide the coalition formation process, since nodes with higher local reward have a higher probability to be offering service closer to user's request under negotiation.

The anytime coalition formation algorithm, seeking distributed QoS optimisation, is described in Algorithm 1. Since the formation of a coalition is aimed at maximising the benefits associated to a cooperative service execution, the quality of each generated coalition can be measured by using the evaluation values of the best proposals for each service's task.

$$Q_{coalition} = \left\lfloor \frac{|coalition|}{|S|} \right\rfloor * \sum_{i=1}^{|coalition|} \frac{1 - Best_{P_i}}{|coalition|} \qquad (1)$$

For an empty set of proposals the quality of the coalition is zero. Note that the quality of the coalition is also zero, if there are not any proposals for one or more tasks T_i of service S.

Algorithm 1 Iterative coalition formation

```
for each $T_i \in S$ do
  Select next candidate proposal $P_{Ki}$, maximising local reward
  $E_{P_{ki}} = evaluate(P_{ki})$
  if $E_{P_{ki}} - Best_{P_i} > \alpha$ then
    $Best_{P_i} = E_{P_{ki}}$
    Update coalition with $N_k$ for task $T_i$
  else if $0 < E_{P_{ki}} - Best_{P_i} < \alpha$ and $R_{P_{ki}} > R_{Best_{P_i}}$ then
    $Best_{P_i} = E_{P_{ki}}$
    Update coalition with $N_k$ for task $T_i$
  end if
end for
```

The algorithm continues, if time permits, to evaluate received service proposals trying to improve the quality of the current solution. It is possible that another node, while achieving a lower local reward, proposes a better service for the specific request under negotiation. The service proposal formulation algorithm, described in the next section, always suggests the best solution for a particular user, even if it has to degrade the provided level of service of previous existing tasks. It is the responsibility of the coalition formation algorithm to select between similar proposals (whose evaluation values differ in less than α) those nodes that achieve higher local rewards, promoting load balancing.

The algorithm terminates when it finds that the quality of a coalition cannot be further improved or the local reward of each node that belongs to that coalition is maximum.

4. TIME-BOUNDED SERVICE PROPOSAL

Requests for cooperative service execution arrive dynamically at any node. Each user's request is formulated as a set of acceptable multi-dimensional QoS levels in decreasing preference order. To guarantee the request locally, the node executes a local QoS optimisation algorithm described in Algorithm 2. Conventional admission control schemes either guarantee or reject each request, implying that future requests may be rejected because resources have already been committed to previous requests. We use a QoS negotiation mechanism that, in cases of overload, or violation of pre-run-time assumptions guarantees graceful degradation. In our model, guaranteeing a user's request is the certification that the service will be provided in *one* of the QoS levels expressed in the request.

A service configuration proposed for a specific task T_i will achieve a reward r_i determined by the proximity of the proposal with respect to the QoS preferences specified in user's service request. Its value is maximum if the task is being served at the highest requested level in all QoS dimensions. Otherwise, it is affected by a penalty factor that increases with the distance for user's preferred values [10].

As introduced in the previous section, each node sends along with the service proposal a measure of global satisfaction resulting from its proposal acceptance. The local reward R_j expresses a degree of global satisfaction for all the users that have tasks being executed by a particular node N_j, with specific QoS levels. For a node N_j, the local reward $R_j = (\sum_{i=1}^{n} r_{T_i})/n$ achieved by a set of tasks is determined combining the reward of each task being locally executed as a measure of global satisfaction of the proposed solution.

Unless all tasks are executed at their highest QoS level, there is a difference between the actual local reward achieved by the currently selected QoS levels and the maximum possible local reward that would be achieved if all local tasks were executed at their highest requested QoS level. This difference can be caused by either resource limitations, which is unavoidable, or poor load balancing, which can be improved by sending actual local rewards in service proposals, and selecting, for proposals with similar evaluation values, those nodes that achieve higher local rewards. Selecting the node with higher local reward for similar service proposals, not only maximises service satisfaction for a particular user, but also maximises global system's utility, since a higher local reward clearly indicates that the previous set of tasks being locally executed had to suffer less QoS degradation in order to accommodate the new task.

In [8], it was demonstrated that the QoS optimisation problem involving multiple resources and multiple QoS dimensions is NP-hard. An optimal solution based on dynamic programming and an approximation scheme based on a local search technique was presented. However, the computation time needed to find an optimal solution can reduce the overall utility of the system. In addition, the deliberation cost is dependent on local resources' availability and user's QoS constraints. Therefore, it is beneficial to build systems that can trade the quality of results against the cost of computation [17].

The proposed anytime algorithm considers two different scenarios when formulating a service proposal. The first one involves guaranteeing the new task without changing the level of service of previously guaranteed tasks. The second one, due to node's overload, demands service degradation in existing tasks in order to accommodate the new requesting task. Our local QoS optimisation (re)computes the set of QoS levels for all local tasks, including the new requested one. Offering QoS degradation as an alternative to task rejection has been proved to achieve higher perceived utility [1].

The algorithm iteratively work on the problem of finding a feasible service configuration that maximises user's satisfaction and produces results that improve in quality over time. Equation 2 shows how the quality of each generated feasible configuration Q_{conf} is calculated by considering the reward achieved by the service proposal configuration for the new arriving task r_{T_a}, the impact on the provided QoS of previous existing tasks and the value of the previous generated feasible configuration Q'_{conf}. Initially, Q'_{conf} is equal to zero.

$$Q_{conf} = \left(r_{T_a} * \frac{\sum_{i=0}^{n} r_{T_i}}{n} \right)^{(1-Q'_{conf})} \tag{2}$$

When a new service request arrives, the algorithm starts by maintaining the QoS levels of previously guaranteed tasks and by selecting the worst requested QoS level, for all dimensions, for the new arrived task. As such, the reward of the initial service configuration for the new task is low (the exact value is determined by the penalty factors used in a particular system), affecting node's local reward. On the other hand, the impact of this new task on the provided level of previously existing tasks is inexistent. Also, this initial solution is the service configuration that has a higher probability of being feasible, considering the new arrived task. The algorithm continues to improve the quality of the initial solution, conducting the search for a better feasible solution in a way that maximises the expected improvement in solution's quality. When there are enough resources the algorithm selects, from the set of possible upgrades, the next configuration that*maximises the reward achieved by the new arrived task.* When QoS degradation is needed, it selects the configuration that *minimises the decrease in local reward.*

At each iteration the algorithm produces a new service configuration that may not be feasible due to local resources availability and user's QoS constraints expressed in request. Since a service proposal can only be considered useful within a feasible set of configurations, the algorithm, if interrupted, always returns the best found feasible solution. However, each intermediate configuration, even if not feasible, is used to calculate the next solution, minimising search effort.

When the new task can be accommodated without degrading the QoS of previously existing tasks, the algorithm incrementally selects the configuration that maximises the increase in obtained reward, according to user's QoS preferences expressed in his request. When QoS degradation is needed, the algorithm incrementally selects the configuration that minimises the decrease in obtained reward of all tasks.

Algorithm 2 Iterative service proposal formulation

```
Each task $T_i$ being locally executed has associated a set of user QoS constraints $Q^i$.
Each $Q^i_{kj} = \{Q^i_{kj}[0], \dots, Q^i_{kj}[n]\}$ is a finite set of $n$ quality choices for the $j^{th}$ attribute of the $k^{th}$ QoS dimension associated with task $T_i$, expressed in decreasing order of preference.
  Step 1: Improve QoS level of the new arrived task $T_a$
  Select the worst requested QoS level, in all $j$ attributes of all $k$ dimensions, $Q^a_{kj}[n]$, for task $T_a$.
  Maintain level of service for all previously guaranteed tasks.
  while the new set of tasks is feasible do
    for each $k$ QoS dimension in $T_a$ receiving service at $Q^a_{kj}[m] > Q_{kj}[0]$ do
      Determine the utility increase by upgrading attribute $j$ to $m-1$
      Find maximum increase and upgrade attribute to the $m-1$'s level
    end for
  end while
  Step 2: Find global minimal degradation to accommodate $T_a$
  Select for all $k$ dimensions of task $T_a$ the final result of Step 1, $Q^a_{kj}[m]$
  while the new set of tasks is not feasible do
    for each task $T_i$ receiving service at $Q^i_{kj}[m] > Q^i_{kj}[n]$ do
      Determine the utility decrease by degrading attribute $j$ to $m+1$
      Find task $T_{min}$ whose reward decrease is minimum and degrade attribute $j$ to the $m+1$'s level
    end for
  end while
```

The algorithm terminates when the time for the reception of proposals has expired (this time is sent in user's request), when it finds a set of feasible QoS levels and the quality of the solution can not be further improved, or when it finds that, even at the lowest QoS level for each task, the new set is not feasible. In this case the new arrived task is rejected. When it is not possible to find a feasible solution to include the new task within available time, the node continues to serve existing tasks at their current QoS levels and does not send any service proposal to the requesting node.

The algorithm always improves or maintains the quality of the solution as it has more time to run. This is done by keeping the best feasible solution so far, if the result of each iteration is not always proposing a feasible set of tasks.

5. CONCLUSIONS AND FUTURE WORK

Resource constrained devices may need to cooperate with neighbour nodes in order to fulfil complex services, with specific user's QoS constraints. Given a set of tasks to be executed, we consider situations where a service is assigned to a group of nodes for cooperative execution in a dynamic heterogeneous environment.

This paper proposes algorithms for coalition formation and service proposal formulation with the ability to trade off deliberation time for quality of results. At each iteration, the search of a better solution is guided by heuristic evaluation functions that optimise the rate at which the quality of the current solution improves overtime. These capabilities are essential for successful operation in dynamic real-time environments, as it may not be feasible to compute an optimal answer before providing a solution for a cooperative service execution.

The proposed anytime algorithms significantly improve the ability of our framework to adapt to changes in dynamic environments by allowing flexibility in the execution times of the algorithms. A complete integration in the existing framework is under development.

ACKNOWLEDGMENTS

This work was supported by FCT, through the CISTER Research Unit (FCT UI 608) and the Reflect project (POSI/EIA/60797/2004).

REFERENCES

[1] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin. Qos negotiation in real-time systems and its application to automated flight control. *IEEE Transactions on Computers, Best of RTAS '97 Special Issue*, 49(11):1170–1183, November 2000.

[2] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 49–54, 1988.

[3] Viktor S. Wold Eide, Frank Eliassen, Ole-Christoffer Granmo, and Olav Lysne. Supporting timeliness and accuracy in distributed real-time content-based video analysis. In *Proceedings of the 11th ACM international conference on Multimedia*, pages 21–32. ACM Press, 2003.

[4] K. Fukuda, N. Wakamiya, M. Murata, and H. Miyahara. Qos mapping between user's preference and bandwidth control for video transport. In *Proceedings of the 5th International Workshop on Quality of Service*, pages 291–302, New York,USA, 1997.

[5] Vera Goebel and Thomas Plagemann. Mapping user-level qos to system-level qos and resources in a distributed lecture-on-demand system. In IEEE Computer Society, editor, *Proceedings of The 7th IEEE Workshop on Future Trends of Distributed Computing Systems*, page 197, 199.

[6] Xiaohui Gu, Alan Messer, Ira Greenberg, Dejan Milojicic, and Klara Nahrstedt. Adaptive offloading for pervasive computing. *IEEE Pervasive Computing Magazine*, 3(3):66–73, 2004.

[7] Ulrich Kermer, Jamey Hicks, and James Rehg. A compilation framework for power and energy management on mobile computers. In *14th International Workshop on Parallel Computing*, pages 115–131, 2001.

[8] Chen Lee, John Lehoczky, Dan Siewiorek, Ragunathan Rajkumar, and Jef Hansen. A scalable solution to the multi-resource qos problem. In *20th IEEE Real-Time Systems Symposium*, pages 315–326, 1999.

[9] L. Marcenaro, F. Oberti, G. L. Foresti, and C. S. Regazzoni. Distributed architectures and logical-task decomposition in multimedia surveillance systems. *Proceedings of the IEEE*, 89(10):1419–1440, October 2001.

[10] Luís Nogueira and Luís Miguel Pinho. Dynamic qos-aware coalition formation. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, Denver, Colorado, April 2005.

[11] Luís Nogueira and Luís Miguel Pinho. Time-bounded distributed qos-aware service configuration in heterogeneous cooperative environments. Technical report, IPP Hurray Research Group. Available at http://hurray.isep.ipp.pt/, January 2006.

[12] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for qos management. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, page 298. IEEE Computer Society, 1997.

[13] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, 2(1):19–26, 1998.

[14] Sven Schmidt, Thomas Legler, Daniel Schaller, and Wolfgang Lehner. Real-time scheduling for data stream management systems. In *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 167–176, 2005.

[15] Michael Stonebraker, Ugur Cetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Record*, 34(4):42–47, 2005.

[16] Cheng Wang and Zhiyuan Li. Parametric analysis for adaptive computation offloading. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation*, pages 119–130. ACM Press, 2004.

[17] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *Artificial Inteligence Magazine*, 17(3):73–83, 1996.

A FAST AND EFFICIENT ISOMORPHIC TASK ALLOCATION SCHEME FOR K-ARY N-CUBE SYSTEMS

D.Doreen Hephzibah Miriam [1], T.Srinivasan [2]
[1] Postgraduate Student, Sri Venkateswara College Of Engineering
[2]Assistant Professor, Department of Computer Science and Engineering
Sri Venkateswara College Of Engineering, Sriperumbudur, India– 602 105.
doreenhm@gmail.com, tsrini@svce.ac.in

Abstract: A good task allocation algorithm should find available processors for incoming jobs, if they exist, with minimum overhead. Due to its topological generality and flexibility the k-ary n-cube architecture has been chosen for the task allocation problem. We propose a fast and efficient isomorphic processor allocation scheme for k-ary n-cube systems by using isomorphic partitioning where the processor space is partitioned into higher dimensional isomorphic subcube and by using Subcube recognition ability algorithm (SRA) which uses simple coordinate calculation and spatial subtraction. Thus the proposed scheme seeks to reduce the search space drastically, and hence can locate a free subcube very quickly providing scalable, faster, processor allocation, complete recognition ability with minimal overhead and minimizes the fragmentation

Keywords: Full subcube recognition, k-ary n-cube systems, isomorphic partitioning, processor allocation.

1. INTRODUCTION

As the incoming tasks to a system can involve different topologies, the k-ary n-cube is desirable to accept and execute topologically different tasks. An exact number of processors with a particular topology must be allocated to an incoming task. That is, processor allocation in a k-ary n-cube system concerns not only about the dimension (m, $0 \leq m \leq n$) but also about the amount of processors in each dimension (r, $2 \leq r \leq k$), apparently more complicated than that in a binary system.

Please use the following format when citing this chapter:

Miriam, D.D.H., Srinivasan, T., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 165–174.

The processor allocation problem is much more challenging for k-ary n-cube networks than hyper cubes or meshes because reducing fragmentation within the system involves recognizing both the dimension of the network (as in hypercubes) and the number of processors in each dimension(as in meshes). However, existing processor allocation strategies for the k-ary n-cube [6] - [9] system either recognize only the dimensionality of the subcubes or allow arbitrary partition sizes at the cost of complex search operations.A critical attribute of a processor allocation algorithm is its ability to find available subcubes for incoming requests is called the *subcube recognition ability*. An allocation algorithm is said to have complete subcube recognition ability when it always find a free subcube for an incoming job if one is available.

The rest of the paper is organized as follows: Section 2 presents the pertinent preliminaries. Section 3 presents earlier allocation algorithms in Mesh systems, Hypercube systems, k-ary n-cube systems. Section 4 presents the proposed isomorphic allocation strategy. Section 5 reports the experimental results. Finally, concluding remarks are discussed in Section 6.

2. PRELIMINARIES

2.1 NOMENCLATURE

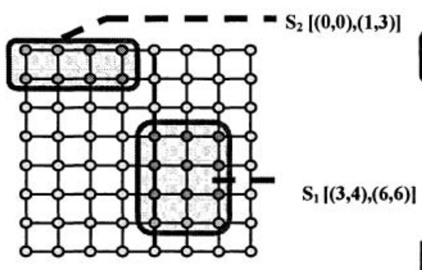
A k-ary n-cube denoted by Q_k^n, has k^n nodes each of which is identified by n tuple $(a_{n-1},\ldots,a_1,a_0)$ of radix k. where a_i represents the node's position in the ith direction.

Definition 1: A (two-dimensional) **subcube** S(p,q) in the k-ary 2 -cube M(w,h) such that $1 \leq p \leq w$ and $1 \leq q \leq h$.A subcube is identified by its base and end and is denoted by S[base, end]. Figure 1 depicts a 8 ary 2 cube systems, two busy subcube are indicated in green circles S_1 [(3,4),(6,6)] and S_2 [(0,0),(1,3)].

Definition 2: The **coverage** of a busy subcube β with respect to a job J, denoted as $\xi_{\beta,J}$ is a set of processors such that the use of any node in $\xi_{\beta,J}$ as the base of a free subcube for the allocation of J will make the job J to be overlapped with β.The coverage set with respect to J denoted C_J is the set of the coverages of all the busy subcubes.

Definition 3: The **reject area** with respect to a job J, R_J is a set of processor such that the use of any node in R_J as the base of a free subcube for the allocation of J will make the job J cross the boundary of the k ary n-cube systems.

Definition 4: The **base block** with respect to a job J is a subcube whose nodes can be used as the bases of free subcubes to accommodate the job J. The base set with respect to a job J,B_J is a set of disjoint base blocks with respect to J. The coverage of two allocated subcubes, the reject area are shown in Figure 2.



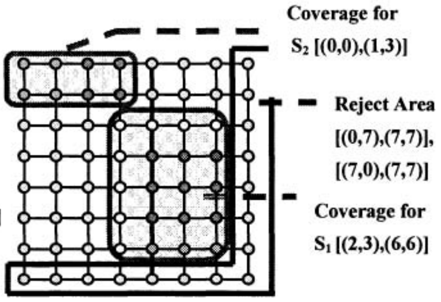

Coverage for S₂ [(0,0),(1,3)]
Reject Area [(0,7),(7,7)], [(7,0),(7,7)]
Coverage for S₁ [(2,3),(6,6)]

Fig 1: A 8-ary 2- cube system with allocated sub cube

Fig. 2: Coverage of two allocated subcubes, the reject area of 8- ary 2 –cube systems

3. PREVIOUS WORKS

3.1 MESH SYSTEMS

3.1.1 FIRST – FIT (FF)/ BEST – FIT (BF):

These were proposed in [1] to improve the FS Strategy. It maintains a busy array representing the allocation status of the mesh The BF is identical to FF except that BF selects the base in such a way that it has maximum number of busy neighbors. This scheme does not have recognition complete algorithm and excessive runtime overhead due to the manipulation of the array.

3.1.2 FREE LIST:[FL]

The key idea of this scheme proposed in [2] is to maintain a FL list of free sub meshes in the system. This scheme is that it is complicated to update the free list when a sub mesh is released in order to maintain a list of largest possible disjoint free sub mesh and recognition completeness is not achieved.

3.1.3 QUICK ALLOCATION

The basic idea of QA proposed in [3] provides complete sub mesh recognition ability with minimal overhead. For each row it defines covered segment. Column wise scan is avoided. Problems in Quick Allocation are the construction of Covered segment requires a series of row wise scan operation which can be a performance bottleneck.

3.2 K –ary n- cube systems

3.2.1 SLICE PARTITIONING:

Slice partitioning scheme proposed in [4],[5],[6] are shown in Figure 3(a).Jobs requesting different sizes are allocated to one or more partitions of base k and the remaining nodes will be wasted results in internal Fragmentation.

3.2.1.1 EXTENDED BUDDY [EB] AND EXTENDED GRAY CODE [EGC]

EB and EGC as proposed in [4], [5] in which higher dimensions partitioned into lower dimensional subcube. Links as well as the processors are underutilized and have longer internodes distance.

3.2.1.2 K-ARY PARTNER AND MULTIPLE GC

K-ary Partner and Multiple GC as proposed in [6], [5] enhances Sub cube recognition ability over EB and EGC. They utilize the fragmented nodes to form a slice along the other dimension. All these strategies limit the job size to be base k.

3.2.2 JOB PARTITIONING

Job partitioning as proposed in [7],[8] are shown in Figure 3(b).It address the internal fragmentation problem by allowing arbitrary partition sizes rather than restricting them to base – k. The allocation algorithm searches the processor space to find an available subcube for the job by sliding a window frame.

3.2.2.1 EXTENDED FREE LIST (EFL) AND EXTENDED TREE COLLAPSING (ETC)

EFL and ETC as proposed in [7],[8].These algorithm improves subcube recognition ability compacted to Sniffing Strategy. For Allocating 4 ary 2 cube job request in an 8 ary cube includes the cases along the other dimensions. Here, the total number of window positioning is n times of the sniffing strategies.

Cube Systems

4. PROPOSED WORK

4.1 THE MAIN APPROACH

The processor allocation algorithm proposed in this paper has complete subcube recognition. This scheme achieves recognition completeness by isomorphically partitioning the k –ary n – cube Systems and manipulating the orientation of the subcube request using SRA algorithm which even reduces both internal and external fragmentation.

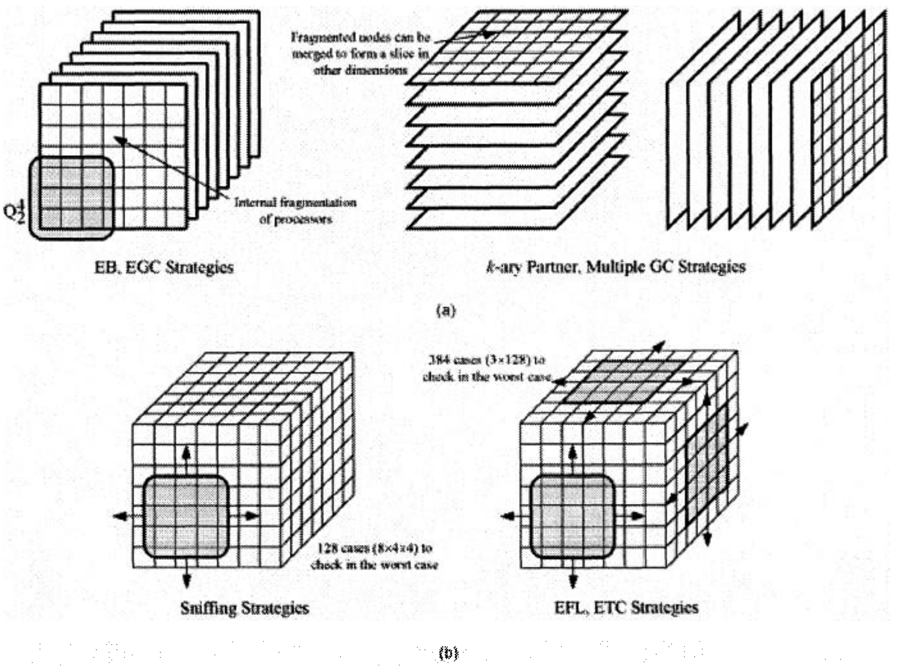

Fragmented nodes can be merged to form a slice in other dimensions
Internal fragmentation of processors
Q_2^4
EB, EGC Strategies
k-ary Partner, Multiple GC Strategies
(a)
384 cases (3×128) to check in the worst case
128 cases (8×4×4) to check in the worst case
Sniffing Strategies
EFL, ETC Strategies
(b)

Fig 3: Slice and job-based allocation strategies on an 8-ary 3-cube system. (a) With slice partitioning. (b) With job-based partitioning

4.2 ISOMORPHIC ALLOCATION STRATEGY

We now introduce the isomorphic allocation strategy for $Q^k{}_n$. The basic idea is Isomorphic partitioning recursively partitions a k-ary n-cube into 2^n number of $k/2^i$ ary n-cubes where i is the partition step. Processor space is partitioned into higher dimensional isomorphic subcubes and keeping the same order of dimension. Graphical representation of isomorphic partitioning is shown in Figure 4. The basic allocation strategy in produces isomorphic subcube partitions and the job requests to be isomorphic ($Q^a{}_n$). Isomorphic partitioning improves Subcube recognition capability, Fragmentation reduction, Complexity compared to other methods.

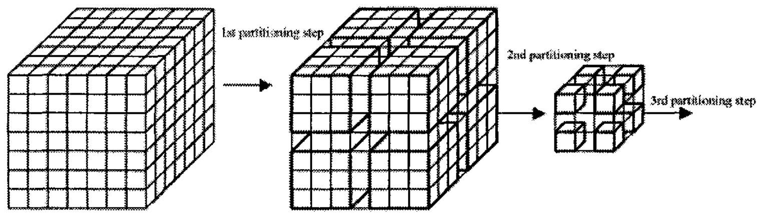


Fig 4: Isomorphic partitioning of an 8-ary 3-cube (8 x 8 x 8).

The resulted partitioned subcubes are said to be "isomorphic" (i-e) n-cube thus they retain properties of k-ary n-cube network Symmetry, Low node degree (2n),Low diameter (kn)

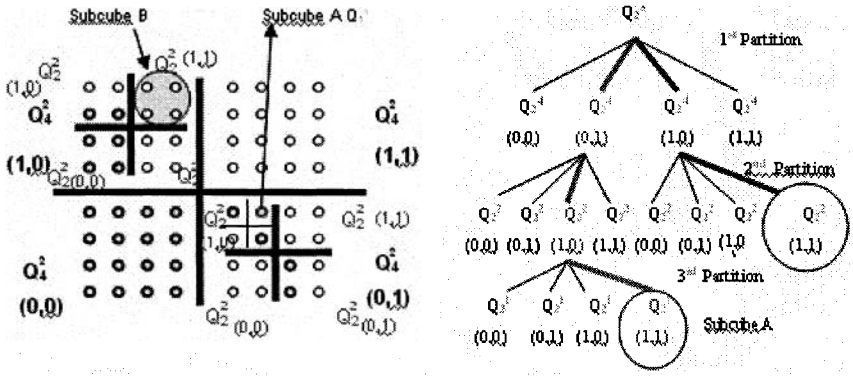


Fig 5 Representation of the isomorphic partitioning of $Q^8{}_2$. (a) Subcubes in $Q^8{}_2$.

(b) 4-ary tree representation of $Q^8{}_2$.

Isomorphic allocation strategy assigns a subcube partition to a job requesting a 2^a-ary n-cube in a 2^k-ary n-cube system, where $a \leq k$.Figure 5a shows the subcubes in an 8-ary 2-cube system (8 x 8 mesh). They can also be described by a 2^n-ary tree (4-ary tree in this example) with k partition steps (three steps in this case), as in Figure 5b.

Consider a subcube A consisting of one node whose address is (3, 5) in Figure 5. (Note that the digits are ordered from right to left, i.e., (a_1, a_0).) .A binary representation of the node is (011, 101). Since $Q^8{}_2 = H_2 \; \Theta \; H_2 \; \Theta \; H_2$, the node can alternatively be represented by ((0, 1), (1, 0),(1, 1)), where (0, 1) is the address of the node in the first sub graph H_2,(1, 0) is the one in the second H_2, and (1 ,1) is the one in the third H_2. In other words, the subcube A can be addressed by selecting (0, 1) $Q^4{}_2$ subcube after the first partition step, (1, 0). $Q^2{}_2$ subcube after the second partition step, and, finally, (1,1) $Q^1{}_2$ subcube after the third partition step. Similarly, subcube B in Figure 5 can be identified by ((1 , 0),(1 , 1))=(11,01)=(11*,01*). In general, a node in a k - ary n-cube, $Q^k{}_n$, is denoted by an n-tuple $(a_{n-1},\ldots,a_1,a_0)$ where $a_i \in \Sigma$ k. We can also denote the node in a full binary representation as

$$(a_{n-1}{}^{(1)},a_{n-1}{}^{(2)}\ldots a_{n-1}{}^{(k/2)},\ldots,a_1{}^{(1)}a_1{}^{(2)}\ldots a_1{}^{(k/2)},a_0{}^{(1)}a_0{}^{(2)}\ldots a_0{}^{(k/2)})$$

where $a_i{}^{(j)} \in \Sigma^2$ .The superscript j in each binary number denotes the partition step.

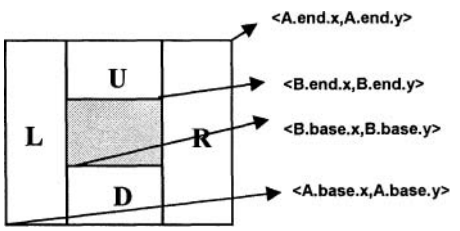
Cube Systems

4.3 SUBCUBE RECOGNITION ABILITY (SRA) ALGORITHM.

The proposed scheme achieves recognition completeness by manipulating the orientation of the subcube request. The proposed scheme quickly finds the base set through simple coordinate calculation and spatial subtraction. First using simple coordinate calculation determine the reject area (R_J) and coverage (C_J) with respect to job that is to be allocated using the busy list, job size, and the system size. The allocation scheme first determines U - R_J and inserts the set difference into the candidate list as initial candidate block. Given an R_J with a sink $<x_s ,y_s>$,the initial candidate block is a subcube I_f $[<0,0>,< x_s -1,y_s -1>$.The initial candidate block along with the coverage with which it intersects is placed in the tray. The coverage in C_J is then spatially subtracted from the initial candidate block.

A spatial subtraction between 2 subcubes is illustrated in Figure 6.The pink and the white rectangles in the figure represent the subtrahend and minuend subcubes respectively.

The newly created candidate block after a spatial subtraction replaces the old block in the candidate list. When more than one candidate block is generated after the spatial subtraction, the length of the candidate list increases and so does the search space. The key idea is to implement the candidate list as a tray. A candidate block on the top of the tray is always compared with the next coverage in the coverage set to see if they intersect with each other. When a new set of candidate blocks is generated after the spatial subtraction, these new candidate blocks are pushed on to the tray replacing the top element. Associated with each candidate block is a pointer to the next coverage that the candidate block should be compared with. When a candidate block is compared with a null pointer appears on the top of the tray, the desired base block is obtained. The node at the top – left corner of the base block is returned as the base of a free subcube.If no such candidate block is found, then the allocation fails. We call this scheme **Subcube Recognition Ability (SRA)** Algorithm.



A-B=
L:<A.base.x,A.base.y>,
<B.base.x-1, A.end.y>
R:<B.end.x+1, A.base.y>,
<A.end.x,A.end.y>
U: <B.base.x, B.end.y+1>,

Fig 6:Spatial subtraction between two 2D subcubes *(A-B)*

Subcube Recognition Ability (SRA) Algorithm
Notations

1. C_J [i] denotes the ith coverage set C_J ,where $i \geq 1$.
2. $f_{coverage}$ denotes the coverage that the candidate block *f* that the is to be compared with.
3. *traytop* represent the candidate block on top of the tray.
4. next(k) returns k + 1 if C_J [k+1]exists.Otherwise,it returns null.

Allocation

```
1.  Construct the coverage set $C_J$ w.r.t.job J
2.  Determine the initial candidate block initial.
3.  Initial $\leftarrow$ next(0).
4.  Push initial on to the tray.
5.  While the tray is not empty do
    If $traytop_{coverage}$ is null then return the base of traytop
    Else $k \leftarrow traytop_{coverage}$
    If traytop intersects with $C_J [k]$ then
       Pop up traytop from the stack.
       Spatially subtract $C_J [k]$ from traytop
       For each new candidate block f created by the spatial subtraction,
          $f_{coverage} \leftarrow$ next(k)
          push f onto the tray.
    Else $next(traytop_{coverage}) \leftarrow traytop_{coverage}$
6.  If all the orientation of J are considered then return fail.
```

The allocation of J(2,2) using the SRA algorithm is illustrated .,where a S_1 (4,3) and S_2(2,4) are allocated in 8 –ary 2 – cube systems. The coverages of two subcubes S_1 [(3,4) ,(6,6)] and S_2 [(0,0),(1,3)] are C_1 [(2,3),(6,6)] and C_2 [(0,0),(1,3)] respectively. First the Reject area R_J [(0,7),(7,7)],[(7,0),(7,7)] is calculated then sink with the <7,7> is determined. Then the initial candidate block I_1 [<0,0>,<6,6>] is obtained using the sink and pushed on to the tray with C_1 is shown in Figure 7.

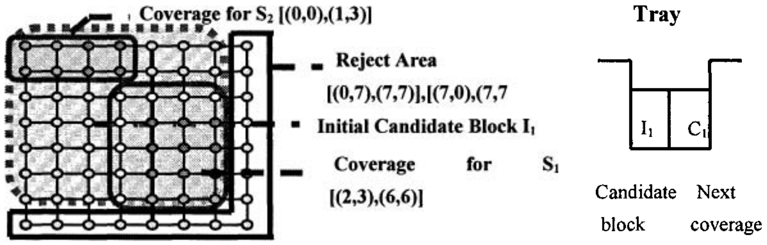


Fig 7: Initial candidate block in first step

Since I_1 and C_1 intersect a spatial subtraction is carried out and two new candidate blocks , I_2 [<0,0>,<6,2>] and I_3 [<0,3>,<1,6>] with the coverages C_2 are pushed onto the tray replacing I_1. I_2 intersects with C_2 and hence a new candidate block I_4 [<(2,0),(6,2)>] is created after spatially subtracting C_2 from B_2 .Since there are no more coverages to compare with, a null pointer is pushed with I_4 indicating that I_4 is the first base block named as B_1 .Any node in B_1 can be used as a base for J. Similarly we find B_2 [<0,4>,<1,6>] are shown in fig 8. An obvious advantage of this versatility is that we can reduce the allocation overhead significantly while preserving the behavior of the allocation scheme being evaluated.

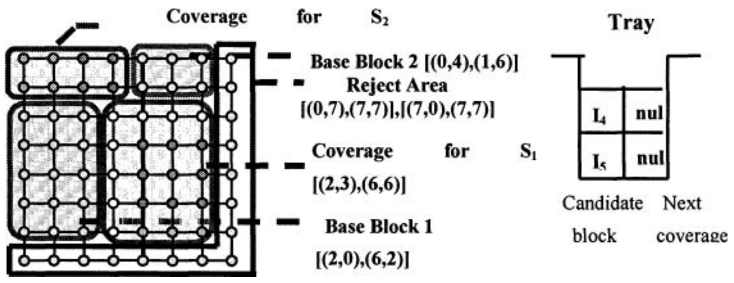


Fig 8: Allocation of 2 x 2 jobs in 8 –ary 2 – cube systems.

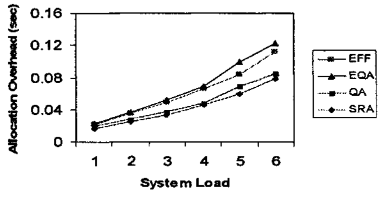
5. EXPERIMENTAL RESULTS

The experimental results of Subcube recognition ability algorithm SRA, Emulated Quick allocation EQA, and Emulated First fit EFF are compared with that of a QA for uniform job size distribution. The allocation overhead and average job response time of the four policies are plotted with respect to system loads are shown in Figure 9.

Table 1. Performance evaluation of Four allocation scheme

SYSTEM LOAD	EFF	EQA	QA	SRA
1	0.0222	0.0228	0.0200	0.0170
2	0.0353	0.0369	0.0284	0.0247
3	0.0489	0.0523	0.0378	0.0338
4	0.0656	0.0688	0.0486	0.0458
5	0.084	0.0989	0.0692	0.0592
6	0.1116	0.1221	0.0850	0.0785

Figure 10. shows the variations in mean response time w.r.t system utilization for a 4-ary 2-cube (Q^4_2).The proposed scheme outperforms.



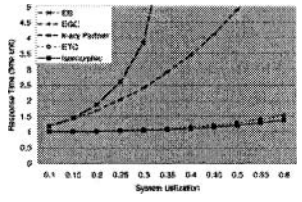


Fig 9: Performance evaluation of four allocation schemes for allocation overhead

Fig 10. Comparison of mean response time in a Q^4_2systems

6. CONCLUSIONS

This paper addresses the processor allocation problem for k-ary n-cube systems. Our proposed isomorphic partitioning efficiently divides a system into the same dimensional subcubes so that the external as well as internal fragmentation is minimized. Moreover, the resulting partitions are characterized by the same order of dimension as the whole system and, thus, retain the advantages of high order architecture. The isomorphic partitioning mechanism is a novel method for partitioning a k-ary n-cube topology. The proposed allocation algorithm Subcube Recognition ability (SRA) finds a free subcube quickly by reducing the search space drastically through the use of simple coordinate calculation and spatial subtraction. The main objective in developing the algorithm is to speed the allocation process while achieving the maximum attainable performance by guaranteeing complete subcube recognition ability. Experimental results shows that the Subcube Recognition ability algorithm is faster than any allocation scheme reported in literature by showing in the graph plotted System load vs. Allocation overhead.

REFERENCES

[1] Y. Zhu, "Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers," *J. Parallel and Distributed Computing*, vol. 16, pp. 328-337, 1992

[2] T. Liu, W. Huang, F. Lombardi and L.N. Bhuyan, "A Submesh Allocation Scheme for Mesh Connected Multiprocessor Systems," *Proc.International Conference on Parallel Processing*, vol. II, pp. 193-200, 1995.

[3] S. Yoo, H.Y. Youn and B. Shirazi , "An Efficient Task Allocation Scheme for 2D Mesh Architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 9, pp. 934-938, 1997.

[4] V. Gautam and V. Chaudhary, "Subcube Allocation Strategies in a K-Ary N-Cube," *Proc. Int'l Conf. Parallel and Distributed Computing and Systems*, pp. 141-146, 1993.

[5] G. Dommety, V. Chaudhary, and B. Sabata, "Strategies for Processor Allocation in k-Ary n-Cubes," *Proc. Int'l Conf. Parallel and Distributed Computing and Systems*, pp. 216-221, 1995.

[6] K. Windisch, V. Lo, and B. Bose, "Contiguous and Non-Contiguous Processor Allocation Algorithms for k-Ary n-Cubes," *Proc. Int'l Conf. Parallel Processing*, 1995.

[7] H.L. Chen and C.T. King, "Efficient Dynamic Processor Allocation for k-Ary n-Cube Massive Parallel Processors," *Computers Math. Applications*, pp. 59-73, 1997.

[8] P.J. Chuang and C.M. Wu, "An Efficient Recognition-Complete Processor Allocation Strategy for k-Ary n-Cube Multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, vol. 11, no. 5, pp. 485-490, May 2000.

[9] *T.Srinivasan, PJS.Srikanth, K.Praveen, L.Harish Subramaniam,*"Parallel AI Game Playing Approach for Faster Processor Allocation in Hypercube Systems using Veitch diagram", *Proc. of 11th IEEE International Conference on Parallel and Distributed Systems - ICPADS 2005*, pp.536-542, July 2005.

COMMUNICATION-AWARE COMPONENT ALLOCATION ALGORITHM FOR A HYBRID ARCHITECTURE *

Marcelo Götz, [1] Achim Rettberg [2] and Carlos Eduardo Pereira [3]

[1]*Heinz Nixdorf Institute*
University of Paderborn, Germany
mgoetz@uni-paderborn.de

[2]*C-LAB*
University of Paderborn, Germany
achim@c-lab.de

[3]*Electrical Engineering Department*
UFRGS, Brazil
cpereira@ece.ufrgs.br

Abstract High computational performance and flexibility are the requirements of nowadays embedded systems and they are increasing constantly. A single architecture must be able to support different application with dynamically requirements (changing environments). As an operating system (OS) is desired to provide support for such systems, it has to use the available resources in an optimal way (competing with the application), since an embedded system architecture usually lack in resources. Therefore, we present here our approach towards a reconfigurable RTOS that is able to distribute itself over a hybrid architecture (comprising FPGA and CPU). In this paper we will concentrate in the strategies used to allocate the OS services over a hybrid architecture, taken into consideration the used resources in the running domain (CPU or FPGA) and the communication costs.

Keywords: Reconfigurable Computing, System-on-Chip, Real Time Operating System

*This work was developed in the course of the Special Research Initiative 614 - Self-optimizing Concepts and Structures in Mechanical Engineering - University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

1. INTRODUCTION

Embedded systems are increasingly requiring more computational performance and flexibility due to the growing application complexity and changing environments where these systems are inserted. Additionally, the used execution platforms are usually lacking in resources, which make the instantiation of a complete system a challenge for a system developer.

In counterpart, the development of an execution platform for such systems may profit from modern Field Programmable Gate Arrays (FPGAs), like the Virtex-II Pro, where a CPU is hardcore embedded into the fabric. Thus, high computation performance can be achieved by implementing components in hardware. Additionally, flexibility is provided due to availability of a CPU and the partial reconfigurable capability of such devices.

In order to easier the activities of the user when developing the application, the used Operating System (OS) needs to tackle the underlying platform properly. Actually, the reasons to use an OS for a Reconfigurable Systems-on-Chip (RSoC) are not different from those for running an OS on any system ([3]).

Towards this objective, we are developing a reconfigurable RTOS that is able to distribute itself over a hybrid architecture, which comprises a CPU and a FPGA. In changing environments, where application requirements are dynamic, the RTOS needs to provide the services currently needed by the running application. However, due to the lack of resources of the underlying platforms, a complete instance of a RTOS is usually not possible. Therefore, we propose to reconfigure the RTOS at run-time over the hybrid architecture in order to better use the available resources, which is also shared by the application tasks.

Our proposal fits into the scope of an in-house ongoing research, where support for self-optimizing systems is being studied. For such systems, a self-optimizing RTOS is also required. In this paper, however, we will concentrate on the strategies used to allocate the OS services over the hybrid architecture, taken into consideration the used resources in the running domain (CPU or FPGA) and the communication costs.

The remaining of this paper is organized as follows. Section 2 describes the related work, followed by a detailed discussion of the RTOS we are using (Section 3) including a briefly description of the previous OS service allocation algorithm. In this section, we also describe the execution platform. We then present our communication-aware allocation algorithm is Section 4. There, the clustering and the allocation of the components are described. In Section 5 we present the evaluation results and we finalize in Section 6 with our conclusions.

2. RELATED WORK

The overhead added by the operating system used for embedded systems need to be carefully considered due to the usual lack of resources provided by the underlying platform. However, up to now all approaches have been based on implementations that are static in nature, see [9], [8], [7], [10] and [11]. It means that they do not change at run-time, even when application requirements may significantly change.

Reconfigurable hardware/software based architectures are very attractive for implementation of run-time reconfigurable embedded systems. The hardware/software allocation of applications tasks to dynamically reconfigurable embedded systems (by means of task migration) allows for customization of their resources during run-time to meet the demands of executing applications, as can be seen in [6].

An example of this trend is the Operating System for Reconfigurable Systems (OS4RS) ([13]). This work proposes an operating system for a heterogeneous reconfigurable System-on-Chip (SoC). It aims to provide an on-the-fly reallocation of specific application tasks, over a hybrid architecture, depending on the computational requirements and on the Quality of Service (QoS) expected from the application. Nevertheless, the RTOS itself is still static. Moreover, the reconfiguration time cost is not a big issue in the design.

Additional research efforts spent in reconfigurable computing field are only focusing on application level, leaving to the RTOS the responsibility to provide the necessary mechanisms and run-time support. The works presented in [1], [14] and [12] are some examples of RTOS services to support the (re)location, scheduling and placement of application tasks on an architecture composed by FPGA with or without an CPU. In our proposal, we expand those concepts and propose new ones to be applied in the RTOS level. Thus, the RTOS can profit from the reconfigurable hybrid architecture in order to make a better usage of the available resources in a flexible manner. Moreover, from our knowledge there are no other works dealing with on-line RTOS services migration between hardware and software execution environments.

Nevertheless, note that in our currently approach, we do not consider that a OS service may be preempted in CPU and resumed at FPGA (or vice-versa). This is different from [13]. In our approach the migration occurs when the OS component is not being used. Beside that, we focus on OS component migration instead of application tasks. In [4] we show how a migration may dynamically be executed without requiring service preemption during migration.

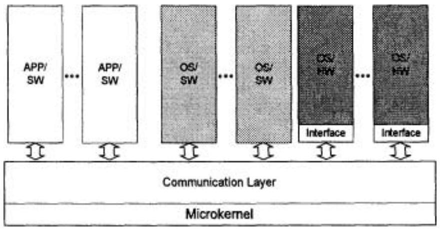


Figure 1. Proposed microkernel based architecture.

3. PRELIMINARIES

Our RTOS is composed of a set of services that may run either on the CPU or on the FPGA. Therefore, the reconfigurable services are provided in two implemented versions: software and hardware. In our approach, most of the application tasks run on the CPU and only application critical tasks use FPGA resources.

The target RTOS architecture follows the microkernel concept, where application and operating system services are seen as components running on top of a small layer which provides basic functionalities. The Figure 1 shows abstractly our architecture. Additionally, the communication infrastructure layer provides the necessary support to allow the communication among components running over the hybrid architecture in an efficient manner. More details about this topic are provided in Section 3.3.

Without loss of generality, we assume that over the hybrid architecture the OS services are seen as components, which uses system resources (FPGA area, CPU workload and communication bandwidth). This view is enforced by the usage of the microkernel architecture model.

3.1 PROBLEM STATEMENT

From a higher point of view, we can see two major problems related with the OS services: their allocation and their reconfiguration. The allocation of the OS services over the hybrid architecture should be done in such a way to optimize the used resources (including the communication costs). This is the focus of the current paper.

Nevertheless, as we considered a changing environment, this allocations need to be continuously evaluated. Whenever the OS components allocations are required to change, a system reconfiguration happens. This reconfiguration activity needs to be carried out respecting the correctness of the running application. In a real-time system, it means that the reconfiguration activities can not violate any time constraint of the running tasks. This problem is handled by modelling theses activities as aperiodic jobs and scheduled together with the running tasks using, therefore, a server ([4]).

3.2 COMMUNICATION UNAWARE ALLOCATION ALGORITHM

A heuristic algorithm presented in [5] determines the allocation OS components. Although presenting good performance, this algorithm does not take into consideration the communication costs. It decides at run-time where to place each OS component taking into consideration its current cost and the remaining available system resources. Here, the resources are: FPGA area (for components being located in hardware) and CPU processor utilization (for components being located in software). Thus, the system has to locate the RTOS components in a limited FPGA area (A_{max}) and limited CPU processor workload (U_{max}).

Every component i has an estimated cost $c_{i,j}$, which represents the percentage of resource from the execution environment used by this component. On the FPGA ($j = 2$) it represents the circuit area needed by the component and on the CPU ($j = 1$) it represents the processor load used by it. The heuristic mentioned above minimizes a objective cost function (Equation 1) subjected to a system resources constraints (Equations 2 and 3).

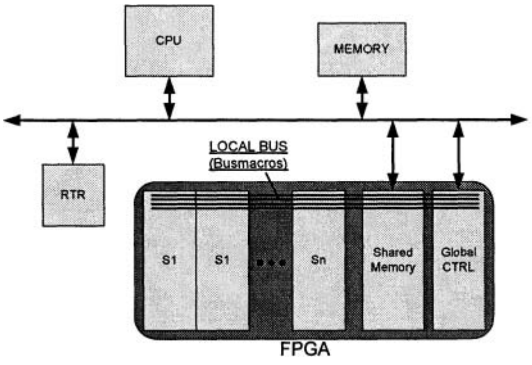
$$min\{\sum_{j=1}^{2}\sum_{i=1}^{n} c_{i,j}x_{i,j}\} \tag{1}$$

$$U = \sum_{i=1}^{n} x_{i,1}c_{i,1} \leq U_{max} \tag{2}$$

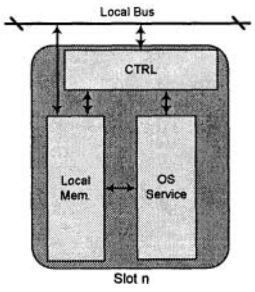
$$A = \sum_{i=1}^{n} x_{i,2}c_{i,2} \leq A_{max} \tag{3}$$

Besides these constraints, an additional one is defined in order to maintain a balanced resource utilization: $B = |w_1U - w_2A| \leq \delta$. Where δ is the maximum allowed unbalanced resource utilization between CPU and FPGA. We also consider that a component i can be assigned just to one of the execution environment. Thus, $\sum_{j=1}^{2} x_{i,j} = 1$ for every $i = 1, ..., n$. The weights w_1 and w_2 are used to proper compare the resource utilization between two different execution environments.

Due to the application dynamism, the assignment decision needs to be checked continuously. Whenever the specified constraint δ is no longer fulfilled, a system reconfiguration takes place. This implies that a set of RTOS component needs to be relocated (reconfigured) by means of migration. In other words, a service may migrate from software to hardware or vice-versa.



(a) Architecture overviwe



(b) OS service slot template

Figure 2. System architecture.

The working algorithm is composed of two phases. First, starting from an empty CPU and FPGA utilization, the components having the smallest costs are selected first and placed on either CPU or FPGA, trying to keep the resource utilization between these two execution domains the same. In the second phase (based on Kernighan-Lin [2]), the allocation is refined by changing the previous location of a component pair (each one locate in different execution domain). This last phase is used in order to improve the balance of resource used and achieve the constraint δ.

3.3 EXECUTION PLATFORM

The kern of our architecture is a Virtex-II Pro fabric, which can be partially reconfigured at run-time and provides additionally a hardcore embedded processor. In Figure 2(a) we show the embedded system architecture in more details. The reconfigurable part of the FPGA is divided in n slots. Each slot provides a OS service framework (Figure 2(b)). The local memory is used to support the communication between local components and the global shared memory is used to perform the communication with components running on the CPU. The local controller is used to manage the access to the local memory and the global controller, which together with its counterpart in software, performs the communication infrastructure mentioned in Section 3. The slots are connected using *Busmacros*. In order to program the FPGA slots, the reconfiguration port is used, which may be local (by using the ICAP Xilinx entity) or an external Run-Time Reconfiguration (RTR) controller.

4. COMMUNICATION-AWARE ALLOCATION ALGORITHM

The new algorithm is build on top of the already available allocation heuristic shortly described in Section 3.2. The idea is to group those components together who present lower communication costs when located at same execution domain. After this clustering process, we do apply the allocation algorithm, where not only single components are assigned to CPU or FPGA, but also meta-components (cluster of components). The previous algorithm is slightly modified in its second phase, when the components assignments are refined, in order to avoid the grouping of made at first.

4.1 DEFINITIONS AND NOTATIONS

The new allocation algorithm is based on component clustering. Therefore, we model our system as an undirected weighted graph $G = (\mathcal{V}, \mathcal{E})$. The edges in $\mathcal{E}$ represent the communication costs CM between two different components. Note that CM depends on two main factors: a static one, related with the architecture (time to deliver a message) and a factor related to the amount of data changed between two components, which is dynamic and depends on the application. Additionally, as each component may be located in one of two different execution environments, the communication cost CM performed between two components is noted by three different values $CM = \{C_\alpha, C_\beta, C_\gamma\}$:

- C_α, when both are on SW domain;
- C_β, when both are on HW domain;
- C_γ, when both are in different domains.

The Figure 3 shows a sample of such a graph. The grey nodes in the graph may be seen as the OS services primitives (API) that are made available for the application tasks (running in software). Note that such node do not have allocation costs as they only is used to properly represent the communication cost between an application task and an OS service.

To measure the connection degree between two communicating components, we define the *local preference* metric, $pl = \frac{2C_\gamma}{2C_\gamma + C_\alpha + C_\beta}$, which is calculated using CM ($pl = f(C_\alpha, C_\beta, C_\gamma)$). The metric pl compare the communication cost between two components when both are placed in the same execution domain in comparison with the case where each of them are placed in different execution domains.

We also define a *global preference* metric, pg, which is pl multiplied by a global factor (see Equation 4). This metric enable us to compare all local preferences with each other by doing the clustering process.

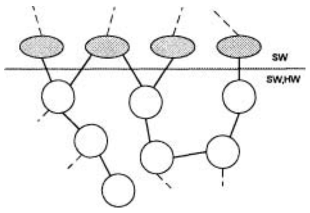


Figure 3. Sample of an OS component graph.

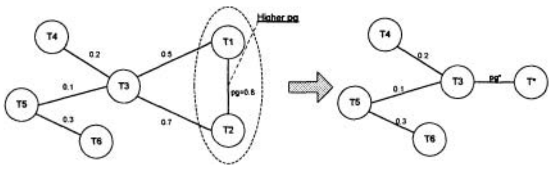


Figure 4. Example of a two component being clustered.

$$pg = \left(\frac{C_\gamma}{\max_V \{C_\gamma\}}\right)pl \tag{4}$$

4.2 CLUSTERING COMPONENTS

The proposed clustering algorithm starts searching for the biggest global preference value, pg', among all edges and tries to cluster the two related components, o and p, respecting two conditions:

- Components o and p have not been clustered;
- $(c_{o,1} + c_{p,1},\ c_{o,2} + c_{p,2}) \leq (\lambda_1,\ \lambda_2)$, where λ_1 and λ_2 are the maximum component costs allowed when performing the combination of o and p. This criterium is used to avoid the deprecation of the allocation algorithm when the allocation costs of the formed components increase.

If a cluster is formed, the two involved components are combined and the search is executed again. This method is repeated until no more components are free for clustering.

When two components are grouped together, a new one is generated T^*. The Figure 4 shows an example. For this case, the CM^* will be generated as follows: $CM^* = CM_{1,3} + CM_{2,3}$. Thus, pg^* is calculated using this new value CM^*: $pg^* = f(C^*_\alpha,\ C^*_\beta,\ C^*_\gamma)$. Note that the communication costs, C_α and C_β, between $T1$ and $T2$ (from the example) are no longer considered for the pg evaluation. Nevertheless, they are stored and used during the balance improvement executed in second phase of the original algorithm ([5]).

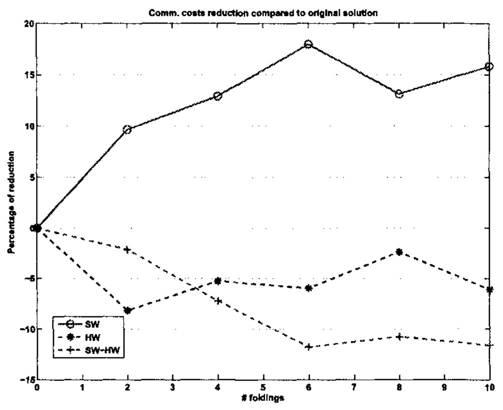


Figure 5. Evaluation results comparison.

4.3 COMPONENT ALLOCATION

When the clustering process ends, we apply the allocation algorithm, which is basically the same one presented in [5]. However, in the second phase of this algorithm, where the allocation decision is refined in order to fulfil the δ constraint (balance improvement), we use the stored C_α and C_β costs. Thus, a pair of components is allowed to change their location, only if this change will improve the balancing and also represent a reduction of the communication costs inside the execution domain.

5. EVALUATION RESULTS

For the evaluation of our algorithm, we did implement it using the MATLAB tool. We create the same environment used in the evaluation of the original algorithm and compare the communication costs achieved by the allocation with and without the clustering process. For our case, we generate randomly the communication graphs of $n = 20$ components respecting the following relation: $C_\alpha < C_\beta < C_\gamma$, which corresponds what we have observed in our architecture. (The communication costs across execution domains are the most expensive ones). The Figure 5 presents the results of the evaluation in every execution domain and also between them. It indicates in percentage, the increase of communication costs for a case using the clustering process in relation to the case where cluster was not used. Therefore, a negative percentage value indicates that a reduction of the communication costs was achieved. The results were performed for different number of clustered (folding) formed.

6. CONCLUSIONS

The paper presents a communication-aware allocation algorithm that is used for mapping OS services (components) of a system to FPGAs or CPUs. We give an idea how to evaluate the communication between the tasks and show how to cluster the tasks. This work is based on the approach presented in [5] and its extension is right now under development, but will be ready for the final version of the paper if it will be accepted.

REFERENCES

[1] Krishnamoorthy Baskaran, Wu Jigang, and Thamipillai Srikanthan. Hardware partitioning algorithm for reconfigurable operating system in embedded systems. In *Sixth Real-Time Linux Workshop*, November 2004. Singapore.

[2] Petru Eles, Krzysztof Kuchcinski, and Zebo Peng. *System Synthesis with VHDL: A Transformational Approach*, chapter 4, pages 114–119. Kluwer Academic Publishers, 1998.

[3] Frank Engel, Ihor Kuz, Stefan M. Petters, and Sergio Ruocco. Operating Systems on SoCs: A Good Idea? In *ERTSI Workshop*, 2004.

[4] Marcelo Götz and Florian Dittmann. Scheduling Reconfiguration Activities of Run-time Reconfigurable RTOS Using an Aperiodic Task Server. In *Proc. of the ARC 2006*, Delft, The Netherlands, March 2006.

[5] Marcelo Götz, Achim Rettberg, and Carlos E. Pereira. Towards Run-time Partitioning of a Real Time Operating System for Reconfigurable Systems on Chip. In *Proc. of IESS*, Manaus, Brazil, August 2005.

[6] J. Harkin, T. M. McGinnity, and L. P. Maguire. Modeling and optimizing run-time reconfiguration using evolutionary computation. *Trans. Embedded Comp. Sys.*, 3(4), 2004.

[7] Paul Kohout, Brinda Ganesh, and Bruce Jacob. Hardware support for real-time operating systems. In *International Symposium on Systems Synthesis*. Proc. of the 1st IEEE/ACM/IFIP Inter. Conf. on HW/SW codesign and system synthesis, 2003.

[8] P. Kuacharoen, M. Shalan, and V. Mooney. A configurable hardware scheduler for real-time systems. In *ERSA*, pages 96–101, June 2003.

[9] J. Lee, K. Ingström, A. Daleby, Tommy Klevin, V.J. Mooney III, and Lennart Lindh. A comparison of the rtu hardware rtos with a hardware/software rtos. In *ASP-DAC*, January 2003.

[10] Jaehwan Lee, Kyeong Ryu, and V. J. Mooney III. A framework for automatic generation of configuration files for a custom hardware/software rtos. In *ERSA*, June 2002.

[11] Lennart Lindh and Frank Stanischewski. Fastchart - a fast time deterministic cpu and hardware based real-time-kernel. In *EUROMICRO*, 1991.

[12] Jean-Yves Mignolet, Vincent Nollet, Paul Coene, Diederik Verkest, Serge Vernalde, and Rudy Lauwereins. Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip. In *DATE*, pages 10986–10993, 2003.

[13] V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Designing an operating system for a heterogeneous reconfigurable soc. In *IPDPS*, Washington, DC, USA, 2003. IEEE Computer Society.

[14] Grant Wigley and David Kearney. The development of an operating system for reconfigurable computing. In *FCCM*, pages 249–250, April 2001.

MULTI-OBJECTIVE DESIGN SPACE EXPLORATION OF EMBEDDED SYSTEM PLATFORMS

Jan Madsen, Thomas K. Stidsen, Peter Kjærulf, Shankar Mahadevan
Informatics and Mathematical Modelling
Technical University of Denmark
{jan,tks,sm}@imm.dtu.dk

Abstract In this paper we present a multi-objective genetic algorithm to solve the problem of mapping a set of task graphs onto a heterogeneous multiprocessor platform. The objective is to meet all real-time deadlines subject to minimizing system cost and power consumption, while staying within bounds on local memory sizes and interface buffer sizes. Our approach allows for mapping onto a fixed platform or onto a flexible platform where architectural changes are explored during the mapping.

We demonstrate our approach through an exploration of a smart phone, where five task graphs with a total of 530 tasks after hyper period extension are mapped onto a multiprocessor platform. The results show four non-inferior solutions which tradeoffs the various objectives.

1. INTRODUCTION

Modern embedded systems are implemented as heterogeneous multiprocessor systems often realized as a single chip solution, System-on-Chip (SoC). Given the high development cost and often short time-to-market demands, these systems are developed as domain specific platforms which can be reconfigured to fit a particular application or set of applications. They are typically designed under rigorous resource constrains, such as speed, size and power consumption. Determining the right platform and efficiently mapping a set of applications onto it, requires hardware/software partitioning, hardware/software interface, processor selection and communication planing.

In this paper, we address the following problem:

> Given a set of applications with individual periods and deadlines, and a heterogenous multiprocessor architecture on which to execute the applications, determine a *mapping* of all tasks on processors and all communications on communication links, such that all deadlines are met subject to power consumption, memory size, buffer sizes of network adapters and overall component cost.

By mapping we mean the allocation of tasks in space and time, i.e. the determination of which tasks to execute on a given processor as well as the detailed

Please use the following format when citing this chapter:

Madsen, J., Stidsen, T.K., Kjærulf, P., Mahadevan, S., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 185–194.

time schedule of each task, and likewise for the communications on communication links.

We address two different variations of the problem;

1 *Fixed* platforms, i.e. no changes of type or number of processors nor any interconnection topology. Hence, the focus is on mapping the applications onto the platform. This variation corresponds to the case where we want to *re-use* an existing platform, which is often the case when moving from one generation to the next of a product family.

2 *Flexible* platforms, i.e. the types and/or number of processors may be changed and the interconnection topology may be changed by adding or removing buses and bus bridges. This variation corresponds to the case where we may change the platform to better fit the requirements of the application.

To demonstrate the capabilities of our approach, we will explore the design of a smart phone, i.e., a heterogeneous multiprocessor platform running five applications with a total of 114 tasks: MP3, JPEG Encoder, JPEG Decoder, GSM Encoder and GSM Decoder. We will demonstrate how our approach can lead to improved solutions for both variations of the optimization problem and in particular for the co-exploration of the architecture selection and application mapping.

The rest of the paper is organized as follows; Section 2 discusses related work. Section 3 presents the application and architecture models. In Section 4 and 5 we present details of our exploration framework. Section 6 present the design space exploration case study of a smart phone. Finally, we present the conclusions in Section 7.

2. RELATED WORK

Static scheduling algorithms for mapping task graphs onto multiprocessor platforms have been studied extensively. A good survey of various heuristic scheduling methods can be found in [5].

Recently, Genetic Algorithms (GA) have been applied to multiprocessor co-synthesis problems due to their property to escape local optima [3, 6–8]. In [6], the goal of the GA-based scheduler is to minimize completion time of all tasks. Although some processor characteristics are taken into account, the approach only addresses homogeneous platforms. In [7] the objectives are to minimize the number of processors required and the total tardiness of tasks for real-time task scheduling. In MOCAG [3] the objectives are extended to also include power consumption beside system price (cost) and task completion time. The approach showed very good results in particular for large systems. The approach described in [2] minimizes schedule length (i.e. the sum of computation, communication and processor wait times) in mixed-machine distributed heterogeneous computing platforms executing up to 200 tasks. The approach uses a fast heuristic with the GA optimization, thereby reducing the exploration time as compared to traditional GA. The approach presented in [8] emphasize energy minimization through the use of dynamic voltage scaling provided by the processors. It is applied to heterogeneous multiprocessor SoC platforms.

Our approach is similar to [2] and [8], but we use a more detailed communication exploration, and in addition to cost, completion time and energy, we explore memory and buffer constrains - with true multi-objective optimization.

3. MODELS

In this section we present the application model and the architecture model on which to execute the application. Both are inputs to our exploration environment.

3.1 APPLICATION MODEL

We consider a real-time application to be modelled as a task graph (expressed as a directed acyclic graph) $G_T = V_T, E_T)$, where $V_T = \{\tau_i : 1 \leq i \leq n\}$ is the set of schedulable tasks, and $E_T = \{e_j : 1 \leq j \leq k\}$ is the set of directed edges representing the data dependencies between the tasks in V_T, i.e., if $\tau_i \prec \tau_j$ then $(\tau_i, \tau_j) \in E_T$. The weight of an edge indicates the size of the message to be transferred between two tasks. Figure 1a shows a example of an application task graph. Each task $\tau_i \in V_T$ is characterized by a five tuple $\langle d_i, T_i, c_i, e_i, m_i \rangle$, i.e. the exact functionality of the task is abstracted away. The relative deadline, d_i, and the period, T_i, are given by external requirements of the application and, hence, are independent of runtime input values, intermediate results or configurations of processing elements. However, the execution time, c_i, the consumed energy, e_i, and the memory usage, m_i, are all determined by the actual mapping of the task onto a particular processor.

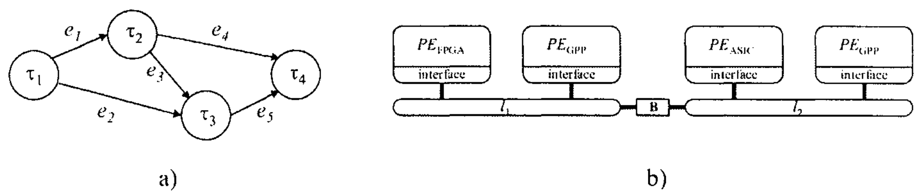


Figure 1. Models, a) Application task graph, b) Architecture graph with 4 PEs, 2 busses and a bus bridge.

The deadline of a real-time application, D_T, is represented by the deadline of the task(s) in V_T with no successors, i.e. no outgoing edges. The task graphs for the different applications are unfolded to cover the hyper period of the complete application. If the different task graphs have different deadlines the period of the hyper period is the least common multiple of all task graphs periods. An instance of a task graph cannot start before the preceding instance has completed its execution.

3.2 ARCHITECTURE MODEL

We consider a heterogeneous multiprocessor architecture to be modelled as an architecture graph $G_A = (V_A, E_A)$. The vertices represent three different types of components, $V_A = V_{PE} \cup V_L \cup V_B$, where $V_{PE} = \{PE_q : 1 \leq q \leq m\}$ is the set of processing elements (PEs), $V_L = \{l_v : 1 \leq v \leq l\}$ is the set of buses which makes up the interconnection network, and $V_B = \{b_k : 1 \leq k \leq r\}$ is the set of bus bridges. Processing elements can be any of dedicated hardware accelerators (PE_{ASIC}), reconfigurable devices (PE_{FPGA}), or general purpose processors (PE_{GPP}). Each PE is characterized by a tuple $\langle f_i, m_i \rangle$, where f_i is the operating frequency of the processor and m_i is the maximum size of the local memory of the processor. Figure 1b shows a example of an architecture graph.

The mapping of the individual tasks, determines if a task will be implemented as hardware logic, ASIC and/or FPGA, or as software running on a GPP. Consequently, by choosing a different processor, the execution characteristics of the task may be changed, which in turn will affect the scheduling of the succeeding tasks; and eventually the completion time of the application.

The interconnections are formed by a (possible hierarchical) network of buses connected through bridges. The communication between two tasks mapped to the same PE is done via accessing shared memory, i.e. we assume that each processing element has local memory, and its access time is negligible. The communication delay between two tasks mapped to different PE's is the property of the size of the message, the sizes of the interface buffers, and the bandwidth of the bus.

Processing elements are connected to buses through network adapters. A network adapter may include buffers, allowing for communication to take place concurrently with computation.

4. DESIGN SPACE EXPLORATION

To solve the presented multi-objective optimization problem, we have used the PISA framework [1] to create a multi-objective Genetic Algorithm (GA). We take as input the set of application task graphs and an architecture graph as described in Section 3. The GA is responsible for design instantiations, i.e. the selection of V_A, and the assignment of the set of tasks V_T onto the set of processing elements $V_{PE} \in V_A$. The selection process can be skipped if the user is only interested in a mapping onto a *fixed* platform, otherwise the platform will be regarded as flexible.

A GA is an iterative and stochastic process that operates on a set of individuals (the population). Each individual represents a potential solution to the problem being solved, and is obtained by decoding the genome of the individual. Initially, the population is randomly generated (in our case based on the input graphs). Each individual in the population is assigned a fitness value which is a measure of its goodness with respect to the problem being considered. This value is the quantitative information used by the algorithm to guide the search for a feasible solution. The basic genetic algorithm consists of repeated execution of three major stages: selection, reproduction, and replacement. Each iteration is called a generation. During the selection stage, individuals with a high fitness value has a higher probability of being selected to create of spring through crossover. A new population is then created by performing crossover followed by mutation. Finally, individuals of the original population is substituted by the newly created individuals in such a way that the most fit individuals are kept deleting the worst ones. A thorough description of genetic algorithms may be found in [4]. There are two important issues which have to be addressed when formulating a problem to be solved by a GA; the representation, i.e. the encoding/decoding mechanism of the genom of an individual, and the evaluation of the fitness of an individual. These issues will be explained in the following sections.

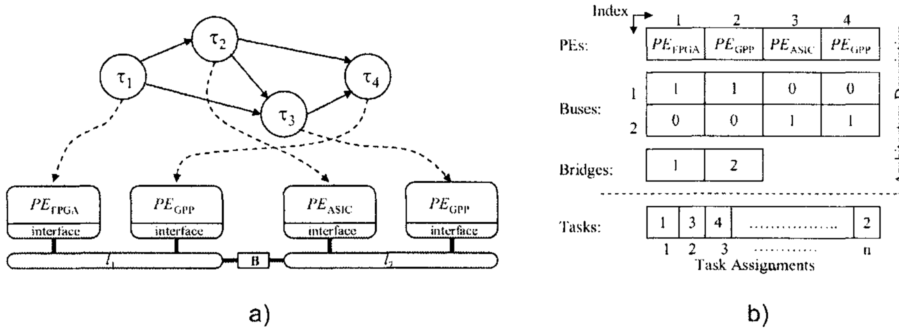


Figure 2. a) Example of a mapping, and b) the corresponding GA representation.

4.1 DESIGN REPRESENTATION

In order for the GA to optimize the designs, each design must be represented as an individual. Figure 2 shows a mapping of an application graph onto an architecture, and the corresponding representation. Each individual consists of two parts: A part specifying the architecture and a part specifying the task assignment. In Figure 2b the architecture representation part contains an array of the deployed processing elements, in this example four *PE*s of three different types (*GPP*, *ASIC* and *FPGA*). The connection between the *PE*s is given by the 2*D* matrix. Each row corresponds to a bus and each element in the row indicates if the corresponding *PE* is connected to the bus ('1') or not ('0'). The bridges which are connecting the busses, are defined as a bridge matrix, where each row represents a bridge and the elements indicates which busses the bridge connects. The task assignment is given as an array, where each index identifies a task and the corresponding element identifies the index of the *PE* to which the task is assigned.

The chosen representation is problem specific and uses internal references. The tasks do *not* identify which *PE* to use, but rather the index of the *PE* in the PE array. Hence, if the type of a PE is changed for an entry, all tasks referring to this index, will have their executing PE changed.

4.2 GENETIC OPERATORS

Initially, a set of individuals are instantiated with unique architecture and application mapping in order to form a population. During each generation we can apply one or more of the following five types of genetic operators,

- *Change PE*: Randomly select an existing PE and change it's type, and randomly select a bus and change its type.
- *Add PE*: Add a new PE to a randomly selected bus, and assign $\lceil \frac{|V_T|}{|V_{PE}|} \rceil$ tasks randomly selected from the other PEs.
- *Remove PE*: Remove a PE from a randomly selected bus, and distribute its tasks among the remaining PEs.
- *Crossover*: Crossover on PE types and tasks mapped to PE. This operator copies the mapping and PE-type from one individual to a PE in another individual.
- *Randomly Re-assign Task*: Move [1;4] randomly selected tasks from a PE to another randomly chosen PE.
- *Heuristically Re-assign Task*: Identify the task graphs which have tasks missing their deadlines, and select a task from these and move it to a PE with no deadline violation.

The first four of the genetic operators enables the GA to find any solution in the problem space. The fifth mutation operator adds a more focused search regarding deadlines and workload balancing. Neither of these operators change the cardinality of V_L, however the GA has full flexibility to reorganize the existing interconnect topology. After applying these operators to individuals, the outcome needs to be evaluated. This is done by a scheduling algorithm which is responsible for determining the start- and the end-times of the computation and communication activities. The scheduling algorithm will be presented in the next section.

5. SCHEDULING

The scheduling task is NP-hard, and it has to be performed for each individual constructed by the GA algorithm. Hence, a fast scheduling method is central for good performance. For a survey of different scheduling algorithms see [5]. We have chosen to use a static list scheduling algorithm which requires a priority for each task. We use a mix of the so called t-levels and b-levels: The t-level of a task is the earliest start time of that task whereas the b-level is the latest start time if time limits are to be satisfied. We use a linear combination of the two measures to produce a task priority-list.

During scheduling tasks are selected from the start of the priority-list but with two important sub conditions

1 For a task to be selected for scheduling, all of its preceding tasks have to have been scheduled already.

2 Tasks with smallest 'earliest start time' is scheduled before other tasks.

5.1 SCHEDULING ALGORITHM

In Figure 3 we outline the pseudo code for the list scheduling algorithm. The list scheduling algorithm initially calculates the t- and b-levels to initialize the $Priority\_List$ (1). Then the list $Num\_Unschedueld\_Predecessors$ is initialized (2). Then the current task to schedule τ_y is set to the task with the highest priority which also satisfies sub-condition 1) and 2) (3). In the main loop, first the earliest possible starting time for the task is found (5). Then τ_y is scheduled to start at this time (6). Afterwards the $Num\_Unschedueld\_Predecessors$ is updated (7). Then the task with the highest priority satisfying sub-condition 1) and 2) is selected as the next task τ_y to schedule (8). Finally the *Earliest Communication Time* (ECT) for all predecessors to τ_y are found, in order to find earliest ready communication resources for mapping and scheduling (9).

```
1:  Calculate $Priority\_List$.
2:  Initialize $Num\_Unschedueld\_Predecessors[..]$
3:  Set $\tau_y$ to the first task in $Priority\_List$ satisfying sub condition 1) and 2)
4:  **repeat**
5:     Find earliest starting time for $\tau_y$
6:     **scheduled** $\tau_y$
7:     Update $Num\_Unschedueld\_Predecessors[..]$
8:     Set next ready task in $Priority\_List$ to $\tau_y$
9:     Calculate $ECT$ to $\tau_y$
10: **until** All tasks scheduled
```

Figure 3. Scheduling Algorithm

Example: Consider a given inter-task communication: $(\tau_x, \tau_y) \in V_T$ (Figure 4a), such that $\tau_x \prec \tau_y$, and $(PE_1, PE_3) \in V_{PE}$, where $\tau_x \rightarrow PE_1$ and $\tau_y \rightarrow PE_3$. Further we assume that the network adapter in PE_3 has no buffers, while PE_1 has both input and output buffers. For the schedulable resources and their interconnectivity, we associate $l_v \in V_L$ a vector of items in the topology set i.e. direct bus (one item) or bridged bus (3 or more items) connecting PE_1 with PE_3. In this case, l_v consists of 3 items: local buses of PE_1 and PE_3, l_1 and l_2, and the bridge, b_1, between l_1 and l_2. Further, we assume the bandwidth of $l_2 > l_1$. Let the message size to be transferred be m. Figure 4b shows a snapshot of the scheduling profile during the communication of interest. For clarity, we assume the transfers over the bridge to be instantaneous and hence ignored in the figure. The shaded portions, imply that the shared resource is busy.

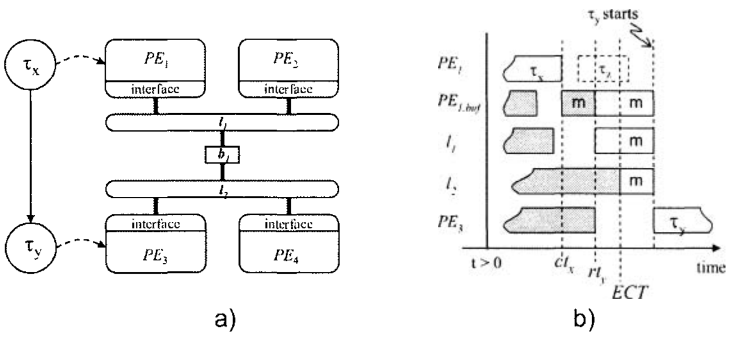


Figure 4. a) Mapping of two tasks, and b) calculation of Earliest Communication Time.

In the following, we are showing how ECT is calculated for the example in Figure 4a. First we calculate the completion time, (ct_x), of τ_x on PE_1. For PE_1, the space in the output buffer, $PE_{1,buf}$, is found to be available, thus the message is moved to $PE_{1,buf}$, freeing PE_1 to start executing another task τ_z. Knowing the precedence constraints and the ordering in the $Priority\_List$, we calculate the earliest possible start time, rt_y, for τ_y on PE_3. ECT is set to the furthest time when either the communication is possible (τ_x completes on PE_1) or required (τ_y ready to start on PE_3), i.e. $ECT = rt_y$. Then we find the topology set, l_v, connecting PE_1 with PE_3, which is $\{l_1, l_2\}$. We evaluate the availability of each of the busses of l_v. Although l_1 is available, the earliest time at which communication can be scheduled is when l_2 is also available. This dictates the ECT. Overall the communication speed is dictated by the slower bus, keeping the output buffer, $PE_{1,buf}$ occupied. The actual start time of τ_y is after the message m has been received. □

5.2 MEMORY ISSUES

During scheduling both interface buffers and local memory are taken into account.

Interface buffers of a processor can be used in two ways 1) to store data coming from the bus to the processor and 2) to store data going from the processor to the bus. It is assumed that buffers can not block. This means that even if a communication task can not be stored in the buffer (e.g. buffer is full), the buffer can still send data to the bus.

When tasks are mapped to processors, the static and dynamic memory consumption of the tasks are taken into account. This assures that the number of tasks mapped to a PE will always fit within the available size of the local memory. The local memory size for each PE is specified as a constraint in the input. However, during scheduling data waiting to be sent to the bus may have to be saved in the local memory of the processors, for instance in the case where the corresponding buffer is full. This can cause a violation of the memory constraint on a given processor. This memory violation is one of the objectives optimized in the multi-objective GA algorithm.

6. CASE STUDY

In this section we explore a smart phone [8] running 5 applications (JPEG encoder and decoder, MP3, and GSM encoder and decoder) with a total of 114 tasks.

After expanding the task graphs into a hyper period, we have a total of 530 tasks to schedule. The GA was run for 100 generations which corresponds to approximately 10 min of run time. In each generation 100 individuals was evaluated. Hence, 10.000 solutions were explored, resulting in four interesting architectures (see figure 5) on the approximated pareto front.

Table 1 lists the cost, energy consumption and memory violation for each of the four architectures.

id	cost($)	Energy (J)	Memory violation (Bytes)
166	1396	22.0	1344
171	1048	29.0	0
184	1396	24.6	0
187	1596	22.0	612

Table 1. Characteristics of four solutions on the approximated pareto front.

The two architectures id 166 and id 184 are identical, but with a different mapping of tasks to processors. This gives id 166 a smaller energy consumption with the cost of a memory violation. The cheapest architecture is id 171, this is however the solution with the largest energy consumption. With regard to energy consumption id 187 is the cheapest but at the same time the most expensive architecture.

As there is no guarantee for optimal solutions the selection of architectures will only be an approximation to the pareto front. However, the experiment shows how the algorithm is a powerful tool to explore the design space for embedded system architectures with both one and multiple busses.

7. CONCLUSIONS

The design of a heterogenous multiprocessor system, is accomplished either by design reuse or incremental modification of existing designs. In this paper, we have presented a multi-objective optimization algorithm which allows to optimize the application mapping on to an existing architecture, or optimize the application mapping and architecture during development. Our algorithm couples GA with list scheduler. The GA allows to instantiate multiple designs which are then evaluated using the scheduler. The outcome is an approximated pareto front of latency, cost, energy consumption and buffer and memory utilization. The case study has shown, that maximum gains are achieved when optimizing both architecture and application simultaneously.

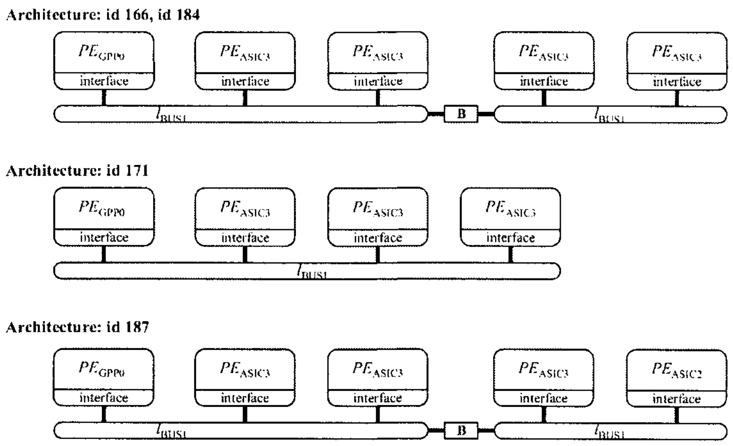


Figure 5. Non-inferior architectures from the optimization runs.

8. ACKNOWLEDGEMENT

This work has been supported by the European project ARTIST2 (IST-004527), Embedded Systems Design.

REFERENCES

[1] Stefan Bleuler, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. PISA — a platform and programming language independent interface for search algorithms. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization (EMO 2003)*, Lecture Notes in Computer Science, pages 494 – 508, Berlin, 2003. Springer.

[2] Muhammad K. Dhodhi, Imtiaz Ahmad, Anwar Yatama, and Ishfaq Ahmad. An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems. In *Journal of Parallel and Distributed Computing*, pages 1338–1361. Elsevier Science, 2002.

[3] Robert P. Dick and Niraj K. Jha. MOGAC: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems. In *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 920–935. IEEE, 1998.

[4] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, 1989.

[5] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, 1999.

[6] Ceyda Oguz and M.Fikret Ercan. A genetic algorithm for multi-layer multiprocessor task scheduling. In *IEEE Region 10 Conference (TENCON)*, pages 168–170. IEEE, 2004.

[7] Jaewon Oh and Chisu Wu. Genetic-algorithm-based real-time task scheduling with multiple goals. In *Journal of Systems and Software*, pages 245–258. Elsevier, 2004.

[8] Marcus T. Schmitz, Bashir M. Al-Hashimi, and Petru Eles. *System-Level Design Techniques for Energy-Efficient Embedded Systems*. Kluwer Academic Publishers, 2004.

DYNAMIC MEMORY MANAGEMENT FOR EMBEDDED REAL-TIME SYSTEMS

A. Crespo, I. Ripoll, M. Masmano
Universidad Politecnica de Valencia
46022 Valencia, Spain
{acrespo,iripoll,mmasmano}@disca.upv.es

Abstract Dynamic memory storage has been widely used during years in computer science. However, its use in real-time systems has not been considered as an important issue because the spatial and temporal worst case for allocation and deallocation operations were unbounded or bounded but with a very large bound.

TLSF (Two Level Segregated Fit) is a new allocator has been designed specifically to meet real-time constraints. These constraints are addressed in two axis: Time and Space. While the temporal behaviour of TLSF is excellent, $O(1)$, the spatial behaviour is as the best of the known allocators. The spatial constraint is specially required in embedded systems with mitided resources. An efficient and guaranteed use of memory is needed for these systems. In this paper we compare the temporal and spatial performances of the TLSF allocator comparing it with the most relevant allocators.

Keywords: Dynamic memory allocation, real-time systems, embedded systems

1. INTRODUCTION

Dynamic storage allocation (DSA) algorithms plays an important role in modern software engineering paradigms (object oriented paradigm) and techniques. Additionally, it allows to increase the flexibility and functionalities of the applications. In fact, there exist in the literature a large number of works and references to this particular issue. However, in the real-time community the use of dynamic memory techniques has not been considered as an important issue because the spatial and temporal worst case for allocation and deallocation operations were insufficiently bounded. It is significant the reduced number of papers about this topic in most relevant real-time events.

Nevertheless, it is not wise to ignore the use of dynamic memory in real-time applications just because it is believed that it is not possible to design an allocator that matches the specific needs of a real-time system. To take profit of, these advantages for the developer of real-time systems a deterministic response of

Please use the following format when citing this chapter:

Crespo, A., Ripoll, I., Masamano, M., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 195–204.

the dynamic memory management is required. This requirement implies the use of dynamic memory mechanisms completely predictable. Allocation and deallocation are the basic mechanisms to handle this issue. Additionally to the temporal cost of these operations is the memory fragmentation incurred by the system when dynamic memory is used. Fragmentation plays an important role in the system efficiency which is more relevant for embedded systems.

TLSF [4] is a new allocator has been designed specifically to meet real-time constraints. These constraints are addressed in two axis: Time and Space. While the temporal behaviour of TLSF is excellent, $O(1)$, the spatial behaviour is as the best of the known allocators. In this paper we compare the temporal and spatial performances of the TLSF allocator comparing it with the most relevant allocators.

1.1 DSA AND REAL-TIME REQUIREMENTS

The requirements of real-time applications regarding dynamic memory can be summarised as:

- **Bounded response time**. The worst-case execution time (WCET) of memory allocation and deallocation has got to be known in advance and be independent of application data. This is the main requirement that must be met.

- **Fast response time**. Besides, having a bounded response time, the response time has got to be short for the DSA algorithm to be usable. A bounded DSA algorithm that is 10 times slower than a conventional one is not useful.

- **Memory requests need to be always satisfied**. No real-time applications can receive a null pointer or just be killed by the OS when the system runs out of memory. Although it is obvious that it is not possible to always grant all the memory requested, the DSA algorithm has got to minimise the chances of exhausting the memory pool by minimising the amount of fragmentation and wasted memory.

- **Efficient use of memory**. The allocator has got to manage memory efficiently, that is, the amount of wasted memory should be as small as possible.

2. DYNAMIC STORAGE ALLOCATION ALGORITHMS

This section presents a categorisation of existing allocators and a brief description of the most representative ones, based on the work of Wilson et al. [13] which provides a complete taxonomy of allocators.

Considering the main mechanism used by an allocator, the following categorisation is proposed [13]. Examples of each category are given. In some cases it is difficult to assign an allocator to a category because it uses more than one mechanism. In that case, tried to determine which is the more relevant mechanism and categorise the allocator accordingly.

Sequential Fits: Sequential Fits algorithms are the most basic mechanisms. They search sequentially free blocks stored in a singly or doubly linked list. Examples are first-fit, next-fit, and best-fit. First-fit and best-fit are two of the most representative sequential fit allocators, both of the are usually implemented with a doubly linked list.

Segregated Free Lists: These algorithms use a set of free lists. Each of these lists store free blocks of a particular predefined size or size range. When a free block is released, it is inserted into the list which corresponds to its size. There are two of these mechanisms: Simple Segregated storage and Segregated

Buddy Systems: Buddy Systems [2] are a particular case of Segregated free lists. Being $\mathcal{H}$ the heap size, there are only $\log_2(\mathcal{H})$ lists since the heap can only be split in powers of two. This restriction yields efficient splitting and merging operations, but it also causes a high memory fragmentation. The Binary-buddy [2] allocator is the most representative of the Buddy Systems allocators, which besides has always been considered as a real-time allocator.

Indexed Fits: This mechanism is based on the use of advanced data structures to index the free blocks using several relevant features. To mention a few examples: algorithms which use Adelson-Velskii and Landin (AVL) trees [10], binary search trees or cartesian trees (Fast-Fit [12]) to store free blocks.

Bitmap Fits: Algorithms in this category use a bitmap to find free blocks rapidly without having to perform an exhaustive search. TLSF and Half-fit [6] are examples of this sort of algorithms. Half-fit groups free blocks in the range $[2^i, 2^{i+1}[$ in a list indexed by i. Bitmaps to keep track of empty lists jointly with bitmap processor instructions are used to speed-up search operations.

Hybrid allocators: Hybrid allocators can use different mechanisms to improve certain characteristics (response time, fragmentation, etc.) The most representative is Doug Lea's allocator [3], which is a combination of several mechanisms. In what follows this allocator will be referred to as DLmalloc.

Table 1 summarises the temporal costs of the worst-case allocation and deallocation of these algorithms

Table 1. Worst-case costs

	Allocation	Deallocation
First-fit/Best-fit	$O\left(\frac{\mathcal{H}}{2\mathcal{M}}\right)$	$O(1)$
Binary-buddy	$O(\log_2\left(\frac{\mathcal{H}}{\mathcal{M}}\right))$	$O(\log_2\left(\frac{\mathcal{H}}{\mathcal{M}}\right))$
AVL-tree	$O(1.44\ \log_2\left(\frac{\mathcal{H}}{\mathcal{M}}\right))$	$O(3 \cdot 1.44\ \log_2\left(\frac{\mathcal{H}}{\mathcal{M}}\right))$
DLmalloc	$O\left(\frac{\mathcal{H}}{\mathcal{M}}\right)$	$O(1)$
Half-fit	$O(1)$	$O(1)$
TLSF	$O(1)$	$O(1)$

Although the notion of fragmentation seems to be well understood, it is hard to define a single method of measuring or even defining what fragmentation is. In [13] fragmentation is defined as "the inability to reuse memory that is free". Historically, two different sources of fragmentation have been considered: internal and external. Internal fragmentation is caused when the allocator returns to the application a block that is bigger than the one requested (due to block round-up, memory alignment, unablility to handle thè remaining memory, etc.). External fragmentation occurs when there is enough free memory but there is not a single block large enough to fulfill the request. Internal fragmentation is caused only by the allocator implementation, while external fragmentation is caused by a combination of the allocation policy and the user request sequence.

Attending to the reason an allocator can not use some parts of the memory, three types of wasted memory, usually called fragmentation can be defined: it Internal fragmentation when the allocator restricts the possible sizes of allocatable blocks because of design constraints or efficiency requirements; *External fragmentation:* The sequence of allocation and deallocation operations may leave some memory areas unused since; and, *wasted memory* which corresponds to the memory managed by the allocator that can not be assigned to the application . In fact, the wasted memory at any time is the difference between the live memory and the memory used by the allocator at that time.

The following table summarises the worst-case memory requirements for several well known allocation policies (see [7–9, 2]).

The table shows the heap size needed when the maximum allocated memory (live memory) is $\mathcal{M}$ and the largest allocated block is m, Robson also showed that an upper bound for the worst-case of any allocator is given by: $\mathcal{M} \times m$.

The other allocation algorithms will have a fragmentation depending on the implemented policy. While AVL-tree, DLmalloc and TLSF tends to a good fit policy, Half-fit implements a binary buddy policy.

Table 2. Fragmentation worst-case

	Heap size
First-fit	$\frac{\mathcal{M}}{\ln 2}\sum_{i=1}^{m}(\frac{1}{i})$
Best-fit	$\mathcal{M}(m-2)$
Binary-buddy	$\mathcal{M}(1+\log_2 m)$

However, several studies [11, 5] based on synthetic workload generated by using well-known distributions (exponential, hyper-exponential, uniform, etc.). The results obtained were not conclusive; these studies show contradictory results with slightly different workload parameters. At that time, it was not clear whether First-fit was better than Best-fit.

Johnstone and Wilson [1] analysed the fragmentation produced by several standard allocators, and concluded that the fragmentation problem is a problem of "poor" allocator implementations rather than an intrinsic characteristic of the allocation problem itself. A

The policy used to search for a suitable free blocks in Half-fit and TLSF introduces a new type of fragmentation or *incomplete memory use* as it is called in [6]: free blocks larger than the base size of the segregated list where they are located, will not be used to serve requests of sizes that are one byte larger than the base size.

TLSF is a bounded-time, Good-fit allocator. The Good-fit policy tries to achieve the same results as Best-fit (which is known to cause a low fragmentation in practice [1]), but introduces implementation optimisations so that it may not find the tightest block, but a block that is close to it. TLSF implements a combination of the Segregated and Bitmap fits mechanisms. The use of bitmaps allow to implement fast, bounded-time mapping and searching functions.

3. TLSF

In a previous paper [4], the authors described a preliminary version of a new algorithm for dynamic memory allocation called TLSF (Two-Level Segregated Fit). We now present a slightly improved version of the algorithm that maintains the original worst-case time bound and efficiency, and reduces the fragmentation problem in order to make TLSF usable for long-lived real-time systems.

TLSF is a bounded-time, Good-fit allocator. The Good-fit policy tries to achieve the same results as Best-fit (which is known to cause a low fragmentation in practice [1]), but introduces implementation optimisations so that it may not find the tightest block, but a block that is close to it. TLSF implements a combination of the Segregated and Bitmap fits mechanisms. The use of

bitmaps allow to implement fast, bounded-time mapping and searching functions.

The TLSF data structure can be represented as a two-dimension array. The first dimension splits free blocks in size-ranges a power of two apart from each other, so that first-level index i refers to free blocks of sizes in the range $[2^i, 2^{i+1}[$. The second dimension splits each first-level range linearly in a number of ranges of an equal width. The number of such ranges, $2^{\mathcal{L}}$, should not exceed the number of bits of the underlying architecture, so that a one-word bitmap can represent the availability of free blocks in all the ranges. According to experience, the recommended values for $\mathcal{L}$ are 4 or, at most, 5 for a 32-bit processor. Figure 1 outlines the data structure for $\mathcal{L} = 3$.

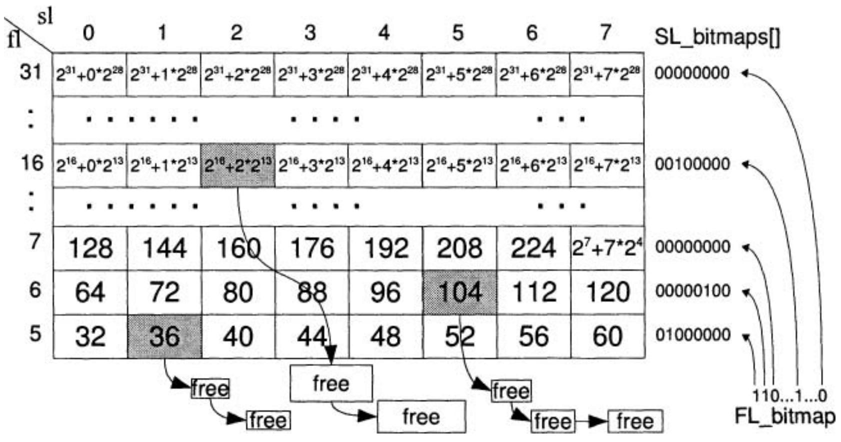


Figure 1. TLSF data structures example.

TLSF uses word-size bitmaps and processor bit instructions to find a suitable list in constant time. For example, using the *ffs* instruction, which returns the position of the first (least significat) bit set to 1, it is possible to find the smaller non-empty list that holds blocks bigger or equal than a given size; and the instruction *fls* (returns the position of the most significant bit set to 1) can be used to compute the $\lfloor \log_2(x) \rfloor$ function. Note that it is not mandatory to have these advanced bit operations implemented in the processor to achieve constant time, since it is possible to implement them by software using less than 6 non-nested conditional blocks (see glibc or Linux implementation).

Given a block of size $r > 0$, the first and second indexes (fl and sl) of the list that holds blocks of its size range are: $fl = \lfloor \log_2(r) \rfloor$ and $sl = \lfloor (r - 2^{fl})/2^{fl-\mathcal{L}} \rfloor$. For efficiency reasons, the actual function used to calcualte sl is $(r/s^{fl-\mathcal{L}}) - 2^{\mathcal{L}}$. The function mapping_insert computes efficiently fl and sl:

4. TEMPORAL ANALYSIS

In this analysis we compare the performance of TLSF with respect to the worst-case scenario. To evaluate the temporal cost we have analysed the worst-case scenario for each allocator and we have measured the cost of all of them under it. A description of the worst-case scenarios of each allocator can be found in [4].

The memory heap size was set to 4Mbytes, and the minimum block size is 16bytes. All testing code is available at http://rtportal.upv.es/rtmalloc/

Table 3. Worst-case (WC) and Bad-case (BC) allocation

Malloc	FF	BF	BB	DL	AVL	HF	TLSF
Processor instructions	81995	98385	1403	721108	3116	164	197
Number of cycles	1613263	1587552	3898	3313253	11739	1690	2448

Table 3 show measurements of a single malloc operation after the worst-case scenario has been constructed. Every allocator has been tested for each worst-case scenario. The result of an allocator when tested in its worst-case scenario is printed in bold face in the tables. The results show that each allocator performs badly in its theoretical worst-case scenarios. This can be easily seen in table 3(b) (instruction count).

As expected, First-fit and Best-fit perform quite badly under their worst-case scenarios. The low data locality produces a high cache miss ratio which makes the temporal response even worse than expected, considering the number of instructions executed. The number of cycles per instruction (CPI) is high: 19 for First-fit and 16 for Best-fit.

The DLmalloc allocator is a good example of an algorithm designed to optimise the average response time, but it has a very long response time in some cases. DLmalloc tries to reduce the time spent coalescing blocks by delaying coalescing as long as possible; and when more space is needed coalescing is done all at once, causing a large overhead on that single request. DLmalloc has the largest response time of all allocators, and therefore it is not advisable to use in real-time systems.

Half-fit, TLSF, Binary-buddy and AVL show a reasonably low allocation cost, Half-fit and TLSF being the ones which show the most uniform response time, both in number of instructions and time. Half-fit shows the best worst-case response; only TLSF is 20% slower. Although Half-fit shows a fast, bounded response time under all tests, it suffers from a considerable theoretical internal fragmentation.

As explained in the fragmentation section, Half-fit provides a very good worst-case response time at the expense of wasting memory. Half-fit is superior in all respects to Binary-buddy. Half-fit is faster than Binary-buddy and handles memory more efficiently.

5. FRAGMENTATION EVALUATION

In order to analyse the fragmentation incurred by the allocators, we have defined a periodic task model which includes the memory requirement of each task. Using this task model, the allocators have been evaluated under different situations.

Let $\tau = \{T_1, ..., T_n\}$ be a periodic task system. Each task $T_i \in \tau$ has the following temporal parameters $T_i = (c_i, p_i, d_i, g_i, h_i)$. Where c_i is the worst execution time, p_i is the period; d_i is the deadline, g_i is the maximum amount of memory that a task T_i can request per period; and h_i is the longest time than task can hold a block after its allocation (holding time).

In this model a task T_i can ask for a maximum of g_i bytes per period which have to be released no later than h_i units of time.

To serve all memory requests, the system provides an area of free memory (also known as heap) of $\mathcal{H}$ bytes. The symbol r denotes the memory size of an allocation request, subindexes will be used to identify the activation that made the request. l_i is the maximum amount of live memory required by task T_i.

$\mathcal{L} = \sum_{i=0}^{n} l_i$ will be the maximum amount of memory need by the periodic task system τ.

The simulation model has been focused on studying the behaviour (spatial response) of the allocators when they are executed in our memory model and how the use of a holding time (h) and a maximum amount of memory per period (g) affects each *real-time* allocator. It is important to note that we have executed $50,000$ u.t. (steps) of simulation in all the simulations of this section, moreover, each test has been repeated *at least* 100 times with different seed for the random generator. That is the reason that we have written up our results as an average, followed, when required by a standard deviation.

To measure fragmentation we have considered the factor $\mathcal{F}$, which is calculated as the point of the maximum memory used by the allocator relative to the point of the maximum amount of memory used by the load (live memory).

In order to evaluate the fragmentation generated by each allocator three different tests have been designed. First test (Test 1) consist in 100 periodic tasks with harmonic periods where the allocation size r_i and the maximum memory per period g_i of each task has been generated using uniform distribution in the range of (16, 4096) and (8, 2048), respectively. The holding time h_i of each task has also been calculated with an uniform distribution, $uniform(4 \cdot p_i, 12 \cdot p_i)$. Second test (Test2) uses 200 periodic tasks with non harmonic periods. with $r_i = uniform(100, 8192)$, $g_i = uniform(200, 16384)$, $h_i = uniform(4 \cdot p_i, 6 \cdot p_i)$, to generate g_i, we have used the same constraint as in Test 1. Third test (Test3) is only conformed by 50 periodic tasks with non harmonic periods. As in the previous tests, we have used an uniform distribution to calculate the parameters of the tasks with the next values: $r_i = uniform(1024, 10240), g_i = uniform(2048, 20480), h_i = uniform(p_i, 16 \cdot p_i)$, to generate g_i, we have used the same constraint as in Test 1.

Table 4 shows the fragmentation obtained by each allocator under these randomly-generated tests.

Table 4. Fragmentation obtained by randomly-generated tests

	FF	BF	BB	HF	TLSF
Test1 Avg.	100.01	4.64	46.37	82.23	**4.33**
Std.Dev.	4.96	0.61	0.74	1.06	**0.55**
Test2 Avg.	85.01	**4.54**	45.00	75.15	4.99
Std.Dev.	4.92	0.67	0.98	1.52	**0.59**
Test3 Avg.	112.51	**7.01**	48.63	99.10	7.69
Std.Dev.	8.53	1.13	1.90	2.61	**0.98**

The results of these tests show that, overall, TLSF is the allocator that requires less memory (less fragmentation) closely followed by Best-Fit. As shown, TLSF behaves better even than Best-Fit in most cases, that can be explained due that TLSF always rounds-up all petitions to fit them to any existing list, allowing TLSF to reuse one block with several petitions with a similar size, independenly of their arrival order. On the other hand, Best-Fit always splits blocks to fit the requested sizes, making impossible to reuse some blocks when these blocks have previously been allocated to slightly smaller petitions.

On the other hand we have Binary-Buddy and Half-Fit, both of them with a fragmentation of up to 99.10% in the case of Half-Fit and 69.59% in Binary-Buddy. As expected, the high fragmentation caused by Binary-buddy is due to the excesive size round up (round up to power of two). All wasted memory of Binary-buddy is caused by internal fragmentation. Half-Fit's fragmentation was also expected because of its *incomplete memory use*. As can be seen, both allocators are quite sensitive request sizes that are not close to power of two, causing a high fragmentation, internal fragmentation in the case of the Binary-Buddy and external one in the Half-Fit case.

6. CONCLUSIONS

TLSF is a dynamic storage allocator designed to meet real-time requirements. This paper has focused on the evaluation of the TLSF in two axis: Time and Space.

The temporal cost of the TLSF is constant which is demonstrated in the experiments. The main conclusion when considering spatial performance is that TLSF algorithm performs as well as Best-fit, which is one of the allocators that better handles and reuses memory. On the other hand Half-Fit handles memory poorly. Half-fit achieves a very fast and constant temporal response time at the expenses of high wasted memory.

We have also extended the already-existing periodic real-time model to include the dynamic memory allocation requirement of the tasks. Based on this model, an experimental framework has been constructed to compare the efficiency, regarding wasted memory, of several dynamic storage allocators. The allocators used in the study were those that meet the requirements needed to be used in real-time systems, i.e., allocation and deallocation is performed in bounded time (constant or logarithmic time).

REFERENCES

[1] M.S. Johnstone and P.R. Wilson. The Memory Fragmentation Problem: Solved? In *Proc. of the Int. Symposium on Memory Management, Vancouver, Canada.* ACM Press, 1998.

[2] D. E. Knuth. *The Art of Computer Programming, volume 1: Fundamental Algorithms.* Addison-Wesley, Reading, Massachusetts, USA, 1973.

[3] D. Lea. A Memory Allocator. *Unix/Mail*, 6/96, 1996.

[4] M. Masmano, I. Ripoll, A. Crespo, and J. Real. TLSF: A new dynamic memory allocator for real-time systems. In *16th Euromicro Conference on Real-Time Systems*, pages 79–88, Catania, Italy, July 2004. IEEE.

[5] Norman R. Nielsen. Dynamic memory allocation in computer simulation. *Commun. ACM*, 20(11):864–873, 1977.

[6] T. Ogasawara. An algorithm with constant execution time for dynamic storage allocation. *2nd Int. Workshop on Real-Time Computing Systems and Applications*, page 21, 1995.

[7] J. M. Robson. An estimate of the store size necessary for dynamic storage allocation. *Journal of the Association for Computing Machinery*, 18(3):416–423, 1971.

[8] J. M. Robson. Bounds for some functions concerning dynamic storage allocation. *Journal of the Association for Computing Machinery*, 21(3):491–499, 1974.

[9] J. M. Robson. Worst case fragmentation of first fit and best fit storage allocation strategies. *Comput. J.*, 20(3):242–244, 1977.

[10] R. Sedgewick. *Algorithms in C. Third Edition.* Addison-Wesley, Reading, Massachusetts, USA, 1998.

[11] J.E. Shore. On the External Storage Fragmentation Produced by First-Fit and Best-Fit Allocation Strategies. *Communications of the ACM*, 18(8):433–440, 1975.

[12] C. J. Stephenson. Fast fits: New methods of dynamic storage allocation. *Operating Systems Review*, 15(5), October 1983. Also in Proceedings of Ninth Symposium on Operating Systems Principles, Bretton Woods, New Hampshire, October 1983.

[13] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles. Dynamic Storage Allocation: A Survey and Critical Review. In H.G. Baker, editor, *Proc. of the Int. Workshop on Memory Management, Kinross, Scotland, UK*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, 1995. Vol:986, pp:1–116.

RELIABILITY-AWARE POWER MANAGEMENT OF MULTI-CORE PROCESSORS *

Jan Haase, Markus Damm, Dennis Hauser, Klaus Waldschmidt
J. W. Goethe-Universitt Frankfurt/Main, Technical Computer Sc. Dep., Box 11 19 32, D-60054 Frankfurt/Main, Germany
{haase|damm|dhauser|waldsch}@ti.informatik.uni-frankfurt.de

Abstract: Long-term reliability of processors is experiencing growing attention since decreasing feature sizes and increasing power consumption have a negative influence on the lifespan. The reliability can also be influenced by Dynamic Power Management (DPM), since it affects the processor's temperature.

In this paper, it is examined how different DPM-strategies for Multi-Core processors alter their lifespan. By simulating such a Multi-Core system using the Self Distributing Virtual Machine (SDVM), thus exploiting dynamic parallelism, it is shown that its long-term reliability can be influenced actively with different DPM strategies.

Keywords: Adaptivity; Power Management; Reliability; SDVM.

1. INTRODUCTION

The long-term reliability resp. lifespan of microprocessors hasn't been much of an issue in the past, since a processor was usually obsolete due to technological aging (and has been replaced) before it began to fail. This is about to change for several reasons. First, microprocessors and multi-core processors are nowadays combined with other components as complete systems on chip (SoCs) or networks on chip (NoCs). Therefore the processor cannot be replaced easily. Secondly, smaller feature sizes and increasing power densities lead to a higher vulnerability to wear-out based failure mechanisms like electromigration or stress migration. The international technology roadmap on semiconductors (ITRS) sees a trend that is threatening "the nearly unlimited lifetime and high level of reliability that customers have come to expect" [1]. The approaches to tackle this problem are mostly design-centric. RAMP [2], for example, is a model to determine lifespan estimates depending on the architecture

*Parts of this work have been supported by the Deutsche Forschungsgemeinschaft (DFG).

of a processor. In a subsequent paper, however, the authors extend their approach to a so called *Dynamic Reliability Management* [3], whose idea is to adjust a processor at runtime (e.g. by voltage scaling) to meet a certain reliability target, though no algorithms for this cause are proposed. Apart from this, no further concepts or algorithms for dynamic reliability management do yet exist.

Our remedy to this problem has two ingredients: Dynamic power management (DPM) and parallel computing. It has been noted in [4] that DPM schemes affect a processor's reliability, since it directly affects a processor's temperature. The essential failure mechanisms like electromigration, corrosion, time-dependent dielectric breakdown (TDDB), hot carrier injection (HCI), surface inversion, and stress migration are more or less temperature dependent [5]. While DPM tends to lower a processor's temperature, which is beneficial, it also leads to the unfavorable effect of temperature cycling, i.e. frequent heating up and cooling down.

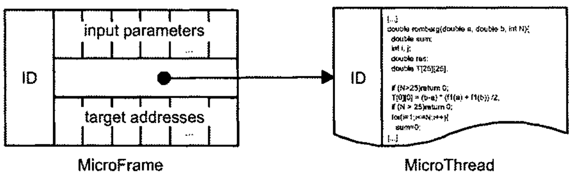
Dynamic power management on multi-core processors, however, has a lot more possibilities to scale the power consumption of a chip: Aside from clock frequency reduction (along with dynamic voltage scaling or adaptive body biasing), whole cores can be switched off without disrupting the execution of applications. Since the workload and thus DPM in parallel computing environments also depends on the parallelizability of an application, it seems to be obvious that this can be done efficiently only with a dynamic approach.

The SDVM (Self Distributing Virtual Machine) as a middleware for the dynamic, automatic distribution of code and data over any network of computing resources seems to be an ideal choice to be run on multi-core processors in NoCs. In particular, it supports adding and removing of computing resources at runtime, making the implementation of the aforementioned dynamic power management on multi-core processors possible in the first place. To permit DPM on an SDVM-driven multi-core processor, an appropriate power managing mechanism has been implemented, which scales the performance of the cores according to the current workload. The reliability-awareness is then achieved by appropriate power management policies.

The goal of this paper is to examine the potential of such reliability-aware power management strategies for multi-core processors by simulation.

2. SDVM - A MIDDLEWARE FOR POWER OPTIMIZED SOCS

The Self Distributing Virtual Machine (SDVM) [6] was designed to feature undisturbed parallel computation flow while adding and removing processing units from computing clusters. These clusters may consist



```
[...]
double romberg(double a, double b, int N){
  double sum;
  int i, j;
  double res;
  double T[25][25];

  if (N>25)return 0;
  T[0][0] = (b-a) * (f1(a) + f1(b)) /2;
  if (N > 25)return 0;
  for(i=1;i<=N;i++){
    sum=0;
[...]
```

Figure 1. The Micro*thread* contains a code fragment whereas the Micro*frame* contains the parameters needed to execute the corresponding code fragment, as well as the IDs of other Microframes the results of the execution then should be sent to.

of several processing cores or even full-grown computers, and any connection network topology is supported.

The SDVM is a middleware, implemented as a daemon to be run on each participating machine or processor, creating a *site* each. The sites communicate by sending messages. Applications must be cut to convenient application fragments, the *microthreads*, which can be executed on any site. The SDVM follows the dataflow principle, therefore a microthread is executable if it has received all its needed input parameters. These parameters are collected in a special data structure, the *microframe* (see Fig. 1). Data, and code, is automatically sent to the sites where it is needed. Therefore, the SDVM is actually real parallel processing though it may look like a multi-threaded concept.

The SDVM supports growing and shrinking the cluster at runtime. When a site is out of work, it requests executable microframe/microthread pairs from other sites automatically. This behavior provides automatic load balancing, even between processing units with different processing speeds, and offers the addition of new sites at runtime. For a detailed description of the scheduling see [6]. If more processing power is available than needed, the local memory data and possibly microthreads are pushed out to another site and then the site can be safely shut down.

The SDVM daemon is organized in several modules with different tasks, which are part of one of three layers:

- The execution layer, where the actual calculationd are performed. It contains the memory (containing data and microframes), the code storage (which contains the needed microthreads or requests them from other sites), the processing manager, the scheduler (see Fig. 2), and a unit for possible input and output.
- The network layer is the part of the daemon which is related to sending messages over the network. Messages are encrypted by a security manager to avoid eavesdropping. The *energy manager* which is described in section 4 is located here.
- The maintenance layer is concerned with the organization of the cluster and the (local) site. Modules are located here which know

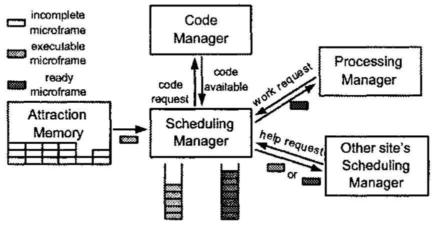


Figure 2. The SDVM's scheduling mechanism: The scheduling manager receives executable microframes (which have got all their parameters) from the attraction memory. It then sends a code request to the code manager. When the corresponding microthread code is locally available (possibly after getting it from another site), the code manager informs the scheduling manager, which then puts the microframe from the executable-queue into the ready-queue. The processing manager is always given a ready microframe. Help requests from other site's scheduling managers are answered by sending ready or—preferably—executable microframes.

the current composition of the cluster, the physical (IP) addresses of other sites, data about the local site (e.g. performance data), and the list of currently running applications and where to find their microthreads.

The SDVM can be used to simulate a multi-core processor on a computer cluster, but it may even be run on a real multi-core processor, as well. Due to these features, the SDVM offers the convenient mechanisms to support different power states of processing units in a SoC. The thereby realized power management is described in the following section.

3. RELIABILITY AND TEMPERATURE

The long-term reliability of a processor is affected by its operating temperature as well as thermal cycling. The effect of the temperature can be modeled by the Arrhenius equation, which describes the influence of the temperature on the rate of chemical reactions. In terms of MTTF(Mean-Time-To-Failure), we then have [2]

$$MTTF \sim e^{\frac{E_a}{kT}} \tag{1}$$

where T is the operating temperature in Kelvin, k is Boltzmann's constant, and E_a is the activation energy in electron volts of the precise failure mechanism considered. The Arrhenius equation is the basis for modeling the temperature-dependence of several failure mechanisms. For instance, failure due to electromigration in interconnects can be modeled with the equation [5]

$$MTTF_{EM} \sim A_0(J - J_{crit})^{-N} e^{\frac{E_a}{kT}} \tag{2}$$

where J is the current density, J_{crit} is the critical current density for electromigration and A_0 and N are empirically determined constants. The activation energy E_a then depends on the material used for the interconnect and varies from 0.5 to 0.9 eV [5]. Other failure mechanisms like stress migration or hot carrier injection have different activation energies.

With the knowledge of the physical and structural construction of a chip, the models for different failure mechanisms can be combined to get a model (like RAMP [2]) for the processor's reliability. As we make no assumptions on the internal structure of the processors or the materials used, it would make no sense to use those detailed models for our purposes. Instead, we use equation 1 as a generic temperature-dependant reliability measure for processors. For E_a we use a value of 0.9 eV.

The temperature of a processor depends on its power consumption, and since dynamic power management lowers the average temperature, it should contribute to the chip's lifespan. But, as it was noted in [4], the switching between different power consumption levels leads to thermal cycling, which can cause various types of failures like lifted bonds, solder fatigue or even a cracked die [5]. The effect of thermal cycling on the reliability of a chip can be modeled by the Coffin-Manson relation, which computes the number of cycles to failure, N_f, as [5]

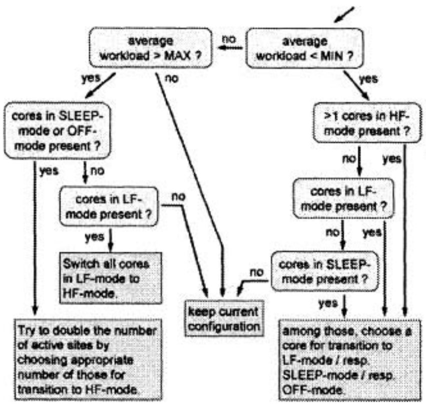
$$N_f = C_0 \cdot (\Delta T)^{-q} \tag{3}$$

where ΔT is the magnitude of thermal cycling, C_0 is a material-dependant constant, and q is the empirically determined Coffin-Manson exponent. This exponent depends on the failure mechanism considered; we use a value of 1.9, which focuses on the reliability of the package [2].

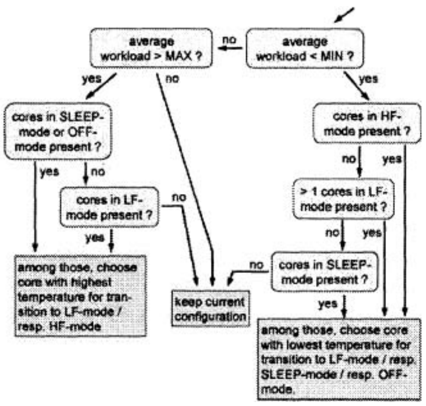
For our purposes, we use equations 1 and 3 for a comparitive analysis of different power management strategies to the non-powermanaged case to get an *acceleration factor* (i.e. the ratio) for each of the PM-strategies described below. Therefore, we don't need to choose a value for C_0 in equation 3, since it then cancels out.

3.1 RELIABILITY AWARE POWER MANAGEMENT

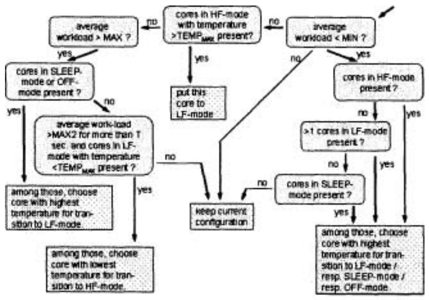
In view of the previous section, a power management strategy which is aware of reliability issues should limit the temperature as well as temperature changes. While the first is a side effect of usual power management strategies, the latter might involve keeping a processor "powered up", although this might not be necessary regarding performance, and is definitely not desirable regarding power consumption. So obviously, there's a trade-off between power consumption, performance and reliability.


average workload < MIN ?
no
average workload > MAX ?
yes
no
>1 cores in HF-mode present ?
cores in SLEEP-mode or OFF-mode present ?
yes
no
cores in LF-mode present ?
yes
Switch all cores in LF-mode to HF-mode.
Try to double the number of active sites by choosing appropriate number of those for transition to HF-mode.
cores in LF-mode present ?
cores in SLEEP-mode present ?
keep current configuration
among those, choose a core for transition to LF-mode / resp. SLEEP-mode / resp. OFF-mode.

(a) fast-upgrade policy


average workload < MIN ?
no
average workload > MAX ?
yes
no
cores in HF-mode present ?
cores in SLEEP-mode or OFF-mode present ?
yes
no
cores in LF-mode present ?
among those, choose core with highest temperature for transition to LF-mode / resp. HF-mode
> 1 cores in LF-mode present ?
cores in SLEEP-mode present ?
keep current configuration
among those, choose core with lowest temperature for transition to LF-mode / resp. SLEEP-mode / resp. OFF-mode.

(b) smooth-temperature policy


average workload < MIN ?
no
cores in HF-mode with temperature >TEMP_MAX present?
yes
put this core to LF-mode
average workload > MAX ?
cores in SLEEP-mode or OFF-mode present ?
average work-load >MAX2 for more than T sec. and cores in LF-mode with temperature <TEMP_MAX present ?
among those, choose core with highest temperature for transition to LF-mode.
among those, choose core with lowest temperature for transition to HF-mode.
keep current configuration
cores in HF-mode present ?
>1 cores in LF-mode present ?
cores in SLEEP-mode present ?
among those, choose core with highest temperature for transition to LF-mode / resp. SLEEP-mode / resp. OFF-mode.

(c) low-temperature policy

- In the PM-state "HFM" (High Frequency Mode), the core runs with maximum clock frequency and supply voltage.
- The PM-state "LFM" (Low Frequency Mode) has lower clock frequency and supply voltage, yielding less performance and power consumption, but full functionality.
- If a core is in the "SLEEP" state, it is switched off while its state is stored, so it can resume computation pretty quickly. The power consumption is reduced.
- The PM-state "OFF" is self-explanatory; the state of the core is not stored and the power consumption is minimal. Resumption of the computation takes more time than in SLEEP-mode.

(d) four PM-states

Figure 3. Power management policies.

In our simulation, we consider two reliability aware dynamic power management (RADPM) strategies: The low-temperature-policy, which tries to keep the temperature as low as possible, and the smooth-temperature-policy, whose goal is to restrict thermal cycling. These policies are compared to the (reliability unaware) fast-upgrade-policy, which tries to optimize performance and serves as a representative of usual power management strategies. The simulated computing environment is a homogenous multi-core-processor with four cores. Each core has four different Power-Management states (see Figure 3(d)).

Figure 3 shows diagrams describing the three PM-policies in detail. Note that the parallelization of the applications is influenced *indirectly* by altering the PM-states and thus the performance of the different cores.

4. IMPLEMENTATION AND RESULTS

The aforementioned power management capabilities were integrated into the SDVM by implementing the so-called *energy manager*. This energy manager has a master mode and a slave mode. Only one core's energy manager is in master mode (the *master core*), which then controls the PM-states of all cores. The main task of the energy managers in slave mode is to listen to the master core and to implement its orders, setting the local site to the desired PM-state. If a slave energy manager observes the absence of the master core (due to a crash or shutdown), it starts an *election* of a new master core.

The basis for the decision for a new power configuration is the temperature and the mean workload of each core. This information is distributed through the cluster by the SDVM's cluster manager's message mechanism.

The test set-up simulates a homogenous multi-core processor with four cores. To this end, the SDVM runs on a cluster of four identical computers. To each PM-state, a typical power consumption value based on an Intel Pentium M processor [7] is assigned (see Table 1).

Table 1. PM-states and their power consumption

PM-state	HF	HF$_{idle}$	LF	LF$_{idle}$	SLEEP	OFF
power consumption	15 W	10 W	7.5 W	4 W	3 W	0.2 W

The temperature T_J of a core is determined out of its power consumption by the formula

$$T_J = T_A + \theta_{AJ} \cdot P_{DISS} \qquad (4)$$

Where T_A is the environmental temperature, θ_{AJ} is the thermal resistance of the core, and P_{DISS} is the power consumption. For θ_{AJ}, a value of $4.5°C/W$ is used.

With this set-up, each PM-strategy (and the "no-PM strategy" as a reference) was simulated using identical workloads composed of multiple instances of a parallelized example application (Romberg integration [6]). The results of the simulations of the PM-strategies are given in figures 4, 5, and 6 showing the workload (area chart) and the temperature (black line) of the four cores for the fast-upgrade, smooth-temperature and low-temperature policy respectively.

The figures 4, 5, and 6 show a clear difference between the three policies. The low-temperature policy restricts the maximum temperature to

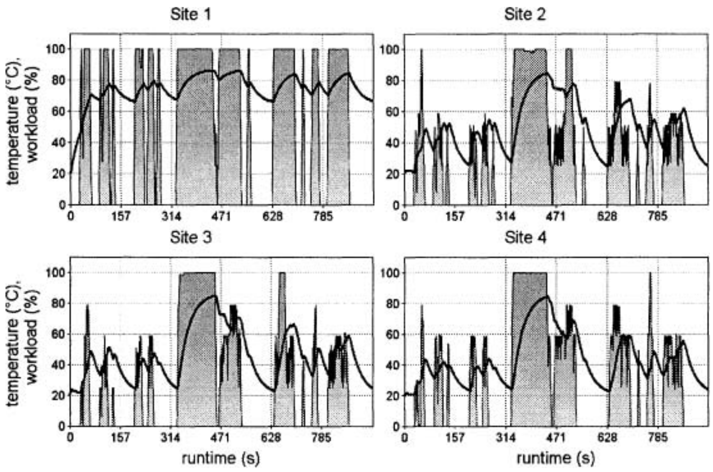


Figure 4. Fast-upgrade policy.

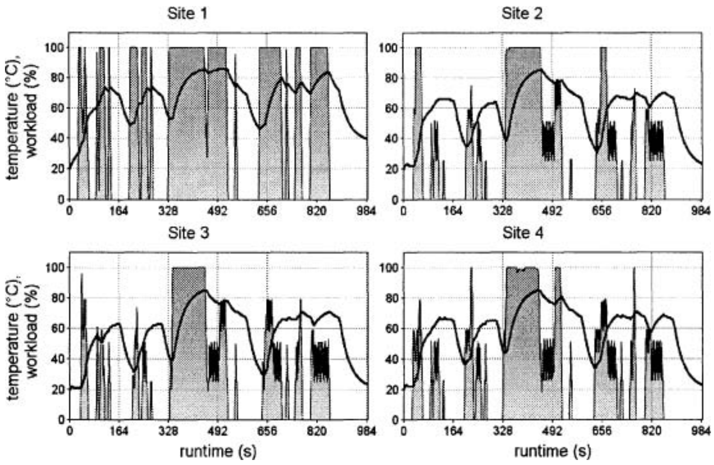


Figure 5. Smooth-temperature policy.

$61°C$, while with the other two policies a maximum temperature of $86°C$ is obtained. The higher temperature of core 1 in figure 4 is caused by the fact that the fast-upgrade policy always leaves one core in HF-mode.

Regarding thermal cycling, we see a reduction both in frequency and magnitude by the smooth-temperature policy compared to the fast-upgrade policy. Because of the temperature limitation, the low-temperature policy causes thermal cycling of lower magnitude, but also with higher frequencies, expecially when the temperature limit is reached.

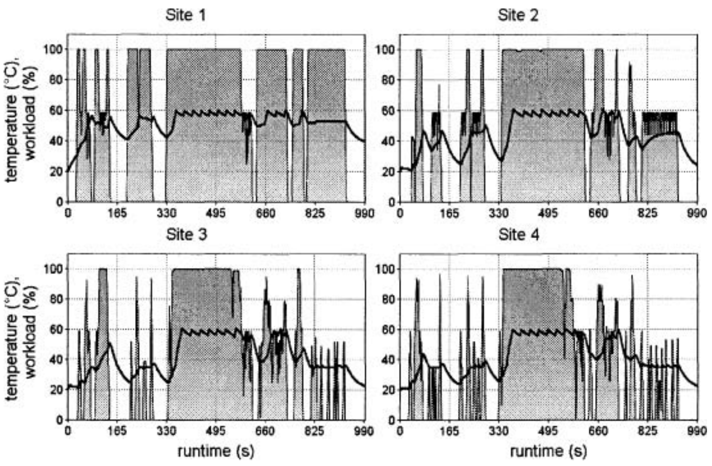


Figure 6. Low-temperature policy.

Table 2. AF_T, AF_{TC}, mean runtime, and mean power consumption of all cores

PM-policy	AF_T	AF_{TC}	mean runtime	power consumption
no PM	1	1	32.7s	48.15 W
fast-upgrade	0.12	3.27	35.0s	31.55 W
smooth-temperature	0.19	1.2	35.9s	37.29 W
low-temperature	0.1	3.28	64.8s	23.18 W

Using the models described in section 3, we computed for each policy the acceleration factors AF_T and AF_{TC} giving the acceleration of the time to failure due to temperature and temperature cycling respectively. Table 2 gives the means of these values over all cores, together with the mean runtime and the mean power consumption.

The acceleration factors without power management are 1, since this case is the reference. We see that all PM-strategies are beneficial regarding failure due to temperature, especially the low-temperature and the fast-upgrade policy. The fact that the low temperature policy is not much better than the fast upgrade policy regarding AF_T (despite the lower maximum temperature) is owed to prolonged computation durations of the first whereas the latter has shorter computation durations which leaves more time for cooling down. In view of thermal cycling, the smooth-temperature policy is the clear winner, while the other two policies show obvious acceleration.

This clearly shows that the reliability of a multi-core chip can be influenced actively with PM-strategies. It should be pointed out that such an approach is only possible using dynamic power management, which in turn can be implemented only within a system which distributes the workload dynamically, as the SDVM does. Incorporating reliability awareness into compile-time power management schemes seems to be almost infeasible.

5. CONCLUSION

In this paper, we proposed *reliability-aware dynamic power management* (RADPM), which incorporates lifespan-controlling goals. The usability of RADPM to prolong system-lifetime was demontrated by simulating a multi-core chip on the Self Distributing Virtual Machine (SDVM). The SDVM was augmented for this purpose with the so-called *energy manager*, which implements different PM-policies. The basic approach, however, could be implemented on any multi-core system which distributes the workload dynamically.

The PM-policies presented are no final solutions for RADPM, but serve as a proof of concept, that the long-term reliability of a multi-core chip can actually be altered deliberately with RADPM. Real implementations for RADPM on mult-icore chips could include, for example, a "reliability account" for each core or could consider the geometric configuration of the cores on the chip to optimize the temperature distribution.

A new insight, however, is that parallelism may not only be used to improve performance, but to improve reliability as well.

The SDVM's homepage containing its complete source code and documentation can be found at: `http://sdvm.ti.cs.uni-frankfurt.de`.

REFERENCES

[1] ITRS, "Critical reliability challenges for the international technology roadmap for semiconductors," 2003, international Sematech Technology Transfer document 03024377A-TR.

[2] J. Srinivasan, S. V. Adve, P. Bose, J. Rivers, and C.-K. Hu, "Ramp: A model for reliability aware microprocessor design," in *IBM Research Report, RC23048 (W0312-122)*, Dec. 2003.

[3] J. Srinivasan and et al., "The case for lifetime reliability-aware microprocessors," in *Proc. of the 31st Annual Intl. Symp. on Comp. Architecture*, 2004.

[4] K. Mihic, T. Simunic, and G. D. Micheli, "Reliability and power management of integrated systems," in *DSD - Euromicro Symposium on Digital System Design*, 2004, pp. 5–11.

[5] JEDEC, "Failure mechanisms and models for semiconductor devices," 2003, jEDEC Publication JEP122-B, Jedec Solid State Technolgy Association.

[6] J. Haase, F. Eschmann, B. Klauer, and K. Waldschmidt, "The SDVM: A Self Distributing Virtual Machine," in *Organic and Pervasive Computing – ARCS 2004: International Conference on Architecture of Computing Systems*, ser. Lecture Notes in Computer Science, vol. 2981. Heidelberg: Springer Verlag, 2004.

[7] Intel, "Pentium M Processor Datasheet," Apr. 2004, http://www.intel.com/design/mobile/datashts/252612.htm.

EVALUATING ENERGY-AWARE TASK ALLOCATION STRATEGIES FOR MPSOCS

Fabio Wronski, Eduardo Wenzel Brião, Flávio Rech Wagner
Universidade Federal do Rio Grande do Sul
Instituto de Informática

Abstract: Because of current market trends, the evaluation of task allocation strategies in multiprocessor system-on-chips (MPSoCs) must take into account both performance and energy consumption. Furthermore, complex interconnection structures, such as networks-on-chip (NoCs), must be considered. Simulators for the evaluation of energy consumption of detailed communication patterns in NoCs are available, as well as performance simulators that consider detailed task execution in processors. However, in order to evaluate task allocation strategies in MPSoCs, these two types of simulation models must be combined, since communication and computation events interfere with each other. Besides that, this simulator must implement low-power mechanisms, such as dynamic voltage scaling (DVS), in order to evaluate allocation algorithms that explore the trade-off between performance and energy. A cycle-accurate simulation of the processor and communication behaviors, however, would be too time-consuming, making impossible a fast exploration of different allocation algorithms. This work presents an MPSoC simulator that implements the appropriate abstractions for a precise evaluation of the energy consumption of task allocation algorithms that explore DVS, which is based on the scheduling of synthetic task graphs. A NoC mesh topology is considered, due to its simplicity and scalability. Experiments that implement the allocation of task graphs using different bin-packing heuristics combined with DVS demonstrate the energy-performance design space that may be explored by task allocation algorithms.

Keywords: Networks-on-Chip, Task Allocation, Energy Estimation, Simulation, Multiprocessor SoCs.

Please use the following format when citing this chapter:

Wronski, F., Brião, E.W., Wagner, F.R., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 215–224.

1. INTRODUCTION

Due to technology and market trends, multiprocessor systems-on-chip (MPSoCs) are being increasingly used as platforms for embedded systems. These systems require a highly scalable and parallel communication infrastructure, such as networks-on-chip (NoCs). Especially mesh networks have been proposed, due to their simplicity. Energy consumption is one of the critical aspects in the design of embedded systems. Low-power techniques, such as voltage scaling[1], are essential and must also be applied in the context of NoC-based MPSoCs.

MPSoCs run several parallel tasks, which must be allocated into the various processors. Allocation algorithms have been usually evaluated in terms of performance and timing metrics, but now energy consumption also becomes a very relevant factor.

In this context, several estimation tools have been proposed. Examples are SimDVS[2], for evaluating dynamic voltage scaling (DVS); CAFES[3], for evaluation of communication costs; and Xpipes[4], which is based on the Orion[5] library, for energy estimation. SimDVS simulates only individual processors, while Xpipes and CAFES simulate only the NoC infrastructure.

For the evaluation of task allocation strategies, however, both the task scheduling and execution in the processors and the communication traffic must be simulated together, since they interfere which each other. A cycle-accurate simulation of the processor and communication behaviors, however, would be too time-consuming, making impossible a fast exploration of different allocation algorithms.

This work presents an MPSoC simulator that implements the appropriate abstractions for a precise evaluation of the energy consumption of task allocation algorithms that explore DVS, which is based on the scheduling of synthetic task graphs. The simulator has been implemented in SystemC TLM (transaction level model), which offers the required abstractions and synchronization mechanisms.

The processor model allows the application of DVS. For the calculation of the energy consumed by each synthetic task running in the processor, a random parameter representing the number of switching gates per cycle has been included in the model. For a more precise evaluation, task scheduling costs are also considered. The energy consumed by the NoC in the communication between tasks allocated to different processors is evaluated by the Orion library, as in Xpipes.

In the experiments, synthetic task graphs have been allocated using simple bin-packing heuristics. Results show the benefits of DVS, even when inefficient allocations are applied and generate too much communication through the network. The trade-offs between energy and performance may be evaluated and suggest that reducing the communication is a good heuristic for minimizing the energy consumption, although the combined use of DVS brings more significant gains.

The remaining of this paper is organized as follows. Section 2 discusses related work. The energy and task models are presented in Section 3. The simulator implementation is discussed in Section 4. Section 5 shows experimental results, and Section 6 draws main conclusions and discusses future work.

2. RELATED WORK

As the dynamic power consumption is dominating in CMOS circuits, several architecture-level simulators have been developed to evaluate and estimate power consumption caused by the charging and discharging of the capacitive load on each gate output of the circuit. Two techniques in the literature are often used: calculation of power consumption by the average number of gates switching in processor instructions[6], and the calculation of power consumption by the number of gates switching in architectural components[7].

A simulator for NoC architectures is Xpipes[4]. It was developed in SystemC and consists of a customizable library of network components, which can be configured and instantiated by the developer. The calculation of energy is achieved using the Orion[5] library. This library has low-level capacitance models of NoC router components that allow a more accurate estimation of dynamic power consumption. These components can be buffers, crossbars, and arbiters.

SimDVS[2] is a unified simulation environment for evaluating dynamic voltage scaling (DVS) algorithms, which has task graphs as input. Some models of state-of-the-art processors are already embedded in the environment, but other models can be developed and included by the users. However, SimDVS supports only single processors, not including models for communication architectures like NoCs.

There are several heuristic approaches for task allocation in NoCs that try to minimize energy consumption or optimize other metrics, under energy constraints. Hu and Marculescu[8], for instance, present an approach for applying a static task allocation strategy in NoC architectures under real-time constraints, aiming at energy reduction. The application is modeled by a graph based on a Communication Weight Model (CWM), where arc weights represent the amount of bits exchanged between vertexes. Their approach, however, does not consider the exact simulation of tasks running in the processors. Besides, our work aims at the evaluation of different allocation approaches and not at the implementation of a single approach.

The CAFES[3] simulator is a framework aiming at the evaluation of several communication models that may be used for the analysis of energy consumption in NoC architectures. However, CAFES does not consider scheduling costs.

In this work, scheduling costs are considered. Besides, our simulator uses the same Orion library for power estimation as Xpipes, but we consider only the mesh architecture with some configurable parameters, instead of a broader library. We rely on a processor model that allows application scheduling, using a task graph model approach similar to SimDVS, while Xpipes uses cycle accurate models for simulating the cores. This Xpipes approach makes high-level energy estimation from task graphs unfeasible.

3. ENERGY AND TASK MODELS

Our energy model considers only dynamic energy consumption, which is still dominant in current technologies, although static power is becoming important. A static energy model is being currently implemented.

In a same time interval, two different task allocations generate different gate switching patterns in the processors and thus different dynamic energy consumptions. In CMOS circuits, the dynamic energy consumption is expressed as:

$$E = \frac{\alpha C}{2} V_{dd}^2$$

where C is a constant that represents the gate capacitance in a given technology; α is a number of gate switchings; and V_{dd} is the circuit supply voltage. If voltage scaling is applied, this parameter is not fixed.

In a processor, each instruction generates a different value for α, which depends on the previous processor state. In a system composed by n tasks $k \in K$, the total α is expressed as:

$$\alpha_{total} = \sum_{i=1}^{n} k_i^{\alpha_{total}}$$

where $k_i^{\alpha_{total}}$ is the total number of switchings of task k_i.

Variable V_{dd} presents a quadratic factor, and it is thus natural that the minimization of supply voltage leads to important energy savings. But reducing the voltage causes the frequency operation to be reduced too, following the equation below[9]:

$$V_{norm} = \beta_1 + \beta_2 f_{norm}$$

where $\beta_1 = V_{th} / V_{max}$ and V_{th} is the threshold voltage, $\beta_2 = 1 - \beta_1$, and V_{norm} and f_{norm} are the voltage and the frequency, both normalized regarding V_{max} and f_{max}. For a 100 nm technology, we have $\beta_1 = 0.3$.

A metric for power consumption evaluation in NoCs is Bit Energy[10]. It defines the power consumed when a data bit is transferred between two routers. This way, we define the energy spent by the transfer of a single bit from node p_i to node p_j as:

$$E_{bit(i,j)} = (\eta - 1) \times (E_{S_{bit}} + E_{B_{bit}} + E_{L_{bit}})$$

where η is the number of routers in the way from p_i to p_j, $E_{S_{bit}}$ is the energy spent in the crossbar inside one router, $E_{L_{bit}}$ is the energy spent in communication links, and $E_{B_{bit}}$ is the energy spent in the buffers. We don't consider the energy $E_{L_{bit}}$ spent in the links, since energy consumption in the routers is much larger.

The Orion[5] library implements an energy estimator for the crossbar and the buffers inside the router. Buffers are usually responsible for 90% of the energy consumption in the router.

The task and communication architecture models used in this work and presented below are based on Hu and Marculescu[8].

Each application is a directed acyclic graph $T = G(K, A)$, where each node $k_i \in K$ is a periodic task and each arc $a_{i,j} \in A$ is a dependency or flow of messages between tasks k_i and k_j. The arc weight $a_{i,j}^W$ represents the amount of bits to be transferred between the tasks.

Each task $k_i \in K$ is a tuple $\{C, T, D, \alpha\}$, where C is the worst case execution, T is the task period, D is the task deadline, and α is the average number of gate switchings per cycle of the task in the core.

The communication architecture is represented by a directed graph $G(P, R)$, where each node $p_i \in P$ represents a core and each directed arc $r_{i,j} \in R$ is a path between p_i and p_j. A core $p_i \in P$ has its own operation frequency p_i^F, cycle period p_i^T and voltage supply p_i^V. In our work, the communication architecture graph represents a mesh network.

4. SIMULATOR

The simulator has been implemented in SystemC, due to its support to the development and simulation of hardware models with the appropriate abstractions for our purposes.

The model of the routers has been implemented at the RT level and is thus very precise in terms of timing and energy. The core model, in turn, in fact considers only the task scheduler, which simulates the states through which the processor goes during the execution of the synthetic tasks. This simplification makes possible a high-level and fast evaluation of the energy consumption, without requiring the development and execution of real applications. As a drawback, accuracy is lost, since the model has a statistical nature.

The simulator uses the RaSoC[11] routers, designed for the synthesis of low power and low area NoC-based embedded systems in FPGAs. The RaSoC architecture utilizes wormhole packet switching, XY routing algorithm, and handshake control flow. Its energy evaluation was implemented with the Orion library. Each router has 5 bi-directional ports with input buffer size of 4 phits. The phit size is 4 bytes.

The task scheduler in each core is based on the implementation of a priority-based scheduling kernel. The scheduling policy implements an EDF (earliest deadline first) algorithm. For implementing the task dependencies, a list-scheduling algorithm was combined with EDF. Additionally, the AVR[12] (average rate) algorithm was implemented for DVS.

Each task is characterized with respective WCETs and gate switchings per cycles. With these data, tasks are scheduled and their energy consumptions are evaluated. Since the worst case execution time rarely occurs when the task is executed, there is a slack in the scheduling. To simulate a more realistic behaviour, a parameter called *slack* is used to define a range between the best and worst case execution times. By default, it has a value of 30%, which means that the minimum value of the best case execution time may represent 70% of the WCET. For each task execution, the simulator randomly chooses a value for its current execution time, within the allowable range. In addition, the scheduler time and costs are also considered, including the cost of an "idle task", which runs when no other task is available.

5. EXPERIMENTS

For the experiments, synthetic task graphs generated by the TGFF tool[13] has been used. The scheduler costs are obtained by simulation, with the CACO-PS[7] tool, of an API RTSJ[14] (Real-Time Specification for Java) for the FemtoJava[15] processor. The technology selected for the experiments is 100 nm. The NoC is a 3 x 3 mesh running at 200 MHz.

The task graphs have been allocated using the Worst-Fit and Best-Fit bin-packing heuristics[16]. In bin-packing, the objective is to pack a set of items with given sizes into bins. Each bin has a fixed capacity, and items whose total size exceeds this capacity cannot be assigned to the bin. The goal is to minimize the number of bins used. In an analogous way, items may represent tasks, or entire task graphs, while bins represent processors with a given processing capacity. The size of each item corresponds to the processor utilization by the task. The Best-Fit (BF) strategy places a new object in the bin whose remaining space will be the smallest one. The Worst-Fit (WF) strategy, in turn, places an object in the bin whose remaining space will be the largest one. As a consequence, WF generates a task distribution with load balancing, while BF generates a distribution that is concentrated in some bins.

Ten task graphs were generated with processor utilizations between 5 and 15% each, resulting in an overall utilization of 95% of one core running at 600 MHz speed and 3 V supply voltage. Each task has a WCET of 1 ms in average. We call this the allocation A (all task graphs running in a single processor).

In the following allocations, the task graphs have been distributed over the NoC. In allocations B and C, each task graph has been entirely allocated in a single processor. Allocation B used BF, while allocation C used WF. We call these strategies BF-TG and WF-TG, respectively. A final allocation D also used WF, but considering individual tasks within each graph (and thus using a much finer grain for the allocation).

Table 1. Allocations and utilizations

Core	BF-TG (B)			WF-TG (C)						WF-TK (D)		
	0	1	2	0		1		2		0	1	2
	%	%	%	%	MHz	%	MHz	%	MHz	%	%	%
0	96	93	96	45	90	39	78	36	72	33	33	28
1	-	-	-	33	66	30	60	27	54	33	29	34
2	-	-	-	24	45	20	40	30	60	29	29%	34

Table 1 shows the processor utilizations resulting from these allocations. The table represents the 3x3 matrix of processors in the NoC. With BF-TG (allocation B), only 3 cores were used, with almost 100 % utilization each, while with WF-TG (allocation C) the load has been distributed among all 9 processors. With WF-TK (allocation D), we could achieve a more evenly distributed load than in allocation C, since allocation items (individual tasks) are much smaller. However, allocation D introduces inter-processor communication, since tasks from a single task graph have been distributed over different processors, while allocations B and C have only intra-processor communication.

In the first experiment, allocations A, B, and C are simulated and compared. Allocation A maps all task graphs into a single core running at 600 MHz, while allocation B divides the graphs between 3 cores, which may run now at 200 MHz each. Allocation C has been simulated twice. In the first simulation, a single frequency of 90 MHz is used for all 9 cores. Since Table 1 shows that there is a maximum utilization of 45% for a processor in allocation C, the frequency can be reduced to 45% of the original 200 MHz. In the second simulation of allocation C, the frequencies of the processors have been individually adjusted from the original 200 MHz, proportionally to the utilizations in Table 1. These individual frequencies are also shown in Table 1 and vary from 40 MHz to 90 MHz.

Table 2 summarizes the results for energy consumption of the first experiment, when task graphs run during 1 second. Results show the impact of frequency operation reduction on system energy consumption. When the frequency is reduced by a factor of 3 (from 600 to 200 MHz), even using 3 cores instead of one (allocation B), the consumption is reduced to 235 mJ (28 % from the energy in allocation A). When the task graphs are divided among 9 cores (first simulation of allocation C), all of them running at 90 MHz, the consumption is reduced to 137 mJ (16% of the value in A). When the frequency of each processor is adjusted to the load it receives, the energy consumption is reduced again to 111 mJ (13% of the value in A). This shows that, when there is a bad load balancing between the processors, it is worthwhile to apply different supply voltages.

Table 2. Experiment 1 results in mJ.

Tasks	(A) on 1 core at 600 MHz	(B) on 3 cores at 200 MHz (A)	(C) on 9 cores at 90 MHz	(C) on 9 cores at different frequencies
Idle	0.00	0.47	4.93	0.44
Scheduling	3.38	0.94	0.27	0.22
Application	842.62	233.59	131.79	110.46
Total	846.00	235.00	137.00	111.12

As expected, allocation C is the best in this first experiment. These results confirm the experiment of Aydin and Yang[17], which shows that a balanced allocation is more energy efficient than a concentrated one, when a voltage scaling approach is applied.

The energy consumption of the scheduler remains with a very low value, due to the relationships between tasks lengths and scheduler length, since the scheduler is called once per task in average. However, this depends on applications and tasks, as well as on the scheduler implementation.

In the second experiment, we compare allocations C (WF-TG) and D (WF-TK), both considering all processors running at 200 MHz. Table 3 shows the average time spent by each processor in different activities (communicating, scheduling, executing the application, or idle). It must be remembered that allocation C concentrates each task graph in a single processor, so that there is no communication between processors. It can be noticed that communication in allocation D consumed 24.45% of the execution time, around 35% of the idle time in allocation C. With an even smaller slack, task deadlines could be missed due to the communication overhead. Scheduling time in allocation D is three times larger than in C, because in this case processors also receive interference from tasks in other processors, such that the scheduler must execute more often.

Table 3. Allocations C and D at 200 MHz.

Task	Allocation C		Allocation D	
	Time (%)	Energy (mJ)	Time (%)	Energy (mJ)
Communication	0.00	0.00	24.45	164.35
Idle	66.88	44.13	36.30	24.65
Scheduling	0.08	0.48	0.26	1.67
Application	33.04	239.38	38.98	289.25
Routers	-	0.00	-	10.09
Total	100.00	283.99	100.00	490.01

The energy consumption increased around 75%, from 284 mJ in allocation C to 490 mJ in D. This is only due to the task allocation strategy and the communication overhead it imposes, since processors have the same frequency. This shows that, considering the inter-processor communication overhead, a good load balancing is not enough for energy minimization.

The energy consumed by the routers is relatively small and does not represent a significant part in this communication overhead.

Even with its communication overhead, when allocation D is compared to the original allocation A (a single processor running all applications and consuming 870 mJ), there is still a reduction of almost 50% in energy consumption. This demonstrates the necessity of mechanisms that combine task allocation strategies that reduce communication overhead with the voltage scaling technique.

6. CONCLUSIONS AND FUTURE WORK

This work introduced a novel model for the simulation of synthetic task graphs that allows the evaluation and comparison of performance and energy consumption of various task allocation strategies in NoC-based systems. This model considers voltage scaling, which must be taken into account by more energy-efficient allocation algorithms, and presents the right abstractions to consider scheduling, communication, and computation costs.

As demonstrated by experiments, two different allocations that have similar timing performances may result in very distinct energy consumptions, because of the communication overhead that may be imposed by the task distribution. It has been also shown that this overhead is mainly imposed to the cores' execution, and not by the NoC itself. However, voltage scaling may considerably mitigate this cost.

Current work includes the development of a static energy consumption model, which will make possible a more precise evaluation of the impact of different task allocation strategies.

This work is a first step in a project aiming at the development and evaluation of on-line task allocation strategies for energy minimization in a dynamic load environment. The allocation algorithms that have been evaluated are very simple and could be applied on-line, without significant overhead, although they may be still improved. In this context, DVS also needs to be integrated to the scheduler.

REFERENCES

[1] M. Weiser, et al. Scheduling for Reduced CPU Energy. in: Symph. on OS Design and Imp., Monterey CA, 1994, pp. 13-23.

[2] D. Shin, et al. SimDVS: An Integrated Simulation Environment for Performance Evaluation of Dynamic Voltage Scaling Algorithms. in: Workshop on Power-Aware Computer Systems, Springer, Cambridge, MA, 2002, pp. 141-156.

[3] C. Marcon, et al. Modeling the Traffic Effect for the Application Cores Mapping Problem onto NoCs. in: VLSI-SoC, Perth, Australia, 2005, pp.

[4] D. Bertozzi and L. Benini, Xpipes: A Network-on-chip Architecture for Gigascale Systems-on-Chip. IEEE Circuits and Systems Magazine, 4(2): 18-31 (2004).

[5] H. Wang. Orion: A power-performance simulator for interconnection networks. in: ACM MICRO, Istanbul, Turkey, 2002, pp. 294-305.

[6] V. Tiwari, et al., Power analysis of embedded software: a first step towards software power minimization. IEEE Trans. on VLSI Systems, 2(4): 437-445 (1994).

[7] A. C. S. Beck, et al. CACO-PS: A General Purpose Cycle-Accurate Configurable Power Simulator. in: SBCCI, Washington, DC, USA, 2003, pp. 349.

[8] J. Hu and R. Marculescu. Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures under Real-Time Constraints. in: DATE, Washington, DC, USA, 2004, pp. 10234.

[9] N. S. Kim, et al., Leakage Current: Moore's Law Meets Static Power. Computer, 36(12): 68-75 (2003).

[10] T. T. Ye, et al. Analysis of Power Consumption on Switch Fabrics in Network Routers. in: DAC, New Orleans, 2002, pp. 524-529.

[11] C. A. Zeferino, et al. RASoC: A Router Soft-Core for Networks-on-Chip. in: DATE, 2004, pp. 198-205.

[12] F. Yao, et al. A scheduling model for reduced CPU energy. in: FOCS, Washington, DC, USA, 1995, pp. 374.

[13] R. P. Dick, et al. TGFF: task graphs for free. in: CODES/CASHE, Washington, USA, 1998, pp. 97-101.

[14] M. A. Wehrmeister, et al. Optimizing Real-Time Embedded Systems Development Using a RTSJ-Based API. in: OTM Workshops, 2004, pp. 292-302.

[15] S. A. Ito, et al. Making Java Work for Microcontroller Applications. in: IEEE Design & Test, Los Alamitos, USA, 2001, pp. 100-110.

[16] D. S. Johnson, Near-optimal bin packing algorithms (Cambridge, Mass., 1973).

[17] H. Aydin and Q. Yang. Energy-Aware Partitioning for Multiprocessor Real-Time Systems. in: IPDPS, 2003, pp. 113.

INTEGRATION OF ENERGY REDUCTION INTO HIGH-LEVEL SYNTHESIS BY PARTITIONING*

Achim Rettberg and Franz Rammig
Paderborn University
Paderborn, Germany
achim@c-lab.de, franz@upb.de

Abstract: The optimization of power consumption at a very high design level is a critical step towards a power-efficient digital system design. The increasing usage of battery-powered and often wireless portable systems is driving the demand for IC and SoC devices consuming the smallest possible amount of energy. The aim of the method presented in this paper is to integrate low power methods within the scheduling process of the High-Level Synthesis by defining partitions. Starting from an Controlled-Data-Flow-Graph (CDFG) the proposed method uses standard scheduling techniques and path analysis on the graph to identify regions that can be combined to partitions. Each partition has a controlled activation or deactivation mechanism. That means, the partition can be switched off when it is not used. As an example design, a part of the MPEG-2 algorithm is used.

1. INTRODUCTION

The optimization of power consumption at a very high design level is a critical step towards a power-efficient digital system design. Furthermore creating optimal low power designs involves making tradeoffs such as timing-versus-power and area-versus-power at different stages of the design flow. Successful power-sensitive designs require engineers, having the ability to accurately and efficiently perform tradeoffs. In order to achieve this, engineers require access to appropriated low power analysis and optimization engines, which need to be integrated with and applied throughout the entire system design flow (see [7]).

We can distinguish between dynamic and static power dissipation. Dynamic power dissipation occurs in logic gates that are in the process of switching from one state to another. During the switching activity of the gates, any internal capacitance associated with the gates transistors has to be charged, thereby

*This work was partly funded by the Deutsche Forschungsgemeinschaft (DFG) in SPP Verfahren zur Verlustarmen Informationsverarbeitung (VIVA), 322 1076

Please use the following format when citing this chapter:

Rettberg, A., Rammig, F.J., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 225–234.

consuming energy. Static power consumption, however, is associated with the logic gates when they are inactive. In this case, these gates should theoretically not be consuming any power, but in reality, there is always some amount of leakage current passing through the transistors. That means those gates do consume a certain amount of power. In order to integrate optimization of power consumption at high design level it is necessary to modify the High-Level Synthesis (HLS) process of the system design flow.

2. RELATED WORK

For a low power behavioral synthesis system, automatic techniques must be developed to minimize the switching activity on globally shared busses and register files, to select low power modules while satisfying the timing constraints, and to schedule operations to minimize the switching activity from one cycle to another cycle. In the past, several algorithms for HLS have been developed. The major objective for all these algorithms was the minimization of the used resources to reduce chip area and the optimization of the system delay time [4]. An interesting approach for the integration of low power techniques into HLS is presented [10]. It focuses on the minimization of resources per cycle whereby energy consumption is reduced. This could be achieved by mapping the same operation types to a real resource. Most of the HLS perform scheduling of the Control and Data Flow Graph (CDFG) before the allocation of the registers and modules, like functional units, and synthesis of the interconnects (see [6] and [13]). Additionally, timing information for the allocation and assignment of various operations are provided. In other systems the resource allocation and binding, before scheduling, is performed to provide more precisely the timing information during the scheduling (see [5]). The work presented in [5] assumes that the scheduling of the CDFG has been done and performs the register allocation before the allocation of modules and interconnection. The work presented in [2] and [8] demonstrates that decisions at the behavioral level have a significant impact on power consumption of the final system implementation. The authors of [3] present a new technique for power optimization of control-dominated designs. This approach is an improvement of the existing techniques described in [9]. For control-dominated designs that typically consist of lots of sequential processes, scheduling is the most critical step during HLS. In our approach, we built intelligent partitions during scheduling. The developed approach is applicable to different target architectures. Especially, self-controlled architectures like the one presented in [12] are addressed. A self-controlled architecture has no global control unit; only distributed small control parts are present in the design. The data is assembled with so-called control information to direct the data content to the operation nodes by intelligent routers. Furthermore, our approach proposes mapping possibilities, to reduce resources with the effect of reducing leakage current.

3. POWER SCHEDULER

The aim of this work is to integrate low power methods within the scheduling process of the High-Level Synthesis (HLS). We call the developed system *Power Scheduler* (see [11]). From the input (CDFG), a scheduled CDFG is generated that supports low power saving. The idea is to have a partitioned graph, whereby each partition can be activated or deactivated by a guard. A guard could be implemented by using gated clocks, guarded evaluation or power down in our approach.
Following, a short overview of the developed method is given in details by specifying the different scheduling phases, but before the cost function for the *Power Scheduler* is described.

3.1 COST FUNCTION

As already known from the dynamic power dissipation is the product from the overall capacity is the square of the supply voltages and the frequency (see [7]).

The power of a node, for example a gate, is given by the following Equation: $P_{node} = \frac{1}{2} * C_L * V_{dd}^2 * f_{clk}$, whereas C_L is the capacity of the node and f_{clk} the clock frequency. When we multiply with the switching activity α_{node} of the node we get the dynamic power dissipation of a single node: $P_{dyn,node} = \frac{1}{2} * C_L * V_{dd}^2 * \alpha_{node} * f_{clk}$.

The switching activity is hard to predict during the HLS process, therefore, we take only the worst case into account $\alpha_{node} = 1$. Now we can calculate the total power dissipation of a design by summing up the dynamic power dissipation of all nodes. This is given in the following Equation: $P_{dyn,tot} = \sum_{i=1}^{n} P_{dyn,i}$, whereas n is the total number of nodes in the design. Later on, we will see that for our approach $P_{dyn,node}$ is used to calculate the dynamic power dissipation for a partition. A partition consists of a number of nodes. Therefore, the dynamic power dissipation of a partition p is given by: $P_{dyn,p} = \sum_{j=1}^{m} P_{dy,j}$, whereas m are the number of nodes inside the partition. It is necessary to add the costs for the components (partition control unit) needed to activate or deactivate the partition. This is illustrated in Figure 1. The left side of Figure 1 shows an un-partitioned design. In opposite to that, the right side shows how a control unit, a so-called guard, which activates or deactivates the execution of this design part, controls Partition 1.

The partition control unit is always active in the part of the circuit where the partition is embedded in opposite to the controlled partition. The power cost

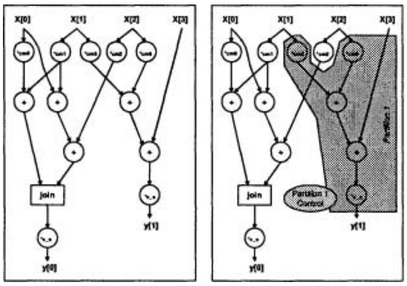


Figure 1. Partition with control unit.

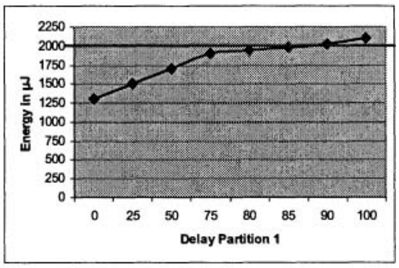


Figure 2. Energy reduction for example from Figure 1.

for a partition control unit (guard) of partition p is called $P_{gc,p}$. The power costs for $P_{gc,p}$ can be calculated by $P_{dyn,tot}$.

Now we can replace the number of nodes n in Equation for $P_{dyn,tot}$ by the number of partitions p of the entire design to calculate the total dynamic power dissipation of the design and adding the additional partitioning costs.

$$P_{design} = \sum_{j=1}^{p} P_{dyn,p} + P_{gc,p}. \tag{1}$$

Let us look at the example in Figure 1. What does equation 1 mean for the example? Let the dynamic power consumption of the entire design without partitioning $P_{exam} = 20\mu W/MHz$. Let $P_{dyn,Part1} = 8\mu W/MHz$. Thus the un-partitioned part has a dynamic power consumption of $P_{dyn,un-part} = 12\mu W/MHz$. In Equation 1 the un-partitioned part of the design is also considered as a partition ($P_{dyn,un-part}$), but without low power control. The power cost for the partition control is $P_{gc,Part1} = 1\mu W/MHz$. Here we assume the partition control consists of a gated clock.

That means, $P_{exam'} = \sum_{j=1}^{p} P_{dyn,p} + P_{gc,p} = P_{dyn,un-part} + (P_{dyn,Part1} + P_{gc,p}) = 12 + (8+1)\mu W/MHz = 12 + 9\mu W/MHz = 21\mu W/MHz$, but $P_{exam'} = 21\mu W/MHz$ is greater than $P_{exam} = 20\mu W/MHz$. Within this calculation we have not considered the run-time. Therefore it is necessary to include the run-time of the system into the Equation.

By taking the run-time of the system into account, we will calculate the *delay d* of entire design and of each partition. In literature, it is well known to evaluate different implementations of a design w.r.t low power, by calculating the *power-delay* product. Equation 2 shows the *power-delay* product for a partition k.

$$PD_k = (P_{dyn,k} * d_k) + (P_{gc,k} * d_l), \tag{2}$$

whereas d_k is the delay of the partition and d_l is the delay of the part of the circuit where partition k is embedded. Let us go back to the example. When we assume that the entire system has a run-time of 100 $cycles$ than the delay for $P_{dyn,un-part}$ is $d_{un-part} = 100\ cycles$. That means, the total energy for the design without low power reduction is $20\mu W/MHz * 100\ cylces = 2000\mu Ws = 2000\mu J$. The delay for the low power control of the partition is also 100 $cycles$. Let us further assume that the partition is only 50 $cycles$ active at run-time, the delay of $d_{Part1} = 50\ cycles$ By including all this values into Equation 2 we get: $PD_{Part1} = P_{dyn,Part1} * d_{Part1} + P_{gc,p} * d_{gc} = 8\mu W/MHz * 50 cycles + 1\mu W/MHz * 100 cycles = 400\mu Ws + 100\mu Ws = 500\mu Ws = 500\mu J$ and for the un-partitioned part of the system $PD_{un-part} = P_{dyn,Part1} * 100 cycles = 12\mu W/MHz * 100 cylces = 1200\mu Ws = 1200\mu J$.
The *power-delay* product of the entire design can now be calculated by summing up the *power-delay* product of all partitions (see Equation 3).

$$TPD = \sum_{x=1}^{l} PD_x, \tag{3}$$

whereas l is the number of design partitions. For our example we get: $TPD = PD_{Part1} + PD_{un-part} = 500 + 1200\mu Ws = 1700\mu Ws = 1700\mu J$.
Remembering that, for the design without our low power reduction method we need $2000\mu J$, we got an energy reduction of $300\mu J$ (15 %). Obviously, the energy reduction depends on the run-time of the system (see Figure 2). If $d_{Partition1}$ is less than $88s$ we reduce the energy, for this case.

Generally a HLS system tries to minimize the delay D and area A of a design. With the *TotalPowerDelay* function we have another constraint power P minimized by a HLS system. Therefore, we use Equation 3 as a cost function for our approach.

3.2 SCHEDULING FLOW

The CDFG consists of two different graphs, a so called Control-Flow-Graph (CFG) and a Data-Flow-Graph (DFG). Generally, the design of digital systems bases on a high-level specification, which is transformed into algorithmic descriptions, like C or behavioral VHDL source code. The HLS transforms the behavioral descriptions into structural ones. During the HLS the algorithmic description (CDFG) is transformed into internal formats. The CFG represents the controller for the DFG which is itself the data-path of the given algorithm. The *Power Scheduler* starts with a CDFG, which describes the design, with each node corresponding to operations and control steps and each directed edge representing data dependency and control order.
The *first* step of the *Power Scheduler* is to read the CDFG that consists of a

CFG and DFG, and store the graphs into the internal data format. The *second* step of the *Power Scheduler* is to use As-Soon-As-Possible (ASAP) and As-Late-As-Possible (ALAP) scheduling on the DFG. ASAP scheduling means that all operations are scheduled as soon as they can processed in the sense of time. Vice versa ALAP means that all operations are scheduled as late as possible. If both scheduling methods are processed it is possible to calculate the mobility of each operation within the DFG. Mobility means the degree of freedom the operation has within the scheduling.
The next important step, the *third* one of the *Power Scheduler*, is to calculate the pathes within the DFG. This step consists of three different phases. The first phase examines all disjoint paths of the DFG. Eventually some of these paths are not active during the entire run-time of the system. In the second phase fork and join nodes of the DFG will be examined. The different paths between the fork and join nodes are the basis for the partitioning construction, because they are alternatively active during the run-time. In the third phase control nodes of the CFG are examined. That means, if it is applicable to schedule them as soon as possible, different paths can be identified which are alternatively active during run-time. The examined paths are the basis of the partitioning and they could be, again combined to partitions. All paths are nodes in a so called compatibility graph.
The *fourth* step of the *Power Scheduler* is to build the partitions by combining the paths that are calculated in the second step To do this it is necessary to examine if there are so-called conflicts between the paths. Two paths are in conflict to each other if they are, for example, both depending from the same fork, join or control node. That means, paths with a conflict have no edge between each other in the compatibility graph. Then a clique search algorithm [1] is used to find cliques in the compatibility graph. Finally, a clique builds a partition than can be activated or deactivated during the run-time to save energy.
From the perspective of the reader of this paper, two questions are opened. Why is it important to build partitions to save energy instead of controlling each single node? Why not use each calculated path as a partition? The answer to both questions is the same. The insertion of activation or deactivation mechanisms into a circuit has also power costs in the data-path as well as in the controller. To reduce these additional costs it is necessary to combine the paths to partitions. Nevertheless, it could be possible that after the clique approach a partition contains only one path.

3.3 PATHS ANALYSIS

As described before the path analysis consist of three different phases. In the first one we examine disjoint pathes, followed by path analysis between

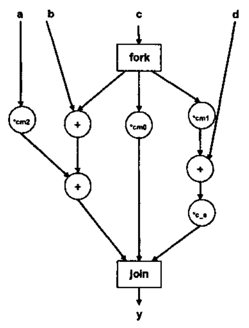


Figure 3. Paths between fork-join nodes.

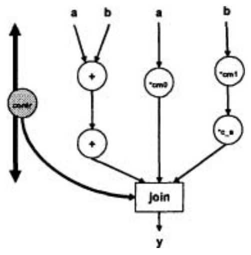


Figure 4. Control nodes path analysis.

fork and join nodes. In the third phase the scheduling of control nodes are important. A path is defined as follows:

DEFINITION 3.1 (PATH) *A path p_{v_s,v_e}, with $i \in I\!N$, is a connection from a source v_s to a destination node v_e to transport a data-word within the DFG $G = (V_d, E_d)$. Whereas v_s, $v_e \in V_d$. All nodes $v_i \in V_d$ between v_s and v_e and all nodes that are necessary to provide the correct operation of the path are objects: $p_{v_s,v_e,i} = \{v_j, \cdots, v_n\}$ with j, $n \in I\!N$ and index $i \in I\!N$.*

The index is necessary if different alternatives paths between the corresponding nodes exist. Furthermore, we need the time of each path p_{v_s,v_e}, which is defined as follows:

DEFINITION 3.2 (PATH-TIME) *Let time($p_{v_s,v_e,i}$) the time necessary to send one data-word from the start node v_s to the end node v_e of $p_{v_s,v_e,i}$.*

The path identification can be realized by *Depth-First-Search* (DFS), starting from source to the destination node and to the primary inputs or outputs that are necessary to realize the path.
To find all disjoint paths we modify the DFG slightly by including a virtual source and destination node. The source node is connected by edges with the primary inputs and constants of the DFG, because these are the only elements from where data goes into the circuit. In similarity, all primary outputs are connected by edges to the destination node, because output data goes only via the primary outputs to the environment where the circuit is embedded in.
For pathes between fork and join nodes, we start with the analysis from the fork towards the join node and add all visited nodes to the path. If we found nodes on the path with additional inputs or outputs we follow them to their primary inputs or outputs and add all visited nodes to the path. If we examine an already visited node, we stop with the determination for the path. For the example given in Figure 3 the following pathes for $p_{fork,join,i}$ (with $i =$

$1 \cdots 3$) can be identified: $p_{fork,join,1} = \{*_{cm2}, +, +\}$, $p_{fork,join,2} = \{*_{cm0}\}$ and $p_{fork,join,3} = \{*_{cm1}, +, *_{c_s}\}$. By assuming that operation $*$ needs two and operation $+$ one timestep the paths times are: $time(p_{fork,join,1}) = 4$, $time(p_{fork,join,2}) = 2$ and $time(p_{fork,join,3}) = 5$.

After the analysis of the fork and join nodes control nodes, are examined. They are scheduled as soon as possible to allow the identification of alternative paths (see also [3] and [9]). Once more, those paths are the basis of the partitioning and they are combined to partitions. Figure 4 gives an example for the path analysis for control nodes. The node *contr* controls the join node (in this case join corresponds to a multiplexer) and selects which of the inputs are directed to the output. If we can schedule and execute *contr* before all other nodes an additional timestep is needed, but we are able to activate only the used input path of the multiplexer. Therefore, we get three paths for our analysis: $p_{+,join,1} = \{+, +\}$, $p_{*_{cm0},join,1} = \{*_{cm0}\}$ and $p_{*_{cm1},join,1} = \{*_{cm1}, *_{c_s}\}$. The path time for these paths are: $time(p_{+,join,1}) = 2$, $time(p_{*_{cm0},join,1}) = 2$ and $time(p_{*_{cm0},join,1}) = 4$.

Obviously, depending on the characteristics and timing requirements of the design it may be not possible to schedule a control node by extending the run-time. Furthermore, all independent graphs in the CDFG forms a path for the partitioning. Before, we discuss the partitioning of the *Power Scheduler* a design example is introduced in the next section.

3.4 PARTITIONING

The example that is used to illustrate the *Power Scheduler* steps is part of the MPEG-2 algorithm. Figure 5 shows the DFG of the vertical and horizontal conversion used for the motion estimation of MPEG-2. Besides this, the DFG depicted in Figure 5calculates the conversion of the image format CIF 4:2:2 to CIF 4:2:0. This conversion halved the chrominance values of the MPEG-2 picture.

The path analysis for the example examines nine paths (named A to I) displayed in Figure 5. These paths are used for the partitioning of the design. Hence, that independent graphs in the CDFG forms also a partition. To build the partitions by combining the paths that are calculated by the path analysis, we construct a compatibility graph, which is defined as follows:

DEFINITION 3.3 (COMPATIBILITY GRAPH (CG)) *Let $G = (V_k; E_k)$ be a undirected graph, with the set of nodes $V_k = v_1, v_2, \cdots, v_n$ equal to the number of paths of the CDFG. An edge (a_s, a_t) with $a_s, a_t \in V_k$ exists in E_k if there is no conflict between the nodes.*

Each path of the path analysis is a node in the compatibility graph. An edge between two nodes exists if they have no conflict and not the same start and end node. Two paths are in conflict to each other if they are depending from the

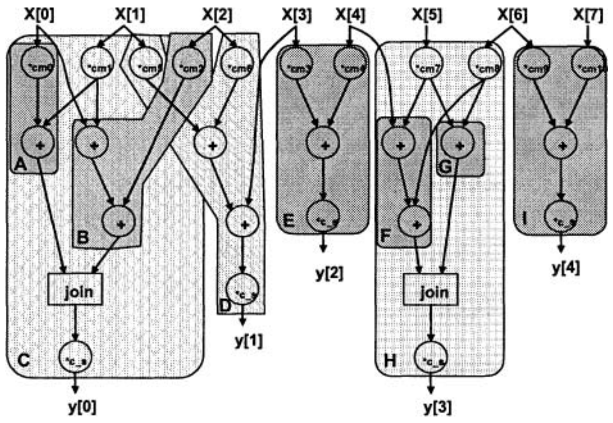
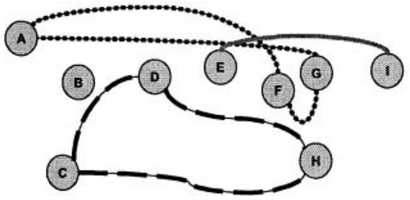


Figure 5. Partitioned design example.



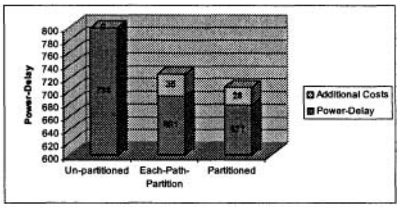


Figure 6. Compatibility graph.

Figure 7. Results for design example.

same fork, join or control node. Therefore, those paths have no edge between each other in the compatibility graph. Furthermore, the path time is recognized. Then a clique search algorithm [1] is used to find cliques in the compatibility graph. The clique algorithm found three cliques for our design example (see Figure 6). Therefore, partitions $P_1 = \{A, F, G\}$, $P_2 = \{C, D, H\}$ and $P_3 = \{E, I\}$ can be activated or deactivated during run-time to save energy. Finally, node B builds an own partition.

4. RESULTS

First results of our proposed method are promising. For the used example, we achieved an energy saving of 15 % in comparison to a not partitioned design (see Figure 7). Also in opposite to a solution where each path build a partition, Figure 7 show, that we achieve a better result. For measurement we used the cost function given in Section 3.1. Furthermore, we implemented the example from the partitioned CDFG in synthesizable VHDL and used the

Power Compiler from Synopsis to prove our measurements. For other parts of the MPEG-2 approach, we achieve similar results.

5. CONCLUSION

In this paper, we presented an approach for low power driven synthesis. Thus, we use standard scheduling algorithms and implement a special partitioning algorithm based on clique search. The implemented *Power Scheduler* contains all developed methods and allows the integration of power saving at a high abstraction level. The presented example, part of the MPEG-2 algorithm, demonstrates the effectiveness of the method.

REFERENCES

[1] Carraghan, R. and Pardalos, P. M. (1990). An exact algorithm for the maximum clique problem. In *In Operations Research Letters 9*, pp 375–382.

[2] Chandrakasan, A., Potkonjak, M., Rabaey, J., and Broderson, R. (1992). Hyper-lp: A system for power minimization using architectural transformations. In *Proc. of ICCAD.*

[3] Chen, C. and Sarrafzadeh, M. (2002). Power-manageable scheduling technique for control dominated high-level synthesis. In *Proc. of Design Automation and Test in Europe (DATE).*

[4] Gajski, D. D. and Ramachandran, L. (1994). Introduction to high-level synthesis. In *IEEE Design and Test of Computers.*

[5] Knudsen, P. and Madsen, J. (1996). Pace: A dynamic programming algorithm for hardware/software partitioning. In *Proc. of IEEE International Workshop on Hardware/Software Codesign.*

[6] Kurdai, F. and Parker, A. (1987). Real: A program for register allocation. In *Proc. of the IEEE-ACM Design Automation Conference.*

[7] Magma-Design-Automation (2004). *Enabling Low Power Design Within an RTL-to-GDSII Implementation Flow.* White Paper.

[8] Mehra, R. and Rabaey, J. (1994). Behavioral level power estimation and exploration. In *Proc. of IWLPD.*

[9] Monteiro, J., Devadas, S., Ashar, P., and Mauskar, A. (1996). Scheduling technique to enable power management. In *Proc. of the 33rd Design Automation Conference, Las Vegas, NV.*

[10] Monteiro, J., Devadas, S., and Li, B. (1994). A methodology for efficient estimation of switching activity in sequential circuits. In *Proc. of the 31st Design Automation Conference, San Diego, CA.*

[11] Rettberg, A. and Rammig, F. J. (2006). A new design partitioning approach for low power high-level synthesis. In *Third IEEE International Workshop on Electronic Desing, Test and Applications (DELTA 2006)*, Kuala Lumpur, Malaysia.

[12] Rettberg, A., Zanella, M. C., Lehmann, T., Dierkes, U., and Rustemeier, C. (2003). Control Development for Mechatronic Systems with a Fully Reconfigurable Pipeline Architecture. In *Proc. of the 16th Symposium on Integrated Circuits and System Design (SBCCI)*, Sao Paulo, Brazil.

[13] Tseng, C. and Siewiorek, D. (1986). Automated synthesis of data paths in digital systems. In *IEEE Transactions on CAD*, pp 5(3): 379–395.

A DEMONSTRATION CASE ON THE TRANSFORMATION OF SOFTWARE ARCHITECTURES FOR SERVICE SPECIFICATION *

João M. Fernandes[1], Ricardo J. Machado[2], Paula Monteiro[2], Helena Rodrigues[2]
[1] Dept. Informática & [2] Dept. Sistemas de Informação
Universidade do Minho, Braga - Guimarães, Portugal

Abstract: This paper presents a demonstration case on the successive application of a model-based technique to assist on the refinement of software logical architectures. The technique is essentially based on the transformation of use cases into object diagrams. The applicability of the technique is illustrated by presenting some results from a mobile application. For mobile software, the definition of the underlying service-oriented architecture must consider as user requirements the services themselves, the mobile operators entry points and the final clients interfaces, and use them to characterize the platform. Within the presented demonstration case, the specification of one service of the mobile application was obtained by successively applying the technique.

1. INTRODUCTION

A Model–Driven Development (MDD) approach is a software development technique that uses models during its execution. With MDD approaches, the development of software is made by successively transforming models into other models, until the final system is obtained.

This article presents the 4-Step Rule Set (4SRS) transformation technique that employ successive transformations of the software architecture, to satisfy the elicited user requirements. It is mainly based on the mapping of use cases into object diagrams. The technique's iterative nature and the use of graphical models ensure that architectures reflect user requirements [1, 2].

Since the 4SRS is an MDD method, its description should contain all elements that are usually present in any software method. It should describe

* Research funded by FCT and FEDER under project *STACOS* (POSI/CHS/48875/2002).

which intermediate and final artefacts should be produced, which notations should be used to create those artefacts and which tasks should be performed, and in which sequence, to create the required artefacts.

4SRS associates, to each object found in analysis, a given category: interface, data, control [3]. This categorization originates object models that, in their essence, are similar to the architectures imposed by the Model-View-Controller [4] or by the Entity-Boundary-Controller [5] patterns.

The 4SRS technique is organized as four steps: (1) object creation, (2) object elimination, (3) object packaging & aggregation, and (4) object association. Additionally, the 2nd step is further subdivided in 7 micro-steps. The application of the 4SRS to obtain the first logical architecture of the demonstration case is described in [6]. After executing all 4SRS steps, the logic architecture for the system that captures all its functional requirements and its non-functional intentionalities is obtained. An object model shows the distribution of significant properties of a system across its parts.

This paper addresses the problem of deriving the logic architecture of a given platform service (called service object diagram), from a functional refinement of the platform architectural model (called platform object diagram), by successively executing the 4SRS technique. The 1st execution, whose details are described in [6] supports the platform requirements analysis by generating one platform object diagram that corresponds to the logic architecture of the system. This paper explains, for the demonstration case considered here, the usage of the 4SRS to derive an object diagram that shows the services the system needs to accomplish its responsibilities.

The 2nd 4SRS execution, which this paper aims to explain, supports service requirements analysis by generating one service object diagram that corresponds to the logic architecture of the service to be specified.

The demonstration case is a platform for mobile applications, supporting usability, openness, interoperability and scalability. It deploys reusable service components to ease the development of context-aware applications that allow citizens to perform a set of government-related activities, and to access the most proper services at any time, anywhere.

2. MODEL-BASED TRANSFORMATIONS

The raw object diagram of the mobile application platform, shown in [6] and obtained after applying the 4SRS, is used in this paper as a starting point for discussing the technique. It identifies the system-level entities, their responsibilities and the relationships among them. Its purpose is to focus at an appropriate decomposition of the system without delving into details.

The components of that object diagram were obtained by reasoning about the characteristics of the service-oriented platform. Applications can be built on top of this architecture by specifying the right composition of services, building a user interface, and orchestrating the data-flow among the various components. Configuring services and applications so they can be reliably reused and composed into larger applications is a major challenge [7].

The resulting raw object diagram (from the 1st execution) can be used in the subsequent phases to define well-delimited sub-projects, by using collapsing and filtering techniques. These techniques redefine the system boundaries, giving origin, for instance, to the database project, services formalization, or platform pattern analysis. Fig. 1 shows the collapsed object diagram that was obtained from the raw object diagram by hiding the packages details. Therefore, links appear at a higher level of abstraction and the resulting object diagram is easier to be read.

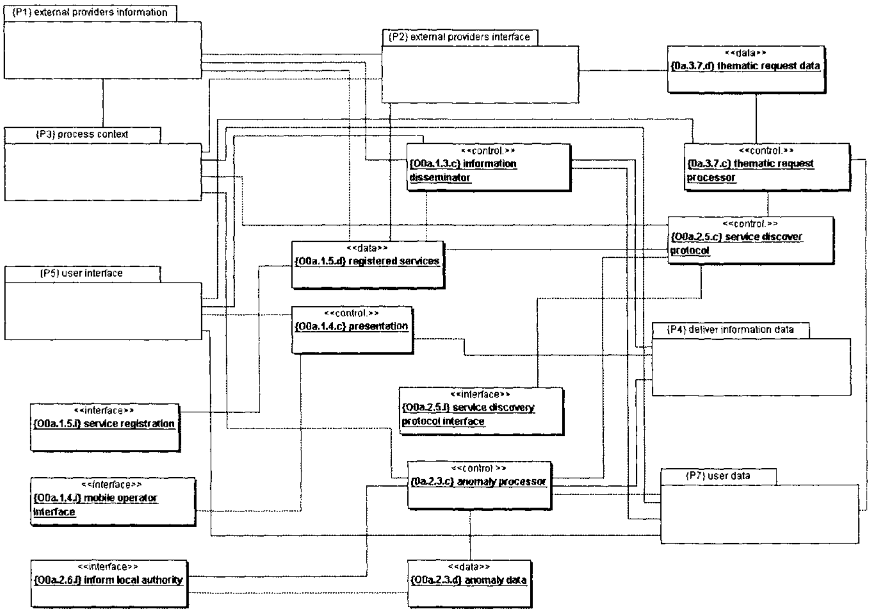

{P1} external providers information
{P2} external providers interface
<<data>>
{0a.3.7.d} thematic request data
{P3} process context
<<control.>>
{O0a.1.3.c} information disseminator
<<control.>>
{0a.3.7.c} thematic request processor
<<control.>>
{O0a.2.5.c} service discover protocol
<<data>>
{O0a.1.5.d} registered services
{P5} user interface
<<control.>>
{O0a.1.4.c} presentation
{P4} deliver information data
<<interface>>
{O0a.2.5.I} service discovery protocol interface
<<interface>>
{O0a.1.5.I} service registration
<<control>>
{0a.2.3.c} anomaly processor
<<interface>>
{O0a.1.4.I} mobile operator interface
{P7} user data
<<interface>>
{O0a.2.6.I} inform local authority
<<data>>
{O0a.2.3.d} anomaly data

Figure 1. Collapsed object diagram.

Fig. 2 shows the filtered object diagram that was obtained by using collapsing and filtering techniques described in [1] by considering package {P5} as a subsystem for design. This diagram was included here as an example of how raw object diagrams can be used during the development process to stress parts of the system and allow subsystem specification and partition of subprojects among various teams.

In this paper, we consider the refinement of package {P5} that has given origin to the AVAccess service. This service is a single point of contact with the platform and should redirect the user to the appropriate service. In particular, when the user intends to report a complaint, he needs to access the AVAccess service and to select the report complaint functionality.

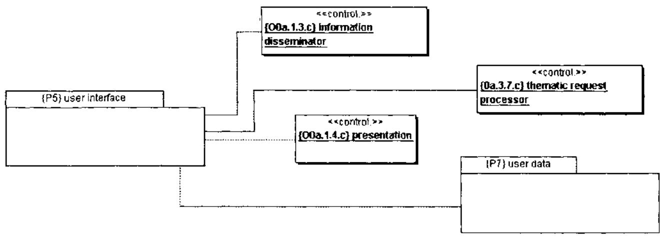


Figure 2. Filtered object diagram for package {P5} service derivation.

Criteria illegible for filtering depend on project management issues, functional implementation domains, etc. Fig. 3 depicts the filtering process executed over Fig. 1 to obtain a {P5}–centric filtered object diagram. During the filtering process, all entities not directly connected to {P5} must be removed from the resulting filtered object diagram.

3. ITERATIVE ARCHITECTURAL REFINEMENT

The development of mobile applications typically follows a service-oriented approach. A service is a software entity running on one or more machines and providing a particular type of function to unknown clients. These services must communicate with each other, to give rise to a service-oriented architecture. The communication can involve either simple data passing or two or more services coordinating some activity. Some means of connecting services to each other is needed, so workflow is a critical part to make services effective. When those services react to changes on user context, applications are said to be context–aware.

For mobile applications, the definition of the underlying service-oriented software architecture must consider the services themselves as user requirements, as well as the mobile operators' entry points and the final clients interfaces, and use them to characterize the platform.

{P5} can be considered as the system to be designed and apply, once more, the 4SRS technique to support its architectural refinement (in Fig. 2). The iterative application of the 4SRS technique suggests the construction of a new use case diagram (called service use case diagram) that captures the users requirements of the new subsystem to refine. From this use case diagram, the corresponding raw object diagram is derived (called service object diagram). This approach contrasts with the one that suggests the application of design patterns [4, 8] to impose into the logical architecture a already proven reference architectural model. Our proposal does not reject this pattern-oriented view, only defers it into latter stages of development.

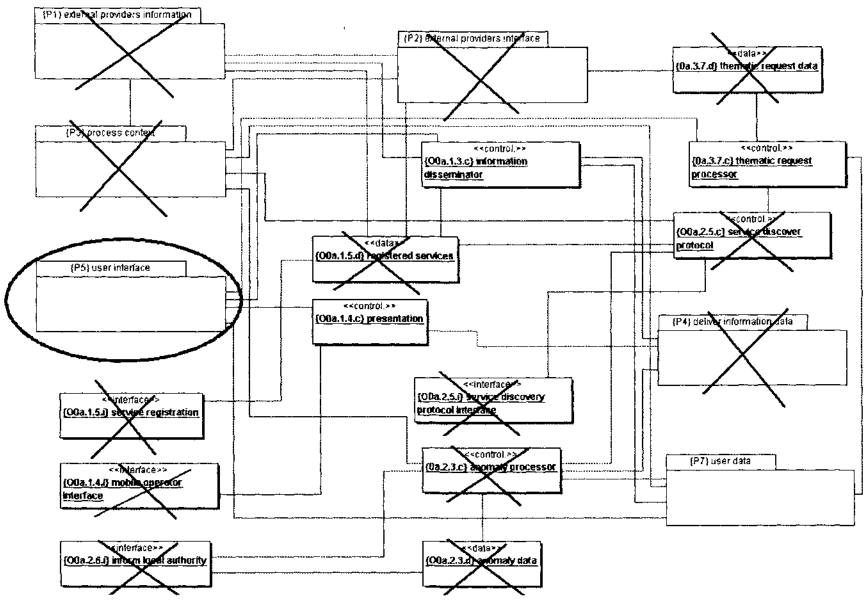

{P1} external providers information
{P2} external providers interface
<<data>> {0a.3.7.d} thematic request data
{P3} process context
<<control.>> {O0a.1.3.c} information disseminator
<<control.>> {0a.3.7.c} thematic request processor
<<control.>> {O0a.2.5.c} service discover protocol
<<data>> {O0a.1.5.d} registered services
{P5} user interface
<<control.>> {O0a.1.4.c} presentation
{P4} deliver information data
<<interface>> {O0a.1.5.i} service registration
<<interface>> {O0a.2.5.i} service discovery protocol interface
<<control.>> {0a.2.3.c} anomaly processor
<<interface>> {O0a.1.4.i} mobile operator interface
{P7} user data
<<interface>> {O0a.2.6.i} inform local authority
<<data>> {O0a.2.3.d} anomaly data

Figure 3. Filtered object diagram for package {P5} service derivation.

The use case diagram in Fig. 4 was created to support the architectural refinement of {P5} to obtain the raw object diagram of the AVAccess service. This service constitutes the example considered in this paper to show the iterative application of the 4SRS technique. All the external entities in this diagram correspond to architectural elements connected to package {P5} in Fig. 2. Object {O0a.1.3.c} in Fig. 2 did not give rise to any actor in Fig. 4, because the architectural refinement of package {P5} did not consider the functionality that is associated with that object. The user actor is present in Fig. 4, since it was already connected to the use cases that gave origin to the objects inside package {P5}, during the development process described in [6]. Actors in Fig. 4 must be viewed as external components, from the point of view of the AVAccess service. To attain better actor semantics within the associations with the obtained use cases, actor {O0a.3.7.c} in Fig. 4 was specialized into two different actors: Application System Context Aggregation Service and Application System Service Repository.

The AVAccess service is the platform component where all user requests are redirected by default. Its service components or end services are architectural components developed and deployed by the local authorities and are the ultimate components to be accessed by the user. They appear as result of other architectural decisions. The AVAccess service is a single point of contact and should redirect the user to the appropriate end service. Users usually start the interaction with the system by contacting this component.

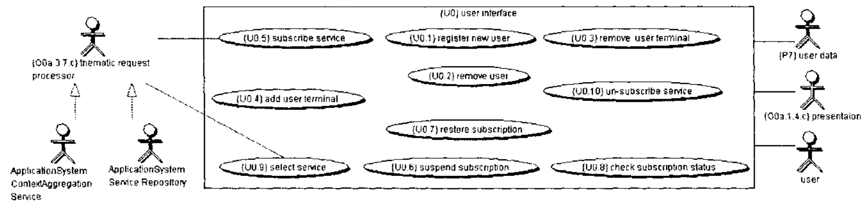

{U0} user interface
{U0.5} subscribe service
{U0.1} register new user
{U0.3} remove user terminal
{U0.2} remove user
{U0.4} add user terminal
{U0.10} un-subscribe service
{U0.7} restore subscription
{U0.9} select service
{U0.6} suspend subscription
{U0.8} check subscription status
{O0a.3.7.c} thematic request processor
ApplicationSystem ContextAggregation Service
ApplicationSystem Service Repository
{P7} user data
{O0a.1.4.c} presentaion
user

Figure 4. Use case diagram for AVAccess service.

One example of a description for the top-level use cases is next presented. Similar descriptions were created for the other top-level use cases.

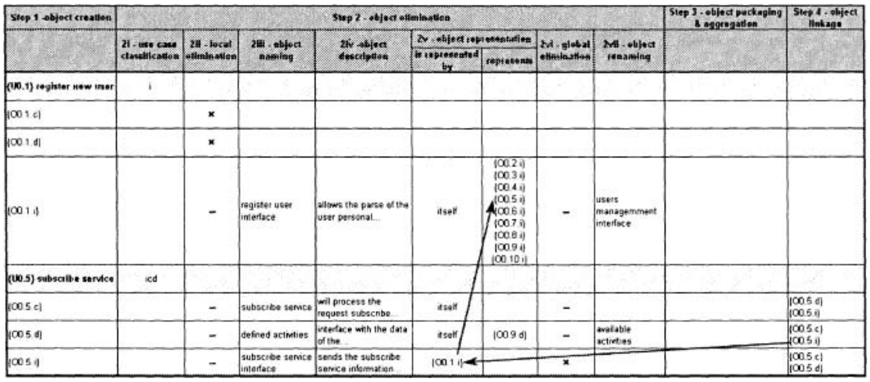
{U0.1} register new user: the user provides user personal information to the AVAccess system. Its personal information consists of userName, password, and, optionally, user profile information. The AVAccess service parses user personal information and sends it to subsystem User. The AVAccess sys-tem sends back the information on success/no success of this operation. The information sent to the user is format-ted by the subsystem Presentation. The system must know terminal model information.

4. TABULAR TRANSFORMATIONS

The execution of the 4SRS transformation steps can be supported in tabular representations. Moreover, the usage of tables permits a set of tools to be devised and built so that the transformations can be partially automated. These tabular representations constitute the main mechanism to automate a set of model transformation steps.

The table for supporting the transformation steps uses one row for each object and one column for each step. The 1st column corresponds to the execution of step 1. The first row allows the insertion of both the reference and the name of the use case. The next three rows allow the insertion of one interface, one data, and one control objects for that use case. For the demonstration case, there is no use case refinement, so step 1 is applicable to all (10) use cases in Fig. 4, which gave origin to 30 objects. Fig. 5 depicts 4 different rows for each of the two previously exemplified use cases.

The 2nd column corresponds to the execution of micro-step 2i. In this micro-step, each use case is classified as one of the 8 different combinations or patterns (Ø, i, c, d, ic, di, cd, icd). This classification helps on the transformation of each use case into objects, and provides hints on which objects to use and how to connect them. For the demonstration case, {U0.1} was classified as "i", meaning that only the interface object is kept (the control and data objects will be eliminated in micro–step 2ii). {U0.5} was classified as "icd", which means that all objects are kept.



Step 1 -object creation	Step 2 - object elimination								Step 3 - object packaging & aggregation	Step 4 - object linkage
	2i - use case classification	2ii - local elimination	2iii - object naming	2iv -object description	2v - object representation		2vi - global elimination	2vii - object renaming		
					is represented by	represents				
{U0.1} register new user	i									
{O0.1.c}		x								
{O0.1.d}		x								
{O0.1.i}		-	register user interface	allows the parse of the user personal...	itself	{O0.2.i} {O0.3.i} {O0.4.i} {O0.5.i} {O0.6.i} {O0.7.i} {O0.8.i} {O0.9.i} {O0.10.i}	-	users managemment interface		
{U0.5} subscribe service	icd									
{O0.5.c}		-	subscribe service	will process the request subscribe...	itself		-			{O0.5.d} {O0.5.i}
{O0.5.d}		-	defined activities	interface with the data of the...	itself	{O0.9.d}	-	available activities		{O0.5.c} {O0.5.i}
{O0.5.i}		-	subscribe service interface	sends the subscribe service information...	{O0.1.i}		x			{O0.5.c} {O0.5.d}

Figure 5. Table for supporting the 4SRS technique.

The 3rd column supports the execution of micro–step 2ii. In this micro-step one decides if each object created in step 1 makes sense in the problem domain, since the creation of objects in step 1 was blindly executed. Objects that are to be eliminated are marked with "x" and objects that are to be kept are marked with "-". For the demonstration case, {U0.1} got two of its originated objects eliminated, since they do not make sense in the problem domain. {U0.1} is only responsible to send the new user information from the user to other subsystems and vice versa, which means that data and control dimensions are not within the scope of this use case.

The 4th column is dedicated to the execution of micro-step 2iii. In this micro–step, objects that have not been eliminated from the previous micro-step must receive a proper name that reflects both the use case from which it is originated and the specific role of the object, considering its main component. {O0.1.i}, for instance, was named register user interface.

The 5th column is related to the execution of micro-step 2iv. Each named object resulting from the previous micro–step must be described, so that the system requirements they represent become included in the object model. These descriptions must be based on the original use case descriptions. For the demonstration case, the following descriptions were obtained:

{O0.1.i} register user interface: allows the parse of the user personal information and sends it to the destination subsystem, and sends back the information on success/no success of the request.

{O0.5.c} subscribe service: will process the request Subscribe service. Will request to the user all the additional information needed to perform the request of the user.

{O0.5.d} defined activities: interface with the data of the available activities in the system (could be a XML file).

{O0.5.i} subscribe service interface: sends the subscribe service information to the destination subsystem, and sends back the information on success/no success of the request.

The 6th and 7th columns correspond to the execution of micro–step 2v. This is the most critical micro–step of the 4SRS technique, since it supports the elimination of redundancy in the user requirements elicitation, and the discovering of missing requirements. The "is represented by" column stores the reference of the object that represents the object being analyzed. If the analyzed object is represented by itself, the corresponding "is represented by" column must refer to itself. The "represents" column stores the references of the objects that the object analyzed will represent. {O0.1.i} does not delegate in other objects its representation and it additionally represents a considerable list of other objects (each one of these objects must refer to {O0.1.i} in their columns "is represented by").

The 8th column is related to micro–step 2vi. This is a fully "automatic" micro–step, since it is based on the results of the previous one. The objects that are represented by other ones must be eliminated, since its system requirements no longer belong to them.

The 9th column is used for micro–step 2vii. Its purpose is to rename the objects not eliminated in the previous micro–step and that represent additional objects. For the demonstration case, object {O0.1.i} was renamed "users management interface" to reflect the list of objects it represents.

The 10th column supports the execution of step 3. Since aggregations and packages were not used in the demonstration case, column 10 is not filled.

The 11th column supports step 4. The associations in the demonstration case were solely derived from the use case classification in step 1. The classification of {U0.5} as type "icd" suggests the existence of three internal links relative to the objects generated from the same use case. However, "id" link (between the interface and the data objects) was not allowed. Additionally, the following two tabular transformations imposed some constrictions to the object connectivity exercise: (1) in step 2v, it was decided that {O0.5.i} is represented by {O0.1.i}; (2) in step 2vi, {O0.5.i} was eliminated. These two decisions imply the existence of the following associations: (1) between {O0.5.c} and {O0.5.d}, suggested by the "icd" classification; (2) between {O0.5.c} and {O0.1.i}, due to the transitivity of the suggested association between {O0.5.c} and {O0.5.i} through the delegation executed by {O0.5.i} in {O0.1.i}.

5. SERVICE SPECIFICATION

Fig. 6 depicts the raw object diagram for the AVAccess service, obtained after a new application of the 4SRS technique over the global logical architecture of the application represented in Fig. 2. Object {O0a.4.1.i} in

Fig. 2 is mapped into object {O0.1.i} in Fig. 6. This object receives user requests for user management and service subscription. In the case of use case {U0.5}, {O0.1.i} uses the functionalities of {O0.5.d} and {O0.5.c}.

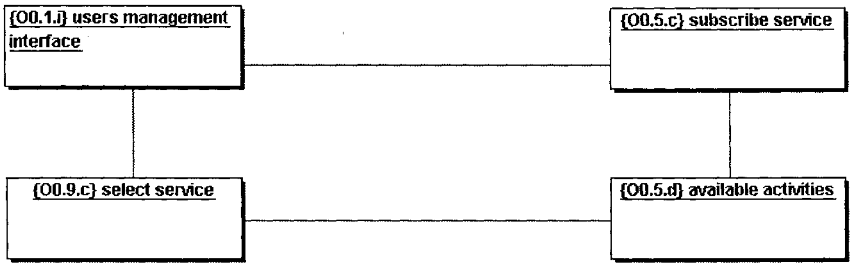

{O0.1.i} users management interface
{O0.5.c} subscribe service
{O0.9.c} select service
{O0.5.d} available activities

Figure 6. Raw object diagram of the AVAccess service.

Object {O0a.2.2.i} in Fig. 2 maps into object {O0.1.i} in Fig. 6. This object receives user requests in case of execution of use case {U0.9} and object {O0.1.i} uses the functionalities associated with objects {O0.9.c} and {O0.5.d}. The obtained raw object model (Fig. 6) constitutes the semantic reference for the service to be designed, since it has emerged from the software logical architecture (Fig. 2) of the platform by adopting a complementary functional refinement at architectural level.

After obtaining this new architectural refined raw object model, the underlying service can be described by a set of diagrams to specify the corresponding architectural component, namely, a class diagram for the static characterization of the service component, a statechart for the life cycle characterization of the service, a set of activity diagrams for methods specification and a set of sequence diagrams for interface and protocol specification. These additional views of the same service are not generated from the application of 4SRS technique, even though they are easier constructed after obtaining the raw object diagram of the service (Fig. 6).

6. CONCLUSIONS AND FUTURE WORK

A software infrastructure for running mobile applications must find, adapt, and deliver the right services to the user computing environment based on his context. The current trend in software industry is for service providers to supply reusable functions via components called services. Building applications involves specifying the right composition of services, building a user interface, and defining the data flow among the components.

For mobile applications, the definition of the underlying service–oriented architecture must consider the services themselves as user requirements, as well as the mobile operators entry points and the final clients interfaces, and use them to characterize the platform. Within the presented demonstration case, the specification of one service of a mobile application was obtained by recursively applying the 4SRS technique. The technique has shown its usefulness by assuring the generation of a seamless specification of the service–oriented architecture requirements.

The proposed iterative usage of the 4SRS technique allows designers to build a new use case diagram that captures the users requirements of the new system to refine a service. From this use case diagram, a raw object diagram can be derived. This approach is complementary to the use of design patterns by allowing a functional refinement of requirements at architectural level, considering the specific aspects of the system under design. This transformational approach shows that model continuity is an important topic and highlights the importance of defining a well-defined process to relate, map and transform requirement models [9]. In the presented case, the 4SRS has allowed the specification of one particular service, considering all the architectural decisions previously taken to specify the platform where the service is supposed to run, by assuring a continuous mapping between the platform and the service models.

As future work, the 4SRS technique will be extended to consider the transformation of objects diagrams into class diagrams, which seem a crucial step for software-intensive systems.

REFERENCES

1. J.M. Fernandes, R.J. Machado. From Use Cases to Objects: An Industrial Information Systems. OOIS 2001, Calgary, Canada, pp. 319-328, Springer, August, 2001.
2. J.M. Fernandes, R.J. Machado, H.D. Santos. Modeling Industrial Embedded Systems with UML. CODES 2000, San Diego, California, U.S.A., pp. 18-22, 2000.
3. I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, 1992.
4. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons, 1996.
5. I. Jacobson, G. Booch, J. Rumbaugh. The Unified Software Development Process. Object Technology. Addison-Wesley, 1999.
6. R.J. Machado, J.M. Fernandes, P. Monteiro, H. Rodrigues. Transformation of UML Models for Service–Oriented Software Architectures. ECBS 2005, Greenbelt, Maryland, U.S.A., pp. 173 182, 2005.
7. G. Banavar, A. Bernstein. Software Infrastructure and Design Challenges for Ubiquitous Computing Applications. Communications of the ACM, vol. 45, no. 12, pp. 92-96, 2002.
8. R. Ahlgren, J. Markkula. Design Patterns and Organisational Memory in Mobile Application Development. PROFES 2005, Oulu, Finland, pp. 143-156, 2005.
9. R.J. Machado, I. Ramos, J.M. Fernandes. Specification of Requirements Models. In A. Aurum and C. Wohlim (Eds.), Engineering and Managing Software Requirements, pp. 47-68, 2005.

MODEL-BASED ANALYSIS OF A WINDMILL COMMUNICATION SYSTEM

Simon Tjell
Department of Computer Science
University of Aarhus
Denmark

Abstract This paper presents the experiences obtained from modeling and analyzing a real-world application of distributed embedded computing. The modeling language Coloured Petri Nets (CPN) has been applied to analyze the properties of a communication system in a windmill, which enables a group of embedded computers to share a group of variables. A CPN-based model of the system is used to analyze certain real-time properties of the system.

Keywords: Real-time systems, communication protocol, Coloured Petri Nets, performance analysis

1. INTRODUCTION

This paper is based on a real life product development problem from the Danish windmill manufacturer Vestas Wind Systems [1] and proposes a solution to the problem. The proposed solution is analyzed by use of a CPN-based model of the system. The paper summarizes parts of a project [7], which has been running for a period of one year. A modern windmill from Vestas is controlled and monitored by a complex application consisting of a group of inter-connected control components. Each control component implements parts of the control algorithms, which cooperate to control and monitor the operation of the windmill through a collection of physical sensors and actuators. The components are executed in a distributed system of embedded computers connected by a network bus. The connection between the components is established by shared access to a group of variables. Basically, a variable represents one of three values: A measurement from a sensor, the output value for an actuator or an intermediate calculation between two components. The correctness of the output-variables from a component is highly dependent of the freshness and consistency of the input-variables of that component. Some of the output-variables are used to control actuators that adjust the wings and other physical parts of the mill. In that way, the correctness of the output-

Please use the following format when citing this chapter:

Tjell, S., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 245–254.

variables has a direct influence of the degree of wear and tear of the machinery over time. This makes it important to optimize the method for sharing the variables in order to maximize the lifetime of the mill - and this task is the main aim of this paper and the project it describes. A design for the communication system and a model-based analysis of the design are provided. The following sections of the paper are structured like this: Firstly the existing software environment is introduced. This is followed by an identification of problems in the existing design of the communication system. The identification is followed by a proposal for an alternative design. The proposed redesign has been modeled, simulated and analyzed. This process is described in the last section of the paper.

2. THE DAO APPLICATION FRAMEWORK

Distributed and Active Objects (DAO) is a proprietary framework for developing windmill control applications. The framework is based on the Active Object design pattern [5] with the addition of distributed capabilities and dynamic attachment mechanisms. The following list gives a description of the key elements of the control application based on the DAO framework. The description defines the terminology for the remaining sections of this paper. Please refer to Figure 1 when reading the list.

1 Node: A node is an embedded computer. Each node has a kernel, which executes a number of components. A typical system consists of up to five nodes.

2 Kernel: The kernel has the responsibility of periodically activating the components on its own node. The activation period for each component is determined by a scheduling registration. The kernel continously runs through a cycle of operations.

3 Component: A component is a software object, which complies with a specified interface. The interface contains an activation method, through which the component is periodically activated by the kernel. The components contain all control algorithms used in the control application.

4 Variable: A variable has a type and a value. The value can be altered and monitored by the components. Each variable is represented in one original instance and a number of copies. The copies for a variable exist on nodes with components attached to that given variable. A number of components can establish dynamic attachments to each variable for reading and/or writing (one component at a time) the value of that variable. Variables exist in two forms on a node: Kernel variables (the originals) and component variables (copies on which the components

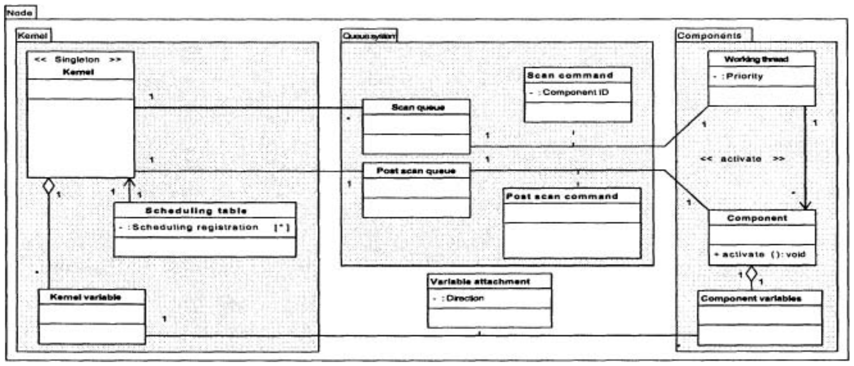


Figure 1. The static structure of the DAO-framework.

operate). The kernel has the responsibility of copying values between the two representations of variables.

5 Communication component: The communication component (CC) is a specialized component that exists in one instance in every node. It differs from the regular components by being able to communicate with its peer CCs through a network interface. The communication is performed in order to maintain variable consistency through-out the nodes. When the CC is activated, it performs two main tasks: It generates and sends update messages containing values of the kernel variables to which components on other nodes are attached and it receives the update messages from other nodes. One message holds values for a number of variables.

6 Queues: The scan queue is used when the kernel activates a component. The kernel does this by placing a scan command containing the ID of the component to be activated in one of the scan queues. The choice of scan queue determines the priority of the activation since commands from each scan queue are consumed by a specific threads. Each thread in the thread pool has an individual priority. When an activated component terminates its activation, this is signaled to the kernel with a post scan command that is placed in the post scan queue. This queue is shared by all working threads. When the kernel receives a post scan command

3. THE PROBLEM

This section gives a description of the existing communication protocol used for communication between the nodes. The description is supplemented by a characterization of a number of problems with this protocol. This description is followed by a proposal for a replacement communication protocol. The communication between the nodes is performed by the communication components in order to keep shared variable values updated. These components have access to a communication interface through which they are able to exchange UDP-datagrams [2] across the network. The CCs are periodically activated by the kernel. During one activation cycle are number of messages are sent and received.

The proposal of a new communication protocol for the DAO-framework is relevant, because problems are being experienced with the existing CC. In its existing design, the communication component uses a simple protocol, in which the CCs send out messages with variable values when variable values are changed - i.e. triggered by the event of changing values. The messages are sent unreliably using multicast through the UDP-protocol with no detection of lost messages. The UDP-protocol features a CRC-mechanism [2], which makes the receiver of a message able to discard the message, if its contents have been altered during the transmission due to electrical noise. This is the only sort of error detection. The nature of the existing design of the CC causes two critical problems:

1 If a message is lost during transmission this is not detected by neither the sender nor the receiver. Messages are sent when a state changes and when this information is lost, it results in the view of the state becoming inconsistent between the sender and receiver(s) of the message. The period of inconsistency continues until a new message is successfully exchanged. This happens the next time the state is changed.

2 If a large number of variables are altered within a short period of time, this will cause a large number of messages to be sent and received in that period. This can cause the CCs in the DAO-nodes to consume too much time within a given activation cycle. This can cause a skew in the time of activation for the other components on a node, which can result in erroneous output from the control algorithms implemented in those components.

4. PROPOSAL FOR AN ALTERNATIVE DESIGN

It has been decided to base the proposal of an alternative design of the communication protocol on the principle of Soft State signaling protocols [2]. The discussion leading to this decision can be found in [7]. The proposed communication protocol is based on a generic Soft State principle [2]. Soft State messaging is a variant of protocols within the family of signaling protocols. This family of protocols spans within two generically defined poles; Soft State and Hard State protocols. Signaling protocols are applied in a wide range of applications. Common to those applications and the specific application in

relation to DAO is the need to maintain a consistent view of a shared state - where the state could be a routing table, a list of shared files or a set of variables. Soft State protocols differ from Hard State protocols by the fact that they rely on the exchange of messages using best-effort-semantics as opposed to reliably exchanging messages. The difference is observable in the way, in which the distributed views of the state are updated across the network. In both cases messages are sent from the sender, which is where the changing of the state occurs to a receiver (or a group of receivers), where the state is observed. Also, in both cases, the sender is the initiator of the message exchange. The difference here lies in which event triggers the sending of the message with the changed state from the sender to the receiver. In the case of the Hard State protocols, the triggering event is the change itself - i.e. when the sender detects a change in its local view of the state, this causes a message with the updated state to be sent to the receiver. On the other hand, in the case of the Soft State protocols, the triggering event is a temporal event caused by a timer running out. Each state is associated with an update timer that causes a periodic event trigger.
Based on the properties of the proposed communication protocol, it is considered a sane choice for replacing the existing protocol, because the proposal is suited for addressing the two main problems with the existing protocol (the two issues refer to the two problems described earlier):

1 All messages are periodically retransmitted. This resolves the problem of potentially long periods of inconsistency in the case of lost messages.

2 The periods for sending messages are planned offline. This makes it possible to predict the maximum amount of messages being exchanged within an activation cycle and thereby predict processing overloads.

The original attachment mechanism with which attachments between components and remote or local variables are registered is adopted and applied in combination with the new protocol design. Each variable is assigned with an update timer that triggers the generation of periodic update messages for that variable. Based on knowledge of the properties of the variables, it is possible to define a method for grouping the variables based on their frequency of modification. In this way, each variable belongs to a certain group in which all variables share the same update rate, which is used for initializing the update timers and thereby determine when to send update messages. Five such groups are identified. Futhermore, the probabilistic distribution of variables in these groups is known based on experience from the existing system. This means that it can be estimated that each group contains a certain percentage of the variables. This knowledge is applied when modeling the application and the modification of the variables. While the modification rate differs between the variables the swiftness of communicating changes to variable values is equally important for all variables in order to optimize consistency.

5. MODELING AND ANALYZING THE PROPOSAL

After having chosen an appropriate communication protocol, time comes to developing the model of the DAO-system with the proposed protocol integrated. This section describes the simulation process. An example of the analyzed data is given. For the full set of analysis results, please refer to [7]. The model is developed using Coloured Petri Nets (CPN) [3], which is a formally specified modeling language with a graphical representation. CPN-models specify the structural and behavioral aspects of (distributed) systems in which the state is represented by places holding tokens. Places are connected by transitions that are able to move tokens between the places of a net and thereby modify the state of the places and the model in general. In the case of modeling DAO and the communication protocol, the tokens represent entities such as messages and variables. In the same way, the transitions represent actions such as sending messages and reading variables. Tokens are also used for representing more abstract aspects of the system; delays, timers, counters etc. Analogously, places are used to represent buffers containing message, tables of variables and so on.

CPN-models differ from traditional Petri Nets by the fact that the tokens have types and values - much like in standard programming languages. This makes it possible to distinguish the tokens and simulate tokens moving around between the places. Figure 2 shows a part of the model. Places are visualized by ellipses and transitions as rectangles. The model consists of a hierarchi of modules like this. This particular part of the model specifies the dynamic behavior of a DAO-node.

The dynamic behavior of each nodes is modeled by a sequence of repeated events, including (ordered for understandability):

1 Generation of update messages: the table of local variables is evaluated in order to find variables with expired update timers. The current values of these variables are collected in a new update messages and the timers are initialized. In each case, the table of attachments from external components is investigated - if no attachments are found, it is not necessary to include the variable in the update message. If no variables comply with the constraints, no update message will be generated.

2 Reception of incoming update messages: The node receives messages through the communication channel.

3 Handling of incoming update messages: When the messages have been received they are processed, which means that all local variables are

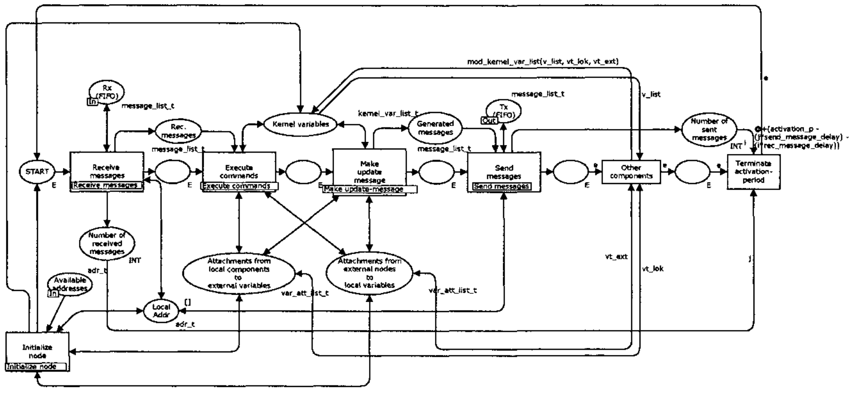


Figure 2. An example of a part of the CPN-model specifying the dynamics of a node.

updated with the values from the update messages. The values in the update messages are not necessarily different from the current values of the variables.

4 Transmission of outgoing update messages: If an update message has been generated, it is transfered to the communication channel.

5 Simulation of the effect of other components: The values of the local variables are changed periodically based on the knowledge of the modification rates for each group of variables. This is done to simulate the effect of the local components modifying the variables.

These events reflect the cycle of events that is handled by the kernel on each node.

One of the main advantages of models expressed in CPN is the fact that they are executable. This makes it possible to perform automated simulations while collecting measurements from the model. In the specific case, simulation has been applied to analyze a group of properties:

1 Consistency of the variables: The most basic purpose of the communication protocol is to maintain consistency within the group of shared variables. The level of consistency is measured by continously performing simple counting operations that lead to a number for the percentage of consistent variables. A variable is considered being consistent when all its instances in all nodes share the same value. Figure 3 shows the result of measuring the average consistency while varying the activation period of the CC and the number of nodes. As expected, the level of

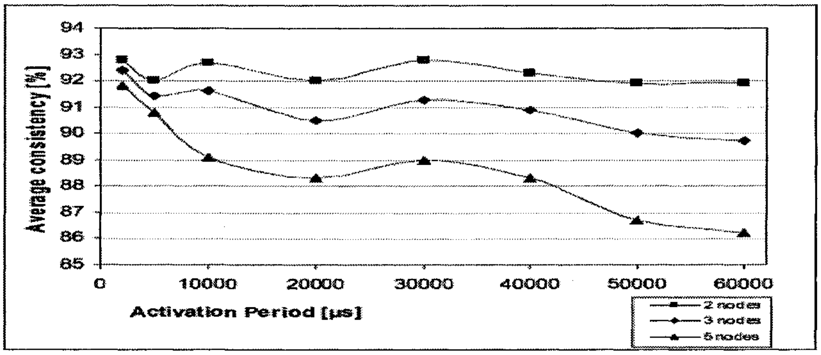


Figure 3. Measuring the average consistency as a function of activation period.

consistency decreases when the activation period is increased, since this will increase the magnitude of the delay that occurs between modification of a variable values and the next-coming transmission of a periodic update message. Furthermore, the consistency level decreases when the number of nodes is increased. This is caused by the fact that more nodes means more components, which results in more writing attachments for each variable.

2 Update delay: Another measure for the quality of the CC is the delay that occurs before all instances of a variable are updated when the value of one of the instances is modified. This is important because the quality of the output from the control algorithms depends directly on this property. The measurement of the delay is performed by supplying the tokens that represent update messages with a timestamp. The timestamp contains the time of modification for the involved variables and is used to calculate the delay when the update message is received by a node.

3 Resource consumption: Analyzing the amount of processing time being consumed by the CC on a node is relevant, because it directly defines the amount of time left for activating the other components on the same node. Measurements from the target platform have shown, that sending and receiving messages consumes significantly more time than any other task of the communication component. This - in combination with the fact that the CC is continously activated with a fixed period - makes it reasonable to estimate the amount of time being spent during each activation of the CC based on the number of messages being handled within that activation cycle. The amount of resource consumption is measured

in each activation cycle by counting the number of incoming and outgoing messages. The maximum consumption will occur when a CC on one node receives messages from all other nodes and at the same time has to send one multicast message itself. This means that the maximum can be calculated based on a collection of parameters: the variable activation cycle, the fixed amount of time used for handling messages and the variable number of nodes. For low activation frequencies, the measured level of resource consumption is similar to the calculated maximum since it will be necessary to send and receive messages in all activation cycles. This is not the case, when the activation frequency is increased. This increases the fraction of activation cycles where less or no messages are handled. On the other hand; when the activation period becomes shorter, each message being sent and received represents a larger fraction of the time in the activation cycle. Measurements on the actual embedded computers have shown that the action of sending and receiving messages is the major source of CPU-time consumption on the nodes. This makes it possible to calculate the percentage wise consumption of CPU-time of the CC by observing the number of messages being processed within a given activation cycle. This approximation is only feasible as long task of processing messages is significantly costlier in time than any other task being handled by the communication component.

Apart from these properties, the tolerance to packet loss is analyzed by varying the probability of losing packets in the communication channel connecting the nodes while observing the influence on the properties described above. The level of consistency and the magnitude of the update delays increase when more packets are lost. At the same time, the resource consumption decreases because the lost packets wont result in consumption of processing time on the receiving nodes.

The analysis illustrates how the configuration of the system is a tradeoff between consistency and resource consumption. With the Soft State protocol, the level of consistency depends on the frequency of sending the periodic update messages - and this frequency directly affects the amount of processing resources being consumed by the CC. In that context, this paper and [7] are ment to illustrate how the model-based analysis of the communication system is applied to gain knowledge of the parameters of this tradeoff. The protocol does not intend to guarantee full reliability at all times but rather a probabilistic reliability of adjustable quality.

6. RELATED WORK

This paper is focused around two main topics: the use of Soft State protocols for sharing states and the use of CPN-models for analyzing an industrial real-time system by simulation. Related work exists within both topics. Firstly, [6] introduces a framework for understanding and discussing the properties of Soft State signaling protocols. The paper presents a Soft State transport protocol accompanied by a formal model, with which the protocol is analyzed by use of queuing theory. In [4] a Markov-model is specified for comparing the performance properties of generic Soft and Hard State protocols in combination with hybrid variants. Both of these works are focused on the performance and consistency issues of the protocols alone. This paper combines the protocol technicalities with the aspects of the application and the hardware platform in the simulation-based analysis. [8] develops CPN-models for analysing the RSVP-protocol, which is partly based on Soft State signaling.

7. SUMMARY

This paper has described the process of identifying the problem in question and has presented a proposal for its solution. This proposal has been modeled and the model has been used for simulation with the purpose of analyzing its real-time properties. CPN-models have proven to be a strong tool in this work and their executability have eased the process of understanding the model and thereby the existing system and the protocol that has been applied. The main weakness of this type of analysis is the lack of a strong method for validating the correctness of the model itself compared to what is being modeled.

REFERENCES

[1] Vestas wind systems. http://www.vestas.com.

[2] David D. Clark. The design philosophy of the DARPA internet protocols. In *SIGCOMM*, pages 106–114, Stanford, CA, August 1988. ACM.

[3] Kurt Jensen. Coloured petri nets: A high level language for system design and analysis. *Lecture Notes in Computer Science; Advances in Petri Nets 1990*, 483:342–416, 1991. NewsletterInfo: 39.

[4] Ping Ji, Zihui Ge, Jim Kurose, and Don Towsley. A comparison of hard-state and soft-state signaling protocols. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 251–262, New York, NY, USA, 2003. ACM Press.

[5] R. Greg Lavender and Douglas C. Schmidt. Active object: an object behavioral pattern for concurrent programming. pages 483–499, 1996.

[6] Suchitra Raman and Steven McCanne. A model, analysis, and protocol framework for soft state-based communication. In *SIGCOMM*, pages 15–25, 1999.

[7] Simon Tjell. Modeling and analysis of a communication protocol for windmills (danish only). Master's thesis, University of Aarhus, 2005. (http://daimi.au.dk/~tjell/thesis.pdf).

[8] M.E. Villapol and J. Billington. Modelling and initial analysis of the resource reservation protocol using coloured petri nets. In *Proc. Of the Workshop on Practical Use of High-Level Petri Nets, within the 21st International Conference on Applications and Theory of Petri Nets*, pages 91–110, 2000.

PRE-RUNTIME SCHEDULING CONSIDERING TIMING AND ENERGY CONSTRAINTS IN EMBEDDED SYSTEMS WITH MULTIPLE PROCESSORS

Eduardo Tavares, Meuse Oliveira Jr, Paulo Maciel, Bruno Souza, Silvino Neto
CIn - UFPE. Recife-PE-Brazil.
{eagt, mnoj, prmm, bs}@cin.ufpe.br, silvinovvneto@yahoo.com.br

Raimundo Barreto, Romulo Freitas, Marcelo Custodio
DCC-UFAM. Manaus-AM-Brazil
{rbarreto, devezas, mmc}@dcc.ufam.edu.br

Abstract In this paper, a pre-runtime scheduling approach for hard real-time embedded systems with multiple processors is presented considering stringent timing and energy constraints. This paper adopts a formal approach, based on time Petri nets, for synthesizing feasible schedules.

1. INTRODUCTION

Some embedded systems are classified as real-time systems, where the correct behavior depends not only on the integrity of the results, but also the time in which such results are produced. In hard real-time systems, if timing constraints are not met, the consequences can be disastrous, including great damage of resources or even loss of human lives. Due to CPU-bound tasks, some hard real-time embedded systems need to rely on multiple processors in order to meet timing constraints.

In addition to timing issues, many hard real-time systems have constraints on autonomy, since, in many cases, they need to be operated in remote areas where energy sources may be highly constrained. Therefore, such systems cannot exceed their respective energy (or power) constraints for executing their associated tasks. Mobile medical devices, for example, have both timing and energy constraints that need to be satisfied. Their tasks cannot miss their re-

Please use the following format when citing this chapter:

Tavares, E., Olivera, M., Jr., Maciel, P., Souza, B., Neto, S., Barreto, R., Freitas, R., Custodio, M., 2006, in IFIP International Federation for Information Processing, Volume 225, From Model-Driven Design to Resource Management for Distributed Embedded Systems, eds. B. Kleinjohann, Kleinjohann L., Machado R., Pereira C., Thiagarajan P.S., (Boston: Springer), pp. 255–264.

spective deadlines, and cannot exceed a specified energy constraint in order to prolong the battery charge usage.

Taking into account such needs, this paper provides a pre-runtime approach based on time Petri nets [6] for synthesizing feasible schedules considering timing and energy constraints. In order to provide a more realistic system behavior, this work models explicitly the worst-case execution time of the dispatcher, since it may affect the tasks' deadline. The approach presented in this paper is a depth-first search method that generates a partial state-space computed from a time Petri net model that represents the task's constraints, thus tackling the state-space growth inherent to such systems.

2. RELATED WORKS

Xu and Parnas [9] present a branch-and-bound algorithm that finds an optimal pre-runtime schedule on a single processor for real-time process segments with release, deadline, and arbitrary exclusion and precedence relations. Despite the importance of their work, real-world experimental results are not presented. Abdelzaher and Shin [1] extended Xu and Parnas' work in order to deal with distributed real-time systems. This algorithm takes into account delays, precedence relations imposed by interprocess communications, and considers many possibilities for improving the scheduling lateness at the cost of complexity.

In [8], Swaminathan and Chakrabarty address the problem of scheduling tasks for minimum I/O energy consumption in hard real-time systems . The work adopts a pre-runtime scheduling approach and employs pruning technique based on time and energy, as well as heuristic methods in order to reduce the problem complexity. However, the proposed approach does not support inter-task relation and does not take into account multiple processors. AlEnawy and Aydin [2] introduce static (pre-runtime) and dynamic (runtime) scheduling mechanisms for dealing with energy-constrained scheduling. The proposed approach does not guarantee that all tasks will be executed, but only selected tasks with high priority. Preemption is supported, but inter-task relations are not taken into account.

3. COMPUTATIONAL MODEL

Computational model syntax is given by a time Petri net [6], and its semantics by a timed labeled transition system. A time Petri net (TPN) is a bipartite directed graph represented by a tuple $\mathcal{P}= (P, T, F, W, m_0, I)$. P (places) and T (transitions) are non-empty disjoint sets of nodes. The edges are represented by $F \subseteq (P \times T) \cup (T \times P)$. $W : F \to \mathbb{N}$ represents the weight of the edges. A TPN marking m_i is a vector $m_i \in \mathbb{N}^{|P|}$, and m_0 is the initial marking. $I : T \to \mathbb{N} \times \mathbb{N}$ represents the timing constraints, where

$I(t) = (EFT(t), LFT(t))\ \forall t \in T$, $EFT(t) \leq LFT(t)$, $EFT(t)$ is the Earliest Firing Time, and $LFT(t)$ is the Latest Firing Time.

An extended time Petri net with energy and priorities is represented by $\mathcal{P}_{\mathcal{E}} = (\mathcal{P}, \mathcal{E}, \pi)$. $\mathcal{P}$ is the underlying time Petri net, $\mathcal{E} : T \rightarrow \mathbb{R}^{+} \cup \{0\}$ is a function that assigns transitions to energy consumption values, and $\pi : T \rightarrow \mathbb{N}$ is a priority function.

A set of enabled transitions is denoted by: $ET(m_i) = \{t \in T \mid m_i(p_j) \geq W(p_j, t)\}$, $\forall p_j \in P$. The time elapsed, since the respective transition enabling, is denoted by a clock vector $c_i \in \mathbb{N}^{|ET(m_i)|}$. The dynamic firing interval ($I_D(t)$) is dynamically modified whenever the respective clock variable $c(t)$ is incremented, and t does not fire. $I_D(t)$ is computed as follows: $I_D(t) = (DLB(t), DUB(t))$, where $DLB(t) = max(0, EFT(t) - c(t))$, $DUB(t) = LFT(t) - c(t)$, $DLB(t)$ is the Dynamic Lower Bound, and $DLB(t)$ is the Dynamic Upper Bound.

Let $\mathcal{P}_{\mathcal{E}}$ be a time Petri net, C be the set of all clock vectors in $\mathcal{P}_{\mathcal{E}}$, and $M_{\mathcal{E}}$ be the set of reachable markings of $\mathcal{P}_{\mathcal{E}}$. The set of states S of $\mathcal{P}_{\mathcal{E}}$ is given by $S \subseteq (M \times \mathbb{N}^{|ET(M)|} \times \mathbb{R})$, that is, a single state is defined by a triple (m, c, e), where m is a marking, c is its respective clock vector for $ET(m)$, and e is the accumulated energy consumption up to this state.

$FT(s)$ is the set of fireable transitions at state s defined by: $FT(s, e_{max}) = \{t_i \in ET(m) \mid (e \leq e_{max}) \wedge (\pi(t_i) = \min(\pi(t_k))) \wedge (DLB(t_i) \leq \min(DUB(t_k)))$ $, \forall t_k \in ET(m)\}$. The *firing domain* for t at state s, is defined by the interval: $FD_s(t) = [DLB(t), \min(DUB(t_k))]$, $\forall t_k \in ET(m)$.

A timed labeled transition system (TLTS) is a quadruple $\mathcal{L} = (S, \Sigma, \rightarrow, s_0)$, where S is a finite set of states, Σ is an alphabet of labels representing actions, $\rightarrow \subseteq S \times \Sigma \times S$ is the transition relation, and $s_0 \in S$ is the initial state.

The semantics of a TPN $\mathcal{P}$ is defined by associating a TLTS $\mathcal{L}_{\mathcal{P}} = (S, \Sigma, \rightarrow, s_0)$: (i) S is the set of states of $\mathcal{P}$; (ii) $\Sigma \subseteq (T \times \mathbb{N})$ is a set of actions labeled with (t, θ) corresponding to the firing of a firable transition (t) at time (θ) in the firing interval $FD(t)$, $\forall s \in S$; (iii) $\rightarrow \subseteq S \times \Sigma \times S$ is the transition relation; (iv) s_0 is the initial state of $\mathcal{P}$.

4. SPECIFICATION MODEL

Let $\mathcal{T}$ be the set of tasks in a system. A periodic task is defined by $\tau_i = (ph_i, r_i, c_i, d_i, p_i, proc_i)$, where ph_i is the initial phase; r_i is the release time; c_i is the worst case computation time required for execution of task τ_i; d_i is the deadline; p_i is the period; and $proc_i$ is the processor allocated to such task. A task is classified as sporadic if it can be randomly activated, but the minimum period between two activations is known. Pre-runtime scheduling can only schedule periodic tasks. However, Mok [7] has proposed a translation from sporadic to periodic tasks. A task τ_i *precedes* task τ_j, if τ_j can only start executing after τ_i has finished. This work considers that communication between tasks allocated to the same processor is treated as a precedence relation. A task τ_i *excludes* task τ_j, if no execution of τ_j can start while task τ_i is executing. If it is considered a single processor, then task τ_i could not be preempted by task τ_j.

When adopting a multiprocessing environment, all inter-processor communications have to be taken into account, since these communications affect the system predictability. A inter-processor communication is represented by a special task, namely, communication task, which is described as follows. Let $\mu_m \in \mathcal{M}$ be a communication task defined by $\mu_m = (\tau_i, \tau_j, ct_m, bus_m)$, where $\tau_i \in \mathcal{T}$ is the sending task, $\tau_j \in \mathcal{T}$ is the receiving task, ct_m is the worst case communication time, $bus_m \in \mathcal{B}$ is the bus, where $\mathcal{B}$ is the set of buses, and $proc_i \neq proc_j$.

5. MODELING REAL-TIME SYSTEMS

In this work, the proposed modeling adopts a formal method for describing systems with timing constraints. The proposed modeling applies composition rules on building blocks models. For lacking of space, this section aims to present just an overview. For more details the reader is referred to [3].

5.1 TASKS MODELING

The considered building blocks are: (i) Fork; (ii) Join; (iii) Periodic Task Arrival; (iv) Deadline Checking; (v) Non-preemptive Task Structure; (vi) Preemptive Task Structure; (vii) Resources; and (viii) Inter-Processor Communication. The blocks are summarized as follows: **a) Fork Block**. Let us suppose that the system has n tasks. The fork block is responsible for starting all tasks in the system. This block models the creation of n concurrent tasks. **b) Join Block**. Usually, concurrent activities need to synchronize with each other. The join block execution states that all tasks in the system have concluded their execution in the schedule period. **c) Periodic Task Arrival Block**. This block models the periodic invocation for all task instances in the schedule period (P_S). **d) Deadline Checking Block**. The proposed modeling method uses elementary net structures to capture deadline missing. The scheduling algorithm (Figure 5) must eliminate states that represent undesirable situations like this one. **e) Task Structure Block**. The task structure may implement either preemptive or non-preemptive scheduling methods. Considering a non-preemptive method, the processor is just released after the entire computation to be finished. The preemptive method implies that a task are implicitly split into all possible subtasks, where the computation time of each subtask is exactly equal to one task time unit (TTU). **g) Resource Block**. The resources modelled are processors (p_{proc_i}) and buses (p_{bus_i}). An individual resource is represented by a single place. The presence of a token in a resource place indicates the availability of this resource.

Figure 1 depicts a Petri net model considering a specification composed of two non-preemptive tasks: $\tau_0 = (0, 0, 2, 7, 8, P1)$ and $\tau_1 = (0, 2, 2, 6, 6, P1)$.

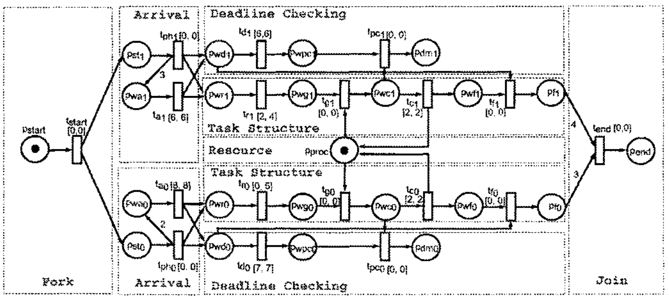


Figure 1. An example considering two tasks.

h) Inter-processor Communication Block. In this work, all inter-processor communications are treated as communication tasks, and message-passing paradigm is adopted. Additionally, the proposed approach for inter-processor communication considers that: (a) after the execution of the sending task, the message transmission is performed; (b) the receiving task can only execute after receiving the complete message; (c) both the sending and receiving processors are ready in the beginning of the communication. In other words, when the sender is transmitting the data, the receiver is prepared at the same moment for getting such data. This mechanism may be interpreted as a synchronous communication. Since interrupts may affect the system predictability, the proposed approach considers polling rather than interrupt handling to implement the receive operation; (d) point-to-point communication (or unicasting); (f) buses are reliable; (g) before communication takes place, the bus and, both sending and receiving processors have to be granted; and (h) communication time is annotated in the respective communication transition.

Figure 2 depicts the inter-processor communication building block. $t_{gb_{ij}}$ represents the granting of the sending processor, the receiving processor and the bus. $t_{send_{ij}}$ represents both the message sending and receiving. It is annotated with timing constraint specification, in this case, ct_m (worst-case communication time) of the respective communication task $\mu_m \in \mathcal{M}$. After the communication, both processors and the bus are released ($t_{comm_{ij}}$). $p_{wgb_{ij}}$ represents the waiting for bus and processors granting. $p_{ws_{ij}}$ indicates that the processors are ready to communicate. $p_{comc_{ij}}$ indicates that the communication was concluded. Lastly, $p_{rbuf_{ij}}$ represents the receiving buffer.

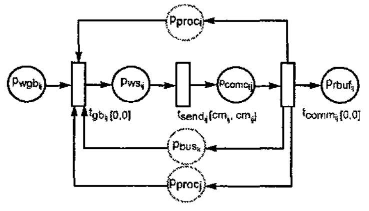


Figure 2. Building block.

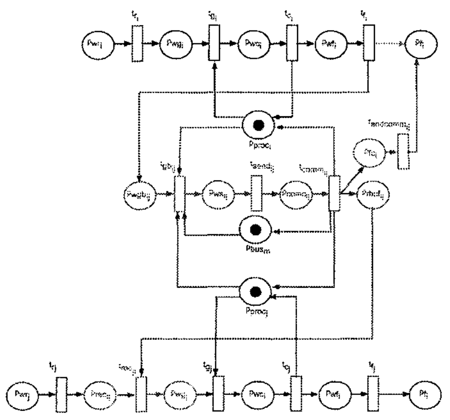


Figure 3. Modeling example.

Figure 3 applies the building block inter-processor communication for modeling the sending and receiving tasks τ_i and τ_j, respectively. It is worth observing that task τ_j has a refined place in order to consume the buffered message. More specifically, the place p_{wg_j} is substituted for the sequence $(p_{rec_{ij}}, t_{rec_{ij}}, p_{wg_j})$.

5.2 DISPATCHER OVERHEAD MODELING

The dispatcher overhead is captured in the grant-processor transition. When the task is non-preemptive, the timing interval of the grant-processor transition corresponds to the worst case execution time of the dispatcher. Since this is a simple solution, in the following presentation, the dispatcher overhead only considers preemptive tasks. When the task is preemptive, the model is slightly more complex. In this case, the proposed modeling adopts the TPN with priorities.

The proposed model considers two grant-processor transitions: grant-processor-with-overhead (t_{gw_i}) and grant-processor-without-overhead (t_{gwo_i}). As it can be seen in Figure 4, the timing interval ($[\alpha, \alpha]$) for transition t_{gw_i} models such timing overhead. Place $p_{proc_k T_i}$ states that task τ_i was the last executed task by the processor $proc_k$. The dispatcher overhead is considered in two situations: (1) when the next task to use the processor is different from the task that used the processor before; or (2) when a task instance ends its execution. The first situation is represented by the place $p_{proc_k T_i}$, where if such place is marked, it implies that the processor was lastly allocated to task τ_i. However, the second situation needs an explanation. Supposing that a task instance i of task τ_j ends its execution, and the following task to be executed is the task instance $i + 1$ of the same task τ_j. In this case, although the two instances are from the same task, the dispatcher calling is mandatory. As presented below, for solving this problem the model consider two final transitions, one that removes the marking in place $p_{proc_k T_i}$ and the other that does not.

In spite of this block may seem complicated, it is worth noting that this modeling is performed automatically by a tool. For more details, the interested reader is referred to [3].

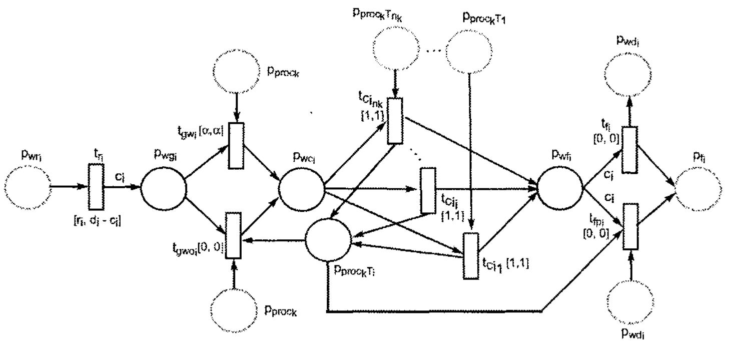


Figure 4. Building block dispatcher overhead.

6. ENERGY CONSUMPTION

Considering the Petri net task model, system energy consumption is associated with transitions representing dispatcher overhead (t_{gw}), task computation (t_c), and message transmission (t_{send}). Taking into account preemptive tasks, the energy consumption value of each computation time unit is equal to E_i/c_i, where c_i is the worst-case computation time (WCET) and E_i is the worst-case energy consumption of a task τ_i. Additionally, it is worth stating that the energy consumption value associated with t_{send} represents the sum of energy consumption values for sending and receiving the respective message.

The energy consumption for the dispatcher, the execution of tasks, and each message exchange must be known beforehand. In this work, the values were measured through a real prototype. The sum of energy dissipated in dispatcher, fired computation, and message transmission transitions results the total energy consumed during an execution of a schedule period.

The usage of the pre-runtime scheduler improves the accuracy of timing and energy consumption estimation. On the other hand, runtime approach cannot assure such an accuracy, since unpredictability of tasks arrival leads to more context-switching, increasing the energy consumption substantially.

The proposed method does not substitute other Lower-Power and Power-Aware techniques (e.g. dynamic voltage-scaling). Instead, the proposed method is a complement to such techniques, since the scheduling synthesis algorithm avoids unnecessary context-switching between tasks. Therefore, the generated schedule contains optimizations in terms of energy consumption.

7. PRE-RUNTIME SCHEDULING SYNTHESIS

This section shows a description of how to minimize the state space size, and the algorithm that implements the proposed method.

Minimizing State Space Size. The analysis based on the interleaving of actions is the fundamental point to be considered when analyzing state space explosion problem. Thus, the analysis of n concurrent actions has to verify all $n!$ interleaving possibilities, unless there are dependencies between these actions. This work proposes three ways for minimizing the state space size:

Modeling. The proposed method models dependencies between actions explicitly. **Partial-Order.** If actions can be executed in any order, such that the system always reaches the same state, these actions are *independent*. In other words, it does not matter in which order these are executed [4]. Independent actions are those that do not disable any other action, such as: arrival, release, precedence, processor releasing, and so on. This reduction method proposes to give a different *choice-priority* level for each class of independent activities. The dependent activities, like *processor granting*, have lowest choice-priority. Therefore, when changing from one state to another state, it is sufficient to analyze the class with highest choice-priority and pruning the other ones. **Removing Undesirable States.** Section 5.1 presents how to model undesirable error states, for instance, states that represent missed deadlines. The method proposed is of interest for schedules that do not reach any of these undesirable states. When generating the TLTS, transitions leading to undesirable error states have to be discarded.

Pre-Runtime Scheduling Algorithm. The algorithm adopted in this work is a depth-first search method on a TLTS. So, the TLTS is partially generated, according to the need. The *stop criterion* is obtained whenever the desirable final marking M^F is reached. For more information about this algorithm the interested reader is referred to [3].

8. CASE STUDY

In order to show the practical usability of the proposed approach in more details, a pulse-oximeter [5] is used as a case study. This equipment is responsible

```
scheduling-synthesis(S,$M^F$,TPN, $e_{max}$)
{
  if (S.M = $M^F$) return TRUE;
  tag(S);
  PT = pruning(firable($S$,$e_{max}$));
  if (|PT| = 0) return FALSE;
  for each ($\langle t,\theta \rangle$ $\in$ PT) {
     S'= fire(S, t, $\theta$);
     if (untagged(S') $\wedge$
        scheduling-synthesis (S',$M^F$,TPN,$e_{max}$)){
        add-in-trans-system (S,S',t,$\theta$);
        return TRUE;
     }
  }
  return FALSE;
}
```

Figure 5. Scheduling synthesis algorithm.

for measuring the oxygen saturation in the blood system using a non-invasive method. A pulse-oximeter may be used in many circumstances, like checking whether the oxygen saturation is lower or not than the acceptable, when a patient is sedated with anesthetics for a surgical procedure. This equipment is widely used in center care units (CCU).

For the sake of this paper, Table 1 shows the pulse oximeter task specification. In addition, the intertask relations are TE1 PRECEDES TE2, TE2 PRECEDES TE3, TE3 PRECEDES TE4, TA1 PRECEDES TA2, TA2 PRECEDES TA3, TA3 PRECEDES TA4, TA4 PRECEDES TA5, TA5 PRECEDES TA6, TA6 PRECEDES TA7, TA7 PRECEDES TA8. All tasks are preemptive. The dispatcher overhead is 200 microseconds and its respective worst-case energy consumption is 3958166,22 nJ. For this case study, the task time unit (TTU) adopted is 100 microseconds and the schedule energy constraint is 2 J .

Using the proposed approach, a feasible schedule was found in 45.9980 seconds, after visiting 42135 states. Due to lack of space, neither the Petri net model nor the found schedule is shown. The energy used by the found schedule totalizes 1,794,314,752.32 nJ ($\equiv$ 1.795J). For this example, the scheduling algorithm found a schedule without any context-switching. To conclude, the scheduling synthesis algorithm was executed on a AMD Duron 1200 Mhz, 256 MB RAM, OS Linux, and compiler GCC 3.3.2.

9. CONCLUSIONS

This paper proposed a pre-runtime scheduling approach considering energy and timing constraints for embedded hard real-time systems with multiple processors. Predictability is an important concern when considering time-critical systems. The scheduling approach presented guarantees that all critical tasks meet their deadlines and the schedule satisfies the energy constraint. In spite of the analysis technique (i.e. state space exploration) is not new, to the best of our present knowledge, there is no similar work reported that uses formal

Table 1. Task specification for the pulse Oximeter

TaskID	r	c	d	p	proc/bus	from	to	Energy
TE1	0	41	1000	2500	P1	-	-	8576,79 nJ
TE2	371	41	1000	2500	P1	-	-	52,48 nJ
TE3	576	41	1000	2500	P1	-	-	8576,79 nJ
TE4	947	41	1000	2500	P1	-	-	52,48 nJ
TE5	0	45	2000	2500	P1	-	-	222,30 nJ
TA1	0	41	5000	16000	P1	-	-	55,76 nJ
TA2	141	50	5000	16000	P1	-	-	222,50 nJ
TA3	191	41	5000	16000	P1	-	-	55,76 nJ
TA4	323	50	5000	16000	P1	-	-	222,50 nJ
TA5	382	41	5000	16000	P1	-	-	55,76 nJ
TA6	523	50	5000	16000	P1	-	-	222,50 nJ
TA7	573	41	5000	16000	P1	-	-	55,76 nJ
TA8	714	50	5000	16000	P1	-	-	222,50 nJ
TC1	764	60	5000	16000	P2	-	-	430,2 nJ
TC2	0	50	10000	16000	P2	-	-	444,00 nJ
TC3	0	50	10000	16000	P2	-	-	1117,5 nJ
TC4	764	45	10000	16000	P2	-	-	935,1 nJ
TC5	0	90	10000	16000	P2	-	-	935,1 nJ
TC6	0	90	10000	160000	P2	-	-	7089,3 nJ
TC7	0	90	10000	80000	P2	-	-	935,1 nJ
M1	-	7	-	-	bus1	TA8	TC1	8797,2 nJ

methods for modeling time-critical systems with energy constraints, considers arbitrary precedence/exclusion relations, and finds pre-runtime schedules. As future work, it is proposed to generate automatically the system source code from a feasible schedule, meeting not only timing constraints but also energy constraints.

REFERENCES

[1] T. Abdelzaher and K. Shin. Combined task and message scheduling in distributed real-time systems. *IEEE Trans. Parallel Distributed Systems*, 10(11):1179–1191, Nov 1999.

[2] T. A. AlEnawy and H. Aydin. On energy-constrained real-time scheduling. *Proceedings of the 16th EuroMicro Conference on Real-Time Systems (ECRTS 04)*, June 2004.

[3] R. Barreto. *A Time Petri Net-Based Methodology for Embedded Hard Real-Time Software Synthesis*. PhD Thesis, Centro de Informática - UFPE, April 2005.

[4] P. Godefroid. *Partial Order Methods for the Verification of Concurrent Systems*. PhD Thesis, University of Liege, Nov. 1994.

[5] M. Nogueira Oliveira Júnior. *Desenvolvimento de Um Protótipo para a Medida Não Invasiva da Saturação Arterial de Oxigênio em Humanos - Oxímetro de Pulso (in portuguese)*. MSc Thesis, Departamento de Biofísica e Radiobiologia, Universidade Federal de Pernambuco, August 1998.

[6] P. Merlin and D. J. Faber. Recoverability of communication protocols. *IEEE Trans. Comm.*, 24(9):1036–1043, Sep. 1976.

[7] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. PhD Thesis, MIT, May 1983.

[8] V. Swaminathan and K. Chakrabarty. Pruning-based, energy-optimal, deterministic i/o device scheduling for hard real-time systems. *ACM Trans. Embedded Comput. Syst. 4(1)*, pages 141–167, 2005.

[9] J. Xu and D. Parnas. Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. Soft. Engineering*, 16(3):360–369, March 1990.

A HIERARCHICAL APPROACH FOR POWER MANAGEMENT ON MOBILE EMBEDDED SYSTEMS*

Arliones Stevert Hoeller Junior, Lucas Francisco Wanner and Antônio Augusto Fröhlich
Laboratory for Software and Hardware Integration
Federal University of Santa Catarina
PO Box 476 - 88049-900 - Florianópolis, SC, Brazil
{ arliones,lucas,guto } @lisha.ufsc.br

Abstract Mobile Embedded Systems usually are simple, battery-powered systems with resource limitations. In some situations, their batteries lifetime becomes a primordial factor for reliability. Because of this, it is very important to handle power consumption of such devices in a non-restrictive and low-overhead way. This power management cannot restrict the wide variety of different low-power modes such devices often feature, thus allowing a wider system configurability. However, once in such devices processing and memory are often scarce, the power management strategy cannot compromise large amounts of system resources. In this paper we propose a simplified interface for power management of software and hardware components. The approach is based on the hierarchical organization of such components in a component-based operating system and allows power management of system components without the need for costly techniques or strategies. A case study including real implementations of system and application is presented to evaluate the technique and shows energy saves of almost 40% by just allowing applications to express when certain components are not being used.

Keywords: Power management, energy consumption management, embedded systems, mobile computing, low-power computing, embedded operating systems.

1. INTRODUCTION

In a mobile, battery-powered embedded system, battery lifetime is a primordial factor for reliability, thus making power management a very important

*This work was partially supported by FINEP (Financiadora de Estudos e Projetos) grant no. 01.04.0903.00.

issue for those systems. Embedded systems hardware usually provides some level of support for low-power operating modes. However, current software methodologies, techniques and standards for power management often focus on general purpose systems, where processing and memory overheads are mostly insignificant. Although these techniques have shown good results [1] [2][3], they impose extra processing costs or require advanced hardware resources, thus making them unusable in restricted embedded systems where processing and memory are very scarce.

Power management standards such as APM and ACPI were created focusing personal computers. These standards require either BIOS support or enough memory and processing capabilities for running a power management virtual machine. These requirements restrict their use to powerful embedded systems, which usually feature fast processors and large amounts of memory and make use of interactive operating systems such as LINUX and WINDOWS. The Advanced Power Management (APM) design assumed that the BIOS might make decisions regarding power consumption solely on monitoring the hardware. The lack of control of the operating system over the power management features of the BIOS, e. g., when the system will change power states, and the missing information on the BIOS level about the characteristics and requirements of the applications have been identified as the main drawbacks of APM [4].

The most important and established power management interface for general purpose computing systems is Advanced Configuration and Power Interface (ACPI), released in 1996 as a replacement of the previous industry standard for power management, Advanced Power Management (APM). ACPI identifies the operating system as the entity which has comprehensive knowledge about the hardware components and their usage and about the characteristics and behavior of the applications which access these hardware components. In contrast to APM, the operating system has full control over the operating modes and power management features of the hardware. ACPI is designed to not rely on the firmware and the exact implementation of the routines to access the hardware. The key to achieve this goal is the use of the ACPI source language (ASL), which is compiled to the machine language AML, similar to JAVA bytecode. Execution of the AML code is done by an interpreter in the operating system, inside a sandbox. This approach has several advantages: The interpretation of AML code prevents erroneous or malicious code to harm the system. AML code abstracts from the operating system as well as the platform or architecture it is executed on, so the burden of supporting drivers for several different operating systems or architectures is released from the hardware manufacturers [5]. However, ACPI abstracts the operating modes of the hardware in a way which may be too restrictive for embedded systems. The four device power modes defined by ACPI (D0 – D3) may be too coarse grained for embedded applications, once most components used in such systems usually feature several low-power operating modes. Furthermore, the use of an interpreted language to access hardware components, though having substantial advantages, poses requirements on the system which by far exceed the limited resources of most embedded devices.

In addition to these standards, several techniques were developed to allow an accurate control of power consumption for individual subsystems such as CPU, memory and I/O devices. These techniques use several strategies to define the best trade-off between performance and power consumption in each situation. For example, Dynamic Voltage and Frequency Scaling (DVFS) [6] is a strategy to slow down the CPU frequency or reduce its voltage supply and, consequently, save energy. Other strategies use event counter registers avaliable in some architectures to identify which parts of the hardware are in use and how these parts must behave to satisfy the system needs in terms of power consumption [1]. Although good results have been achieved, heuristics used to dynamicly guide the application of such techniques also impose extra processing costs or require extra hardware resources, thus becoming mostly unusable in deeply embedded systems.

In order to enable power management in embedded systems without incurring excessive overhead, we propose a simple and uniform interface for power management of software and hardware components. The mechanism behind this interface is based on the hierarchical organization of software and hardware components, and allows consistent power state migration of individual components, subsystems or the whole system. A case study is presented to demonstrate the use of the technique on a real implementation of this strategy in our component-based embedded operating system, EPOS.

This paper is organized as follows. Section 2 introduces the system power management interface for software and hardware components. Section 3 presents an application to exemplify the use of the power management interface. Section 4 gives an overview of related work. Section 5 finalizes.

2. POWER MANAGEMENT INTERFACE FOR SOFTWARE AND HARDWARE COMPONENTS

Power management policies in operating systems such as LINUX and WINDOWS dynamically analyze the behavior of applications and the system in order to determine when a hardware component should change its operating mode through an ACPI-compliant interface. However, most embedded systems cannot afford the overhead of such dynamic power management strategies. Furthermore, considering that a deeply embedded system is usually comprised by a single application, the best place to determine a power management strategy is in the application itself.

Embedded Parallel Operating System (EPOS) is a component-based, application-oriented operating system. In EPOS, high-level system abstractions, such as `File`, `Thread`, `Scheduler` and `Communicator`, are exported to applications through a component interface, and interact with the underlying hardware through hardware mediators. Through the system component hierarchy, each system abstraction and hardware mediator knows the state of its resources.

Through the definition of an uniform power management interface for system components, we allow the application programmer to change the power consumption status of each component individually. The interface is comprised by two methods: one to verify the component power state (`power()`) and other to change it (`power(user_desired_status)`). The mechanism behind this interface makes use of the hierarchical organization of software and hardware components in EPOS to allow consistent state migration among system operating modes.

Low-power hardware typically used in embedded systems often present a large set of operating modes. Enabling the use of all available operating modes is likely to enhance the system configurability, but might also increase the application complexity when managing the system power consumption. In order to solve this issue, we established a set of high-level definitions for the power consumption states, which will ease the application programmer from having to understand every hardware component in the system. As in ACPI [5], four universal modes were defined: `FULL`, `LIGHT`, `STANDBY` and `OFF`. These may be extended by system components whenever needed. When the device is fully operational, it is in the `FULL` state. The `LIGHT` state will consume less energy, but will grant the proper behavior of the device, it will probably incur in performance loss. In `STANDBY`, the device will have its behavior changed. This state will probably be a sleep mode. When `OFF` the device is switched off or switched to its smallest energy consumption state.

As embedded applications grow in complexity they make use of a large number of individual system components. As such, it may be impracticable for application programmers to take care of the power consumption of each component individually. To solve this problem we allow applications to manage individual subsystems or the system as a whole.

In order to exemplify how an entire subsystem may change its operating mode, we present a brief description of the EPOS communication subsystem. This subsystem is shown in Figure 1 and is basically comprised by four families of components: `Communicator`, `Channel`, `Network` and `NIC`. `NIC` is a family of hardware mediators, which abstracts the hardware device to the `Network` family. `Network` is responsible for abstracting the network (e. g., Ethernet, CAN, ATM, etc). `Channel` is responsible for inter-process communication and uses `Network` to build a logical communication channel through which messages are exchanged. Finally, a `Communicator` is an end-point for communications.

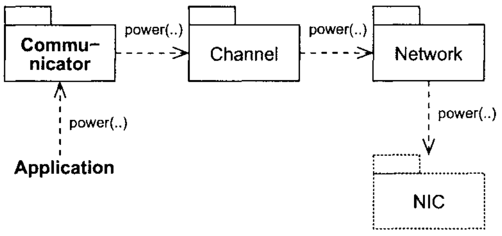


Figure 1. EPOS communication subsystem.

To grant portability of application code, the application programmer is suggested to use higher level abstractions, such as members of the `Communicator` family in the communication subsystem. In this context, our power management strategy must provide ways for the application programmer to change the power state of a communicator and this component must consistently propagate power state migrations to all software and hardware components in its hierarchy. For example, an implementation of a `Communicator` will use a `Channel` and probably an `Alarm` component to handle time-outs in the communication protocol. When the application executes a command asking the `Communicator` component to switch the operating mode to `OFF`, the `Communicator` will finish all started communications by flushing its buffers and waiting for all acknowledgment signals before shutting down other components in its hierarchy.

System-wide power management actions are handled by the `System` component in EPOS. The `System` component contains references to all subsystems used by the application. Thus, if an application wants to switch the whole system to a different operating mode, it may use the interface on the `System` component, which will propagate this request to all subsystems.

Figure 2 illustrates the system-wide power management interface may be accessed. It shows the components instantiated for a hypothetical sensing system. In this instance, the system is comprised by four components: the `CPU`, a `Communicator`, a `Sensor` and the `System` component. Each component has its own interface, which may be called by the application at anytime, and a set of power consumption levels. If the application wants to switch a specific subsystem to another power consumption level, it can access its components directly. If it wants to modify the whole system power consumption level, it may access the `System` component, which will propagate the modification through the system.

The main challenge identified on the development of power-aware components was the need for consistent operating mode propagation. This propa-

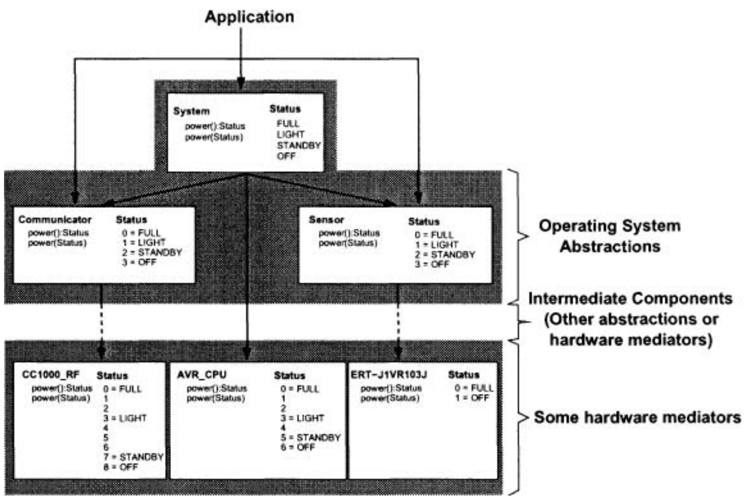


Figure 2. Accessing the power management interface.

gation must guarantee that no data will be lost and no unfinished actions will be interrupted. By letting each component handle its responsibilities (e. g., a `Communicator` flushing all its buffers and waiting for all acknowledgment signals) before propagating the power state propagation (e. g. shutting down `Alarm` and `Channel`), it is possible to guarantee consistent operating mode propagation of an entire subsystem.

In this strategy, the application programmer is expected to specify in the application when certain components aren't being used. It is done by issuing "`power`" commands to individual components, subsystems or the system. In order to free the application programmer from having to wake-up these components, such components are implemented to automatically switch on when a call is done to any of their methods. When this happens, components are switched to the their previous states or to the less energy spendable power state in which is possible to perform the required actions.

3. CASE STUDY: THERMOMETER

In order to demonstrate the usability of the defined interface, a thermometer was implemented using a simple prototype with a 10 kilo ohm thermistor connected to an analog-to-digital converter channel of an Atmel ATMega16 [7] microcontroller. The embedded application is presented in Figure 3. This application uses four system components: `System`, `Alarm`, `Thermometer` (member of the `Sentient` family [8]) and `UART`. The EPOS hierarchical organization binds, for example, the `Thermometer` abstraction with the micro-controller's analog-to-digital converter hardware mediator.

```
System sys;
Thermometer therm;
UART uart;

void alarm_handler() {
    uart.put(therm.get());
}

int main() {
    Handler_Function handler(&alarm_handler);
    Alarm alarm(1000000, &handler);

    while(1) {
        sys.power(STANDBY);
    }
}
```

Figure 3. The Thermometer application.

When the application starts, all used components are initialized by their constructors and a periodical event is registered with the `Alarm` component. The power state of the whole system is then switched to `STANDBY` through a power command issued to `System`. When this happens, the `System` component switches all system components, except for the `Alarm`, to *sleeping* modes. The `Alarm` component uses a timer to generate interrupts at a given frequency. Each time an interrupt occurs, the CPU wakes-up and the `Alarm` component handles all registered events currently due for execution. In this example, every two seconds the `Thermometer` and `UART` components are automatically switched on when accessed and a temperature reading is forwarded through the serial port. When all registered events are handled, the application continues normal execution on a loop which puts the `System` back in the `STANDBY` mode.

The graphics presented in Figure 4 show energy measurements for this application with and without system power management capabilities. Both graphics show the results of a mean between ten measurements. Each measurement was ten seconds long. In graphic (a) is noticed that system power consumption oscillates between 2.5 and 4 Watts. In graphic (b), the oscillation stays between 2 and 2.7 Watts. By calculating the integral of these graphics is possible to obtain energy consumption for these system instances during the time it was running. The results were 3.96 Joules for (a) and 2.45 Joules for (b), i.e., the system saved 38.1% of energy without compromising its functionality.

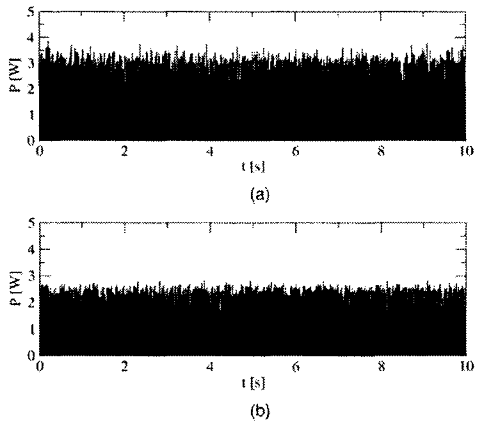


Figure 4. Power consumption for the Thermometer application *without (a) with (b)* power management.

4. RELATED WORK

TINYOS and MANTIS are embedded operating systems focused on wireless sensor networks. In these systems energy-awareness is mostly based on low-power MACs [9, 10] and multi-hop routing power scheduling [11, 12]. This makes sense in the context of wireless sensor networks, for a significant amount of energy is spent on the communication mechanism. Although this approach shows expressive results, it often focuses on the development of low-power components instead of power-aware ones. Another drawback in these systems is the lack of configurability and standardization of a configuration interface.

SPEU (System Properties Estimation with UML) [13] is an optimization tool which takes into account performance, system footprint and energy constraints to generate either a performance-efficient, size-efficient or energy-efficient system. These informations are extracted from an UML model of the embedded application. This model must include class and sequence diagrams, so the tool can estimate performance, code-size and energy consumption of each application. The generated system is a Java software and is intended to run over the FEMTOJAVA [14] soft-core processor. Once SPEU only takes into account the UML diagrams, its estimations show errors as big as 85%, making it only useful to compare different design decisions. It also lacks configurability, once the optimization process is only guided by one variable, i. e., if the application programmer's design choice is performance, the system will never enter power-aware states, even if it is not using certain devices. This certainly limits its use in real-world applications.

IMPACCT (which stands for Integrated Management of Power-Aware Computing and Communication Technologies) [15] is a system-level tool for exploring power/performance tradeoffs by means of power-aware scheduling and architectural configuration. The idea behind the IMPACCT system is the embedded application analysis through a timing simulation to define the widest possible dynamic range of power/performance tradeoffs and the power mode in which each component should operate over time. This tool chain also includes a power-aware scheduler implementation for hard real-time systems. IMPACCT tools deliver a very interesting way to configure the power-aware scheduler and the power-modes of an embedded system, but is far from delivering a fast prototyping environment.

5. CONCLUSION

In this paper we presented an strategy to enable application-driven power management in deeply embedded systems. In order to achieve this goal we allowed application programmers to express when certain components are not being used. This is expressed through a simple power management interface which allows power mode switching of system components, subsystems or the system as a whole, making all combinations of components operating modes feasible. By using the hierarchical architecture by which system components are organized in our system, effective power management was achieved for deeply embedded systems without the need for costly techniques or strategies, thus incurring in no unnecessary processing or memory overheads.

A case study using a 8-bit microcontroller to monitor temperature in an indoor ambient showed that almost 40% of energy could be saved when using this strategy.

ACKNOWLEDGMENTS

Authors would like to thank Augusto Born de Oliveira, Hugo Marcondes and Rafael Cancian from LISHA for very helpful discussion. We also would like to thank the Department of Computer Sciences 4 at Friedrich-Alexander Universität (Germany), its head Prof. Schröder-Preikschat and Andreas Weissel for providing equipment and some advise for this work.

REFERENCES

[1] Bellosa, Frank, Weissel, Andreas, Waitz, Martin, and Kellner, Simon (2003). Event-driven energy accounting for dynamic thermal management. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, pages 04–1 – 04–10, New Orleans, USA.

[2] Sorber, Jacob, Banerjee, Nilanjan, Corner, Mark D., and Rollins, Sami (2005). Turducken: hierarchical power management for mobile devices. In *MobiSys '05: Proceedings of the*

3rd international conference on Mobile systems, applications, and services, pages 261–274, New York, NY, USA. ACM Press.

[3] Pering, T. and Broderson, R. (1998). Dynamic voltage scaling and the design of a low-power microprocessor system. In *Proceedings of the International Symposium on Computer Architecture ISCA '98.*

[4] Intel Corp. and Microsoft Corp. (1996). *Advanced Power Management (APM) BIOS Interface Specification*, 1.2 edition.

[5] Hewlett-Packard Corp., Intel Corp., Microsoft Corp., Phoenix Technologies Ltd., and Toshiba Corp. (2004). *Advanced Configuration and Power Interface Specification*, 3.0 edition.

[6] Benini, Luca, Bogliolo, Alessandro, and Micheli, Giovanni De (1998). Dynamic power management of electronic systems. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 696–702, New York, NY, USA. ACM Press.

[7] Atmel Corp. (2004). *ATMega16L Datasheet.* San Jose, CA, 2466j edition.

[8] Wanner, Lucas Francisco, Junior, Arliones Stevert Hoeller, Polpeta, Fauze Valerio, and Frohlich, Antonio Augusto (2005). Operating system support for handling heterogeneity in wireless sensor networks. In *Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation*, Catania, Italy. IEEE.

[9] Polastre, Joseph, Szewczyk, Robert, Sharp, Cory, and Culler, David (2004). The mote revolution: Low power wireless sensor network devices. In *Proceedings of Hot Chips 16: A Symposium on High Performance Chips.*

[10] Sheth, Anmol and Han, Richard (2004). Shush: A mac protocol for transmit power controlled wireless networks. Technical Report CU-CS-986-04, Department of Computer Science, University of Colorado, Boulder.

[11] Hohlt, Barbara, Doherty, Lance, and Brewer, Eric (2004). Flexible power scheduling for sensor networks. In *Proceedings of The Third International Symposium on Information Processing in Sensor Networks*, pages 205–214, Berkley, USA. IEEE.

[12] Sheth, Anmol and Han, Richard (2003). Adaptive power control and selective radio activation for low-power infrastructure-mode 802.11 lans. In *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops*, pages 797–802, Providence, USA. IEEE.

[13] da S. Oliveira, Marcion F., de Brisolara, Lisiane B., Carro, Luigi, and Wagner, Flávio R. (2005). An embedded sw design exploration approach based on xml estimation tools. In Rettberg, Achim, mauro C. Zanella, and Rammig, Franz J., editors, *From Specification to Embedded Systems Application*, pages 45–54, Manaus, Brazil. IFIP, Springer.

[14] Ito, S.A., Carro, L., and Jacobi, R.P. (2001). Making java work for microcontroller applications. *IEEE Design and Test of Computers*, 18(5):100–110.

[15] Chou, Pai H., Liu, Jinfeng, Li, Dexin, and Bagherzadeh, Nader (2002). Impacct: Methodology and tools for power-aware embedded systems. *DESIGN AUTOMATION FOR EMBEDDED SYSTEMS, Special Issue on Design Methodologies and Tools for Real-Time Embedded Systems*, 7(3):205–232.